SuperLU

3.1

Generated by Doxygen 1.5.9

Sat Aug 29 03:52:29 2009

Contents

1 Data Structure Index **1**

1.1 Data Structures 1

2 File Index **3**

2.1 File List 3

3 Data Structure Documentation **7**

3.1 Colamd_Col_struct Struct Reference 7

3.1.1 Field Documentation 8

3.1.1.1 degree_next 8

3.1.1.2 hash 8

3.1.1.3 hash_next 8

3.1.1.4 headhash 8

3.1.1.5 length 8

3.1.1.6 order 8

3.1.1.7 parent 8

3.1.1.8 prev 8

3.1.1.9 score 8

3.1.1.10 shared1 8

3.1.1.11 shared2 8

3.1.1.12 shared3 8

3.1.1.13 shared4 8

3.1.1.14 start 8

3.1.1.15 thickness 8

3.2 Colamd_Row_struct Struct Reference 9

3.2.1 Field Documentation 9

3.2.1.1 degree 9

3.2.1.2 first_column 9

3.2.1.3 length 9

3.2.1.4 mark . . . 9
3.2.1.5 p . . . 9
3.2.1.6 shared1 . . . 9
3.2.1.7 shared2 . . . 9
3.2.1.8 start . . . 9
3.3 ColInfo_struct Struct Reference . . . 10
3.3.1 Field Documentation . . . 11
3.3.1.1 degree_next . . . 11
3.3.1.2 hash . . . 11
3.3.1.3 hash_next . . . 11
3.3.1.4 headhash . . . 11
3.3.1.5 length . . . 11
3.3.1.6 order . . . 11
3.3.1.7 parent . . . 11
3.3.1.8 prev . . . 11
3.3.1.9 score . . . 11
3.3.1.10 shared1 . . . 11
3.3.1.11 shared2 . . . 11
3.3.1.12 shared3 . . . 11
3.3.1.13 shared4 . . . 11
3.3.1.14 start . . . 11
3.3.1.15 thickness . . . 11
3.4 complex Struct Reference . . . 12
3.4.1 Field Documentation . . . 12
3.4.1.1 i . . . 12
3.4.1.2 r . . . 12
3.5 DNformat Struct Reference . . . 13
3.5.1 Field Documentation . . . 13
3.5.1.1 lda . . . 13
3.5.1.2 nzval . . . 13
3.6 doublecomplex Struct Reference . . . 14
3.6.1 Field Documentation . . . 14
3.6.1.1 i . . . 14
3.6.1.2 r . . . 14
3.7 e_node Struct Reference . . . 15
3.7.1 Field Documentation . . . 15

3.7.1.1 mem 15
3.7.1.2 size 15
3.8 GlobalLU_t Struct Reference 16
3.8.1 Field Documentation 18
3.8.1.1 lsub 18
3.8.1.2 lusup 18
3.8.1.3 lusup 18
3.8.1.4 lusup 18
3.8.1.5 lusup 18
3.8.1.6 MemModel 18
3.8.1.7 n 18
3.8.1.8 nzlmax 18
3.8.1.9 nzlumax 18
3.8.1.10 nzumax 18
3.8.1.11 supno 18
3.8.1.12 ucol 18
3.8.1.13 ucol 18
3.8.1.14 ucol 18
3.8.1.15 ucol 18
3.8.1.16 usub 18
3.8.1.17 xlsub 18
3.8.1.18 xlusup 18
3.8.1.19 xsup 18
3.8.1.20 xusub 18
3.9 LU_stack_t Struct Reference 19
3.9.1 Field Documentation 19
3.9.1.1 array 19
3.9.1.2 size 19
3.9.1.3 top1 19
3.9.1.4 top2 19
3.9.1.5 used 19
3.10 mem_usage_t Struct Reference 20
3.10.1 Field Documentation 20
3.10.1.1 expansions 20
3.10.1.2 for_lu 20
3.10.1.3 total_needed 20

3.11 NCformat Struct Reference 21
3.11.1 Field Documentation 21
3.11.1.1 colptr 21
3.11.1.2 nnz 21
3.11.1.3 nzval 21
3.11.1.4 rowind 21
3.12 NCPformat Struct Reference 22
3.12.1 Field Documentation 22
3.12.1.1 colbeg 22
3.12.1.2 colend 22
3.12.1.3 nnz 22
3.12.1.4 nzval 22
3.12.1.5 rowind 22
3.13 NRformat Struct Reference 23
3.13.1 Field Documentation 23
3.13.1.1 colind 23
3.13.1.2 nnz 23
3.13.1.3 nzval 23
3.13.1.4 rowptr 23
3.14 NRformat_loc Struct Reference 24
3.14.1 Field Documentation 24
3.14.1.1 colind 24
3.14.1.2 fst_row 24
3.14.1.3 m_loc 24
3.14.1.4 nnz_loc 24
3.14.1.5 nzval 24
3.14.1.6 rowptr 24
3.15 RowInfo_struct Struct Reference 25
3.15.1 Field Documentation 25
3.15.1.1 degree 25
3.15.1.2 first_column 25
3.15.1.3 length 25
3.15.1.4 mark 25
3.15.1.5 p 25
3.15.1.6 shared1 25
3.15.1.7 shared2 25

3.15.1.8 start 25
3.16 SCformat Struct Reference 26
3.16.1 Field Documentation 26
3.16.1.1 col_to_sup 26
3.16.1.2 nnz 26
3.16.1.3 nsuper 26
3.16.1.4 nzval 26
3.16.1.5 nzval_colptr 26
3.16.1.6 rowind 26
3.16.1.7 rowind_colptr 26
3.16.1.8 sup_to_col 26
3.17 SCPformat Struct Reference 27
3.17.1 Field Documentation 27
3.17.1.1 col_to_sup 27
3.17.1.2 nnz 27
3.17.1.3 nsuper 27
3.17.1.4 nzval 27
3.17.1.5 nzval_colbeg 27
3.17.1.6 nzval_colend 27
3.17.1.7 rowind 27
3.17.1.8 rowind_colbeg 27
3.17.1.9 rowind_colend 27
3.17.1.10 sup_to_colbeg 27
3.17.1.11 sup_to_colend 27
3.18 superlu_options_t Struct Reference 28
3.18.1 Field Documentation 29
3.18.1.1 ColPerm 29
3.18.1.2 ConditionNumber 29
3.18.1.3 DiagPivotThresh 29
3.18.1.4 Equil 29
3.18.1.5 Fact 29
3.18.1.6 IterRefine 29
3.18.1.7 PivotGrowth 29
3.18.1.8 PrintStat 29
3.18.1.9 RefineInitialized 29
3.18.1.10 ReplaceTinyPivot 29

3.18.1.11 RowPerm . . . 29
3.18.1.12 SolveInitialized . . . 29
3.18.1.13 SymmetricMode . . . 29
3.18.1.14 Trans . . . 29
3.19 SuperLUStat_t Struct Reference . . . 30
3.19.1 Field Documentation . . . 30
3.19.1.1 ops . . . 30
3.19.1.2 panel_histo . . . 30
3.19.1.3 RefineSteps . . . 30
3.19.1.4 TinyPivots . . . 30
3.19.1.5 utime . . . 30
3.20 SuperMatrix Struct Reference . . . 31
3.20.1 Field Documentation . . . 31
3.20.1.1 Dtype . . . 31
3.20.1.2 Mtype . . . 31
3.20.1.3 ncol . . . 31
3.20.1.4 nrow . . . 31
3.20.1.5 Store . . . 31
3.20.1.6 Stype . . . 31

4 File Documentation 33

4.1 EXAMPLE/clinsol.c File Reference . . . 33
4.1.1 Function Documentation . . . 35
4.1.1.1 main . . . 35
4.2 EXAMPLE/clinsol1.c File Reference . . . 36
4.2.1 Function Documentation . . . 38
4.2.1.1 main . . . 38
4.3 EXAMPLE/clinsolx.c File Reference . . . 39
4.3.1 Function Documentation . . . 41
4.3.1.1 main . . . 41
4.3.1.2 parse_command_line . . . 42
4.4 EXAMPLE/clinsolx1.c File Reference . . . 43
4.4.1 Function Documentation . . . 45
4.4.1.1 main . . . 45
4.4.1.2 parse_command_line . . . 46
4.5 EXAMPLE/clinsolx2.c File Reference . . . 47
4.5.1 Function Documentation . . . 49

4.5.1.1 main 49
4.5.1.2 parse_command_line 50
4.6 EXAMPLE/dlinsol.c File Reference 51
4.6.1 Function Documentation 53
4.6.1.1 main 53
4.7 EXAMPLE/dlinsol1.c File Reference 54
4.7.1 Function Documentation 56
4.7.1.1 main 56
4.8 EXAMPLE/dlinsolx.c File Reference 57
4.8.1 Function Documentation 59
4.8.1.1 main 59
4.8.1.2 parse_command_line 60
4.9 EXAMPLE/dlinsolx1.c File Reference 61
4.9.1 Function Documentation 63
4.9.1.1 main 63
4.9.1.2 parse_command_line 64
4.10 EXAMPLE/dlinsolx2.c File Reference 65
4.10.1 Function Documentation 67
4.10.1.1 main 67
4.10.1.2 parse_command_line 68
4.11 EXAMPLE/dreadtriple.c File Reference 69
4.11.1 Function Documentation 69
4.11.1.1 dreadrhs 69
4.11.1.2 dreadtriple 69
4.12 EXAMPLE/slinsol.c File Reference 70
4.12.1 Function Documentation 72
4.12.1.1 main 72
4.13 EXAMPLE/slinsol1.c File Reference 73
4.13.1 Function Documentation 75
4.13.1.1 main 75
4.14 EXAMPLE/slinsolx.c File Reference 76
4.14.1 Function Documentation 78
4.14.1.1 main 78
4.14.1.2 parse_command_line 79
4.15 EXAMPLE/slinsolx1.c File Reference 80
4.15.1 Function Documentation 82

4.15.1.1 main 82
4.15.1.2 parse_command_line 83
4.16 EXAMPLE/slinsolx2.c File Reference 84
4.16.1 Function Documentation 86
4.16.1.1 main 86
4.16.1.2 parse_command_line 87
4.17 EXAMPLE/superlu.c File Reference 88
4.17.1 Detailed Description 88
4.17.2 Function Documentation 90
4.17.2.1 main 90
4.18 EXAMPLE/zlinsol.c File Reference 91
4.18.1 Function Documentation 93
4.18.1.1 main 93
4.19 EXAMPLE/zlinsol1.c File Reference 94
4.19.1 Function Documentation 96
4.19.1.1 main 96
4.20 EXAMPLE/zlinsolx.c File Reference 97
4.20.1 Function Documentation 99
4.20.1.1 main 99
4.20.1.2 parse_command_line 100
4.21 EXAMPLE/zlinsolx1.c File Reference 101
4.21.1 Function Documentation 103
4.21.1.1 main 103
4.21.1.2 parse_command_line 104
4.22 EXAMPLE/zlinsolx2.c File Reference 105
4.22.1 Function Documentation 107
4.22.1.1 main 107
4.22.1.2 parse_command_line 108
4.23 EXAMPLE/zreadtriple.c File Reference 109
4.23.1 Function Documentation 109
4.23.1.1 zreadtriple 109
4.24 SRC/ccolumn_bmod.c File Reference 110
4.24.1 Detailed Description 110
4.24.2 Function Documentation 111
4.24.2.1 ccolumn_bmod 111
4.24.2.2 clsolve 111

4.24.2.3 cmatvec . . . 112
4.24.2.4 cusolve . . . 112
4.25 SRC/ccolumn_dfs.c File Reference . . . 113
4.25.1 Detailed Description . . . 113
4.25.2 Define Documentation . . . 114
4.25.2.1 T2_SUPER . . . 114
4.25.3 Function Documentation . . . 114
4.25.3.1 ccolumn_dfs . . . 114
4.26 SRC/ccopy_to_ucol.c File Reference . . . 116
4.26.1 Detailed Description . . . 116
4.26.2 Function Documentation . . . 117
4.26.2.1 ccopy_to_ucol . . . 117
4.27 SRC/cgscon.c File Reference . . . 118
4.27.1 Detailed Description . . . 118
4.27.2 Function Documentation . . . 118
4.27.2.1 cgscon . . . 118
4.28 SRC/cgsequ.c File Reference . . . 121
4.28.1 Detailed Description . . . 121
4.28.2 Function Documentation . . . 121
4.28.2.1 cgsequ . . . 121
4.29 SRC/cgsrfs.c File Reference . . . 124
4.29.1 Detailed Description . . . 124
4.29.2 Define Documentation . . . 125
4.29.2.1 ITMAX . . . 125
4.29.3 Function Documentation . . . 125
4.29.3.1 cgsrfs . . . 125
4.30 SRC/cgssv.c File Reference . . . 128
4.30.1 Detailed Description . . . 128
4.30.2 Function Documentation . . . 128
4.30.2.1 cgssv . . . 128
4.31 SRC/cgssvx.c File Reference . . . 133
4.31.1 Detailed Description . . . 133
4.31.2 Function Documentation . . . 133
4.31.2.1 cgssvx . . . 133
4.32 SRC/cgstrf.c File Reference . . . 142
4.32.1 Detailed Description . . . 142

4.32.2 Function Documentation 142
4.32.2.1 cgstrf 142
4.33 SRC/cgstrs.c File Reference 148
4.33.1 Detailed Description 148
4.33.2 Function Documentation 149
4.33.2.1 cgstrs 149
4.33.2.2 clsolve 150
4.33.2.3 cmatvec 150
4.33.2.4 cprint_soln 150
4.33.2.5 cusolve 151
4.34 SRC/clacon.c File Reference 152
4.34.1 Detailed Description 152
4.34.2 Function Documentation 152
4.34.2.1 clacon_ 152
4.35 SRC/clangs.c File Reference 154
4.35.1 Detailed Description 154
4.35.2 Function Documentation 154
4.35.2.1 clangs 154
4.36 SRC/claqgs.c File Reference 156
4.36.1 Detailed Description 156
4.36.2 Define Documentation 156
4.36.2.1 THRESH 156
4.36.3 Function Documentation 156
4.36.3.1 claqgs 156
4.37 SRC/cmemory.c File Reference 159
4.37.1 Detailed Description 160
4.37.2 Define Documentation 161
4.37.2.1 DoubleAlign 161
4.37.2.2 GluIntArray 161
4.37.2.3 NO_MEMTYPE 161
4.37.2.4 NotDoubleAlign 161
4.37.2.5 Reduce 161
4.37.2.6 StackFull 161
4.37.2.7 TempSpace 161
4.37.3 Typedef Documentation 161
4.37.3.1 ExpHeader 161

4.37.4 Function Documentation . . . 161
4.37.4.1 callocateA . . . 161
4.37.4.2 cexpand . . . 162
4.37.4.3 cLUMemInit . . . 162
4.37.4.4 cLUMemXpand . . . 163
4.37.4.5 cLUWorkFree . . . 164
4.37.4.6 cLUWorkInit . . . 164
4.37.4.7 cmemory_usage . . . 165
4.37.4.8 complexCalloc . . . 165
4.37.4.9 complexMalloc . . . 165
4.37.4.10 copy_mem_complex . . . 166
4.37.4.11 copy_mem_int . . . 167
4.37.4.12 cQuerySpace . . . 167
4.37.4.13 cSetRWork . . . 168
4.37.4.14 cSetupSpace . . . 168
4.37.4.15 cStackCompress . . . 169
4.37.4.16 cuser_free . . . 169
4.37.4.17 cuser_malloc . . . 169
4.37.4.18 user_bcopy . . . 170
4.37.5 Variable Documentation . . . 170
4.37.5.1 expanders . . . 170
4.37.5.2 no_expand . . . 170
4.37.5.3 stack . . . 170
4.38 SRC/cmyblas2.c File Reference . . . 171
4.38.1 Detailed Description . . . 171
4.38.2 Function Documentation . . . 171
4.38.2.1 clsolve . . . 171
4.38.2.2 cmatvec . . . 172
4.38.2.3 cusolve . . . 172
4.39 SRC/colamd.c File Reference . . . 173
4.39.1 Detailed Description . . . 174
4.39.2 Define Documentation . . . 187
4.39.2.1 ALIVE . . . 187
4.39.2.2 ASSERT . . . 187
4.39.2.3 COL_IS_ALIVE . . . 187
4.39.2.4 COL_IS_DEAD . . . 187

4.39.2.5 COL_IS_DEAD_PRINCIPAL 187
4.39.2.6 DEAD 187
4.39.2.7 DEAD_NON_PRINCIPAL 187
4.39.2.8 DEAD_PRINCIPAL 187
4.39.2.9 DEBUG0 187
4.39.2.10 DEBUG1 187
4.39.2.11 DEBUG2 187
4.39.2.12 DEBUG3 187
4.39.2.13 DEBUG4 187
4.39.2.14 EMPTY 187
4.39.2.15 FALSE 187
4.39.2.16 INDEX 187
4.39.2.17 KILL_NON_PRINCIPAL_COL 187
4.39.2.18 KILL_PRINCIPAL_COL 187
4.39.2.19 KILL_ROW 187
4.39.2.20 MAX 187
4.39.2.21 MIN 187
4.39.2.22 ONES_COMPLEMENT 187
4.39.2.23 PRINTF 187
4.39.2.24 PRIVATE 187
4.39.2.25 PUBLIC 187
4.39.2.26 ROW_IS_ALIVE 187
4.39.2.27 ROW_IS_DEAD 187
4.39.2.28 ROW_IS_MARKED_DEAD 187
4.39.2.29 TRUE 187
4.39.3 Function Documentation 187
4.39.3.1 clear_mark 187
4.39.3.2 colamd 188
4.39.3.3 colamd_recommended 189
4.39.3.4 colamd_report 189
4.39.3.5 colamd_set_defaults 189
4.39.3.6 detect_super_cols 190
4.39.3.7 find_ordering 190
4.39.3.8 garbage_collection 191
4.39.3.9 init_rows_cols 191
4.39.3.10 init_scoring 192

4.39.3.11 order_children 192
4.39.3.12 print_report 192
4.39.3.13 symamd 193
4.39.3.14 symamd_report 193
4.40 SRC/colamd.h File Reference 194
4.40.1 Detailed Description 195
4.40.2 Define Documentation 197
4.40.2.1 COLAMD_C 197
4.40.2.2 COLAMD_DEFRAG_COUNT 197
4.40.2.3 COLAMD_DENSE_COL 197
4.40.2.4 COLAMD_DENSE_ROW 197
4.40.2.5 COLAMD_ERROR_A_not_present 197
4.40.2.6 COLAMD_ERROR_A_too_small 197
4.40.2.7 COLAMD_ERROR_col_length_negative 197
4.40.2.8 COLAMD_ERROR_internal_error 197
4.40.2.9 COLAMD_ERROR_ncol_negative 197
4.40.2.10 COLAMD_ERROR_nnz_negative 197
4.40.2.11 COLAMD_ERROR_nrow_negative 197
4.40.2.12 COLAMD_ERROR_out_of_memory 197
4.40.2.13 COLAMD_ERROR_p0_nonzero 197
4.40.2.14 COLAMD_ERROR_p_not_present 197
4.40.2.15 COLAMD_ERROR_row_index_out_of_bounds 197
4.40.2.16 COLAMD_INFO1 197
4.40.2.17 COLAMD_INFO2 197
4.40.2.18 COLAMD_INFO3 197
4.40.2.19 COLAMD_KNOBS 197
4.40.2.20 COLAMD_OK 197
4.40.2.21 COLAMD_OK_BUT_JUMBLED 197
4.40.2.22 COLAMD_R 197
4.40.2.23 COLAMD_RECOMMENDED 197
4.40.2.24 COLAMD_STATS 198
4.40.2.25 COLAMD_STATUS 198
4.40.3 Typedef Documentation 198
4.40.3.1 Colamd_Col 198
4.40.3.2 Colamd_Row 198
4.40.4 Function Documentation 198

4.40.4.1 colamd . . . 198
4.40.4.2 colamd_recommended . . . 199
4.40.4.3 colamd_report . . . 199
4.40.4.4 colamd_set_defaults . . . 199
4.40.4.5 symamd . . . 199
4.40.4.6 symamd_report . . . 199
4.41 SRC/cpanel_bmod.c File Reference . . . 200
4.41.1 Detailed Description . . . 200
4.41.2 Function Documentation . . . 201
4.41.2.1 ccheck_tempv . . . 201
4.41.2.2 clsolve . . . 201
4.41.2.3 cmatvec . . . 201
4.41.2.4 cpanel_bmod . . . 201
4.42 SRC/cpanel_dfs.c File Reference . . . 202
4.42.1 Detailed Description . . . 202
4.42.2 Function Documentation . . . 202
4.42.2.1 cpanel_dfs . . . 202
4.43 SRC/cpivotgrowth.c File Reference . . . 204
4.43.1 Detailed Description . . . 204
4.43.2 Function Documentation . . . 204
4.43.2.1 cPivotGrowth . . . 204
4.44 SRC/cpivotL.c File Reference . . . 206
4.44.1 Detailed Description . . . 206
4.44.2 Function Documentation . . . 206
4.44.2.1 cpivotL . . . 206
4.45 SRC/cpruneL.c File Reference . . . 208
4.45.1 Detailed Description . . . 208
4.45.2 Function Documentation . . . 208
4.45.2.1 cpruneL . . . 208
4.46 SRC/creadhb.c File Reference . . . 210
4.46.1 Detailed Description . . . 210
4.46.2 Function Documentation . . . 212
4.46.2.1 cDumpLine . . . 212
4.46.2.2 cParseFloatFormat . . . 212
4.46.2.3 cParseIntFormat . . . 212
4.46.2.4 creadhb . . . 212

4.46.2.5 cReadValues . . . 213
4.46.2.6 cReadVector . . . 213
4.47 SRC/csnode_bmod.c File Reference . . . 214
4.47.1 Detailed Description . . . 214
4.47.2 Function Documentation . . . 215
4.47.2.1 csnode_bmod . . . 215
4.48 SRC/csnode_dfs.c File Reference . . . 216
4.48.1 Detailed Description . . . 216
4.48.2 Function Documentation . . . 216
4.48.2.1 csnode_dfs . . . 216
4.49 SRC/csp_blas2.c File Reference . . . 218
4.49.1 Detailed Description . . . 218
4.49.2 Function Documentation . . . 219
4.49.2.1 clsolve . . . 219
4.49.2.2 cmatvec . . . 219
4.49.2.3 cusolve . . . 219
4.49.2.4 sp_cgemv . . . 219
4.49.2.5 sp_ctrsv . . . 220
4.50 SRC/csp_blas3.c File Reference . . . 223
4.50.1 Detailed Description . . . 223
4.50.2 Function Documentation . . . 223
4.50.2.1 sp_cgemm . . . 223
4.51 SRC/cutil.c File Reference . . . 226
4.51.1 Detailed Description . . . 227
4.51.2 Function Documentation . . . 227
4.51.2.1 ccheck_tempv . . . 227
4.51.2.2 cCompRow_to_CompCol . . . 227
4.51.2.3 cCopy_CompCol_Matrix . . . 228
4.51.2.4 cCopy_Dense_Matrix . . . 228
4.51.2.5 cCreate_CompCol_Matrix . . . 228
4.51.2.6 cCreate_CompRow_Matrix . . . 228
4.51.2.7 cCreate_Dense_Matrix . . . 228
4.51.2.8 cCreate_SuperNode_Matrix . . . 228
4.51.2.9 cfill . . . 228
4.51.2.10 cFillRHS . . . 229
4.51.2.11 cGenXtrue . . . 229

4.51.2.12 cinf_norm_error 229
4.51.2.13 cPrint_CompCol_Matrix 229
4.51.2.14 cPrint_Dense_Matrix 229
4.51.2.15 cprint_lu_col 229
4.51.2.16 cPrint_SuperNode_Matrix 230
4.51.2.17 cPrintPerf 230
4.51.2.18 print_complex_vec 230
4.52 SRC/dcolumn_bmod.c File Reference 231
4.52.1 Detailed Description 231
4.52.2 Function Documentation 232
4.52.2.1 dcolumn_bmod 232
4.52.2.2 dlsolve 232
4.52.2.3 dmatvec 233
4.52.2.4 dusolve 233
4.53 SRC/dcolumn_dfs.c File Reference 234
4.53.1 Detailed Description 234
4.53.2 Define Documentation 235
4.53.2.1 T2_SUPER 235
4.53.3 Function Documentation 235
4.53.3.1 dcolumn_dfs 235
4.54 SRC/dcomplex.c File Reference 237
4.54.1 Detailed Description 237
4.54.2 Function Documentation 238
4.54.2.1 d_cnjg 238
4.54.2.2 d_imag 238
4.54.2.3 z_abs 238
4.54.2.4 z_abs1 238
4.54.2.5 z_div 238
4.54.2.6 z_exp 238
4.55 SRC/dcopy_to_ucol.c File Reference 239
4.55.1 Detailed Description 239
4.55.2 Function Documentation 240
4.55.2.1 dcopy_to_ucol 240
4.56 SRC/dGetDiagU.c File Reference 241
4.56.1 Detailed Description 241
4.56.2 Function Documentation 242

4.56.2.1 dGetDiagU . . . 242
4.57 SRC/dgscon.c File Reference . . . 243
4.57.1 Detailed Description . . . 243
4.57.2 Function Documentation . . . 243
4.57.2.1 dgscon . . . 243
4.58 SRC/dgsequ.c File Reference . . . 246
4.58.1 Detailed Description . . . 246
4.58.2 Function Documentation . . . 246
4.58.2.1 dgsequ . . . 246
4.59 SRC/dgsrfs.c File Reference . . . 249
4.59.1 Detailed Description . . . 249
4.59.2 Define Documentation . . . 250
4.59.2.1 ITMAX . . . 250
4.59.3 Function Documentation . . . 250
4.59.3.1 dgsrfs . . . 250
4.60 SRC/dgssv.c File Reference . . . 253
4.60.1 Detailed Description . . . 253
4.60.2 Function Documentation . . . 253
4.60.2.1 dgssv . . . 253
4.61 SRC/dgssvx.c File Reference . . . 258
4.61.1 Detailed Description . . . 258
4.61.2 Function Documentation . . . 258
4.61.2.1 dgssvx . . . 258
4.62 SRC/dgstrf.c File Reference . . . 267
4.62.1 Detailed Description . . . 267
4.62.2 Function Documentation . . . 267
4.62.2.1 dgstrf . . . 267
4.63 SRC/dgstrs.c File Reference . . . 273
4.63.1 Detailed Description . . . 273
4.63.2 Function Documentation . . . 274
4.63.2.1 dgstrs . . . 274
4.63.2.2 dlsolve . . . 275
4.63.2.3 dmatvec . . . 275
4.63.2.4 dprint_soln . . . 275
4.63.2.5 dusolve . . . 276
4.64 SRC/dgstrsL.c File Reference . . . 277

4.64.1 Detailed Description 277
4.64.2 Function Documentation 278
4.64.2.1 dgstrsL 278
4.64.2.2 dlsolve 279
4.64.2.3 dmatvec 279
4.64.2.4 dprint_soln 279
4.64.2.5 dusolve 279
4.65 SRC/dgstrsU.c File Reference 280
4.65.1 Detailed Description 280
4.65.2 Function Documentation 281
4.65.2.1 dgstrsU 281
4.65.2.2 dlsolve 282
4.65.2.3 dmatvec 282
4.65.2.4 dusolve 282
4.66 SRC/dlacon.c File Reference 283
4.66.1 Detailed Description 283
4.66.2 Define Documentation 283
4.66.2.1 d_sign 283
4.66.2.2 i_dnnt 283
4.66.3 Function Documentation 283
4.66.3.1 dlacon_ 283
4.67 SRC/dlamch.c File Reference 285
4.67.1 Detailed Description 285
4.67.2 Define Documentation 286
4.67.2.1 abs 286
4.67.2.2 FALSE_ 286
4.67.2.3 max 286
4.67.2.4 min 286
4.67.2.5 TRUE_ 286
4.67.3 Function Documentation 286
4.67.3.1 dlamc1_ 286
4.67.3.2 dlamc2_ 287
4.67.3.3 dlamc3_ 289
4.67.3.4 dlamc4_ 290
4.67.3.5 dlamc5_ 291
4.67.3.6 dlamch_ 292

4.67.3.7 pow_di 294
4.68 SRC/dlangs.c File Reference 295
4.68.1 Detailed Description 295
4.68.2 Function Documentation 295
4.68.2.1 dlangs 295
4.69 SRC/dlaqgs.c File Reference 297
4.69.1 Detailed Description 297
4.69.2 Define Documentation 297
4.69.2.1 THRESH 297
4.69.3 Function Documentation 297
4.69.3.1 dlaqgs 297
4.70 SRC/dmemory.c File Reference 300
4.70.1 Detailed Description 301
4.70.2 Define Documentation 302
4.70.2.1 DoubleAlign 302
4.70.2.2 GluIntArray 302
4.70.2.3 NO_MEMTYPE 302
4.70.2.4 NotDoubleAlign 302
4.70.2.5 Reduce 302
4.70.2.6 StackFull 302
4.70.2.7 TempSpace 302
4.70.3 Typedef Documentation 302
4.70.3.1 ExpHeader 302
4.70.4 Function Documentation 302
4.70.4.1 copy_mem_double 302
4.70.4.2 copy_mem_int 303
4.70.4.3 dallocateA 303
4.70.4.4 dexpand 303
4.70.4.5 dLUMemInit 303
4.70.4.6 dLUMemXpand 304
4.70.4.7 dLUWorkFree 305
4.70.4.8 dLUWorkInit 305
4.70.4.9 dmemory_usage 306
4.70.4.10 doubleCalloc 306
4.70.4.11 doubleMalloc 306
4.70.4.12 dQuerySpace 307

4.70.4.13 dSetRWork 307
4.70.4.14 dSetupSpace 307
4.70.4.15 dStackCompress 308
4.70.4.16 duser_free 308
4.70.4.17 duser_malloc 308
4.70.4.18 user_bcopy 309
4.70.5 Variable Documentation 309
4.70.5.1 expanders 309
4.70.5.2 no_expand 309
4.70.5.3 stack 309
4.71 SRC/dmyblas2.c File Reference 310
4.71.1 Detailed Description 310
4.71.2 Function Documentation 310
4.71.2.1 dlsolve 310
4.71.2.2 dmatvec 311
4.71.2.3 dusolve 311
4.72 SRC/dpanel_bmod.c File Reference 312
4.72.1 Detailed Description 312
4.72.2 Function Documentation 313
4.72.2.1 dcheck_tempv 313
4.72.2.2 dlsolve 313
4.72.2.3 dmatvec 313
4.72.2.4 dpanel_bmod 313
4.73 SRC/dpanel_dfs.c File Reference 314
4.73.1 Detailed Description 314
4.73.2 Function Documentation 314
4.73.2.1 dpanel_dfs 314
4.74 SRC/dpivotgrowth.c File Reference 316
4.74.1 Detailed Description 316
4.74.2 Function Documentation 316
4.74.2.1 dPivotGrowth 316
4.75 SRC/dpivotL.c File Reference 318
4.75.1 Detailed Description 318
4.75.2 Function Documentation 318
4.75.2.1 dpivotL 318
4.76 SRC/dpruneL.c File Reference 320

4.76.1 Detailed Description 320
4.76.2 Function Documentation 320
4.76.2.1 dpruneL 320
4.77 SRC/dreadhb.c File Reference 322
4.77.1 Detailed Description 322
4.77.2 Function Documentation 324
4.77.2.1 dDumpLine 324
4.77.2.2 dParseFloatFormat 324
4.77.2.3 dParseIntFormat 324
4.77.2.4 dreadhb 324
4.77.2.5 dReadValues 325
4.77.2.6 dReadVector 325
4.78 SRC/dsnode_bmod.c File Reference 326
4.78.1 Detailed Description 326
4.78.2 Function Documentation 327
4.78.2.1 dsnode_bmod 327
4.79 SRC/dsnode_dfs.c File Reference 328
4.79.1 Detailed Description 328
4.79.2 Function Documentation 328
4.79.2.1 dsnode_dfs 328
4.80 SRC/dsp_blas2.c File Reference 330
4.80.1 Detailed Description 330
4.80.2 Function Documentation 331
4.80.2.1 dlsolve 331
4.80.2.2 dmatvec 331
4.80.2.3 dusolve 331
4.80.2.4 sp_dgemv 331
4.80.2.5 sp_dtrsv 332
4.81 SRC/dsp_blas3.c File Reference 335
4.81.1 Detailed Description 335
4.81.2 Function Documentation 335
4.81.2.1 sp_dgemm 335
4.82 SRC/dutil.c File Reference 338
4.82.1 Detailed Description 339
4.82.2 Function Documentation 340
4.82.2.1 dcheck_tempv 340

4.82.2.2 dCompRow_to_CompCol . . . 340
4.82.2.3 dCopy_CompCol_Matrix . . . 340
4.82.2.4 dCopy_Dense_Matrix . . . 340
4.82.2.5 dCreate_CompCol_Matrix . . . 340
4.82.2.6 dCreate_CompRow_Matrix . . . 340
4.82.2.7 dCreate_Dense_Matrix . . . 340
4.82.2.8 dCreate_SuperNode_Matrix . . . 341
4.82.2.9 dfill . . . 341
4.82.2.10 dFillRHS . . . 341
4.82.2.11 dGenXtrue . . . 341
4.82.2.12 dinf_norm_error . . . 342
4.82.2.13 dPrint_CompCol_Matrix . . . 342
4.82.2.14 dPrint_Dense_Matrix . . . 342
4.82.2.15 dprint_lu_col . . . 342
4.82.2.16 dPrint_SuperNode_Matrix . . . 342
4.82.2.17 dPrintPerf . . . 342
4.82.2.18 print_double_vec . . . 342
4.83 SRC/dzsum1.c File Reference . . . 343
4.83.1 Detailed Description . . . 343
4.83.2 Define Documentation . . . 343
4.83.2.1 CX . . . 343
4.83.3 Function Documentation . . . 343
4.83.3.1 dzsum1_ . . . 343
4.84 SRC/get_perm_c.c File Reference . . . 345
4.84.1 Detailed Description . . . 345
4.84.2 Function Documentation . . . 345
4.84.2.1 at_plus_a . . . 345
4.84.2.2 genmmd_ . . . 346
4.84.2.3 get_colamd . . . 347
4.84.2.4 get_perm_c . . . 347
4.84.2.5 getata . . . 349
4.85 SRC/heap_relax_snode.c File Reference . . . 350
4.85.1 Detailed Description . . . 350
4.85.2 Function Documentation . . . 350
4.85.2.1 heap_relax_snode . . . 350
4.86 SRC/icmax1.c File Reference . . . 352

4.86.1 Detailed Description . . . 352
4.86.2 Define Documentation . . . 352
4.86.2.1 CX . . . 352
4.86.3 Function Documentation . . . 352
4.86.3.1 icmax1_ . . . 352
4.87 SRC/izmax1.c File Reference . . . 354
4.87.1 Detailed Description . . . 354
4.87.2 Define Documentation . . . 354
4.87.2.1 CX . . . 354
4.87.3 Function Documentation . . . 354
4.87.3.1 izmax1_ . . . 354
4.88 SRC/lsame.c File Reference . . . 356
4.88.1 Detailed Description . . . 356
4.88.2 Function Documentation . . . 356
4.88.2.1 lsame_ . . . 356
4.89 SRC/memory.c File Reference . . . 358
4.89.1 Detailed Description . . . 358
4.89.2 Function Documentation . . . 359
4.89.2.1 copy_mem_int . . . 359
4.89.2.2 intCalloc . . . 360
4.89.2.3 intMalloc . . . 361
4.89.2.4 SetIWork . . . 362
4.89.2.5 superlu_free . . . 362
4.89.2.6 superlu_malloc . . . 362
4.89.2.7 user_bcopy . . . 363
4.90 SRC/mmd.c File Reference . . . 364
4.90.1 Typedef Documentation . . . 364
4.90.1.1 shortint . . . 364
4.90.2 Function Documentation . . . 364
4.90.2.1 genmmd_ . . . 364
4.90.2.2 mmdelm_ . . . 365
4.90.2.3 mmdint_ . . . 366
4.90.2.4 mmdnum_ . . . 366
4.90.2.5 mmdupd_ . . . 366
4.91 SRC/old_colamd.c File Reference . . . 368
4.91.1 Define Documentation . . . 371

4.91.1.1 ALIVE . . . 371
4.91.1.2 COL_IS_ALIVE . . . 371
4.91.1.3 COL_IS_DEAD . . . 371
4.91.1.4 COL_IS_DEAD_PRINCIPAL . . . 371
4.91.1.5 DEAD . . . 371
4.91.1.6 DEAD_NON_PRINCIPAL . . . 371
4.91.1.7 DEAD_PRINCIPAL . . . 371
4.91.1.8 DEBUG0 . . . 371
4.91.1.9 DEBUG1 . . . 371
4.91.1.10 DEBUG2 . . . 371
4.91.1.11 DEBUG3 . . . 371
4.91.1.12 DEBUG4 . . . 371
4.91.1.13 EMPTY . . . 371
4.91.1.14 FALSE . . . 371
4.91.1.15 KILL_NON_PRINCIPAL_COL . . . 371
4.91.1.16 KILL_PRINCIPAL_COL . . . 371
4.91.1.17 KILL_ROW . . . 371
4.91.1.18 MAX . . . 371
4.91.1.19 MIN . . . 371
4.91.1.20 ONES_COMPLEMENT . . . 371
4.91.1.21 PRIVATE . . . 371
4.91.1.22 PUBLIC . . . 371
4.91.1.23 ROW_IS_ALIVE . . . 371
4.91.1.24 ROW_IS_DEAD . . . 371
4.91.1.25 ROW_IS_MARKED_DEAD . . . 371
4.91.1.26 TRUE . . . 371
4.91.2 Typedef Documentation . . . 371
4.91.2.1 ColInfo . . . 371
4.91.2.2 RowInfo . . . 371
4.91.3 Function Documentation . . . 371
4.91.3.1 clear_mark . . . 371
4.91.3.2 colamd . . . 371
4.91.3.3 colamd_recommended . . . 372
4.91.3.4 colamd_set_defaults . . . 372
4.91.3.5 detect_super_cols . . . 372
4.91.3.6 find_ordering . . . 372

4.91.3.7 garbage_collection 372
4.91.3.8 init_rows_cols 372
4.91.3.9 init_scoring 372
4.91.3.10 order_children 372
4.92 SRC/old_colamd.h File Reference 373
4.92.1 Define Documentation 373
4.92.1.1 COLAMD_DEFRAG_COUNT 373
4.92.1.2 COLAMD_DENSE_COL 373
4.92.1.3 COLAMD_DENSE_ROW 373
4.92.1.4 COLAMD_JUMBLED_COLS 373
4.92.1.5 COLAMD_KNOBS 373
4.92.1.6 COLAMD_STATS 373
4.92.2 Function Documentation 373
4.92.2.1 colamd 373
4.92.2.2 colamd_recommended 374
4.92.2.3 colamd_set_defaults 374
4.93 SRC/relax_snode.c File Reference 375
4.93.1 Detailed Description 375
4.93.2 Function Documentation 375
4.93.2.1 relax_snode 375
4.94 SRC/scolumn_bmod.c File Reference 377
4.94.1 Detailed Description 377
4.94.2 Function Documentation 378
4.94.2.1 scolumn_bmod 378
4.94.2.2 slsolve 378
4.94.2.3 smatvec 379
4.94.2.4 susolve 379
4.95 SRC/scolumn_dfs.c File Reference 380
4.95.1 Detailed Description 380
4.95.2 Define Documentation 381
4.95.2.1 T2_SUPER 381
4.95.3 Function Documentation 381
4.95.3.1 scolumn_dfs 381
4.96 SRC/scomplex.c File Reference 383
4.96.1 Detailed Description 383
4.96.2 Function Documentation 384

4.96.2.1 c_abs . . . 384
4.96.2.2 c_abs1 . . . 384
4.96.2.3 c_div . . . 384
4.96.2.4 c_exp . . . 384
4.96.2.5 r_cnjg . . . 384
4.96.2.6 r_imag . . . 384
4.97 SRC/scopy_to_ucol.c File Reference . . . 385
4.97.1 Detailed Description . . . 385
4.97.2 Function Documentation . . . 386
4.97.2.1 scopy_to_ucol . . . 386
4.98 SRC/scsum1.c File Reference . . . 387
4.98.1 Detailed Description . . . 387
4.98.2 Define Documentation . . . 387
4.98.2.1 CX . . . 387
4.98.3 Function Documentation . . . 387
4.98.3.1 scsum1_ . . . 387
4.99 SRC/sgscon.c File Reference . . . 389
4.99.1 Detailed Description . . . 389
4.99.2 Function Documentation . . . 389
4.99.2.1 sgscon . . . 389
4.100 SRC/sgsequ.c File Reference . . . 391
4.100.1 Detailed Description . . . 391
4.100.2 Function Documentation . . . 391
4.100.2.1 sgsequ . . . 391
4.101 SRC/sgsrfs.c File Reference . . . 394
4.101.1 Detailed Description . . . 394
4.101.2 Define Documentation . . . 395
4.101.2.1 ITMAX . . . 395
4.101.3 Function Documentation . . . 395
4.101.3.1 sgsrfs . . . 395
4.102 SRC/sgssv.c File Reference . . . 398
4.102.1 Detailed Description . . . 398
4.102.2 Function Documentation . . . 398
4.102.2.1 sgssv . . . 398
4.103 SRC/sgssvx.c File Reference . . . 403
4.103.1 Detailed Description . . . 403

4.103.2 Function Documentation . . . 403
4.103.2.1 sgssvx . . . 403
4.104 SRC/sgstrf.c File Reference . . . 412
4.104.1 Detailed Description . . . 412
4.104.2 Function Documentation . . . 412
4.104.2.1 sgstrf . . . 412
4.105 SRC/sgstrs.c File Reference . . . 417
4.105.1 Detailed Description . . . 417
4.105.2 Function Documentation . . . 418
4.105.2.1 sgstrs . . . 418
4.105.2.2 slsolve . . . 419
4.105.2.3 smatvec . . . 419
4.105.2.4 sprint_soln . . . 419
4.105.2.5 susolve . . . 420
4.106 SRC/slacon.c File Reference . . . 421
4.106.1 Detailed Description . . . 421
4.106.2 Define Documentation . . . 421
4.106.2.1 d_sign . . . 421
4.106.2.2 i_dnnt . . . 421
4.106.3 Function Documentation . . . 421
4.106.3.1 slacon_ . . . 421
4.107 SRC/slamch.c File Reference . . . 423
4.107.1 Detailed Description . . . 423
4.107.2 Define Documentation . . . 424
4.107.2.1 abs . . . 424
4.107.2.2 dabs . . . 424
4.107.2.3 FALSE_ . . . 424
4.107.2.4 max . . . 424
4.107.2.5 min . . . 424
4.107.2.6 TRUE_ . . . 424
4.107.3 Function Documentation . . . 424
4.107.3.1 pow_ri . . . 424
4.107.3.2 slamc1_ . . . 424
4.107.3.3 slamc2_ . . . 426
4.107.3.4 slamc3_ . . . 428
4.107.3.5 slamc4_ . . . 429

4.107.3.6 slamc5_ . . . 430
4.107.3.7 slamch_ . . . 431
4.108 SRC/slangs.c File Reference . . . 434
4.108.1 Detailed Description . . . 434
4.108.2 Function Documentation . . . 434
4.108.2.1 slangs . . . 434
4.109 SRC/slaqgs.c File Reference . . . 436
4.109.1 Detailed Description . . . 436
4.109.2 Define Documentation . . . 436
4.109.2.1 THRESH . . . 436
4.109.3 Function Documentation . . . 436
4.109.3.1 slaqgs . . . 436
4.110 SRC/slu_cdefs.h File Reference . . . 439
4.110.1 Detailed Description . . . 442
4.110.2 Typedef Documentation . . . 443
4.110.2.1 int_t . . . 443
4.110.3 Function Documentation . . . 443
4.110.3.1 callocateA . . . 443
4.110.3.2 ccolumn_bmod . . . 443
4.110.3.3 ccolumn_dfs . . . 444
4.110.3.4 cCompRow_to_CompCol . . . 445
4.110.3.5 cCopy_CompCol_Matrix . . . 445
4.110.3.6 cCopy_Dense_Matrix . . . 445
4.110.3.7 ccopy_to_ucol . . . 445
4.110.3.8 cCreate_CompCol_Matrix . . . 445
4.110.3.9 cCreate_CompRow_Matrix . . . 446
4.110.3.10 cCreate_Dense_Matrix . . . 446
4.110.3.11 cCreate_SuperNode_Matrix . . . 446
4.110.3.12 cfill . . . 446
4.110.3.13 cFillRHS . . . 446
4.110.3.14 cGenXtrue . . . 447
4.110.3.15 cgscon . . . 447
4.110.3.16 cgsequ . . . 448
4.110.3.17 cgsrfs . . . 450
4.110.3.18 cgssv . . . 452
4.110.3.19 cgssvx . . . 456

4.110.3.20 cgstrf . . . 463
4.110.3.21 cgstrs . . . 467
4.110.3.22 check_tempv . . . 468
4.110.3.23 cinf_norm_error . . . 468
4.110.3.24 claqgs . . . 469
4.110.3.25 cLUMemInit . . . 470
4.110.3.26 cLUMemXpand . . . 471
4.110.3.27 cLUWorkFree . . . 472
4.110.3.28 cmemory_usage . . . 472
4.110.3.29 complexCalloc . . . 472
4.110.3.30 complexMalloc . . . 473
4.110.3.31 countnz . . . 473
4.110.3.32 cpanel_bmod . . . 473
4.110.3.33 cpanel_dfs . . . 474
4.110.3.34 cPivotGrowth . . . 475
4.110.3.35 cpivotL . . . 476
4.110.3.36 cPrint_CompCol_Matrix . . . 477
4.110.3.37 cPrint_Dense_Matrix . . . 477
4.110.3.38 cPrint_SuperNode_Matrix . . . 477
4.110.3.39 cpruneL . . . 477
4.110.3.40 cQuerySpace . . . 477
4.110.3.41 creadhb . . . 478
4.110.3.42 creadmt . . . 478
4.110.3.43 cSetRWork . . . 478
4.110.3.44 csnode_bmod . . . 479
4.110.3.45 csnode_dfs . . . 479
4.110.3.46 fixupL . . . 480
4.110.3.47 floatCalloc . . . 480
4.110.3.48 floatMalloc . . . 480
4.110.3.49 print_lu_col . . . 481
4.110.3.50 PrintPerf . . . 481
4.110.3.51 sp_cgemm . . . 481
4.110.3.52 sp_cgemv . . . 483
4.110.3.53 sp_ctrsv . . . 484
4.111 SRC/slu_Cnames.h File Reference . . . 486
4.111.1 Detailed Description . . . 486

4.111.2 Define Documentation 486
4.111.2.1 ADD_ 486
4.111.2.2 ADD__ 486
4.111.2.3 C_CALL 486
4.111.2.4 F77_CALL_C 486
4.111.2.5 NOCHANGE 486
4.111.2.6 UPCASE 486
4.112 SRC/slu_dcomplex.h File Reference 487
4.112.1 Detailed Description 488
4.112.2 Define Documentation 488
4.112.2.1 z_add 488
4.112.2.2 z_eq 488
4.112.2.3 z_sub 488
4.112.2.4 zd_mult 488
4.112.2.5 zz_conj 488
4.112.2.6 zz_mult 489
4.112.3 Function Documentation 489
4.112.3.1 d_cnjg 489
4.112.3.2 d_imag 489
4.112.3.3 z_abs 489
4.112.3.4 z_abs1 489
4.112.3.5 z_div 490
4.112.3.6 z_exp 490
4.113 SRC/slu_ddefs.h File Reference 491
4.113.1 Detailed Description 494
4.113.2 Typedef Documentation 495
4.113.2.1 int_t 495
4.113.3 Function Documentation 495
4.113.3.1 check_tempv 495
4.113.3.2 countnz 495
4.113.3.3 dallocateA 495
4.113.3.4 dcolumn_bmod 495
4.113.3.5 dcolumn_dfs 496
4.113.3.6 dCompRow_to_CompCol 497
4.113.3.7 dCopy_CompCol_Matrix 497
4.113.3.8 dCopy_Dense_Matrix 497

4.113.3.9 dcopy_to_ucol 497
4.113.3.10 dCreate_CompCol_Matrix 498
4.113.3.11 dCreate_CompRow_Matrix 498
4.113.3.12 dCreate_Dense_Matrix 498
4.113.3.13 dCreate_SuperNode_Matrix 498
4.113.3.14 dfill 498
4.113.3.15 dFillRHS 499
4.113.3.16 dGenXtrue 499
4.113.3.17 dgscon 499
4.113.3.18 dgsequ 500
4.113.3.19 dgsrfs 502
4.113.3.20 dgssv 505
4.113.3.21 dgssvx 509
4.113.3.22 dgstrf 516
4.113.3.23 dgstrs 520
4.113.3.24 dinf_norm_error 521
4.113.3.25 dlaqgs 521
4.113.3.26 dLUMemInit 523
4.113.3.27 dLUMemXpand 524
4.113.3.28 dLUWorkFree 524
4.113.3.29 dmemory_usage 525
4.113.3.30 doubleCalloc 525
4.113.3.31 doubleMalloc 525
4.113.3.32 dpanel_bmod 525
4.113.3.33 dpanel_dfs 526
4.113.3.34 dPivotGrowth 526
4.113.3.35 dpivotL 527
4.113.3.36 dPrint_CompCol_Matrix 528
4.113.3.37 dPrint_Dense_Matrix 528
4.113.3.38 dPrint_SuperNode_Matrix 528
4.113.3.39 dpruneL 528
4.113.3.40 dQuerySpace 529
4.113.3.41 dreadhb 529
4.113.3.42 dreadmt 530
4.113.3.43 dSetRWork 530
4.113.3.44 dsnode_bmod 530

4.113.3.45 dsnode_dfs 530
4.113.3.46 fixupL 531
4.113.3.47 print_lu_col 531
4.113.3.48 PrintPerf 531
4.113.3.49 sp_dgemm 531
4.113.3.50 sp_dgemv 533
4.113.3.51 sp_dtrsv 534
4.114 SRC/slu_scomplex.h File Reference 537
4.114.1 Detailed Description 538
4.114.2 Define Documentation 538
4.114.2.1 c_add 538
4.114.2.2 c_eq 538
4.114.2.3 c_sub 538
4.114.2.4 cc_conj 538
4.114.2.5 cc_mult 538
4.114.2.6 cs_mult 539
4.114.3 Function Documentation 539
4.114.3.1 c_abs 539
4.114.3.2 c_abs1 539
4.114.3.3 c_div 540
4.114.3.4 c_exp 540
4.114.3.5 r_cnjg 540
4.114.3.6 r_imag 540
4.115 SRC/slu_sdefs.h File Reference 541
4.115.1 Detailed Description 543
4.115.2 Typedef Documentation 545
4.115.2.1 int_t 545
4.115.3 Function Documentation 545
4.115.3.1 check_tempv 545
4.115.3.2 countnz 545
4.115.3.3 fixupL 545
4.115.3.4 floatCalloc 545
4.115.3.5 floatMalloc 545
4.115.3.6 print_lu_col 545
4.115.3.7 PrintPerf 545
4.115.3.8 sallocateA 545

4.115.3.9 scolumn_bmod . . . 545
4.115.3.10 scolumn_dfs . . . 546
4.115.3.11 sCompRow_to_CompCol . . . 547
4.115.3.12 sCopy_CompCol_Matrix . . . 547
4.115.3.13 sCopy_Dense_Matrix . . . 547
4.115.3.14 scopy_to_ucol . . . 547
4.115.3.15 sCreate_CompCol_Matrix . . . 548
4.115.3.16 sCreate_CompRow_Matrix . . . 548
4.115.3.17 sCreate_Dense_Matrix . . . 548
4.115.3.18 sCreate_SuperNode_Matrix . . . 548
4.115.3.19 sfill . . . 549
4.115.3.20 sFillRHS . . . 549
4.115.3.21 sGenXtrue . . . 549
4.115.3.22 sgscon . . . 549
4.115.3.23 sgsequ . . . 551
4.115.3.24 sgsrfs . . . 552
4.115.3.25 sgssv . . . 555
4.115.3.26 sgssvx . . . 559
4.115.3.27 sgstrf . . . 566
4.115.3.28 sgstrs . . . 570
4.115.3.29 sinf_norm_error . . . 571
4.115.3.30 slaqgs . . . 571
4.115.3.31 sLUMemInit . . . 573
4.115.3.32 sLUMemXpand . . . 573
4.115.3.33 sLUWorkFree . . . 574
4.115.3.34 smemory_usage . . . 574
4.115.3.35 sp_sgemm . . . 575
4.115.3.36 sp_sgemv . . . 577
4.115.3.37 sp_strsv . . . 578
4.115.3.38 spanel_bmod . . . 579
4.115.3.39 spanel_dfs . . . 580
4.115.3.40 sPivotGrowth . . . 581
4.115.3.41 spivotL . . . 582
4.115.3.42 sPrint_CompCol_Matrix . . . 582
4.115.3.43 sPrint_Dense_Matrix . . . 582
4.115.3.44 sPrint_SuperNode_Matrix . . . 582

4.115.3.45 spruneL 582
4.115.3.46 sQuerySpace 583
4.115.3.47 sreadhb 583
4.115.3.48 sreadmt 584
4.115.3.49 sSetRWork 584
4.115.3.50 ssnode_bmod 584
4.115.3.51 ssnode_dfs 584
4.116 SRC/slu_util.h File Reference 586
4.116.1 Detailed Description 588
4.116.2 Define Documentation 588
4.116.2.1 ABORT 588
4.116.2.2 CHECK_MALLOC 589
4.116.2.3 EMPTY 590
4.116.2.4 FALSE 590
4.116.2.5 FIRSTCOL_OF_SNODE 590
4.116.2.6 L_FST_SUPC 590
4.116.2.7 L_NZ_START 590
4.116.2.8 L_SUB 590
4.116.2.9 L_SUB_START 590
4.116.2.10 NO_MARKER 590
4.116.2.11 NUM_TEMPV 590
4.116.2.12 SUPERLU_FREE 590
4.116.2.13 SUPERLU_MALLOC 590
4.116.2.14 SUPERLU_MAX 590
4.116.2.15 SUPERLU_MIN 590
4.116.2.16 TRUE 590
4.116.2.17 U_NZ_START 590
4.116.2.18 U_SUB 590
4.116.2.19 USER_ABORT 590
4.116.2.20 USER_FREE 590
4.116.2.21 USER_MALLOC 590
4.116.3 Typedef Documentation 590
4.116.3.1 flops_t 590
4.116.3.2 Logical 590
4.116.4 Enumeration Type Documentation 590
4.116.4.1 colperm_t 590

4.116.4.2 DiagScale_t 591
4.116.4.3 fact_t 591
4.116.4.4 IterRefine_t 591
4.116.4.5 LU_space_t 591
4.116.4.6 MemType 591
4.116.4.7 PhaseType 592
4.116.4.8 rowperm_t 592
4.116.4.9 stack_end_t 592
4.116.4.10 trans_t 592
4.116.4.11 yes_no_t 592
4.116.5 Function Documentation 593
4.116.5.1 check_repfnz 593
4.116.5.2 Destroy_CompCol_Matrix 593
4.116.5.3 Destroy_CompCol_Permuted 593
4.116.5.4 Destroy_CompRow_Matrix 593
4.116.5.5 Destroy_Dense_Matrix 593
4.116.5.6 Destroy_SuperMatrix_Store 594
4.116.5.7 Destroy_SuperNode_Matrix 594
4.116.5.8 get_perm_c 594
4.116.5.9 heap_relax_snode 595
4.116.5.10 ifill 597
4.116.5.11 intCalloc 597
4.116.5.12 intMalloc 598
4.116.5.13 lsame_ 599
4.116.5.14 print_panel_seg 601
4.116.5.15 PrintSumm 601
4.116.5.16 relax_snode 601
4.116.5.17 resetrep_col 602
4.116.5.18 set_default_options 602
4.116.5.19 SetIWork 602
4.116.5.20 snode_profile 603
4.116.5.21 sp_coletree 603
4.116.5.22 sp_ienv 604
4.116.5.23 sp_preorder 607
4.116.5.24 spcoletree 608
4.116.5.25 StatFree 608

4.116.5.26 StatInit . . . 609
4.116.5.27 StatPrint . . . 609
4.116.5.28 super_stats . . . 609
4.116.5.29 superlu_abort_and_exit . . . 609
4.116.5.30 superlu_free . . . 609
4.116.5.31 superlu_malloc . . . 609
4.116.5.32 SuperLU_timer_ . . . 610
4.116.5.33 TreePostorder . . . 610
4.116.5.34 xerbla_ . . . 612
4.117 SRC/slu_zdefs.h File Reference . . . 613
4.117.1 Detailed Description . . . 616
4.117.2 Typedef Documentation . . . 617
4.117.2.1 int_t . . . 617
4.117.3 Function Documentation . . . 617
4.117.3.1 check_tempv . . . 617
4.117.3.2 countnz . . . 617
4.117.3.3 doubleCalloc . . . 617
4.117.3.4 doublecomplexCalloc . . . 617
4.117.3.5 doublecomplexMalloc . . . 617
4.117.3.6 doubleMalloc . . . 618
4.117.3.7 fixupL . . . 618
4.117.3.8 print_lu_col . . . 618
4.117.3.9 PrintPerf . . . 618
4.117.3.10 sp_zgemm . . . 618
4.117.3.11 sp_zgemv . . . 620
4.117.3.12 sp_ztrsv . . . 621
4.117.3.13 zallocateA . . . 623
4.117.3.14 zcolumn_bmod . . . 623
4.117.3.15 zcolumn_dfs . . . 624
4.117.3.16 zCompRow_to_CompCol . . . 625
4.117.3.17 zCopy_CompCol_Matrix . . . 625
4.117.3.18 zCopy_Dense_Matrix . . . 625
4.117.3.19 zcopy_to_ucol . . . 626
4.117.3.20 zCreate_CompCol_Matrix . . . 626
4.117.3.21 zCreate_CompRow_Matrix . . . 626
4.117.3.22 zCreate_Dense_Matrix . . . 626

4.117.3.23 zCreate_SuperNode_Matrix 627
4.117.3.24 zfill 627
4.117.3.25 zFillRHS 627
4.117.3.26 zGenXtrue 627
4.117.3.27 zgscon 627
4.117.3.28 zgsequ 629
4.117.3.29 zgsrfs 631
4.117.3.30 zgssv 633
4.117.3.31 zgssvx 637
4.117.3.32 zgstrf 644
4.117.3.33 zgstrs 648
4.117.3.34 zinf_norm_error 649
4.117.3.35 zlaqgs 649
4.117.3.36 zLUMemInit 651
4.117.3.37 zLUMemXpand 652
4.117.3.38 zLUWorkFree 652
4.117.3.39 zmemory_usage 653
4.117.3.40 zpanel_bmod 653
4.117.3.41 zpanel_dfs 654
4.117.3.42 zPivotGrowth 654
4.117.3.43 zpivotL 655
4.117.3.44 zPrint_CompCol_Matrix 656
4.117.3.45 zPrint_Dense_Matrix 656
4.117.3.46 zPrint_SuperNode_Matrix 656
4.117.3.47 zpruneL 656
4.117.3.48 zQuerySpace 656
4.117.3.49 zreadhb 657
4.117.3.50 zreadmt 658
4.117.3.51 zSetRWork 658
4.117.3.52 zsnode_bmod 658
4.117.3.53 zsnode_dfs 658
4.118 SRC/smemory.c File Reference 660
4.118.1 Detailed Description 661
4.118.2 Define Documentation 662
4.118.2.1 DoubleAlign 662
4.118.2.2 GluIntArray 662

4.118.2.3 NO_MEMTYPE 662
4.118.2.4 NotDoubleAlign 662
4.118.2.5 Reduce 662
4.118.2.6 StackFull 662
4.118.2.7 TempSpace 662
4.118.3 Typedef Documentation 662
4.118.3.1 ExpHeader 662
4.118.4 Function Documentation 662
4.118.4.1 copy_mem_float 662
4.118.4.2 copy_mem_int 663
4.118.4.3 floatCalloc 663
4.118.4.4 floatMalloc 663
4.118.4.5 sallocateA 663
4.118.4.6 sexpand 664
4.118.4.7 sLUMemInit 664
4.118.4.8 sLUMemXpand 665
4.118.4.9 sLUWorkFree 666
4.118.4.10 sLUWorkInit 666
4.118.4.11 smemory_usage 667
4.118.4.12 sQuerySpace 667
4.118.4.13 sSetRWork 667
4.118.4.14 sSetupSpace 668
4.118.4.15 sStackCompress 668
4.118.4.16 suser_free 668
4.118.4.17 suser_malloc 669
4.118.4.18 user_bcopy 669
4.118.5 Variable Documentation 669
4.118.5.1 expanders 669
4.118.5.2 no_expand 669
4.118.5.3 stack 669
4.119 SRC/smyblas2.c File Reference 670
4.119.1 Detailed Description 670
4.119.2 Function Documentation 670
4.119.2.1 slsolve 670
4.119.2.2 smatvec 671
4.119.2.3 susolve 671

4.120 SRC/sp_coletree.c File Reference 672
4.120.1 Detailed Description 672
4.120.2 Function Documentation 673
4.120.2.1 etdfs 673
4.120.2.2 finalize_disjoint_sets 673
4.120.2.3 find 674
4.120.2.4 initialize_disjoint_sets 674
4.120.2.5 link 675
4.120.2.6 make_set 675
4.120.2.7 mxCallocInt 676
4.120.2.8 nr_etdfs 676
4.120.2.9 sp_coletree 677
4.120.2.10 sp_symetree 677
4.120.2.11 TreePostorder 678
4.121 SRC/sp_ienv.c File Reference 679
4.121.1 Function Documentation 679
4.121.1.1 sp_ienv 679
4.122 EXAMPLE/sp_ienv.c File Reference 681
4.122.1 Function Documentation 681
4.122.1.1 sp_ienv 681
4.123 SRC/sp_preorder.c File Reference 684
4.123.1 Detailed Description 684
4.123.2 Function Documentation 684
4.123.2.1 check_perm 684
4.123.2.2 sp_preorder 685
4.124 SRC/spanel_bmod.c File Reference 687
4.124.1 Detailed Description 687
4.124.2 Function Documentation 688
4.124.2.1 scheck_tempv 688
4.124.2.2 slsolve 688
4.124.2.3 smatvec 688
4.124.2.4 spanel_bmod 688
4.125 SRC/spanel_dfs.c File Reference 689
4.125.1 Detailed Description 689
4.125.2 Function Documentation 689
4.125.2.1 spanel_dfs 689

4.126 SRC/spivotgrowth.c File Reference . . . 691
4.126.1 Detailed Description . . . 691
4.126.2 Function Documentation . . . 691
4.126.2.1 sPivotGrowth . . . 691
4.127 SRC/spivotL.c File Reference . . . 693
4.127.1 Detailed Description . . . 693
4.127.2 Function Documentation . . . 693
4.127.2.1 spivotL . . . 693
4.128 SRC/spruneL.c File Reference . . . 695
4.128.1 Detailed Description . . . 695
4.128.2 Function Documentation . . . 695
4.128.2.1 spruneL . . . 695
4.129 SRC/sreadhb.c File Reference . . . 697
4.129.1 Detailed Description . . . 697
4.129.2 Function Documentation . . . 699
4.129.2.1 sDumpLine . . . 699
4.129.2.2 sParseFloatFormat . . . 699
4.129.2.3 sParseIntFormat . . . 699
4.129.2.4 sreadhb . . . 699
4.129.2.5 sReadValues . . . 700
4.129.2.6 sReadVector . . . 700
4.130 SRC/ssnode_bmod.c File Reference . . . 701
4.130.1 Detailed Description . . . 701
4.130.2 Function Documentation . . . 702
4.130.2.1 ssnode_bmod . . . 702
4.131 SRC/ssnode_dfs.c File Reference . . . 703
4.131.1 Detailed Description . . . 703
4.131.2 Function Documentation . . . 703
4.131.2.1 ssnode_dfs . . . 703
4.132 SRC/ssp_blas2.c File Reference . . . 705
4.132.1 Detailed Description . . . 705
4.132.2 Function Documentation . . . 706
4.132.2.1 slsolve . . . 706
4.132.2.2 smatvec . . . 706
4.132.2.3 sp_sgemv . . . 706
4.132.2.4 sp_strsv . . . 707

4.132.2.5 susolve 709
4.133 SRC/ssp_blas3.c File Reference 710
4.133.1 Detailed Description 710
4.133.2 Function Documentation 710
4.133.2.1 sp_sgemm 710
4.134 SRC/superlu_timer.c File Reference 713
4.134.1 Detailed Description 713
4.134.2 Define Documentation 714
4.134.2.1 CLK_TCK 714
4.134.3 Function Documentation 714
4.134.3.1 SuperLU_timer_ 714
4.135 SRC/supermatrix.h File Reference 715
4.135.1 Detailed Description 715
4.135.2 Enumeration Type Documentation 715
4.135.2.1 Dtype_t 715
4.135.2.2 Mtype_t 716
4.135.2.3 Stype_t 716
4.136 SRC/sutil.c File Reference 717
4.136.1 Detailed Description 718
4.136.2 Function Documentation 719
4.136.2.1 print_float_vec 719
4.136.2.2 scheck_tempv 719
4.136.2.3 sCompRow_to_CompCol 719
4.136.2.4 sCopy_CompCol_Matrix 719
4.136.2.5 sCopy_Dense_Matrix 719
4.136.2.6 sCreate_CompCol_Matrix 719
4.136.2.7 sCreate_CompRow_Matrix 719
4.136.2.8 sCreate_Dense_Matrix 719
4.136.2.9 sCreate_SuperNode_Matrix 720
4.136.2.10 sfill 720
4.136.2.11 sFillRHS 720
4.136.2.12 sGenXtrue 720
4.136.2.13 sinf_norm_error 721
4.136.2.14 sPrint_CompCol_Matrix 721
4.136.2.15 sPrint_Dense_Matrix 721
4.136.2.16 sprint_lu_col 721

4.136.2.17 sPrint_SuperNode_Matrix . . . 721
4.136.2.18 sPrintPerf . . . 721
4.137 SRC/util.c File Reference . . . 722
4.137.1 Detailed Description . . . 723
4.137.2 Define Documentation . . . 724
4.137.2.1 NBUCKS . . . 724
4.137.3 Function Documentation . . . 724
4.137.3.1 check_repfnz . . . 724
4.137.3.2 countnz . . . 724
4.137.3.3 DenseSize . . . 724
4.137.3.4 Destroy_CompCol_Matrix . . . 724
4.137.3.5 Destroy_CompCol_Permuted . . . 725
4.137.3.6 Destroy_CompRow_Matrix . . . 725
4.137.3.7 Destroy_Dense_Matrix . . . 725
4.137.3.8 Destroy_SuperMatrix_Store . . . 725
4.137.3.9 Destroy_SuperNode_Matrix . . . 726
4.137.3.10 fixupL . . . 726
4.137.3.11 ifill . . . 726
4.137.3.12 LUFactFlops . . . 727
4.137.3.13 LUSolveFlops . . . 727
4.137.3.14 print_int_vec . . . 727
4.137.3.15 print_options . . . 727
4.137.3.16 print_panel_seg . . . 727
4.137.3.17 PrintSumm . . . 727
4.137.3.18 resetrep_col . . . 727
4.137.3.19 set_default_options . . . 728
4.137.3.20 SpaSize . . . 728
4.137.3.21 StatFree . . . 728
4.137.3.22 StatInit . . . 728
4.137.3.23 StatPrint . . . 728
4.137.3.24 super_stats . . . 728
4.137.3.25 superlu_abort_and_exit . . . 729
4.137.4 Variable Documentation . . . 729
4.137.4.1 max_sup_size . . . 729
4.138 SRC/xerbla.c File Reference . . . 730
4.138.1 Function Documentation . . . 732

4.138.1.1 xerbla_ 732
4.139 SRC/zcolumn_bmod.c File Reference 733
4.139.1 Detailed Description 733
4.139.2 Function Documentation 734
4.139.2.1 zcolumn_bmod 734
4.139.2.2 zlsolve 734
4.139.2.3 zmatvec 735
4.139.2.4 zusolve 735
4.140 SRC/zcolumn_dfs.c File Reference 736
4.140.1 Detailed Description 736
4.140.2 Define Documentation 737
4.140.2.1 T2_SUPER 737
4.140.3 Function Documentation 737
4.140.3.1 zcolumn_dfs 737
4.141 SRC/zcopy_to_ucol.c File Reference 739
4.141.1 Detailed Description 739
4.141.2 Function Documentation 740
4.141.2.1 zcopy_to_ucol 740
4.142 SRC/zgscon.c File Reference 741
4.142.1 Detailed Description 741
4.142.2 Function Documentation 741
4.142.2.1 zgscon 741
4.143 SRC/zgsequ.c File Reference 744
4.143.1 Detailed Description 744
4.143.2 Function Documentation 744
4.143.2.1 zgsequ 744
4.144 SRC/zgsrfs.c File Reference 747
4.144.1 Detailed Description 747
4.144.2 Define Documentation 748
4.144.2.1 ITMAX 748
4.144.3 Function Documentation 748
4.144.3.1 zgsrfs 748
4.145 SRC/zgssv.c File Reference 751
4.145.1 Detailed Description 751
4.145.2 Function Documentation 751
4.145.2.1 zgssv 751

4.146 SRC/zgssvx.c File Reference 756
4.146.1 Detailed Description 756
4.146.2 Function Documentation 756
4.146.2.1 zgssvx 756
4.147 SRC/zgstrf.c File Reference 765
4.147.1 Detailed Description 765
4.147.2 Function Documentation 765
4.147.2.1 zgstrf 765
4.148 SRC/zgstrs.c File Reference 770
4.148.1 Detailed Description 770
4.148.2 Function Documentation 771
4.148.2.1 zgstrs 771
4.148.2.2 zlsolve 772
4.148.2.3 zmatvec 772
4.148.2.4 zprint_soln 772
4.148.2.5 zusolve 773
4.149 SRC/zlacon.c File Reference 774
4.149.1 Detailed Description 774
4.149.2 Function Documentation 774
4.149.2.1 zlacon_ 774
4.150 SRC/zlangs.c File Reference 776
4.150.1 Detailed Description 776
4.150.2 Function Documentation 776
4.150.2.1 zlangs 776
4.151 SRC/zlaqgs.c File Reference 778
4.151.1 Detailed Description 778
4.151.2 Define Documentation 778
4.151.2.1 THRESH 778
4.151.3 Function Documentation 778
4.151.3.1 zlaqgs 778
4.152 SRC/zmemory.c File Reference 781
4.152.1 Detailed Description 782
4.152.2 Define Documentation 783
4.152.2.1 DoubleAlign 783
4.152.2.2 GluIntArray 783
4.152.2.3 NO_MEMTYPE 783

4.152.2.4 NotDoubleAlign 783
4.152.2.5 Reduce 783
4.152.2.6 StackFull 783
4.152.2.7 TempSpace 783
4.152.3 Typedef Documentation 783
4.152.3.1 ExpHeader 783
4.152.4 Function Documentation 783
4.152.4.1 copy_mem_doublecomplex 783
4.152.4.2 copy_mem_int 784
4.152.4.3 doublecomplexCalloc 784
4.152.4.4 doublecomplexMalloc 784
4.152.4.5 user_bcopy 784
4.152.4.6 zallocateA 784
4.152.4.7 zexpand 785
4.152.4.8 zLUMemInit 785
4.152.4.9 zLUMemXpand 786
4.152.4.10 zLUWorkFree 787
4.152.4.11 zLUWorkInit 787
4.152.4.12 zmemory_usage 788
4.152.4.13 zQuerySpace 788
4.152.4.14 zSetRWork 788
4.152.4.15 zSetupSpace 789
4.152.4.16 zStackCompress 789
4.152.4.17 zuser_free 789
4.152.4.18 zuser_malloc 790
4.152.5 Variable Documentation 790
4.152.5.1 expanders 790
4.152.5.2 no_expand 790
4.152.5.3 stack 790
4.153 SRC/zmyblas2.c File Reference 791
4.153.1 Detailed Description 791
4.153.2 Function Documentation 791
4.153.2.1 zlsolve 791
4.153.2.2 zmatvec 792
4.153.2.3 zusolve 792
4.154 SRC/zpanel_bmod.c File Reference 793

4.154.1 Detailed Description 793
4.154.2 Function Documentation 794
4.154.2.1 zcheck_tempv 794
4.154.2.2 zlsolve 794
4.154.2.3 zmatvec 794
4.154.2.4 zpanel_bmod 794
4.155 SRC/zpanel_dfs.c File Reference 795
4.155.1 Detailed Description 795
4.155.2 Function Documentation 795
4.155.2.1 zpanel_dfs 795
4.156 SRC/zpivotgrowth.c File Reference 797
4.156.1 Detailed Description 797
4.156.2 Function Documentation 797
4.156.2.1 zPivotGrowth 797
4.157 SRC/zpivotL.c File Reference 799
4.157.1 Detailed Description 799
4.157.2 Function Documentation 799
4.157.2.1 zpivotL 799
4.158 SRC/zpruneL.c File Reference 801
4.158.1 Detailed Description 801
4.158.2 Function Documentation 801
4.158.2.1 zpruneL 801
4.159 SRC/zreadhb.c File Reference 803
4.159.1 Detailed Description 803
4.159.2 Function Documentation 805
4.159.2.1 zDumpLine 805
4.159.2.2 zParseFloatFormat 805
4.159.2.3 zParseIntFormat 805
4.159.2.4 zreadhb 805
4.159.2.5 zReadValues 806
4.159.2.6 zReadVector 806
4.160 SRC/zsnode_bmod.c File Reference 807
4.160.1 Detailed Description 807
4.160.2 Function Documentation 808
4.160.2.1 zsnode_bmod 808
4.161 SRC/zsnode_dfs.c File Reference 809

4.161.1 Detailed Description 809
4.161.2 Function Documentation 809
4.161.2.1 zsnode_dfs 809
4.162 SRC/zsp_blas2.c File Reference 811
4.162.1 Detailed Description 811
4.162.2 Function Documentation 812
4.162.2.1 sp_zgemv 812
4.162.2.2 sp_ztrsv 813
4.162.2.3 zlsolve 814
4.162.2.4 zmatvec 815
4.162.2.5 zusolve 815
4.163 SRC/zsp_blas3.c File Reference 816
4.163.1 Detailed Description 816
4.163.2 Function Documentation 816
4.163.2.1 sp_zgemm 816
4.164 SRC/zutil.c File Reference 819
4.164.1 Detailed Description 820
4.164.2 Function Documentation 821
4.164.2.1 print_doublecomplex_vec 821
4.164.2.2 zcheck_tempv 821
4.164.2.3 zCompRow_to_CompCol 821
4.164.2.4 zCopy_CompCol_Matrix 821
4.164.2.5 zCopy_Dense_Matrix 821
4.164.2.6 zCreate_CompCol_Matrix 821
4.164.2.7 zCreate_CompRow_Matrix 821
4.164.2.8 zCreate_Dense_Matrix 821
4.164.2.9 zCreate_SuperNode_Matrix 822
4.164.2.10 zfill 822
4.164.2.11 zFillRHS 822
4.164.2.12 zGenXtrue 822
4.164.2.13 zinf_norm_error 823
4.164.2.14 zPrint_CompCol_Matrix 823
4.164.2.15 zPrint_Dense_Matrix 823
4.164.2.16 zprint_lu_col 823
4.164.2.17 zPrint_SuperNode_Matrix 823
4.164.2.18 zPrintPerf 823

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Colamd_Col_struct 7
Colamd_Row_struct 9
ColInfo_struct 10
complex 12
DNformat 13
doublecomplex 14
e_node (Headers for 4 types of dynamatically managed memory) 15
GlobalLU_t 16
LU_stack_t 19
mem_usage_t 20
NCformat 21
NCPformat 22
NRformat 23
NRformat_loc 24
RowInfo_struct 25
SCformat 26
SCPformat 27
superlu_options_t 28
SuperLUStat_t 30
SuperMatrix 31

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

EXAMPLE/clinsol.c 33
EXAMPLE/clinsol1.c 36
EXAMPLE/clinsolx.c 39
EXAMPLE/clinsolx1.c 43
EXAMPLE/clinsolx2.c 47
EXAMPLE/dlinsol.c 51
EXAMPLE/dlinsol1.c 54
EXAMPLE/dlinsolx.c 57
EXAMPLE/dlinsolx1.c 61
EXAMPLE/dlinsolx2.c 65
EXAMPLE/dreadtriple.c 69
EXAMPLE/slinsol.c 70
EXAMPLE/slinsol1.c 73
EXAMPLE/slinsolx.c 76
EXAMPLE/slinsolx1.c 80
EXAMPLE/slinsolx2.c 84
EXAMPLE/sp_ienv.c 681
EXAMPLE/superlu.c (Small 5x5 example) 88
EXAMPLE/zlinsol.c 91
EXAMPLE/zlinsol1.c 94
EXAMPLE/zlinsolx.c 97
EXAMPLE/zlinsolx1.c 101
EXAMPLE/zlinsolx2.c 105
EXAMPLE/zreadtriple.c 109
SRC/ccolumn_bmod.c (Performs numeric block updates) 110
SRC/ccolumn_dfs.c (Performs a symbolic factorization) 113
SRC/ccopy_to_ucol.c (Copy a computed column of U to the compressed data structure) 116
SRC/cgscon.c (Estimates reciprocal of the condition number of a general matrix) 118
SRC/cgsequ.c (Computes row and column scalings) 121
SRC/cgsrfs.c (Improves computed solution to a system of inear equations) 124
SRC/cgssv.c (Solves the system of linear equations A∗X=B) 128
SRC/cgssvx.c (Solves the system of linear equations A∗X=B or A'∗X=B) 133
SRC/cgstrf.c (Computes an LU factorization of a general sparse matrix) 142

SRC/cgstrs.c (Solves a system using LU factorization) . . . 148
SRC/clacon.c (Estimates the 1-norm) . . . 152
SRC/clangs.c (Returns the value of the one norm) . . . 154
SRC/claqgs.c (Equlibrates a general sprase matrix) . . . 156
SRC/cmemory.c (Memory details) . . . 159
SRC/cmyblas2.c (Level 2 Blas operations) . . . 171
SRC/colamd.c (A sparse matrix column ordering algorithm) . . . 173
SRC/colamd.h (Colamd prototypes and definitions) . . . 194
SRC/cpanel_bmod.c (Performs numeric block updates) . . . 200
SRC/cpanel_dfs.c (Peforms a symbolic factorization on a panel of symbols) . . . 202
SRC/cpivotgrowth.c (Computes the reciprocal pivot growth factor) . . . 204
SRC/cpivotL.c (Performs numerical pivoting) . . . 206
SRC/cpruneL.c (Prunes the L-structure) . . . 208
SRC/creadhb.c (Read a matrix stored in Harwell-Boeing format) . . . 210
SRC/csnode_bmod.c (Performs numeric block updates within the relaxed snode) . . . 214
SRC/csnode_dfs.c (Determines the union of row structures of columns within the relaxed node) 216
SRC/csp_blas2.c (Sparse BLAS 2, using some dense BLAS 2 operations) . . . 218
SRC/csp_blas3.c (Sparse BLAS3, using some dense BLAS3 operations) . . . 223
SRC/cutil.c (Matrix utility functions) . . . 226
SRC/dcolumn_bmod.c (Performs numeric block updates) . . . 231
SRC/dcolumn_dfs.c (Performs a symbolic factorization) . . . 234
SRC/dcomplex.c (Common arithmetic for complex type) . . . 237
SRC/dcopy_to_ucol.c (Copy a computed column of U to the compressed data structure) . . . 239
SRC/dGetDiagU.c (Extracts main diagonal of matrix) . . . 241
SRC/dgscon.c (Estimates reciprocal of the condition number of a general matrix) . . . 243
SRC/dgsequ.c (Computes row and column scalings) . . . 246
SRC/dgsrfs.c (Improves computed solution to a system of inear equations) . . . 249
SRC/dgssv.c (Solves the system of linear equations A∗X=B) . . . 253
SRC/dgssvx.c (Solves the system of linear equations A∗X=B or A'∗X=B) . . . 258
SRC/dgstrf.c (Computes an LU factorization of a general sparse matrix) . . . 267
SRC/dgstrs.c (Solves a system using LU factorization) . . . 273
SRC/dgstrsL.c (Performs the L-solve using the LU factorization computed by DGSTRF) . . . 277
SRC/dgstrsU.c (Performs the U-solve using the LU factorization computed by DGSTRF) . . . 280
SRC/dlacon.c (Estimates the 1-norm) . . . 283
SRC/dlamch.c (Determines double precision machine parameters) . . . 285
SRC/dlangs.c (Returns the value of the one norm) . . . 295
SRC/dlaqgs.c (Equlibrates a general sprase matrix) . . . 297
SRC/dmemory.c (Memory details) . . . 300
SRC/dmyblas2.c (Level 2 Blas operations) . . . 310
SRC/dpanel_bmod.c (Performs numeric block updates) . . . 312
SRC/dpanel_dfs.c (Peforms a symbolic factorization on a panel of symbols) . . . 314
SRC/dpivotgrowth.c (Computes the reciprocal pivot growth factor) . . . 316
SRC/dpivotL.c (Performs numerical pivoting) . . . 318
SRC/dpruneL.c (Prunes the L-structure) . . . 320
SRC/dreadhb.c (Read a matrix stored in Harwell-Boeing format) . . . 322
SRC/dsnode_bmod.c (Performs numeric block updates within the relaxed snode) . . . 326
SRC/dsnode_dfs.c (Determines the union of row structures of columns within the relaxed node) 328
SRC/dsp_blas2.c (Sparse BLAS 2, using some dense BLAS 2 operations) . . . 330
SRC/dsp_blas3.c (Sparse BLAS3, using some dense BLAS3 operations) . . . 335
SRC/dutil.c (Matrix utility functions) . . . 338
SRC/dzsum1.c (Takes sum of the absolute values of a complex vector and returns a double precision result) . . . 343
SRC/get_perm_c.c (Matrix permutation operations) . . . 345
SRC/heap_relax_snode.c (Identify the initial relaxed supernodes) . . . 350

SRC/icmax1.c (Finds the index of the element whose real part has maximum absolute value) . . 352
SRC/izmax1.c (Finds the index of the element whose real part has maximum absolute value) . . 354
SRC/lsame.c (Check if CA is the same letter as CB regardless of case) 356
SRC/memory.c (Precision-independent memory-related routines) 358
SRC/mmd.c 364
SRC/old_colamd.c 368
SRC/old_colamd.h 373
SRC/relax_snode.c (Identify initial relaxed supernodes) 375
SRC/scolumn_bmod.c (Performs numeric block updates) 377
SRC/scolumn_dfs.c (Performs a symbolic factorization) 380
SRC/scomplex.c (Common arithmetic for complex type) 383
SRC/scopy_to_ucol.c (Copy a computed column of U to the compressed data structure) 385
SRC/scsum1.c (Takes sum of the absolute values of a complex vector and returns a single precision result) 387
SRC/sgscon.c (Estimates reciprocal of the condition number of a general matrix) 389
SRC/sgsequ.c (Computes row and column scalings) 391
SRC/sgsrfs.c (Improves computed solution to a system of inear equations) 394
SRC/sgssv.c (Solves the system of linear equations A∗X=B) 398
SRC/sgssvx.c (Solves the system of linear equations A∗X=B or A'∗X=B) 403
SRC/sgstrf.c (Computes an LU factorization of a general sparse matrix) 412
SRC/sgstrs.c (Solves a system using LU factorization) 417
SRC/slacon.c (Estimates the 1-norm) 421
SRC/slamch.c (Determines single precision machine parameters and other service routines) . . 423
SRC/slangs.c (Returns the value of the one norm) 434
SRC/slaqgs.c (Equlibrates a general sprase matrix) 436
SRC/slu_cdefs.h (Header file for real operations) 439
SRC/slu_Cnames.h (Macros defining how C routines will be called) 486
SRC/slu_dcomplex.h (Header file for complex operations) 487
SRC/slu_ddefs.h (Header file for real operations) 491
SRC/slu_scomplex.h (Header file for complex operations) 537
SRC/slu_sdefs.h (Header file for real operations) 541
SRC/slu_util.h (Utility header file) 586
SRC/slu_zdefs.h (Header file for real operations) 613
SRC/smemory.c (Memory details) 660
SRC/smyblas2.c (Level 2 Blas operations) 670
SRC/sp_coletree.c (Tree layout and computation routines) 672
SRC/sp_ienv.c 679
SRC/sp_preorder.c (Permute and performs functions on columns of orginal matrix) 684
SRC/spanel_bmod.c (Performs numeric block updates) 687
SRC/spanel_dfs.c (Peforms a symbolic factorization on a panel of symbols) 689
SRC/spivotgrowth.c (Computes the reciprocal pivot growth factor) 691
SRC/spivotL.c (Performs numerical pivoting) 693
SRC/spruneL.c (Prunes the L-structure) 695
SRC/sreadhb.c (Read a matrix stored in Harwell-Boeing format) 697
SRC/ssnode_bmod.c (Performs numeric block updates within the relaxed snode) 701
SRC/ssnode_dfs.c (Determines the union of row structures of columns within the relaxed node) 703
SRC/ssp_blas2.c (Sparse BLAS 2, using some dense BLAS 2 operations) 705
SRC/ssp_blas3.c (Sparse BLAS3, using some dense BLAS3 operations) 710
SRC/superlu_timer.c (Returns the time used) 713
SRC/supermatrix.h (Defines matrix types) 715
SRC/sutil.c (Matrix utility functions) 717
SRC/util.c (Utility functions) 722
SRC/xerbla.c 730
SRC/zcolumn_bmod.c (Performs numeric block updates) 733

SRC/zcolumn_dfs.c (Performs a symbolic factorization) 736
SRC/zcopy_to_ucol.c (Copy a computed column of U to the compressed data structure) 739
SRC/zgscon.c (Estimates reciprocal of the condition number of a general matrix) 741
SRC/zgsequ.c (Computes row and column scalings) 744
SRC/zgsrfs.c (Improves computed solution to a system of inear equations) 747
SRC/zgssv.c (Solves the system of linear equations A∗X=B) 751
SRC/zgssvx.c (Solves the system of linear equations A∗X=B or A'∗X=B) 756
SRC/zgstrf.c (Computes an LU factorization of a general sparse matrix) 765
SRC/zgstrs.c (Solves a system using LU factorization) 770
SRC/zlacon.c (Estimates the 1-norm) 774
SRC/zlangs.c (Returns the value of the one norm) 776
SRC/zlaqgs.c (Equlibrates a general sprase matrix) 778
SRC/zmemory.c (Memory details) 781
SRC/zmyblas2.c (Level 2 Blas operations) 791
SRC/zpanel_bmod.c (Performs numeric block updates) 793
SRC/zpanel_dfs.c (Peforms a symbolic factorization on a panel of symbols) 795
SRC/zpivotgrowth.c (Computes the reciprocal pivot growth factor) 797
SRC/zpivotL.c (Performs numerical pivoting) 799
SRC/zpruneL.c (Prunes the L-structure) 801
SRC/zreadhb.c (Read a matrix stored in Harwell-Boeing format) 803
SRC/zsnode_bmod.c (Performs numeric block updates within the relaxed snode) 807
SRC/zsnode_dfs.c (Determines the union of row structures of columns within the relaxed node) 809
SRC/zsp_blas2.c (Sparse BLAS 2, using some dense BLAS 2 operations) 811
SRC/zsp_blas3.c (Sparse BLAS3, using some dense BLAS3 operations) 816
SRC/zutil.c (Matrix utility functions) 819

Chapter 3

Data Structure Documentation

3.1 Colamd_Col_struct Struct Reference

```
#include <colamd.h>
```

Data Fields

- int start
- int length
- union {
  int thickness
  int parent
  } shared1
- union {
  int score
  int order
  } shared2
- union {
  int headhash
  int hash
  int prev
  } shared3
- union {
  int degree_next
  int hash_next
  } shared4

3.1.1 Field Documentation

3.1.1.1 int Colamd_Col_struct::degree_next

3.1.1.2 int Colamd_Col_struct::hash

3.1.1.3 int Colamd_Col_struct::hash_next

3.1.1.4 int Colamd_Col_struct::headhash

3.1.1.5 int Colamd_Col_struct::length

3.1.1.6 int Colamd_Col_struct::order

3.1.1.7 int Colamd_Col_struct::parent

3.1.1.8 int Colamd_Col_struct::prev

3.1.1.9 int Colamd_Col_struct::score

3.1.1.10 union { ... } Colamd_Col_struct::shared1

3.1.1.11 union { ... } Colamd_Col_struct::shared2

3.1.1.12 union { ... } Colamd_Col_struct::shared3

3.1.1.13 union { ... } Colamd_Col_struct::shared4

3.1.1.14 int Colamd_Col_struct::start

3.1.1.15 int Colamd_Col_struct::thickness

The documentation for this struct was generated from the following file:

- SRC/colamd.h

3.2 Colamd_Row_struct Struct Reference

```
#include <colamd.h>
```

Data Fields

- int start
- int length
- union {
    int degree
    int p
  } shared1
- union {
    int mark
    int first_column
  } shared2

3.2.1 Field Documentation

3.2.1.1 int Colamd_Row_struct::degree

3.2.1.2 int Colamd_Row_struct::first_column

3.2.1.3 int Colamd_Row_struct::length

3.2.1.4 int Colamd_Row_struct::mark

3.2.1.5 int Colamd_Row_struct::p

3.2.1.6 union { ... } Colamd_Row_struct::shared1

3.2.1.7 union { ... } Colamd_Row_struct::shared2

3.2.1.8 int Colamd_Row_struct::start

The documentation for this struct was generated from the following file:

- SRC/colamd.h

3.3 ColInfo_struct Struct Reference

Data Fields

- int start
- int length
- union {
    int thickness
    int parent
  } shared1
- union {
    int score
    int order
  } shared2
- union {
    int headhash
    int hash
    int prev
  } shared3
- union {
    int degree_next
    int hash_next
  } shared4

3.3.1 Field Documentation

3.3.1.1 int ColInfo_struct::degree_next

3.3.1.2 int ColInfo_struct::hash

3.3.1.3 int ColInfo_struct::hash_next

3.3.1.4 int ColInfo_struct::headhash

3.3.1.5 int ColInfo_struct::length

3.3.1.6 int ColInfo_struct::order

3.3.1.7 int ColInfo_struct::parent

3.3.1.8 int ColInfo_struct::prev

3.3.1.9 int ColInfo_struct::score

3.3.1.10 union { ... } ColInfo_struct::shared1

3.3.1.11 union { ... } ColInfo_struct::shared2

3.3.1.12 union { ... } ColInfo_struct::shared3

3.3.1.13 union { ... } ColInfo_struct::shared4

3.3.1.14 int ColInfo_struct::start

3.3.1.15 int ColInfo_struct::thickness

The documentation for this struct was generated from the following file:

- SRC/old_colamd.c

3.4 complex Struct Reference

```
#include <slu_scomplex.h>
```

Data Fields

- float r
- float i

3.4.1 Field Documentation

3.4.1.1 float complex::i

3.4.1.2 float complex::r

The documentation for this struct was generated from the following file:

- SRC/slu_scomplex.h

3.5 DNformat Struct Reference

```
#include <supermatrix.h>
```

Data Fields

- int_t lda
- void * nzval

3.5.1 Field Documentation

3.5.1.1 int_t DNformat::lda

3.5.1.2 void* DNformat::nzval

The documentation for this struct was generated from the following file:

- SRC/supermatrix.h

3.6 doublecomplex Struct Reference

`#include <slu_dcomplex.h>`

Data Fields

- double r
- double i

3.6.1 Field Documentation

3.6.1.1 double doublecomplex::i

3.6.1.2 double doublecomplex::r

The documentation for this struct was generated from the following file:

- SRC/slu_dcomplex.h

3.7 e_node Struct Reference

Headers for 4 types of dynamatically managed memory.

Data Fields

- int size
- void ∗ mem

3.7.1 Field Documentation

3.7.1.1 void ∗ e_node::mem

3.7.1.2 int e_node::size

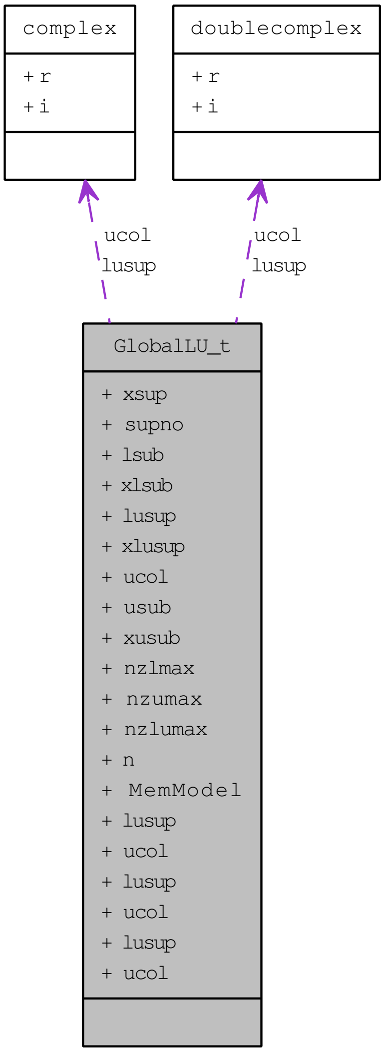
The documentation for this struct was generated from the following files:

- SRC/cmemory.c
- SRC/dmemory.c
- SRC/smemory.c
- SRC/zmemory.c

3.8 GlobalLU_t Struct Reference

`#include <slu_cdefs.h>`

Collaboration diagram for GlobalLU_t:



Data Fields

- int ∗ xsup
- int ∗ supno
- int ∗ lsub
- int ∗ xlsub
- complex ∗ lusup
- int ∗ xlusup
- complex ∗ ucol
- int ∗ usub
- int ∗ xusub

- int nzlmax
- int nzumax
- int nzlumax
- int n
- LU_space_t MemModel
- double * lusup
- double * ucol
- float * lusup
- float * ucol
- doublecomplex * lusup
- doublecomplex * ucol

3.8.1 Field Documentation

3.8.1.1 int * GlobalLU_t::lsub

3.8.1.2 doublecomplex* GlobalLU_t::lusup

3.8.1.3 float* GlobalLU_t::lusup

3.8.1.4 double* GlobalLU_t::lusup

3.8.1.5 complex* GlobalLU_t::lusup

3.8.1.6 LU_space_t GlobalLU_t::MemModel

3.8.1.7 int GlobalLU_t::n

3.8.1.8 int GlobalLU_t::nzlmax

3.8.1.9 int GlobalLU_t::nzlumax

3.8.1.10 int GlobalLU_t::nzumax

3.8.1.11 int * GlobalLU_t::supno

3.8.1.12 doublecomplex* GlobalLU_t::ucol

3.8.1.13 float* GlobalLU_t::ucol

3.8.1.14 double* GlobalLU_t::ucol

3.8.1.15 complex* GlobalLU_t::ucol

3.8.1.16 int * GlobalLU_t::usub

3.8.1.17 int * GlobalLU_t::xlsub

3.8.1.18 int * GlobalLU_t::xlusup

3.8.1.19 int * GlobalLU_t::xsup

3.8.1.20 int * GlobalLU_t::xusub

The documentation for this struct was generated from the following files:

- SRC/slu_cdefs.h
- SRC/slu_ddefs.h
- SRC/slu_sdefs.h
- SRC/slu_zdefs.h

3.9 LU_stack_t Struct Reference

Data Fields

- int size
- int used
- int top1
- int top2
- void * array

3.9.1 Field Documentation

3.9.1.1 void * LU_stack_t::array

3.9.1.2 int LU_stack_t::size

3.9.1.3 int LU_stack_t::top1

3.9.1.4 int LU_stack_t::top2

3.9.1.5 int LU_stack_t::used

The documentation for this struct was generated from the following files:

- SRC/cmemory.c
- SRC/dmemory.c
- SRC/smemory.c
- SRC/zmemory.c

3.10 mem_usage_t Struct Reference

```
#include <slu_util.h>
```

Data Fields

- float for_lu
- float total_needed
- int expansions

3.10.1 Field Documentation

3.10.1.1 int mem_usage_t::expansions

3.10.1.2 float mem_usage_t::for_lu

3.10.1.3 float mem_usage_t::total_needed

The documentation for this struct was generated from the following file:

- SRC/slu_util.h

3.11 NCformat Struct Reference

```
#include <supermatrix.h>
```

Data Fields

- int_t nnz
- void * nzval
- int_t * rowind
- int_t * colptr

3.11.1 Field Documentation

3.11.1.1 int_t* NCformat::colptr

3.11.1.2 int_t NCformat::nnz

3.11.1.3 void* NCformat::nzval

3.11.1.4 int_t* NCformat::rowind

The documentation for this struct was generated from the following file:

- SRC/supermatrix.h

3.12 NCPformat Struct Reference

```
#include <supermatrix.h>
```

Data Fields

- int_t nnz
- void * nzval
- int_t * rowind
- int_t * colbeg
- int_t * colend

3.12.1 Field Documentation

3.12.1.1 int_t* NCPformat::colbeg

3.12.1.2 int_t* NCPformat::colend

3.12.1.3 int_t NCPformat::nnz

3.12.1.4 void* NCPformat::nzval

3.12.1.5 int_t* NCPformat::rowind

The documentation for this struct was generated from the following file:

- SRC/supermatrix.h

3.13 NRformat Struct Reference

`#include <supermatrix.h>`

Data Fields

- int_t nnz
- void * nzval
- int_t * colind
- int_t * rowptr

3.13.1 Field Documentation

3.13.1.1 int_t* NRformat::colind

3.13.1.2 int_t NRformat::nnz

3.13.1.3 void* NRformat::nzval

3.13.1.4 int_t* NRformat::rowptr

The documentation for this struct was generated from the following file:

- SRC/supermatrix.h

3.14 NRformat_loc Struct Reference

`#include <supermatrix.h>`

Data Fields

- int_t nnz_loc
- int_t m_loc
- int_t fst_row
- void * nzval
- int_t * rowptr
- int_t * colind

3.14.1 Field Documentation

3.14.1.1 int_t* NRformat_loc::colind

3.14.1.2 int_t NRformat_loc::fst_row

3.14.1.3 int_t NRformat_loc::m_loc

3.14.1.4 int_t NRformat_loc::nnz_loc

3.14.1.5 void* NRformat_loc::nzval

3.14.1.6 int_t* NRformat_loc::rowptr

The documentation for this struct was generated from the following file:

- SRC/supermatrix.h

3.15 RowInfo_struct Struct Reference

Data Fields

- int start
- int length
- union {
  int degree
  int p
  } shared1

- union {
  int mark
  int first_column
  } shared2

3.15.1 Field Documentation

3.15.1.1 int RowInfo_struct::degree

3.15.1.2 int RowInfo_struct::first_column

3.15.1.3 int RowInfo_struct::length

3.15.1.4 int RowInfo_struct::mark

3.15.1.5 int RowInfo_struct::p

3.15.1.6 union { ... } RowInfo_struct::shared1

3.15.1.7 union { ... } RowInfo_struct::shared2

3.15.1.8 int RowInfo_struct::start

The documentation for this struct was generated from the following file:

- SRC/old_colamd.c

3.16 SCformat Struct Reference

```
#include <supermatrix.h>
```

Data Fields

- int_t nnz
- int_t nsuper
- void ∗ nzval
- int_t ∗ nzval_colptr
- int_t ∗ rowind
- int_t ∗ rowind_colptr
- int_t ∗ col_to_sup
- int_t ∗ sup_to_col

3.16.1 Field Documentation

3.16.1.1 int_t∗ SCformat::col_to_sup

3.16.1.2 int_t SCformat::nnz

3.16.1.3 int_t SCformat::nsuper

3.16.1.4 void∗ SCformat::nzval

3.16.1.5 int_t∗ SCformat::nzval_colptr

3.16.1.6 int_t∗ SCformat::rowind

3.16.1.7 int_t∗ SCformat::rowind_colptr

3.16.1.8 int_t∗ SCformat::sup_to_col

The documentation for this struct was generated from the following file:

- SRC/supermatrix.h

3.17 SCPformat Struct Reference

```
#include <supermatrix.h>
```

Data Fields

- int_t nnz
- int_t nsuper
- void * nzval
- int_t * nzval_colbeg
- int_t * nzval_colend
- int_t * rowind
- int_t * rowind_colbeg
- int_t * rowind_colend
- int_t * col_to_sup
- int_t * sup_to_colbeg
- int_t * sup_to_colend

3.17.1 Field Documentation

3.17.1.1 int_t* SCPformat::col_to_sup

3.17.1.2 int_t SCPformat::nnz

3.17.1.3 int_t SCPformat::nsuper

3.17.1.4 void* SCPformat::nzval

3.17.1.5 int_t* SCPformat::nzval_colbeg

3.17.1.6 int_t* SCPformat::nzval_colend

3.17.1.7 int_t* SCPformat::rowind

3.17.1.8 int_t* SCPformat::rowind_colbeg

3.17.1.9 int_t* SCPformat::rowind_colend

3.17.1.10 int_t* SCPformat::sup_to_colbeg

3.17.1.11 int_t* SCPformat::sup_to_colend

The documentation for this struct was generated from the following file:

- SRC/supermatrix.h

3.18 superlu_options_t Struct Reference

```
#include <slu_util.h>
```

Data Fields

- fact_t Fact
- yes_no_t Equil
- colperm_t ColPerm
- trans_t Trans
- IterRefine_t IterRefine
- double DiagPivotThresh
- yes_no_t PivotGrowth
- yes_no_t ConditionNumber
- rowperm_t RowPerm
- yes_no_t SymmetricMode
- yes_no_t PrintStat
- yes_no_t ReplaceTinyPivot
- yes_no_t SolveInitialized
- yes_no_t RefineInitialized

3.18.1 Field Documentation

3.18.1.1 colperm_t superlu_options_t::ColPerm

3.18.1.2 yes_no_t superlu_options_t::ConditionNumber

3.18.1.3 double superlu_options_t::DiagPivotThresh

3.18.1.4 yes_no_t superlu_options_t::Equil

3.18.1.5 fact_t superlu_options_t::Fact

3.18.1.6 IterRefine_t superlu_options_t::IterRefine

3.18.1.7 yes_no_t superlu_options_t::PivotGrowth

3.18.1.8 yes_no_t superlu_options_t::PrintStat

3.18.1.9 yes_no_t superlu_options_t::RefineInitialized

3.18.1.10 yes_no_t superlu_options_t::ReplaceTinyPivot

3.18.1.11 rowperm_t superlu_options_t::RowPerm

3.18.1.12 yes_no_t superlu_options_t::SolveInitialized

3.18.1.13 yes_no_t superlu_options_t::SymmetricMode

3.18.1.14 trans_t superlu_options_t::Trans

The documentation for this struct was generated from the following file:

- SRC/slu_util.h

3.19 SuperLUStat_t Struct Reference

`#include <slu_util.h>`

Data Fields

- int * panel_histo
- double * utime
- flops_t * ops
- int TinyPivots
- int RefineSteps

3.19.1 Field Documentation

3.19.1.1 flops_t* SuperLUStat_t::ops

3.19.1.2 int* SuperLUStat_t::panel_histo

3.19.1.3 int SuperLUStat_t::RefineSteps

3.19.1.4 int SuperLUStat_t::TinyPivots

3.19.1.5 double* SuperLUStat_t::utime

The documentation for this struct was generated from the following file:

- SRC/slu_util.h

3.20 SuperMatrix Struct Reference

```
#include <supermatrix.h>
```

Data Fields

- Stype_t Stype
- Dtype_t Dtype
- Mtype_t Mtype
- int_t nrow
- int_t ncol
- void * Store

3.20.1 Field Documentation

3.20.1.1 Dtype_t SuperMatrix::Dtype

3.20.1.2 Mtype_t SuperMatrix::Mtype

3.20.1.3 int_t SuperMatrix::ncol

3.20.1.4 int_t SuperMatrix::nrow

3.20.1.5 void* SuperMatrix::Store

3.20.1.6 Stype_t SuperMatrix::Stype

The documentation for this struct was generated from the following file:

- SRC/supermatrix.h

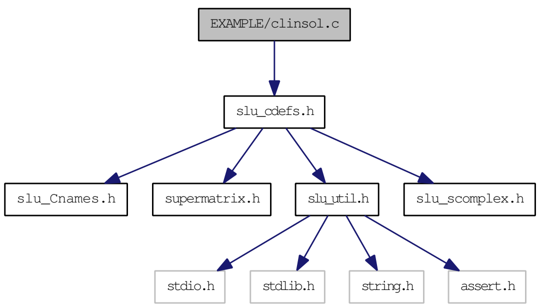
Chapter 4

File Documentation

4.1 EXAMPLE/clinsol.c File Reference

```
#include "slu_cdefs.h"
```

Include dependency graph for clinsol.c:



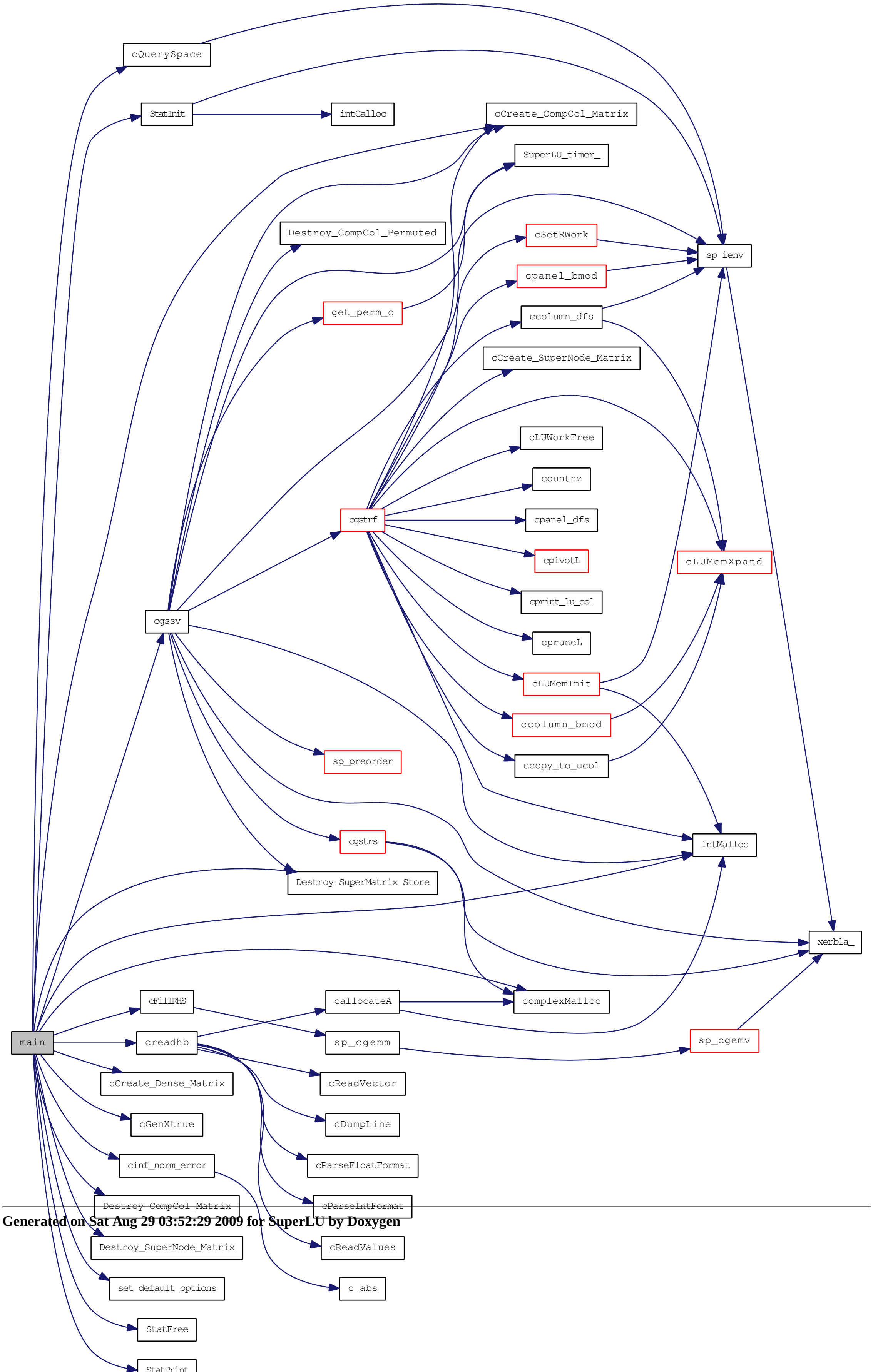
Functions

- main (int argc, char ∗argv[ ])

4.1.1 Function Documentation

4.1.1.1 main (int *argc*, char ∗ *argv*[ ])

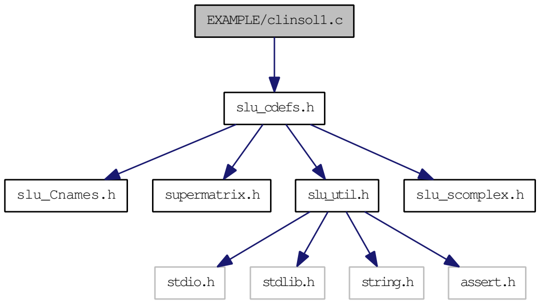
Here is the call graph for this function:



4.2 EXAMPLE/clinsol1.c File Reference

```
#include "slu_cdefs.h"
```

Include dependency graph for clinsol1.c:



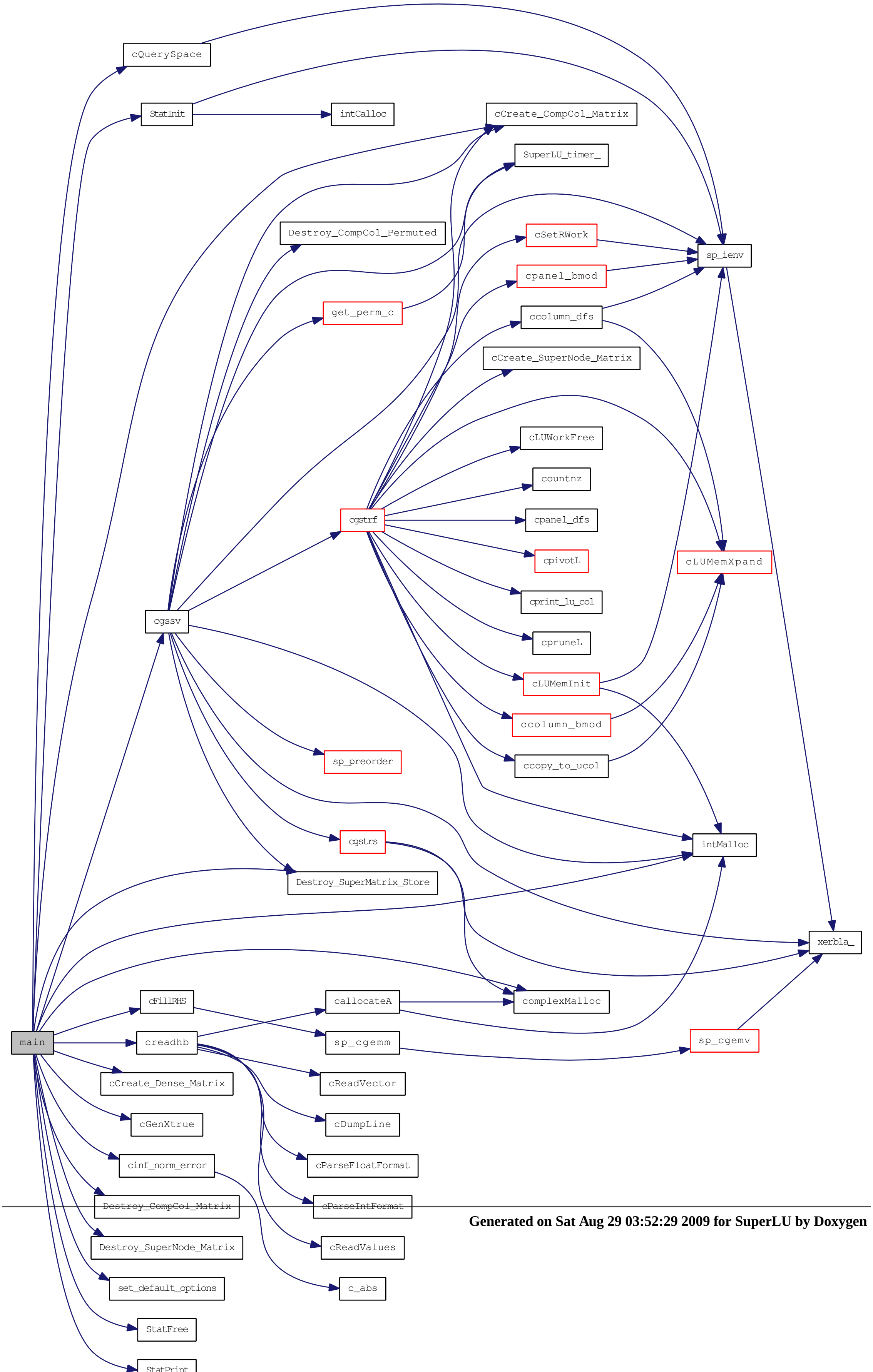
Functions

- main (int argc, char ∗argv[ ])

4.2.1 Function Documentation

4.2.1.1 main (int *argc*, char ∗ *argv*[ ])

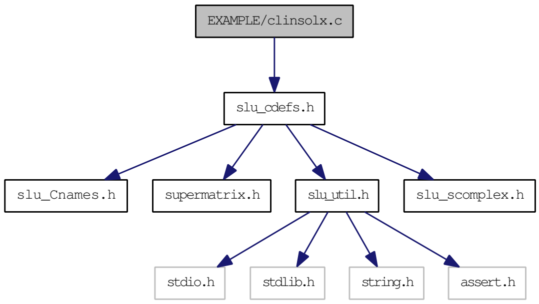
Here is the call graph for this function:



4.3 EXAMPLE/clinsolx.c File Reference

`#include "slu_cdefs.h"`

Include dependency graph for clinsolx.c:



Functions

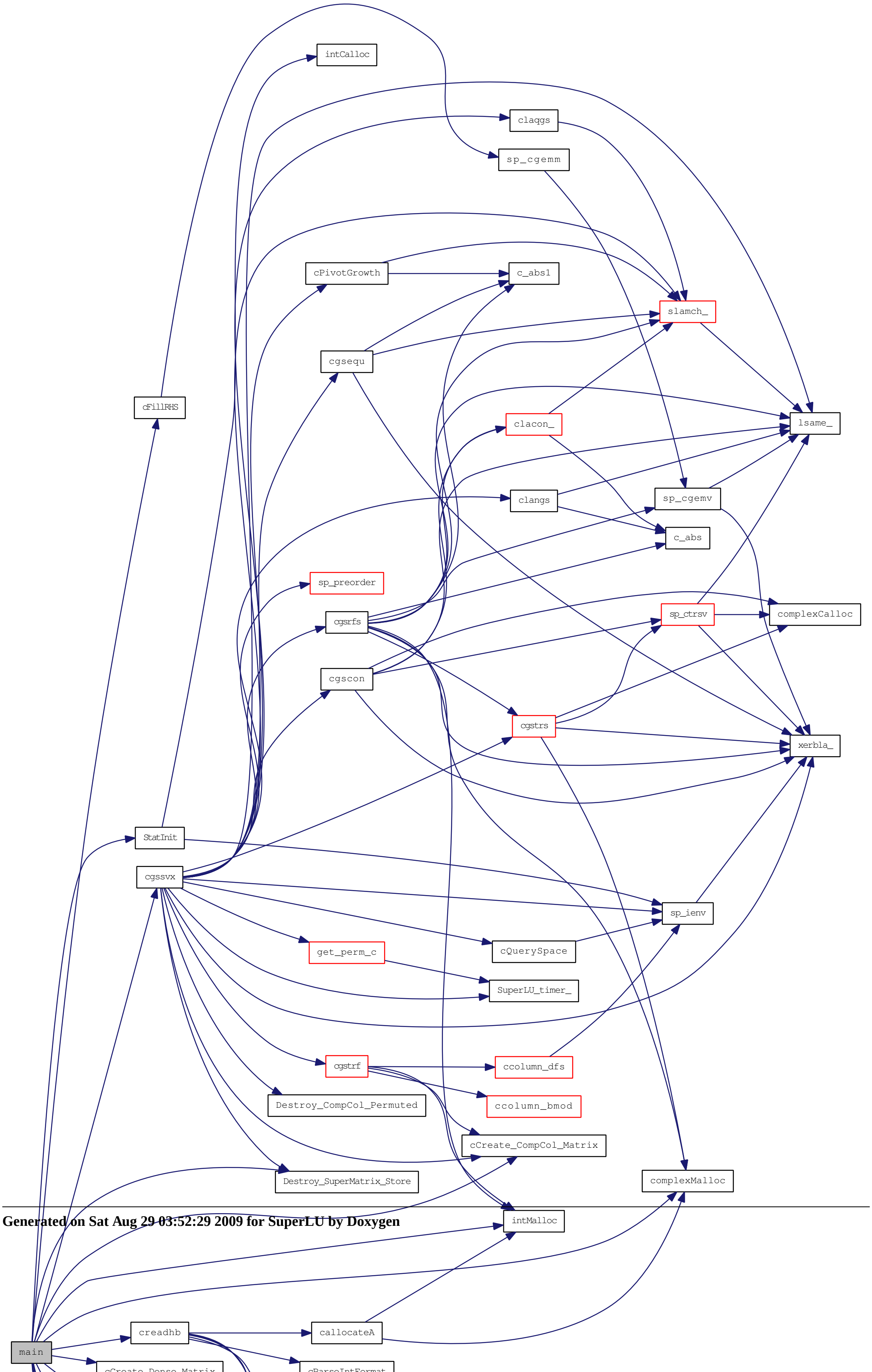
- main (int argc, char ∗argv[ ])

- void parse_command_line (int argc, char ∗argv[ ], int ∗lwork, float ∗u, yes_no_t ∗equil, trans_-t ∗trans)

4.3.1 Function Documentation

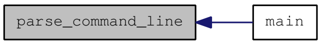
4.3.1.1 main (int *argc*, char ∗ *argv*[ ])

Here is the call graph for this function:



4.3.1.2 void parse_command_line (int *argc*, char * *argv*[ ], int * *lwork*, float * *u*, yes_no_t * *equil*, trans_t * *trans*)

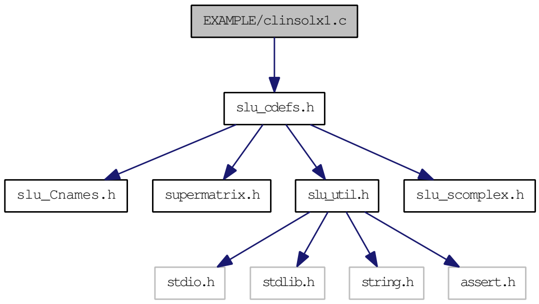
Here is the caller graph for this function:



4.4 EXAMPLE/clinsolx1.c File Reference

```
#include "slu_cdefs.h"
```

Include dependency graph for clinsolx1.c:



Functions

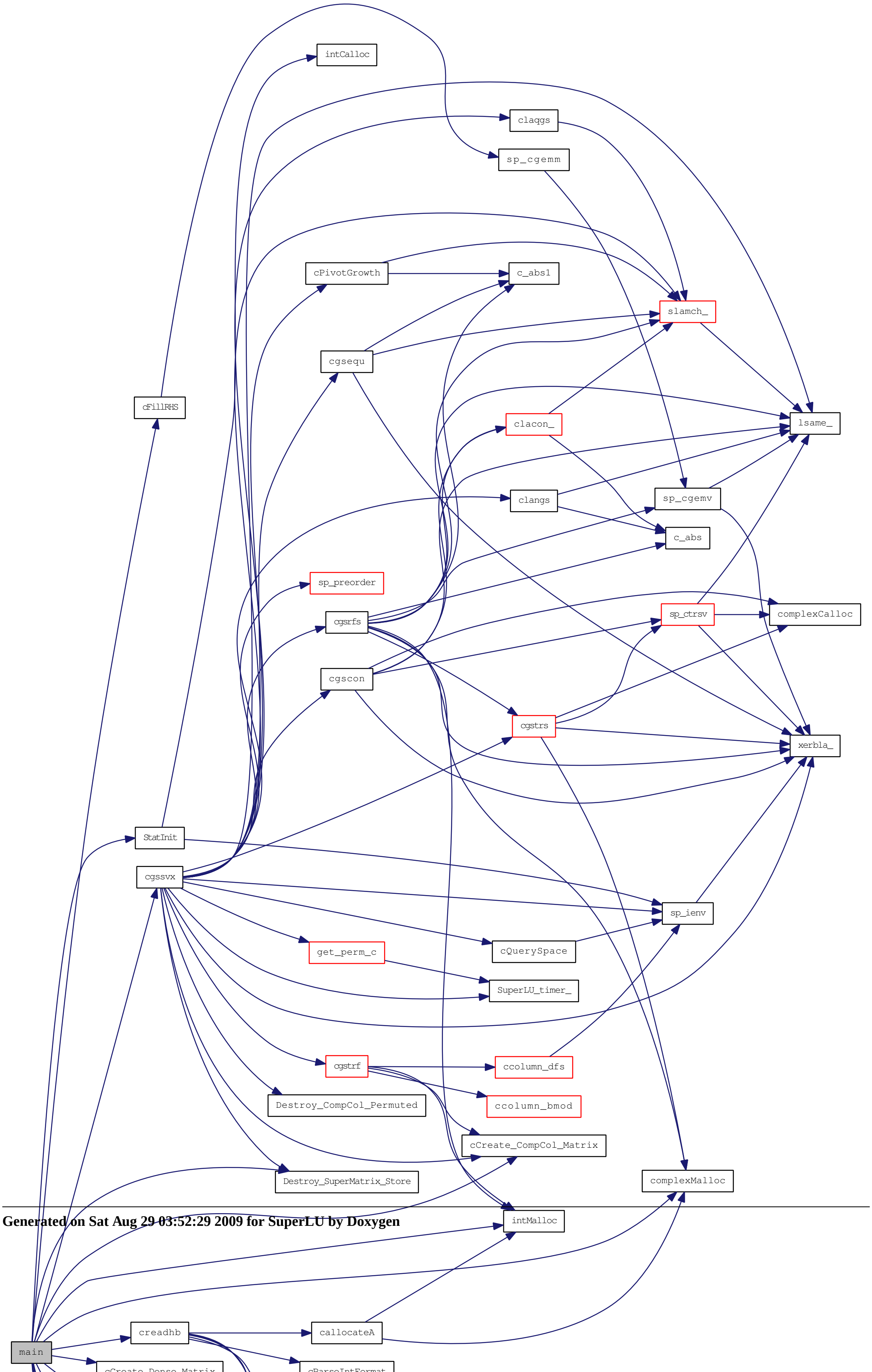
- main (int argc, char ∗argv[ ])

- void parse_command_line (int argc, char ∗argv[ ], int ∗lwork, float ∗u, yes_no_t ∗equil, trans_-t ∗trans)

4.4.1 Function Documentation

4.4.1.1 main (int *argc*, char ∗ *argv*[ ])

Here is the call graph for this function:

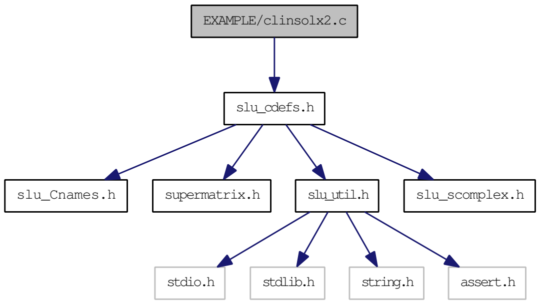


4.4.1.2 void parse_command_line (int *argc*, char * *argv*[ ], int * *lwork*, float * *u*, yes_no_t * *equil*, trans_t * *trans*)

4.5 EXAMPLE/clinsolx2.c File Reference

`#include "slu_cdefs.h"`

Include dependency graph for clinsolx2.c:



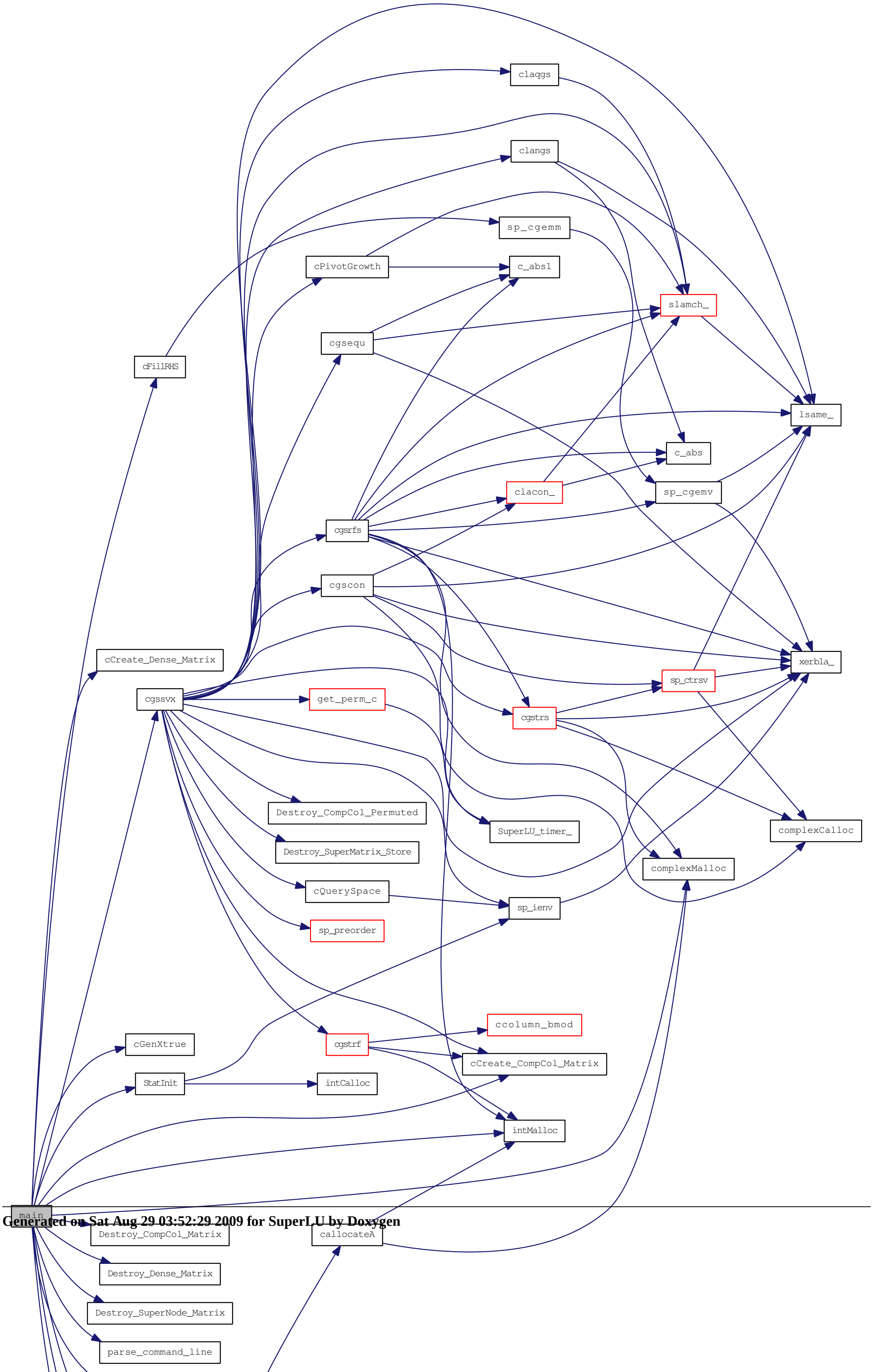
Functions

- main (int argc, char ∗argv[ ])
- void parse_command_line (int argc, char ∗argv[ ], int ∗lwork, double ∗u, yes_no_t ∗equil, trans_t ∗trans)

4.5.1 Function Documentation

4.5.1.1 main (int *argc*, char ∗ *argv*[ ])

Here is the call graph for this function:

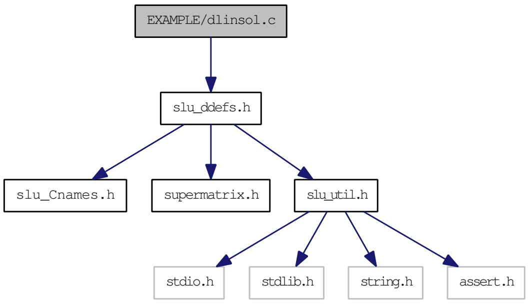


4.5.1.2 void parse_command_line (int *argc*, char ∗ *argv*[ ], int ∗ *lwork*, double ∗ *u*, yes_no_t ∗ *equil*, trans_t ∗ *trans*)

4.6 EXAMPLE/dlinsol.c File Reference

```
#include "slu_ddefs.h"
```

Include dependency graph for dlinsol.c:



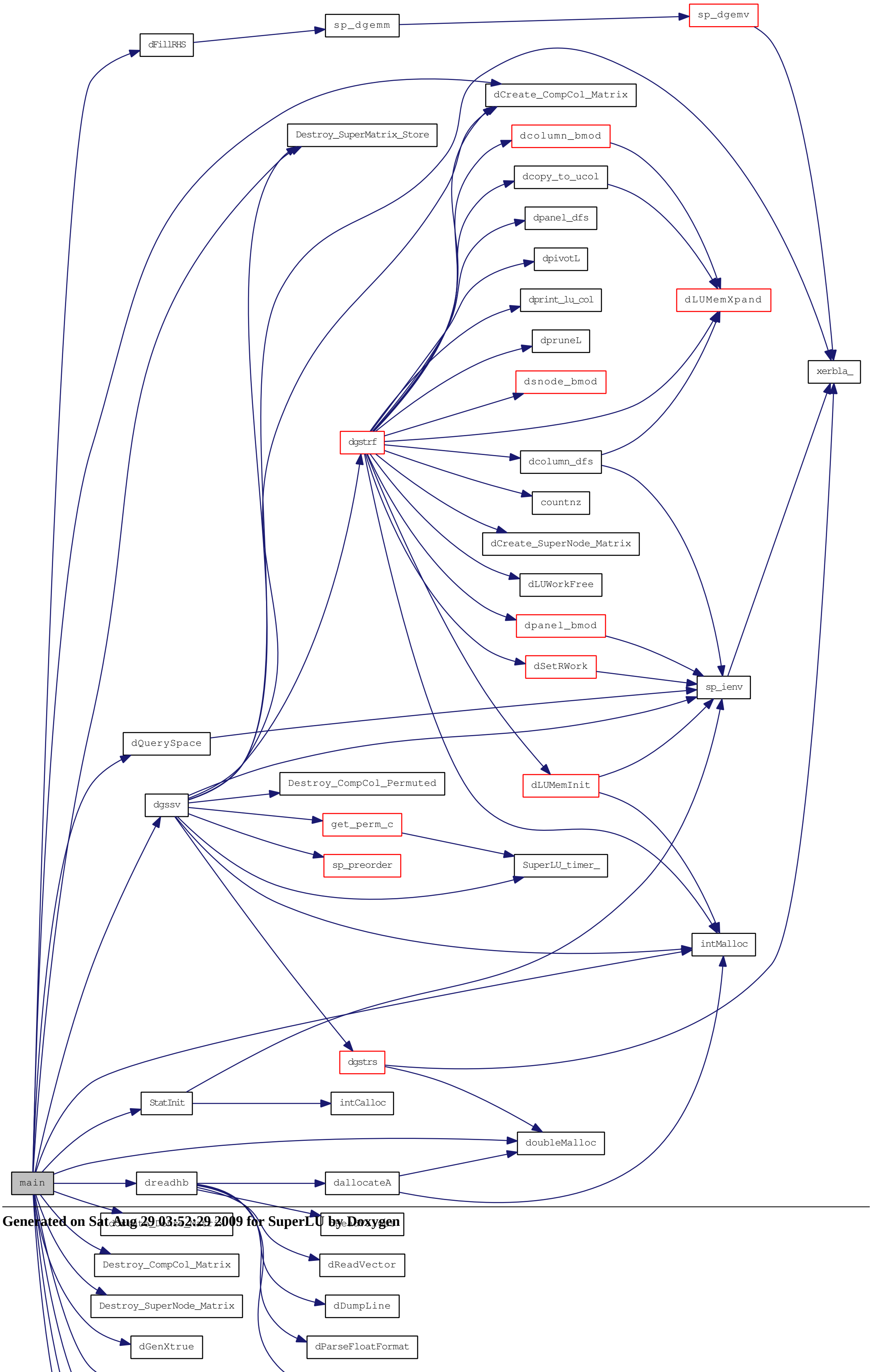
Functions

- main (int argc, char ∗argv[ ])

4.6.1 Function Documentation

4.6.1.1 main (int *argc*, char ∗ *argv*[ ])

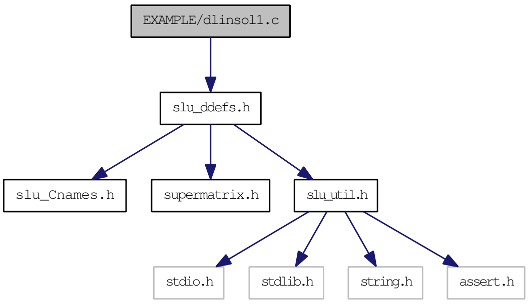
Here is the call graph for this function:



4.7 EXAMPLE/dlinsol1.c File Reference

```
#include "slu_ddefs.h"
```

Include dependency graph for dlinsol1.c:



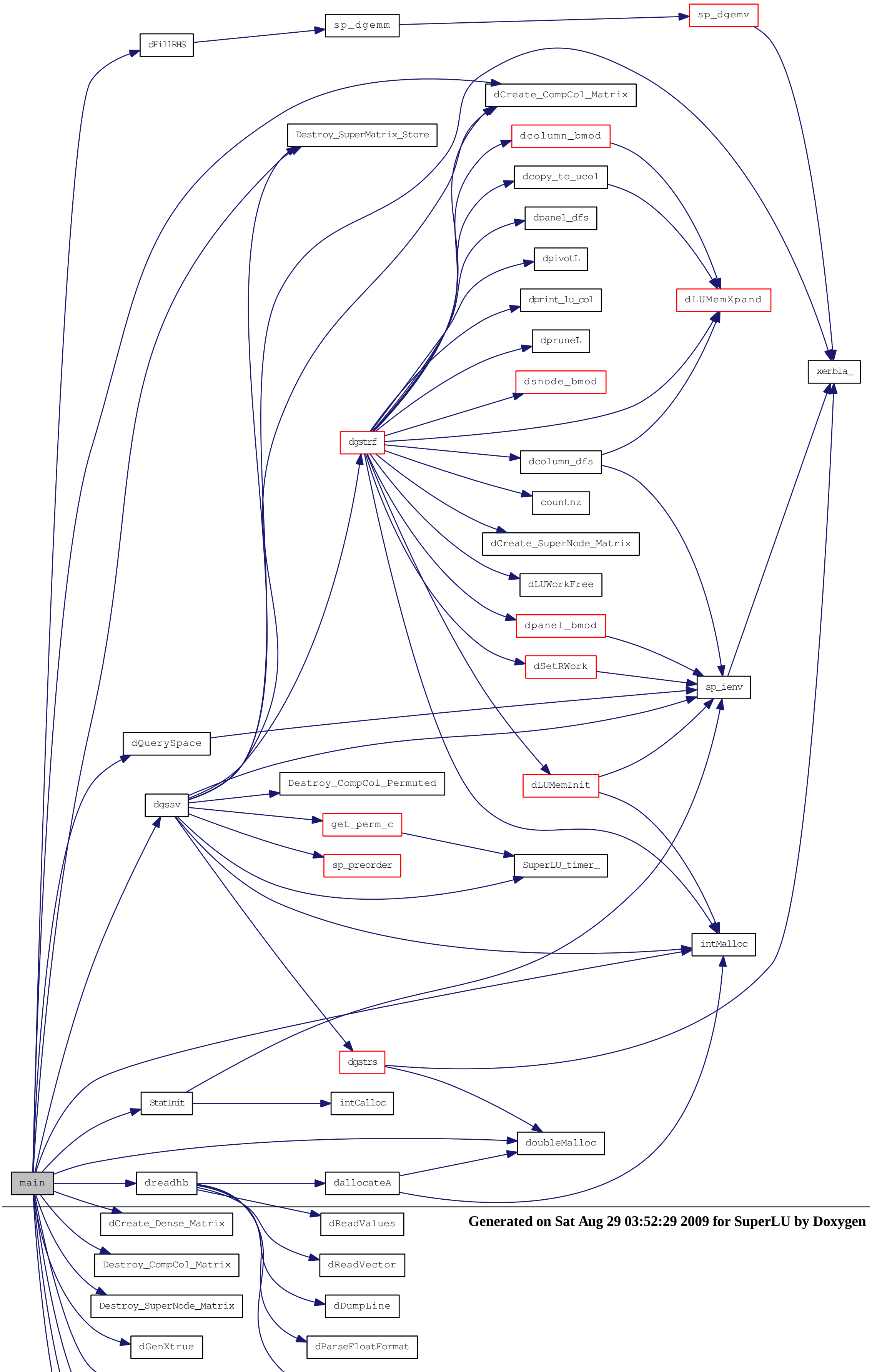
Functions

- main (int argc, char ∗argv[ ])

4.7.1 Function Documentation

4.7.1.1 main (int *argc*, char ∗ *argv*[ ])

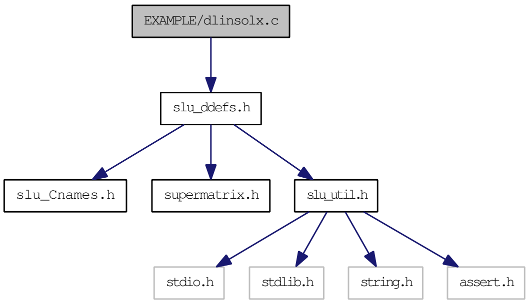
Here is the call graph for this function:



4.8 EXAMPLE/dlinsolx.c File Reference

```
#include "slu_ddefs.h"
```

Include dependency graph for dlinsolx.c:



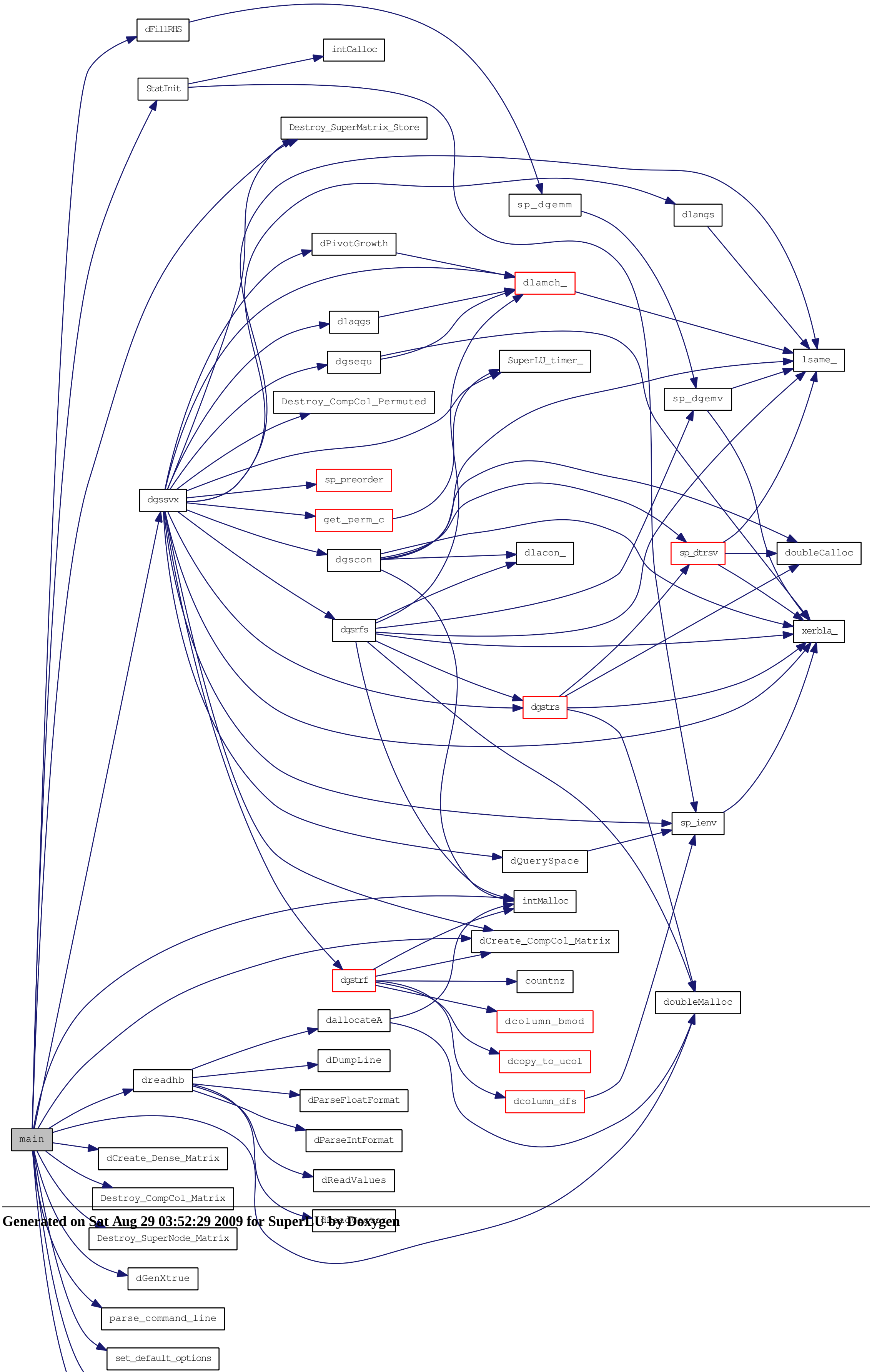
Functions

- main (int argc, char *argv[ ])
- void parse_command_line (int argc, char *argv[ ], int *lwork, double *u, yes_no_t *equil, trans_t *trans)

4.8.1 Function Documentation

4.8.1.1 main (int *argc*, char ∗ *argv*[ ])

Here is the call graph for this function:

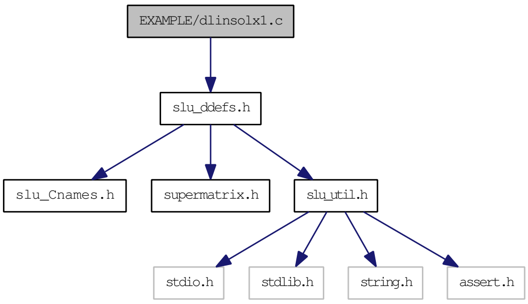


4.8.1.2 void parse_command_line (int *argc*, char * *argv*[ ], int * *lwork*, double * *u*, yes_no_t * *equil*, trans_t * *trans*)

4.9 EXAMPLE/dlinsolx1.c File Reference

```
#include "slu_ddefs.h"
```

Include dependency graph for dlinsolx1.c:



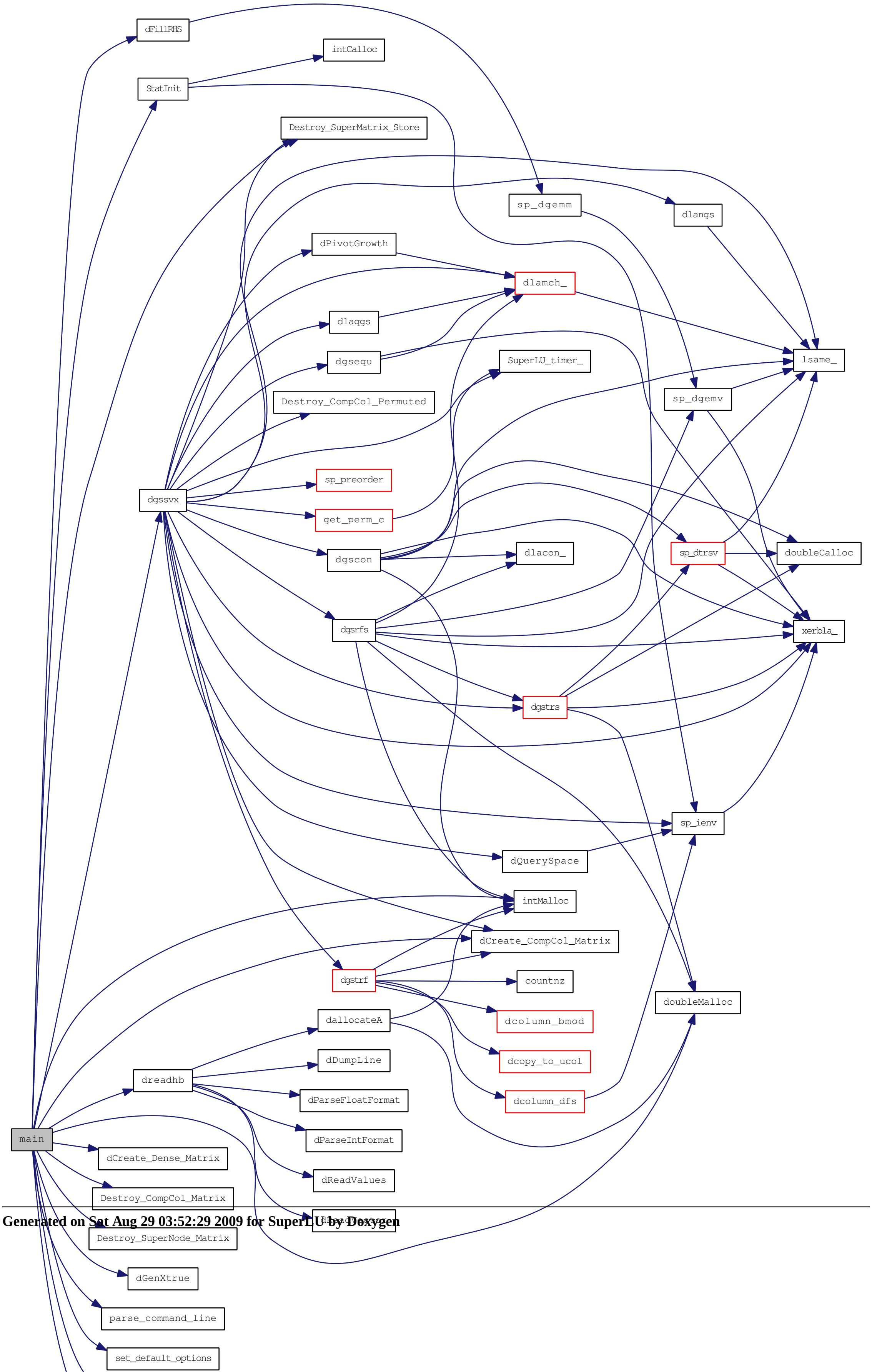
Functions

- main (int argc, char *argv[ ])
- void parse_command_line (int argc, char *argv[ ], int *lwork, double *u, yes_no_t *equil, trans_t *trans)

4.9.1 Function Documentation

4.9.1.1 main (int *argc*, char ∗ *argv*[ ])

Here is the call graph for this function:

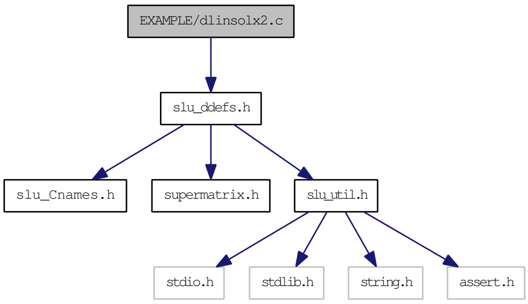


4.9.1.2 void parse_command_line (int *argc*, char ∗ *argv*[ ], int ∗ *lwork*, double ∗ *u*, yes_no_t ∗ *equil*, trans_t ∗ *trans*)

4.10 EXAMPLE/dlinsolx2.c File Reference

`#include "slu_ddefs.h"`

Include dependency graph for dlinsolx2.c:



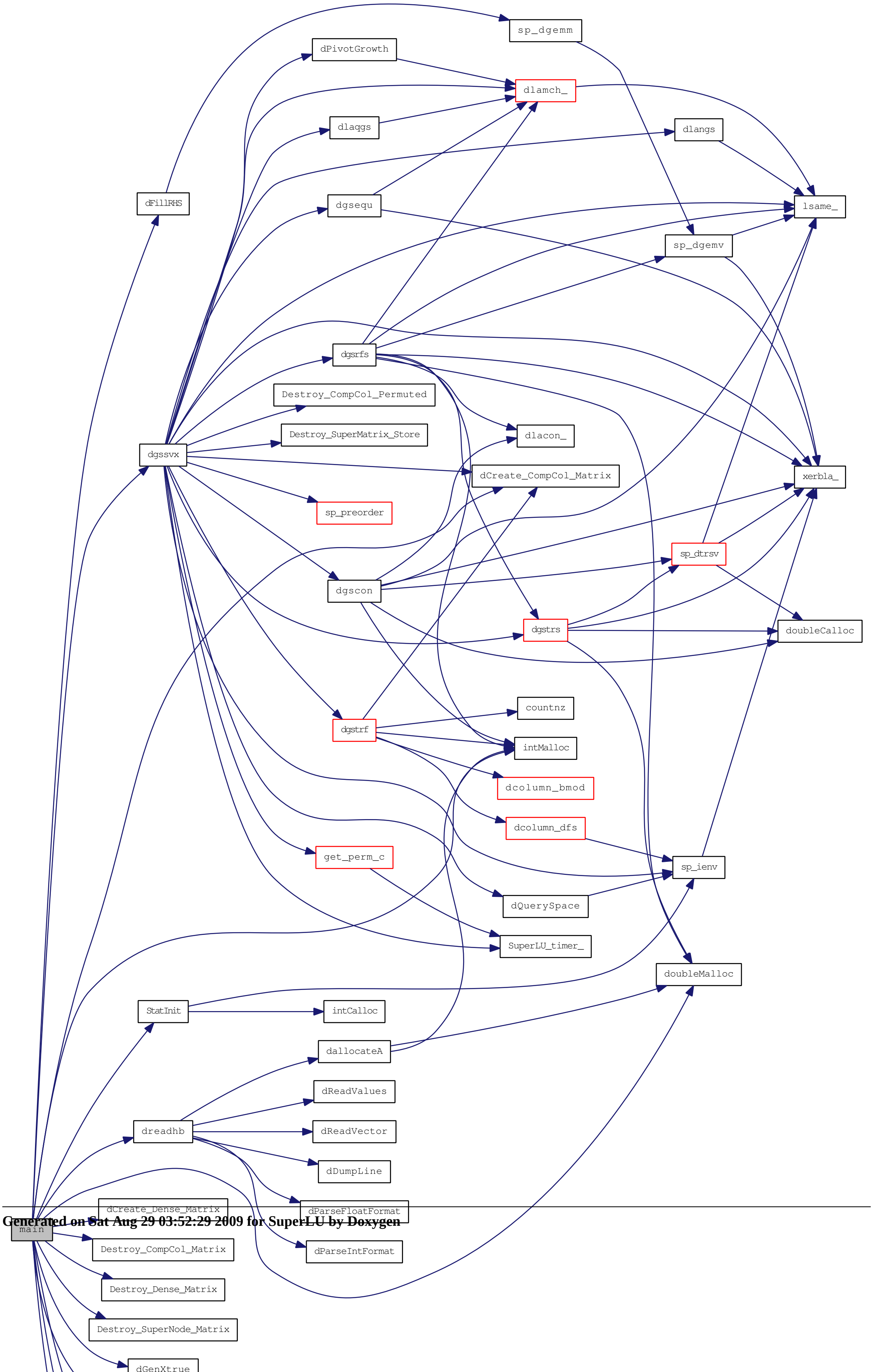
Functions

- main (int argc, char ∗argv[ ])
- void parse_command_line (int argc, char ∗argv[ ], int ∗lwork, double ∗u, yes_no_t ∗equil, trans_t ∗trans)

4.10.1 Function Documentation

4.10.1.1 main (int *argc*, char ∗ *argv*[ ])

Here is the call graph for this function:

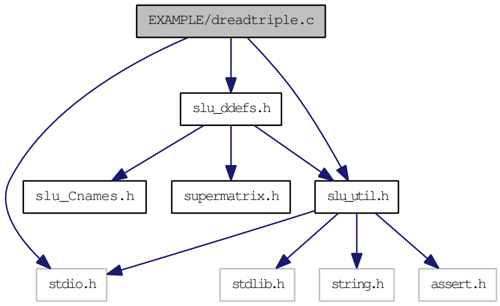


4.10.1.2 void parse_command_line (int *argc*, char $*$ *argv*[ ], int $*$ *lwork*, double $*$ *u*, yes_no_t $*$ *equil*, trans_t $*$ *trans*)

4.11 EXAMPLE/dreadtriple.c File Reference

```
#include <stdio.h>
#include "slu_ddefs.h"
#include "slu_util.h"
```

Include dependency graph for dreadtriple.c:



Functions

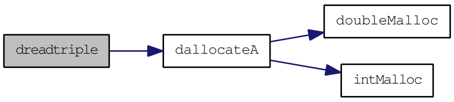
- void dreadtriple (int ∗m, int ∗n, int ∗nonz, double ∗∗nzval, int ∗∗rowind, int ∗∗colptr)
- void dreadrhs (int m, double ∗b)

4.11.1 Function Documentation

4.11.1.1 void dreadrhs (int *m*, double ∗ *b*)

4.11.1.2 void dreadtriple (int ∗ *m*, int ∗ *n*, int ∗ *nonz*, double ∗∗ *nzval*, int ∗∗ *rowind*, int ∗∗ *colptr*)

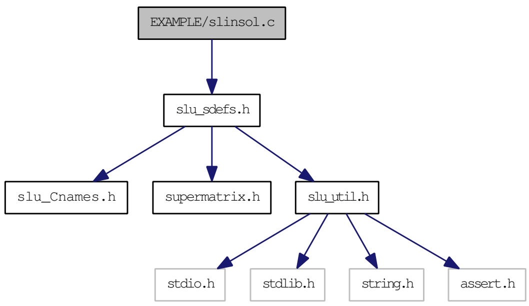
Here is the call graph for this function:



4.12 EXAMPLE/slinsol.c File Reference

```
#include "slu_sdefs.h"
```

Include dependency graph for slinsol.c:



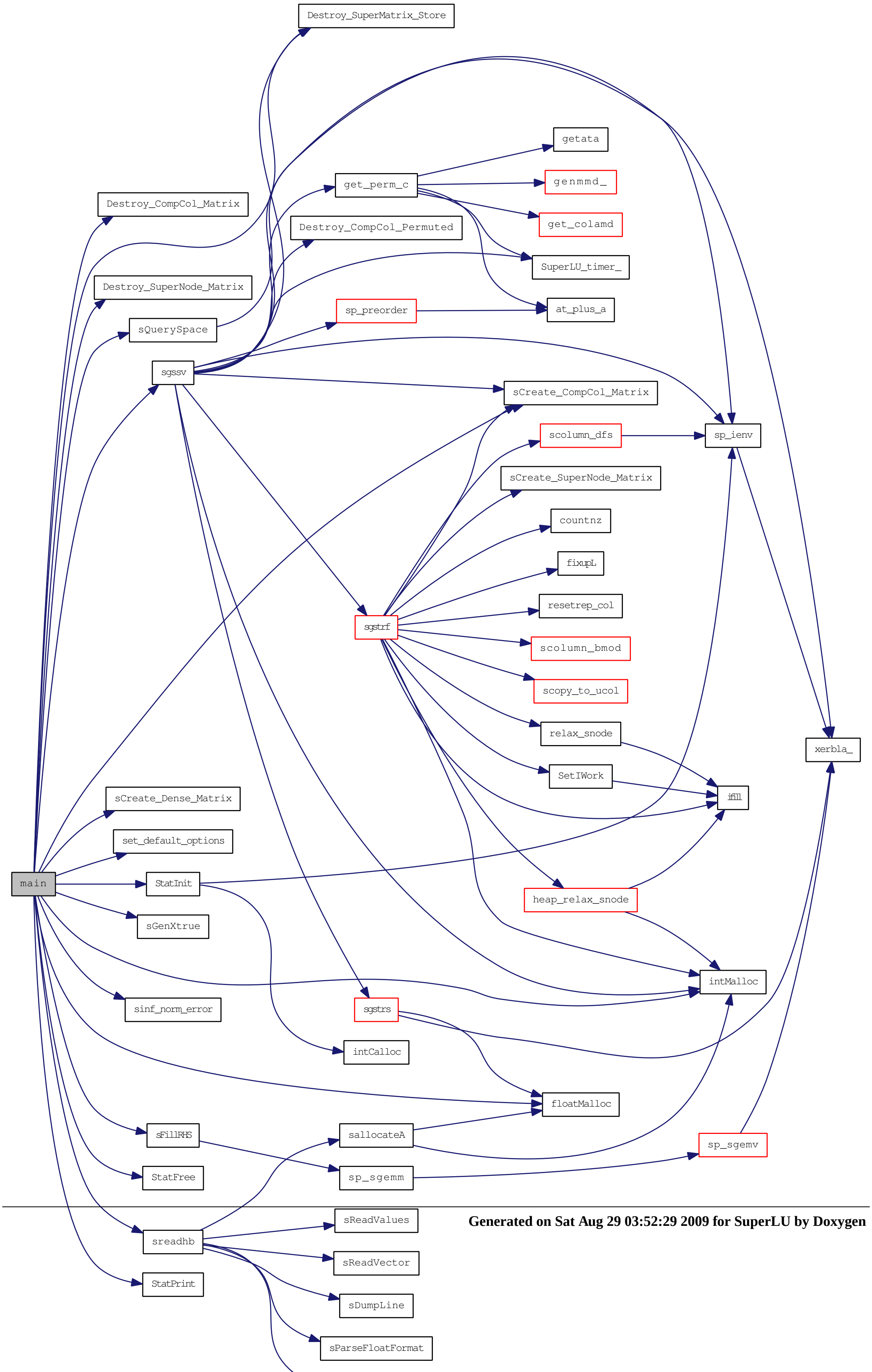
Functions

- main (int argc, char ∗argv[ ])

4.12.1 Function Documentation

4.12.1.1 main (int *argc*, char ∗ *argv*[ ])

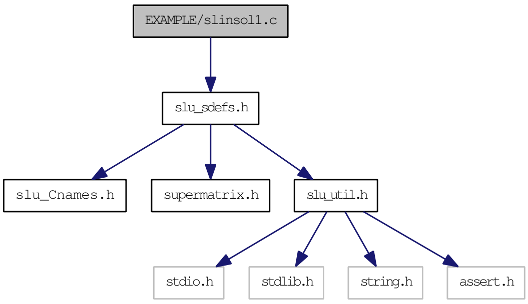
Here is the call graph for this function:



4.13 EXAMPLE/slinsol1.c File Reference

```
#include "slu_sdefs.h"
```

Include dependency graph for slinsol1.c:



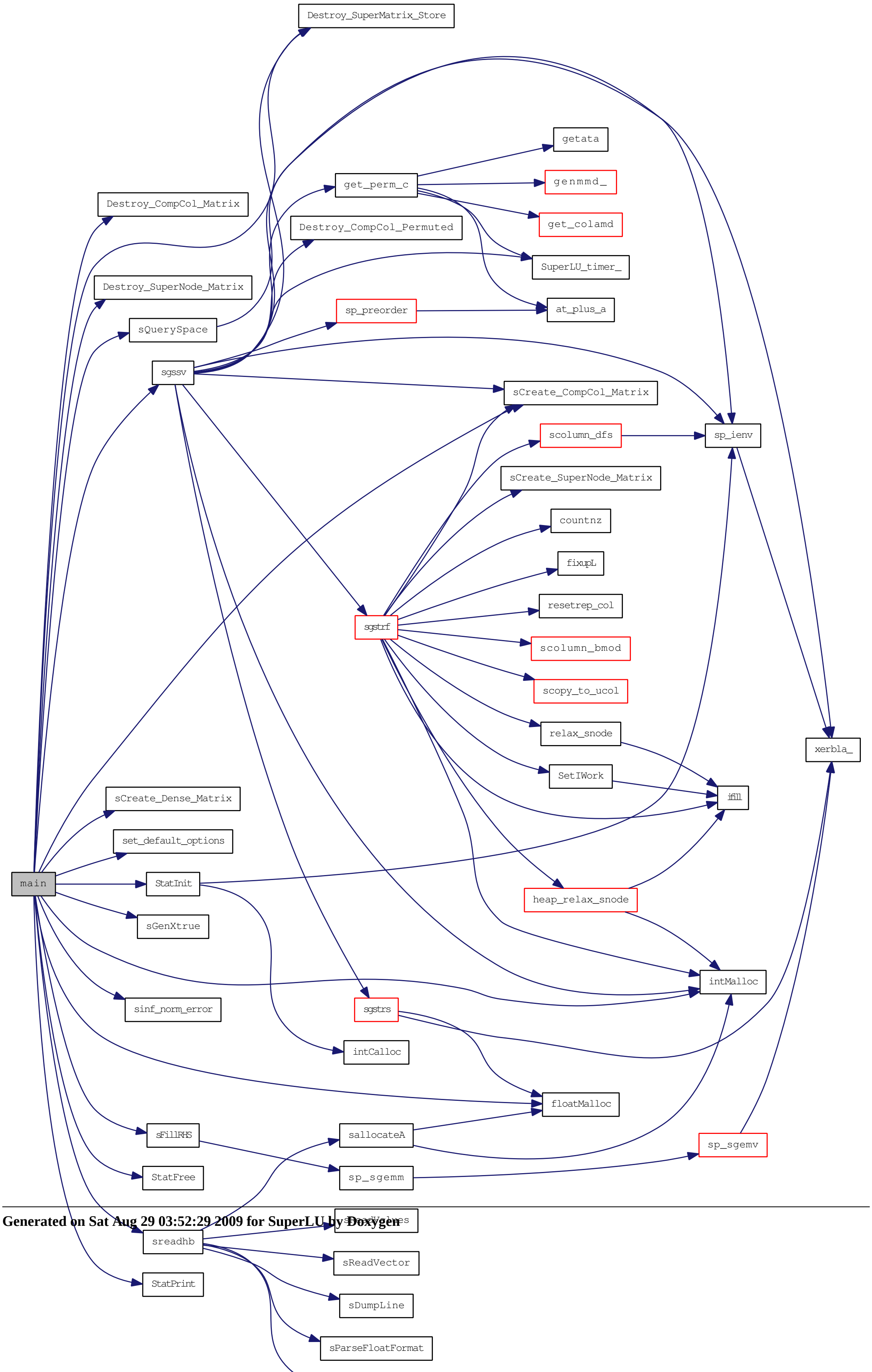
Functions

- main (int argc, char ∗argv[ ])

4.13.1 Function Documentation

4.13.1.1 main (int *argc*, char ∗ *argv*[ ])

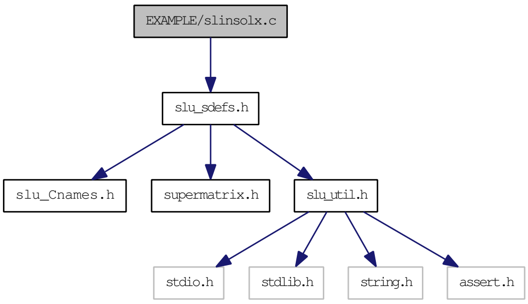
Here is the call graph for this function:



4.14 EXAMPLE/slinsolx.c File Reference

```
#include "slu_sdefs.h"
```

Include dependency graph for slinsolx.c:



Functions

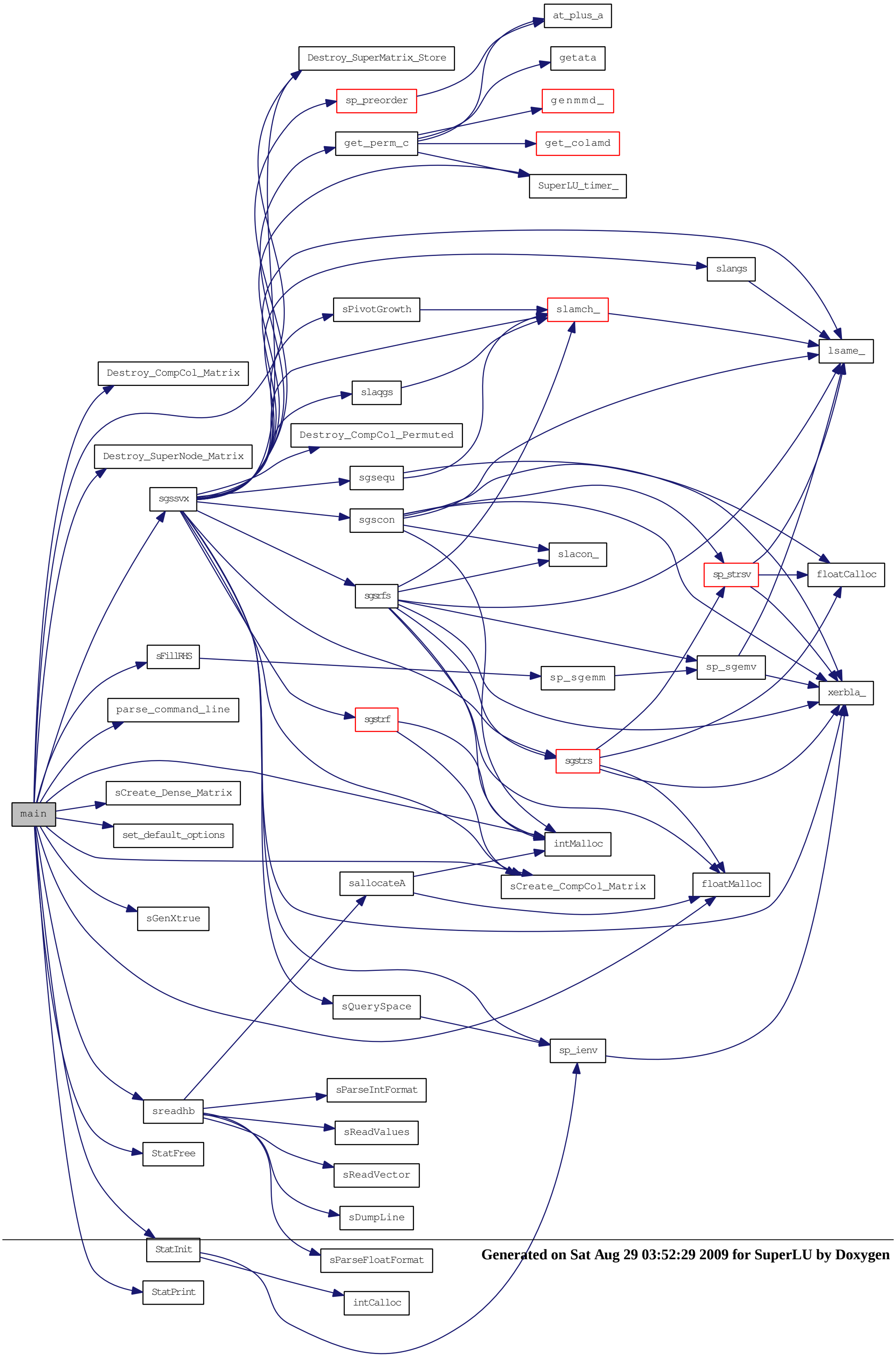
- main (int argc, char *argv[ ])

- void parse_command_line (int argc, char *argv[ ], int *lwork, float *u, yes_no_t *equil, trans_t *trans)

4.14.1 Function Documentation

4.14.1.1 main (int *argc*, char ∗ *argv*[ ])

Here is the call graph for this function:

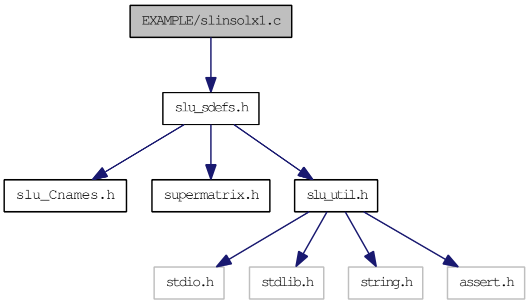


4.14.1.2 void parse_command_line (int *argc*, char ∗ *argv*[ ], int ∗ *lwork*, float ∗ *u*, yes_no_t ∗ *equil*, trans_t ∗ *trans*)

4.15 EXAMPLE/slinsolx1.c File Reference

```
#include "slu_sdefs.h"
```

Include dependency graph for slinsolx1.c:



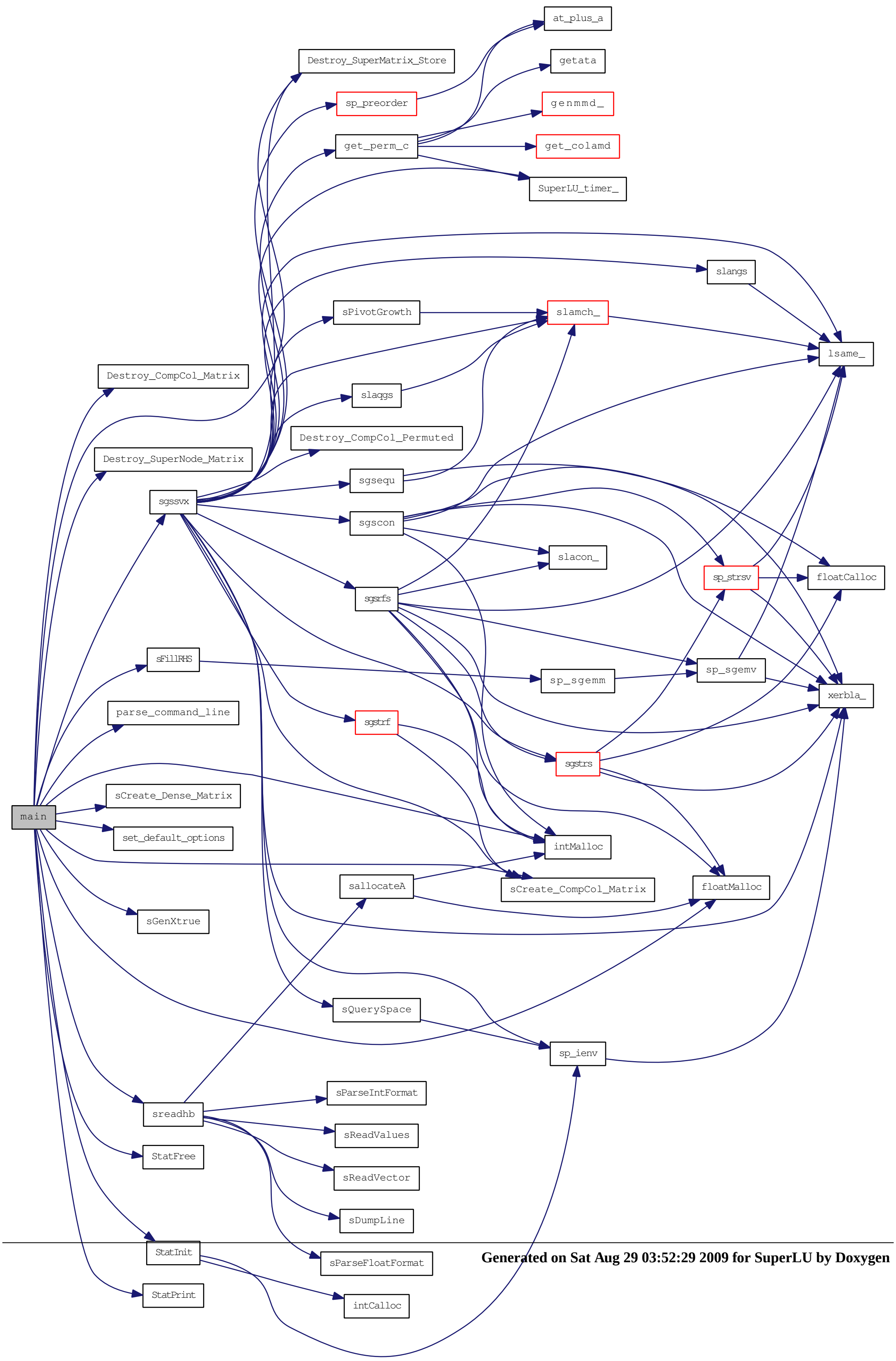
Functions

- main (int argc, char ∗argv[ ])
- void parse_command_line (int argc, char ∗argv[ ], int ∗lwork, float ∗u, yes_no_t ∗equil, trans_t ∗trans)

4.15.1 Function Documentation

4.15.1.1 main (int *argc*, char ∗ *argv*[ ])

Here is the call graph for this function:

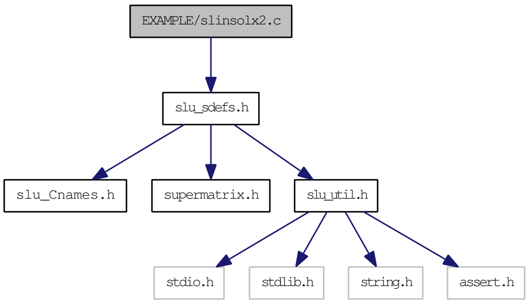


4.15.1.2 void parse_command_line (int *argc*, char ∗ *argv*[ ], int ∗ *lwork*, float ∗ *u*, yes_no_t ∗ *equil*, trans_t ∗ *trans*)

4.16 EXAMPLE/slinsolx2.c File Reference

`#include "slu_sdefs.h"`

Include dependency graph for slinsolx2.c:



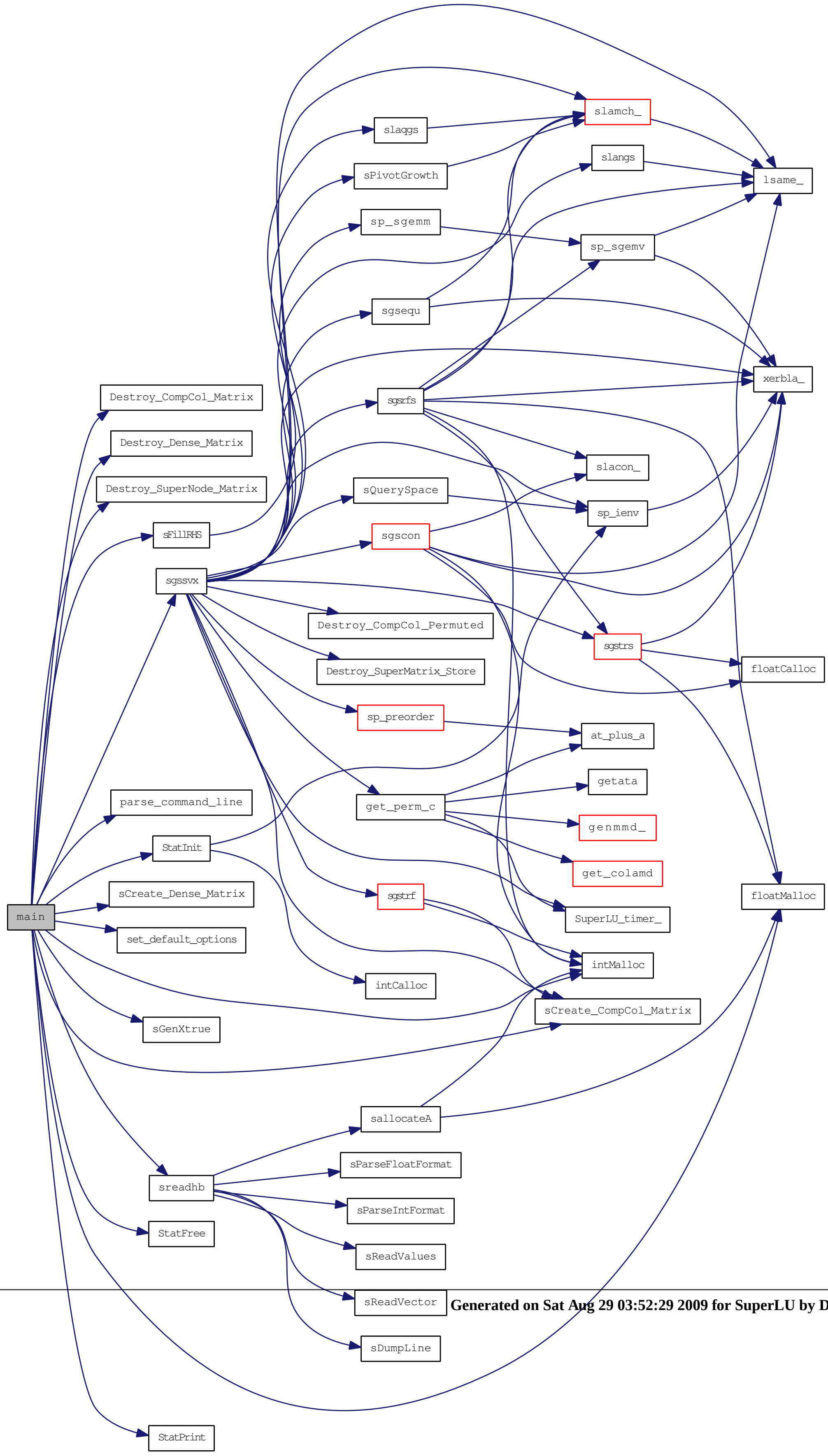
Functions

- main (int argc, char *argv[ ])
- void parse_command_line (int argc, char *argv[ ], int *lwork, double *u, yes_no_t *equil, trans_t *trans)

4.16.1 Function Documentation

4.16.1.1 main (int *argc*, char ∗ *argv*[ ])

Here is the call graph for this function:



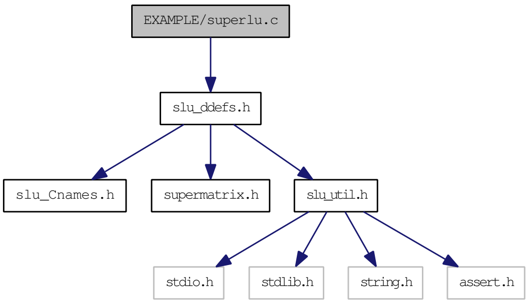
4.16.1.2 void parse_command_line (int *argc*, char ∗ *argv*[ ], int ∗ *lwork*, double ∗ *u*, yes_no_t ∗ *equil*, trans_t ∗ *trans*)

4.17 EXAMPLE/superlu.c File Reference

a small 5x5 example

`#include "slu_ddefs.h"`

Include dependency graph for superlu.c:



Functions

- main (int argc, char ∗argv[ ])

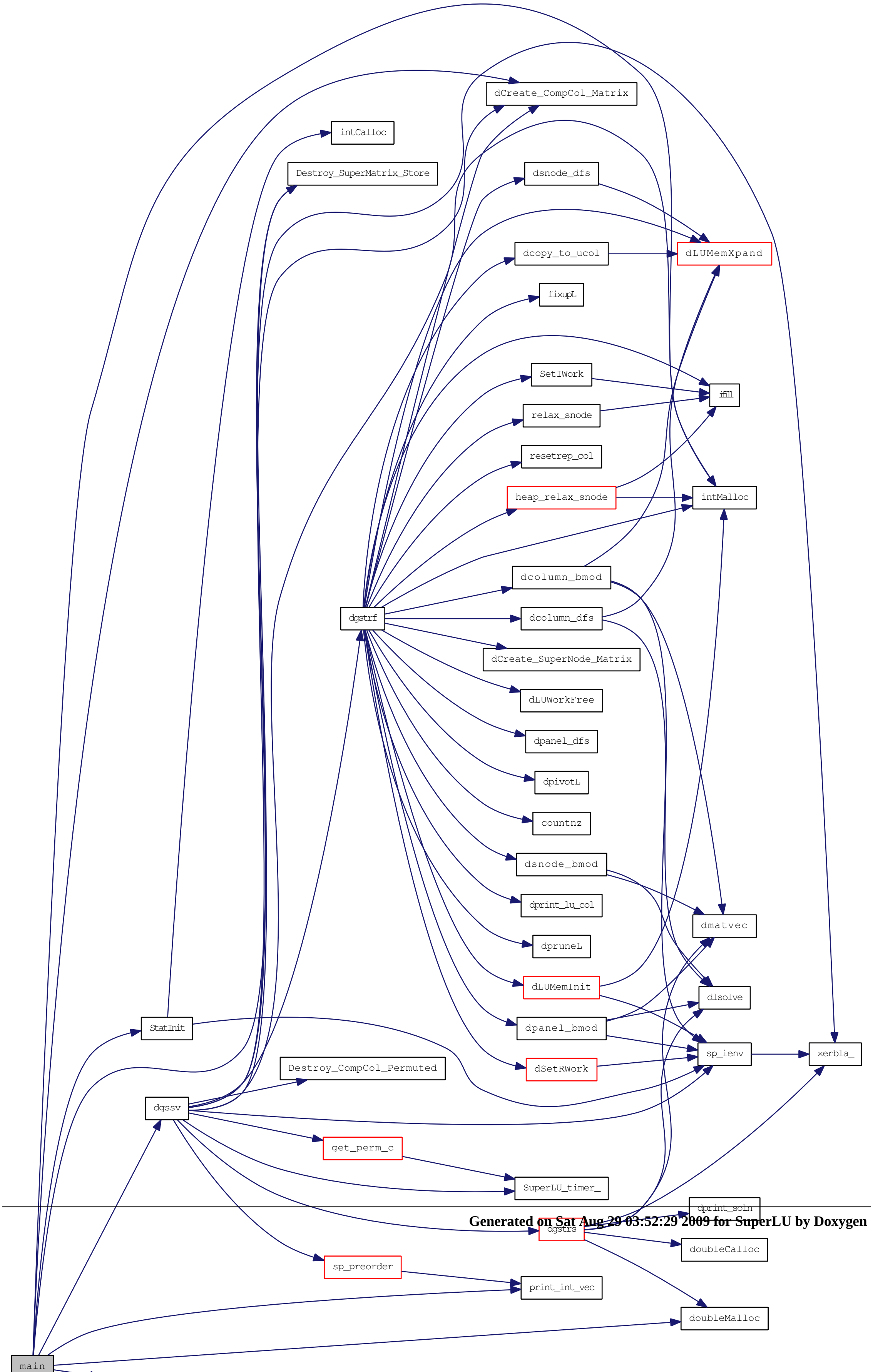
4.17.1 Detailed Description

```
* -- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997
```

4.17.2 Function Documentation

4.17.2.1 main (int *argc*, char ∗ *argv*[ ])

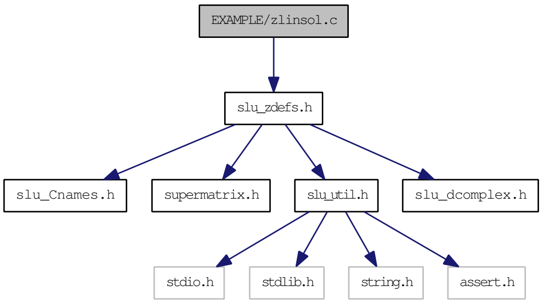
Here is the call graph for this function:



4.18 EXAMPLE/zlinsol.c File Reference

`#include "slu_zdefs.h"`

Include dependency graph for zlinsol.c:



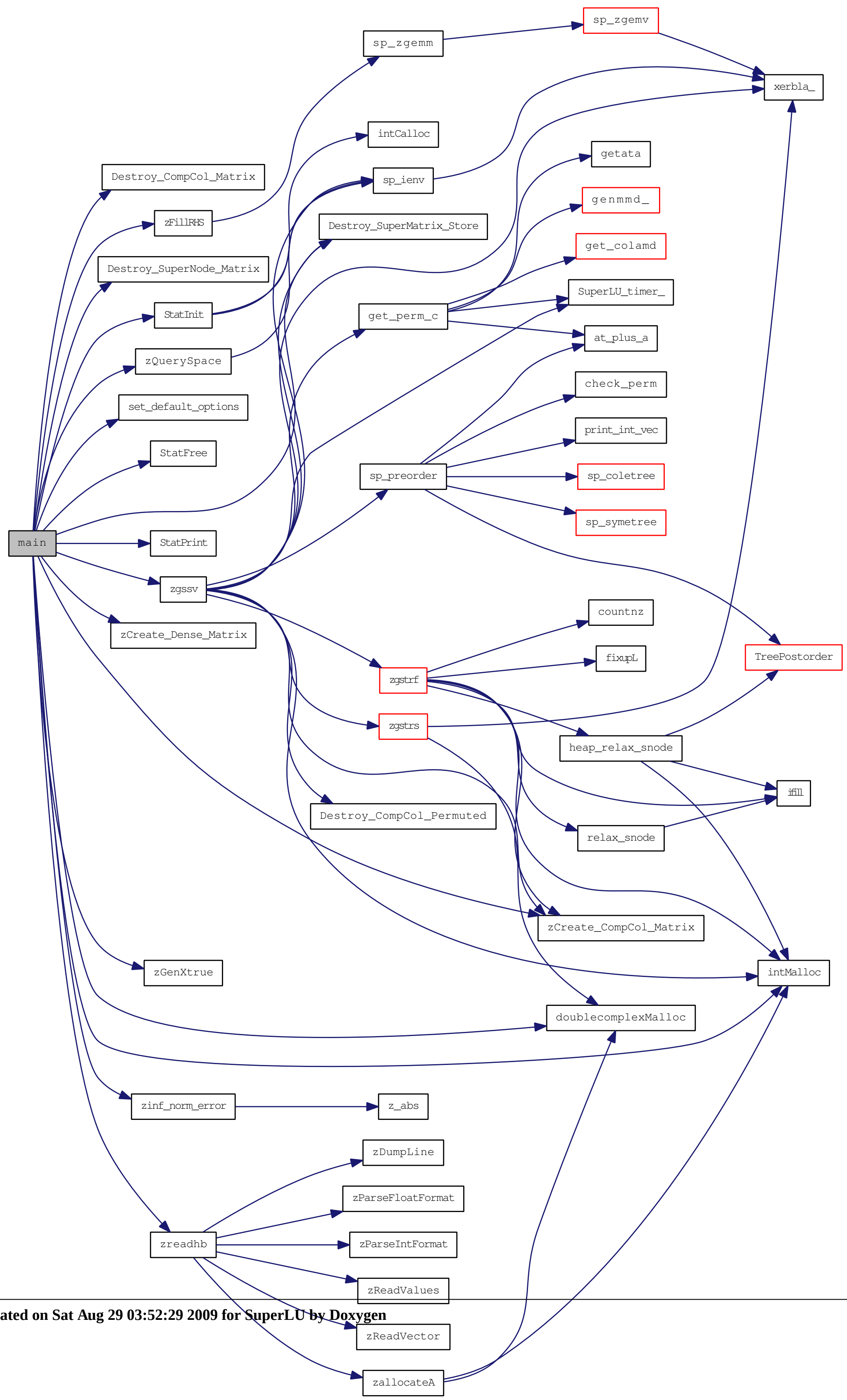
Functions

- main (int argc, char ∗argv[ ])

4.18.1 Function Documentation

4.18.1.1 main (int *argc*, char ∗ *argv*[ ])

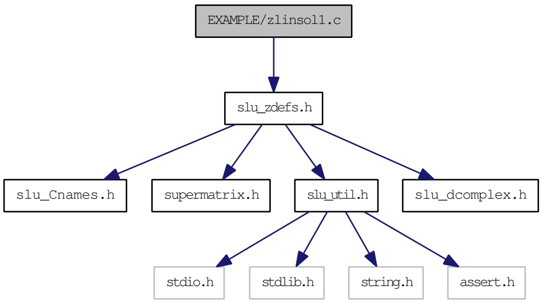
Here is the call graph for this function:



4.19 EXAMPLE/zlinsol1.c File Reference

```
#include "slu_zdefs.h"
```

Include dependency graph for zlinsol1.c:



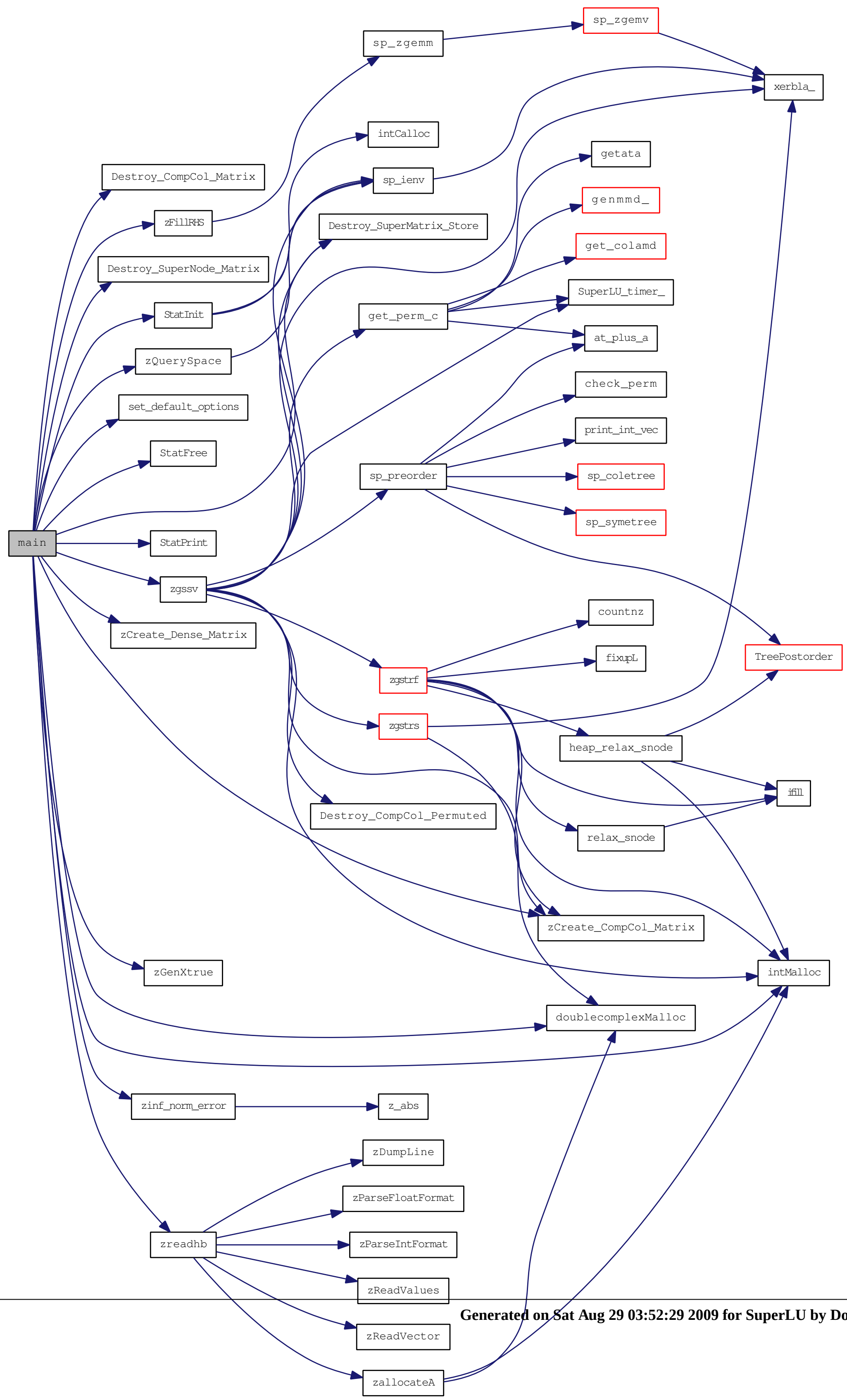
Functions

- main (int argc, char ∗argv[ ])

4.19.1 Function Documentation

4.19.1.1 main (int *argc*, char ∗ *argv*[ ])

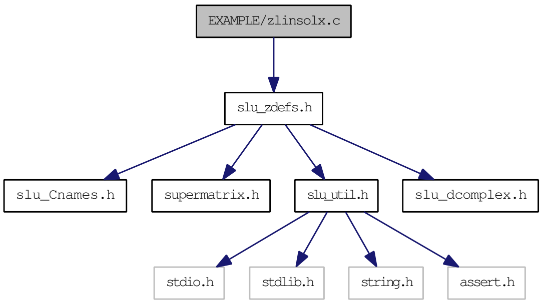
Here is the call graph for this function:



4.20 EXAMPLE/zlinsolx.c File Reference

```
#include "slu_zdefs.h"
```

Include dependency graph for zlinsolx.c:



Functions

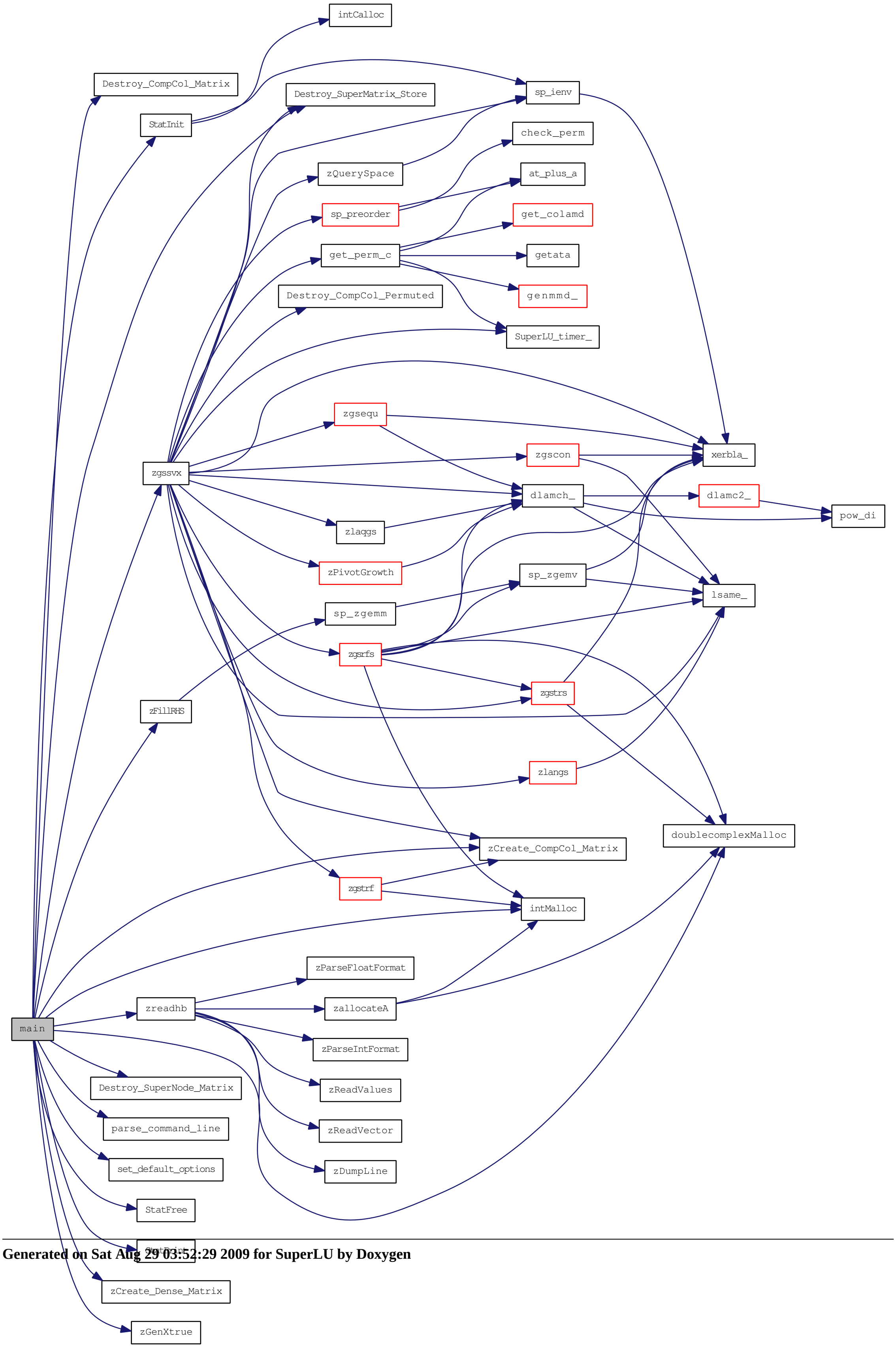
- main (int argc, char ∗argv[ ])

- void parse_command_line (int argc, char ∗argv[ ], int ∗lwork, double ∗u, yes_no_t ∗equil, trans_t ∗trans)

4.20.1 Function Documentation

4.20.1.1 main (int *argc*, char ∗ *argv*[ ])

Here is the call graph for this function:

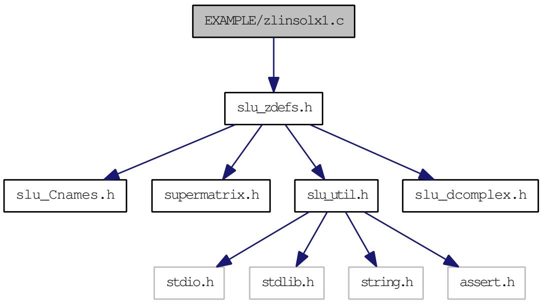


4.20.1.2 void parse_command_line (int *argc*, char ∗ *argv*[ ], int ∗ *lwork*, double ∗ *u*, yes_no_t ∗ *equil*, trans_t ∗ *trans*)

4.21 EXAMPLE/zlinsolx1.c File Reference

`#include "slu_zdefs.h"`

Include dependency graph for zlinsolx1.c:



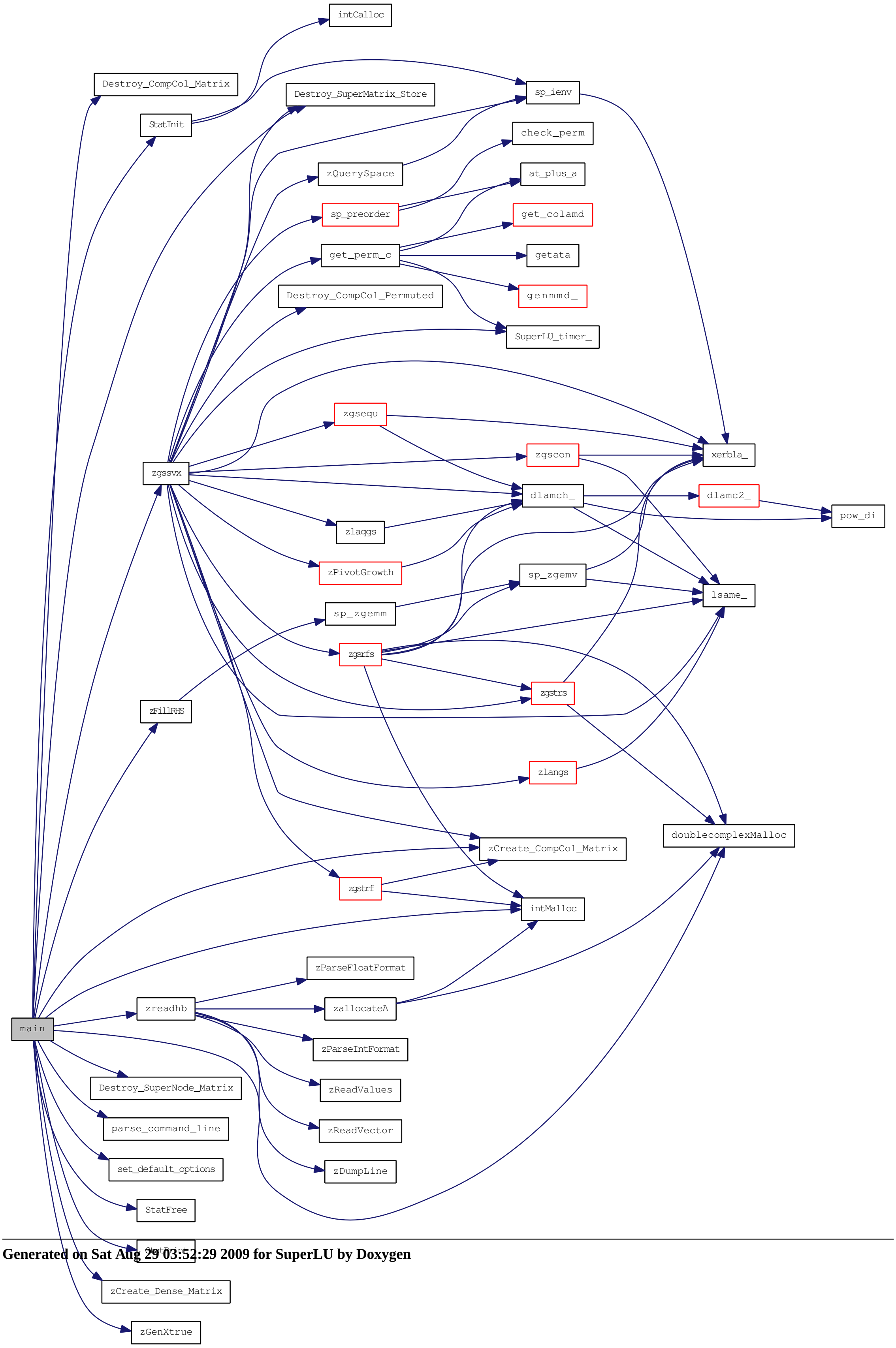
Functions

- main (int argc, char ∗argv[ ])
- void parse_command_line (int argc, char ∗argv[ ], int ∗lwork, double ∗u, yes_no_t ∗equil, trans_t ∗trans)

4.21.1 Function Documentation

4.21.1.1 main (int *argc*, char ∗ *argv*[ ])

Here is the call graph for this function:

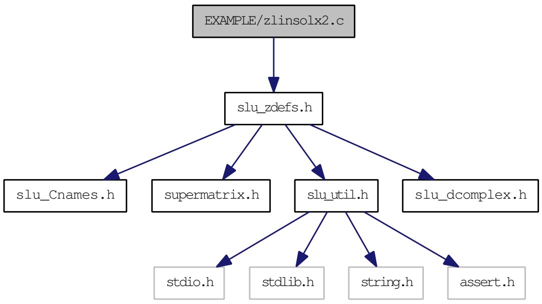


4.21.1.2 void parse_command_line (int *argc*, char * *argv*[ ], int * *lwork*, double * *u*, yes_no_t * *equil*, trans_t * *trans*)

4.22 EXAMPLE/zlinsolx2.c File Reference

```
#include "slu_zdefs.h"
```

Include dependency graph for zlinsolx2.c:



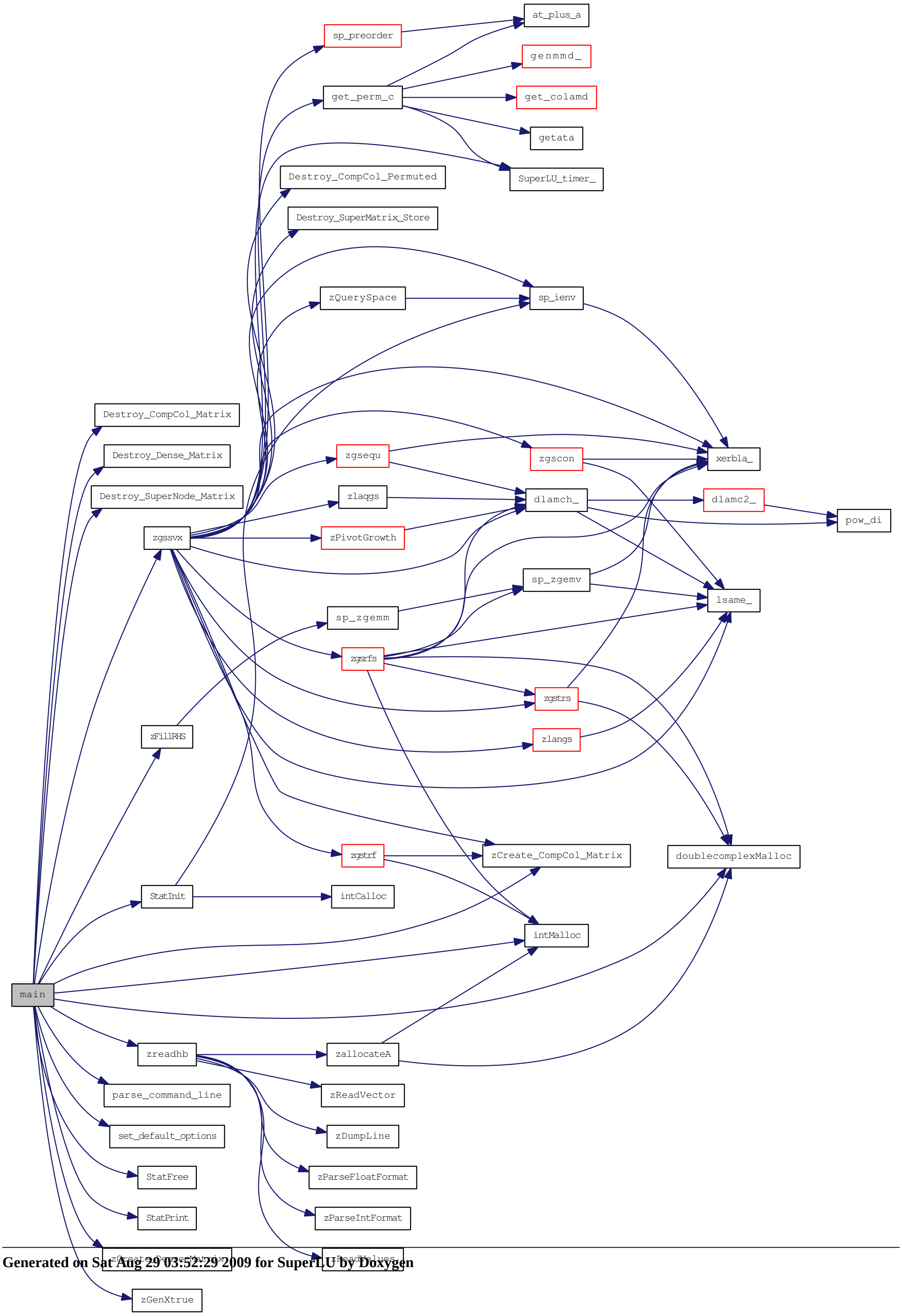
Functions

- main (int argc, char ∗argv[ ])
- void parse_command_line (int argc, char ∗argv[ ], int ∗lwork, double ∗u, yes_no_t ∗equil, trans_t ∗trans)

4.22.1 Function Documentation

4.22.1.1 main (int *argc*, char ∗ *argv*[ ])

Here is the call graph for this function:

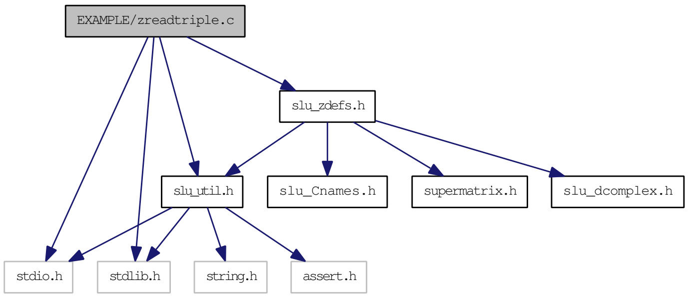


4.22.1.2 void parse_command_line (int *argc*, char ∗ *argv*[ ], int ∗ *lwork*, double ∗ *u*, yes_no_t ∗ *equil*, trans_t ∗ *trans*)

4.23 EXAMPLE/zreadtriple.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "slu_zdefs.h"
#include "slu_util.h"
```

Include dependency graph for zreadtriple.c:



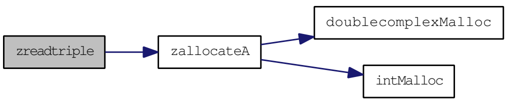
Functions

- void zreadtriple (int ∗m, int ∗n, int ∗nonz, doublecomplex ∗∗nzval, int ∗∗rowind, int ∗∗colptr)

4.23.1 Function Documentation

4.23.1.1 void zreadtriple (int ∗ *m*, int ∗ *n*, int ∗ *nonz*, doublecomplex ∗∗ *nzval*, int ∗∗ *rowind*, int ∗∗ *colptr*)

Here is the call graph for this function:



4.24 SRC/ccolumn_bmod.c File Reference

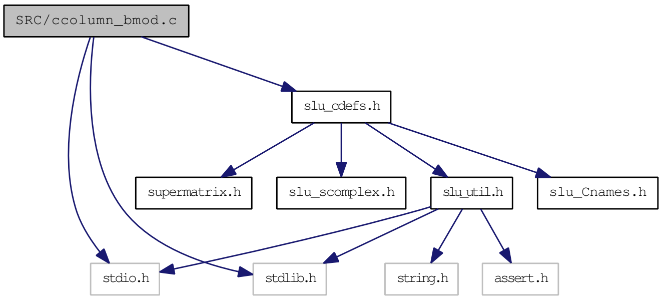
performs numeric block updates

`#include <stdio.h>`

`#include <stdlib.h>`

`#include "slu_cdefs.h"`

Include dependency graph for ccolumn_bmod.c:



Functions

- void cusolve (int, int, complex ∗, complex ∗)

  Solves a dense upper triangular system.

- void clsolve (int, int, complex ∗, complex ∗)

  Solves a dense UNIT lower triangular system.

- void cmatvec (int, int, int, complex ∗, complex ∗, complex ∗)

  Performs a dense matrix-vector multiply: Mxvec = Mxvec + M ∗ vec.

- int ccolumn_bmod (const int jcol, const int nseg, complex ∗dense, complex ∗tempv, int ∗segrep, int ∗repfnz, int fpanelc, GlobalLU_t ∗Glu, SuperLUStat_t ∗stat)

4.24.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

 Permission is hereby granted to use or copy this program for any
 purpose, provided the above notices are retained on all copies.
 Permission to modify the code and to distribute modified code is
```

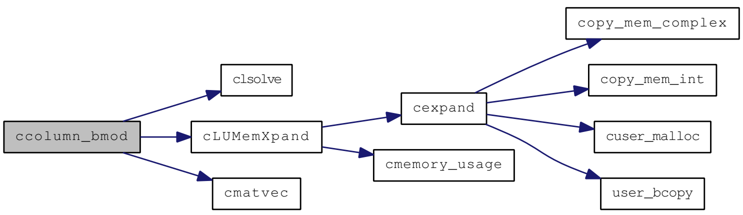
```
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.24.2 Function Documentation

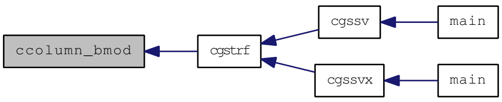
4.24.2.1 int ccolumn_bmod (const int *jcol*, const int *nseg*, complex ∗ *dense*, complex ∗ *tempv*, int ∗ *segrep*, int ∗ *repfnz*, int *fpanelc*, GlobalLU_t ∗ *Glu*, SuperLUStat_t ∗ *stat*)

```
Purpose:
========
Performs numeric block updates (sup-col) in topological order.
It features: col-col, 2cols-col, 3cols-col, and sup-col updates.
Special processing on the supernodal portion of L[*,j]
Return value:   0 - successful return
              > 0 - number of bytes allocated when run out of space
```

Here is the call graph for this function:



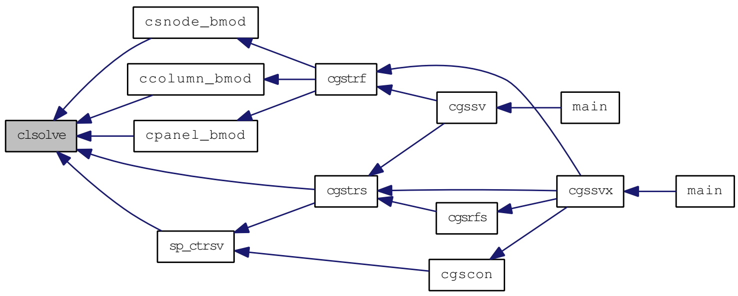
Here is the caller graph for this function:



4.24.2.2 void clsolve (int *ldm*, int *ncol*, complex ∗ *M*, complex ∗ *rhs*)

The unit lower triangular matrix is stored in a 2D array M(1:nrow,1:ncol). The solution will be returned in the rhs vector.

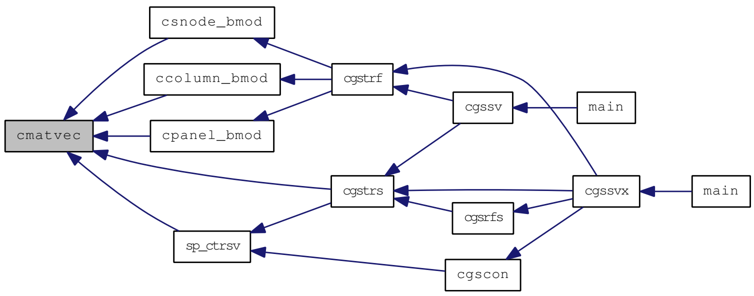
Here is the caller graph for this function:



4.24.2.3 void cmatvec (int *ldm*, int *nrow*, int *ncol*, complex ∗ *M*, complex ∗ *vec*, complex ∗ *Mxvec*)

The input matrix is M(1:nrow,1:ncol); The product is returned in Mxvec[].

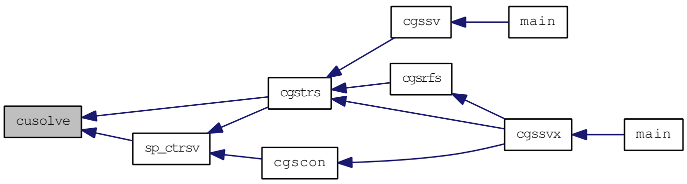
Here is the caller graph for this function:



4.24.2.4 void cusolve (int *ldm*, int *ncol*, complex ∗ *M*, complex ∗ *rhs*)

The upper triangular matrix is stored in a 2-dim array M(1:ldm,1:ncol). The solution will be returned in the rhs vector.

Here is the caller graph for this function:

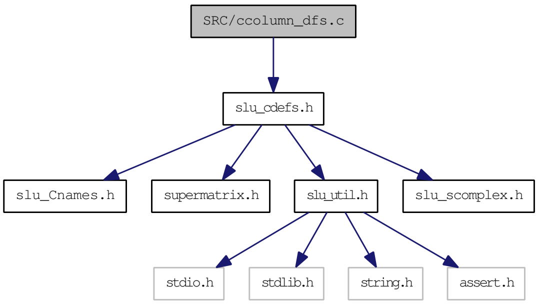


4.25 SRC/ccolumn_dfs.c File Reference

Performs a symbolic factorization.

`#include "slu_cdefs.h"`

Include dependency graph for ccolumn_dfs.c:



Defines

- #define T2_SUPER

  What type of supernodes we want.

Functions

- int ccolumn_dfs (const int m, const int jcol, int ∗perm_r, int ∗nseg, int ∗lsub_col, int ∗segrep, int ∗repfnz, int ∗xprune, int ∗marker, int ∗parent, int ∗xplore, GlobalLU_t ∗Glu)

4.25.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003


Copyright (c) 1994 by Xerox Corporation.  All rights reserved.


THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.


Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.25.2 Define Documentation

4.25.2.1 #define T2_SUPER

4.25.3 Function Documentation

4.25.3.1 int ccolumn_dfs (const int *m*, const int *jcol*, int ∗ *perm_r*, int ∗ *nseg*, int ∗ *lsub_col*, int ∗ *segrep*, int ∗ *repfnz*, int ∗ *xprune*, int ∗ *marker*, int ∗ *parent*, int ∗ *xplore*, GlobalLU_t ∗ *Glu*)

```
 Purpose
 =======
   "column_dfs" performs a symbolic factorization on column jcol, and
   decide the supernode boundary.


   This routine does not use numeric values, but only use the RHS
   row indices to start the dfs.

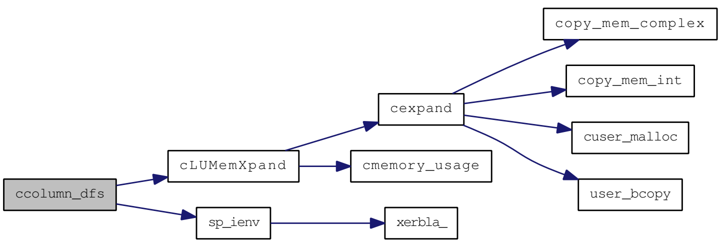

   A supernode representative is the last column of a supernode.
   The nonzeros in U[*,j] are segments that end at supernodal
   representatives. The routine returns a list of such supernodal
   representatives in topological order of the dfs that generates them.
   The location of the first nonzero in each such supernodal segment
   (supernodal entry location) is also returned.


 Local parameters
 ================
   nseg: no of segments in current U[*,j]
   jsuper: jsuper=EMPTY if column j does not belong to the same
supernode as j-1. Otherwise, jsuper=nsuper.

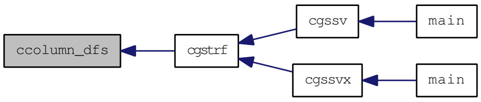

   marker2: A-row --> A-row/col (0/1)
   repfnz: SuperA-col --> PA-row
   parent: SuperA-col --> SuperA-col
   xplore: SuperA-col --> index to L-structure


 Return value
 ============
     0  success;
   > 0  number of bytes allocated when run out of space.
```

Here is the call graph for this function:



Here is the caller graph for this function:

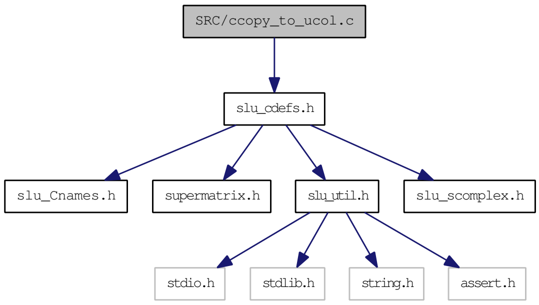


4.26 SRC/ccopy_to_ucol.c File Reference

Copy a computed column of U to the compressed data structure.

`#include "slu_cdefs.h"`

Include dependency graph for ccopy_to_ucol.c:



Functions

- int ccopy_to_ucol (int jcol, int nseg, int ∗segrep, int ∗repfnz, int ∗perm_r, complex ∗dense, GlobalLU_t ∗Glu)

4.26.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997
Copyright (c) 1994 by Xerox Corporation.  All rights reserved.



THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

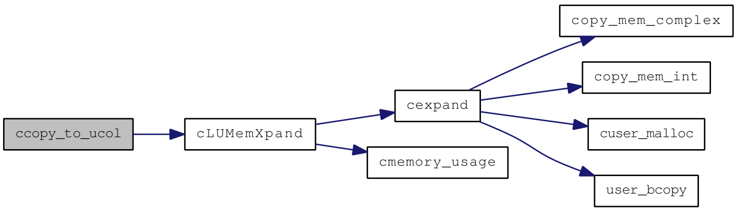


Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

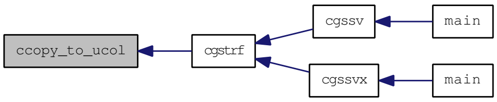
4.26.2 Function Documentation

4.26.2.1 int ccopy_to_ucol (int *jcol*, int *nseg*, int ∗ *segrep*, int ∗ *repfnz*, int ∗ *perm_r*, complex ∗ *dense*, GlobalLU_t ∗ *Glu*)

Here is the call graph for this function:



Here is the caller graph for this function:



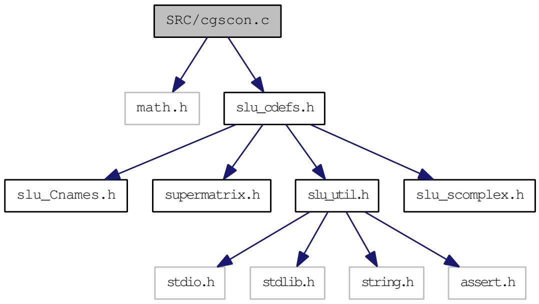
4.27 SRC/cgscon.c File Reference

Estimates reciprocal of the condition number of a general matrix.

`#include <math.h>`

`#include "slu_cdefs.h"`

Include dependency graph for cgscon.c:



Functions

- void cgscon (char *norm, SuperMatrix *L, SuperMatrix *U, float anorm, float *rcond, SuperLUStat_t *stat, int *info)

4.27.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003


Modified from lapack routines CGECON.
```

4.27.2 Function Documentation

4.27.2.1 void cgscon (char * *norm*, SuperMatrix * *L*, SuperMatrix * *U*, float *anorm*, float * *rcond*, SuperLUStat_t * *stat*, int * *info*)

```
Purpose
=======


CGSCON estimates the reciprocal of the condition number of a general
real matrix A, in either the 1-norm or the infinity-norm, using
the LU factorization computed by CGETRF.   *


An estimate is obtained for norm(inv(A)), and the reciprocal of the
condition number is computed as
   RCOND = 1 / ( norm(A) * norm(inv(A)) ).
```

```
See supermatrix.h for the definition of 'SuperMatrix' structure.



Arguments
=========



 NORM    (input) char*
         Specifies whether the 1-norm condition number or the
         infinity-norm condition number is required:
         = '1' or 'O':  1-norm;
         = 'I':         Infinity-norm.



 L       (input) SuperMatrix*
         The factor L from the factorization Pr*A*Pc=L*U as computed by
         cgstrf(). Use compressed row subscripts storage for supernodes,
         i.e., L has types: Stype = SLU_SC, Dtype = SLU_C, Mtype = SLU_TRLU.



 U       (input) SuperMatrix*
         The factor U from the factorization Pr*A*Pc=L*U as computed by
         cgstrf(). Use column-wise storage scheme, i.e., U has types:
         Stype = SLU_NC, Dtype = SLU_C, Mtype = SLU_TRU.



 ANORM   (input) float
         If NORM = '1' or 'O', the 1-norm of the original matrix A.
         If NORM = 'I', the infinity-norm of the original matrix A.



 RCOND   (output) float*
        The reciprocal of the condition number of the matrix A,
        computed as RCOND = 1/(norm(A) * norm(inv(A))).

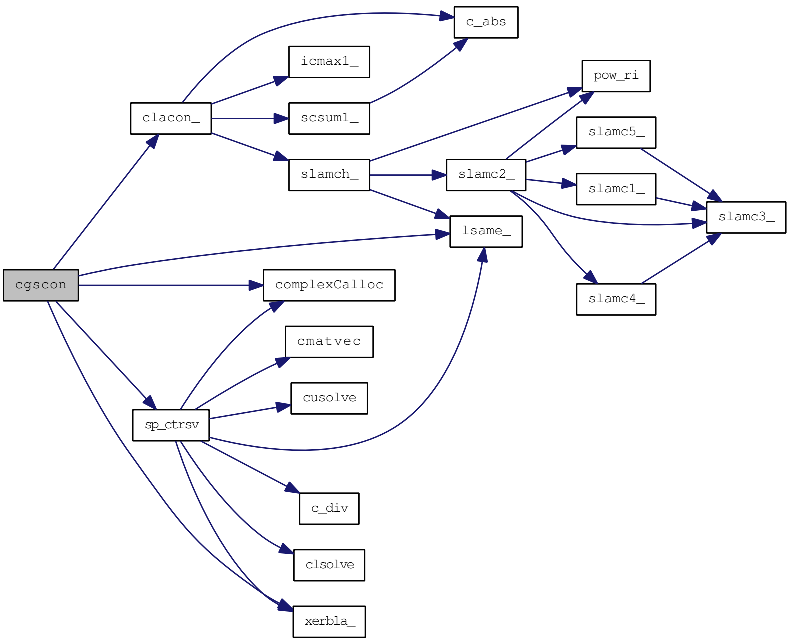


 INFO    (output) int*
        = 0:  successful exit
        < 0:  if INFO = -i, the i-th argument had an illegal value



 ====================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:

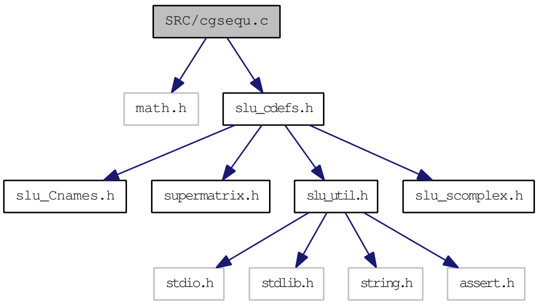
4.28 SRC/cgsequ.c File Reference

Computes row and column scalings.

`#include <math.h>`

`#include "slu_cdefs.h"`

Include dependency graph for cgsequ.c:



Functions

- void cgsequ (SuperMatrix ∗A, float ∗r, float ∗c, float ∗rowcnd, float ∗colcnd, float ∗amax, int ∗info)

  Driver related.

4.28.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997


Modified from LAPACK routine CGEEQU
```

4.28.2 Function Documentation

4.28.2.1 void cgsequ (SuperMatrix ∗ *A*, float ∗ *r*, float ∗ *c*, float ∗ *rowcnd*, float ∗ *colcnd*, float ∗ *amax*, int ∗ *info*)

```
Purpose
  =======


  CGSEQU computes row and column scalings intended to equilibrate an
  M-by-N sparse matrix A and reduce its condition number. R returns the row
  scale factors and C the column scale factors, chosen to try to make
  the largest element in each row and column of the matrix B with
  elements B(i,j)=R(i)*A(i,j)*C(j) have absolute value 1.
```

```
R(i) and C(j) are restricted to be between SMLNUM = smallest safe
number and BIGNUM = largest safe number.  Use of these scaling
factors is not guaranteed to reduce the condition number of A but
works well in practice.


See supermatrix.h for the definition of 'SuperMatrix' structure.


Arguments
=========


A       (input) SuperMatrix*
        The matrix of dimension (A->nrow, A->ncol) whose equilibration
        factors are to be computed. The type of A can be:
        Stype = SLU_NC; Dtype = SLU_C; Mtype = SLU_GE.


R       (output) float*, size A->nrow
        If INFO = 0 or INFO > M, R contains the row scale factors
        for A.


C       (output) float*, size A->ncol
        If INFO = 0,  C contains the column scale factors for A.


ROWCND  (output) float*
        If INFO = 0 or INFO > M, ROWCND contains the ratio of the
        smallest R(i) to the largest R(i).  If ROWCND >= 0.1 and
        AMAX is neither too large nor too small, it is not worth
        scaling by R.

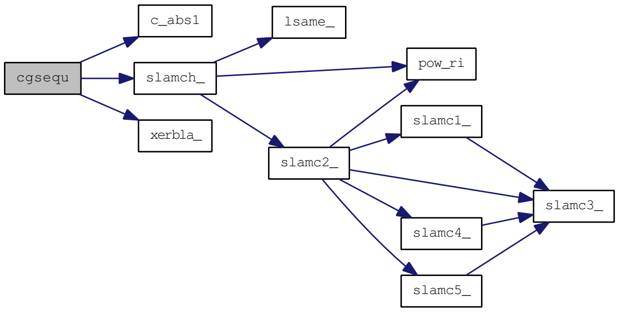

COLCND  (output) float*
        If INFO = 0, COLCND contains the ratio of the smallest
        C(i) to the largest C(i).  If COLCND >= 0.1, it is not
        worth scaling by C.


AMAX    (output) float*
        Absolute value of largest matrix element.  If AMAX is very
        close to overflow or very close to underflow, the matrix
        should be scaled.


INFO    (output) int*
        = 0:  successful exit
        < 0:  if INFO = -i, the i-th argument had an illegal value
        > 0:  if INFO = i,  and i is
              <= A->nrow:  the i-th row of A is exactly zero
              >  A->ncol:  the (i-M)-th column of A is exactly zero


=====================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:

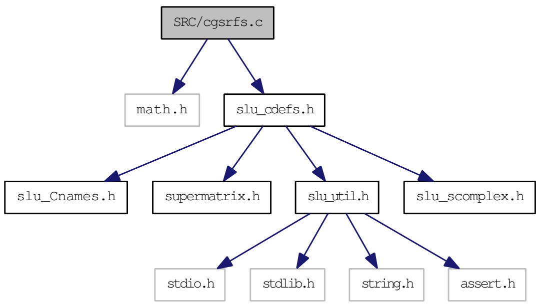
4.29 SRC/cgsrfs.c File Reference

Improves computed solution to a system of inear equations.

`#include <math.h>`

`#include "slu_cdefs.h"`

Include dependency graph for cgsrfs.c:



Defines

- #define ITMAX 5

Functions

- void cgsrfs (trans_t trans, SuperMatrix ∗A, SuperMatrix ∗L, SuperMatrix ∗U, int ∗perm_c, int ∗perm_r, char ∗equed, float ∗R, float ∗C, SuperMatrix ∗B, SuperMatrix ∗X, float ∗ferr, float ∗berr, SuperLUStat_t ∗stat, int ∗info)

4.29.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003


Modified from lapack routine CGERFS
```

4.29.2 Define Documentation

4.29.2.1 #define ITMAX 5

4.29.3 Function Documentation

4.29.3.1 void cgsrfs (trans_t *trans*, SuperMatrix ∗ *A*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, int ∗ *perm_c*, int ∗ *perm_r*, char ∗ *equed*, float ∗ *R*, float ∗ *C*, SuperMatrix ∗ *B*, SuperMatrix ∗ *X*, float ∗ *ferr*, float ∗ *berr*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
  Purpose
  =======

  CGSRFS improves the computed solution to a system of linear
  equations and provides error bounds and backward error estimates for
  the solution.

  If equilibration was performed, the system becomes:
          (diag(R)*A_original*diag(C)) * X = diag(R)*B_original.

  See supermatrix.h for the definition of 'SuperMatrix' structure.

  Arguments
  =========

 trans   (input) trans_t
          Specifies the form of the system of equations:
          = NOTRANS: A * X = B  (No transpose)
          = TRANS:   A'* X = B  (Transpose)
          = CONJ:    A**H * X = B  (Conjugate transpose)

  A       (input) SuperMatrix*
          The original matrix A in the system, or the scaled A if
          equilibration was done. The type of A can be:
          Stype = SLU_NC, Dtype = SLU_C, Mtype = SLU_GE.

  L       (input) SuperMatrix*
    The factor L from the factorization Pr*A*Pc=L*U. Use
          compressed row subscripts storage for supernodes,
          i.e., L has types: Stype = SLU_SC, Dtype = SLU_C, Mtype = SLU_TRLU.

  U       (input) SuperMatrix*
          The factor U from the factorization Pr*A*Pc=L*U as computed by
          cgstrf(). Use column-wise storage scheme,
          i.e., U has types: Stype = SLU_NC, Dtype = SLU_C, Mtype = SLU_TRU.

  perm_c  (input) int*, dimension (A->ncol)
    Column permutation vector, which defines the
          permutation matrix Pc; perm_c[i] = j means column i of A is
          in position j in A*Pc.

  perm_r  (input) int*, dimension (A->nrow)
          Row permutation vector, which defines the permutation matrix Pr;
          perm_r[i] = j means row i of A is in position j in Pr*A.
```

```
equed   (input) Specifies the form of equilibration that was done.
        = 'N': No equilibration.
        = 'R': Row equilibration, i.e., A was premultiplied by diag(R).
        = 'C': Column equilibration, i.e., A was postmultiplied by
               diag(C).
        = 'B': Both row and column equilibration, i.e., A was replaced
               by diag(R)*A*diag(C).

R       (input) float*, dimension (A->nrow)
        The row scale factors for A.
        If equed = 'R' or 'B', A is premultiplied by diag(R).
        If equed = 'N' or 'C', R is not accessed.

C       (input) float*, dimension (A->ncol)
        The column scale factors for A.
        If equed = 'C' or 'B', A is postmultiplied by diag(C).
        If equed = 'N' or 'R', C is not accessed.

B       (input) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_C, Mtype = SLU_GE.
        The right hand side matrix B.
        if equed = 'R' or 'B', B is premultiplied by diag(R).

X       (input/output) SuperMatrix*
        X has types: Stype = SLU_DN, Dtype = SLU_C, Mtype = SLU_GE.
        On entry, the solution matrix X, as computed by cgstrs().
        On exit, the improved solution matrix X.
        if *equed = 'C' or 'B', X should be premultiplied by diag(C)
            in order to obtain the solution to the original system.

FERR    (output) float*, dimension (B->ncol)
        The estimated forward error bound for each solution vector
        X(j) (the j-th column of the solution matrix X).
        If XTRUE is the true solution corresponding to X(j), FERR(j)
        is an estimated upper bound for the magnitude of the largest
        element in (X(j) - XTRUE) divided by the magnitude of the
        largest element in X(j).  The estimate is as reliable as
        the estimate for RCOND, and is almost always a slight
        overestimate of the true error.

BERR    (output) float*, dimension (B->ncol)
        The componentwise relative backward error of each solution
        vector X(j) (i.e., the smallest relative change in
        any element of A or B that makes X(j) an exact solution).

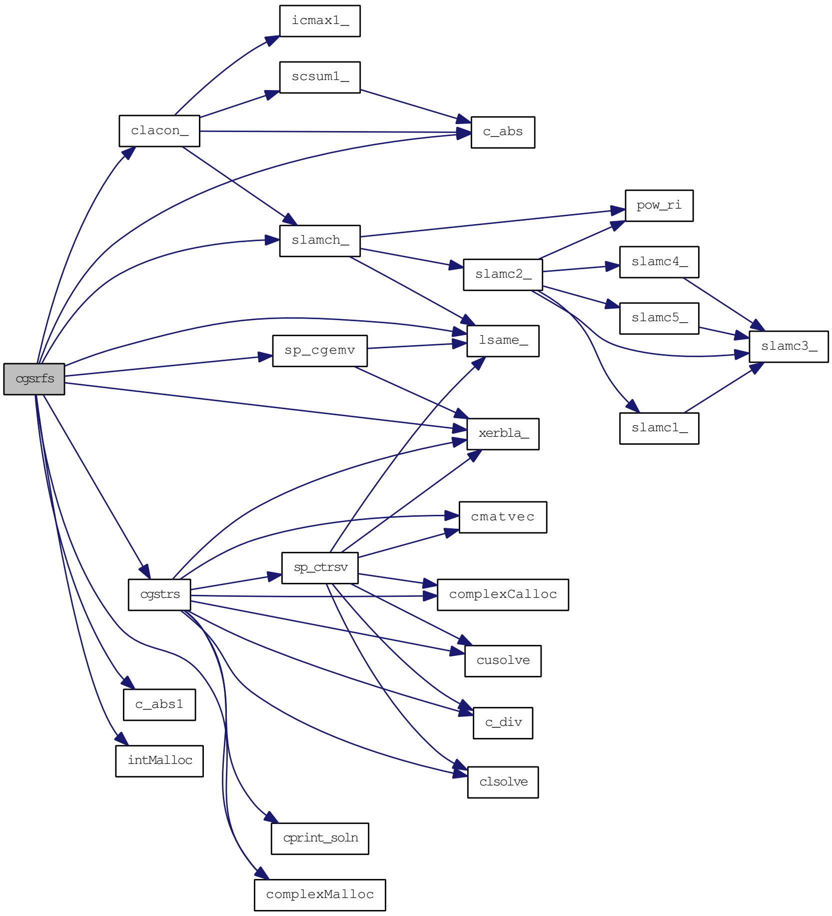
stat     (output) SuperLUStat_t*
         Record the statistics on runtime and floating-point operation count.
         See util.h for the definition of 'SuperLUStat_t'.

info    (output) int*
        = 0:  successful exit
         < 0:  if INFO = -i, the i-th argument had an illegal value

 Internal Parameters
 ===================
```

```
ITMAX is the maximum number of steps of iterative refinement.
```

Here is the call graph for this function:



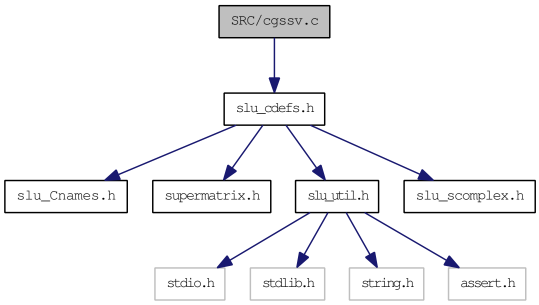
Here is the caller graph for this function:

4.30 SRC/cgssv.c File Reference

Solves the system of linear equations A∗X=B.

`#include "slu_cdefs.h"`

Include dependency graph for cgssv.c:



Functions

- void cgssv (superlu_options_t ∗options, SuperMatrix ∗A, int ∗perm_c, int ∗perm_r, SuperMatrix ∗L, SuperMatrix ∗U, SuperMatrix ∗B, SuperLUStat_t ∗stat, int ∗info)

  Driver routines.

4.30.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003
```

4.30.2 Function Documentation

4.30.2.1 void cgssv (superlu_options_t ∗ *options*, SuperMatrix ∗ *A*, int ∗ *perm_c*, int ∗ *perm_r*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, SuperMatrix ∗ *B*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======

CGSSV solves the system of linear equations A*X=B, using the
LU factorization from CGSTRF. It performs the following steps:

  1. If A is stored column-wise (A->Stype = SLU_NC):

     1.1. Permute the columns of A, forming A*Pc, where Pc
          is a permutation matrix. For more details of this step,
          see sp_preorder.c.
```

```
      1.2. Factor A as Pr*A*Pc=L*U with the permutation Pr determined
           by Gaussian elimination with partial pivoting.
           L is unit lower triangular with offdiagonal entries
           bounded by 1 in magnitude, and U is upper triangular.

      1.3. Solve the system of equations A*X=B using the factored
           form of A.

   2. If A is stored row-wise (A->Stype = SLU_NR), apply the
      above algorithm to the transpose of A:

      2.1. Permute columns of transpose(A) (rows of A),
           forming transpose(A)*Pc, where Pc is a permutation matrix.
           For more details of this step, see sp_preorder.c.

      2.2. Factor A as Pr*transpose(A)*Pc=L*U with the permutation Pr
           determined by Gaussian elimination with partial pivoting.
           L is unit lower triangular with offdiagonal entries
           bounded by 1 in magnitude, and U is upper triangular.

      2.3. Solve the system of equations A*X=B using the factored
           form of A.

   See supermatrix.h for the definition of 'SuperMatrix' structure.

 Arguments
 =========

 options (input) superlu_options_t*
         The structure defines the input parameters to control
         how the LU decomposition will be performed and how the
         system will be solved.

 A       (input) SuperMatrix*
         Matrix A in A*X=B, of dimension (A->nrow, A->ncol). The number
         of linear equations is A->nrow. Currently, the type of A can be:
         Stype = SLU_NC or SLU_NR; Dtype = SLU_C; Mtype = SLU_GE.
         In the future, more general A may be handled.

 perm_c  (input/output) int*
         If A->Stype = SLU_NC, column permutation vector of size A->ncol
         which defines the permutation matrix Pc; perm_c[i] = j means
         column i of A is in position j in A*Pc.
         If A->Stype = SLU_NR, column permutation vector of size A->nrow
         which describes permutation of columns of transpose(A)
         (rows of A) as described above.

         If options->ColPerm = MY_PERMC or options->Fact = SamePattern or
            options->Fact = SamePattern_SameRowPerm, it is an input argument.
            On exit, perm_c may be overwritten by the product of the input
            perm_c and a permutation that postorders the elimination tree
            of Pc'*A'*Pc; perm_c is not changed if the elimination tree
            is already in postorder.
         Otherwise, it is an output argument.
```

```
perm_r  (input/output) int*
        If A->Stype = SLU_NC, row permutation vector of size A->nrow,
        which defines the permutation matrix Pr, and is determined
        by partial pivoting.  perm_r[i] = j means row i of A is in
        position j in Pr*A.
        If A->Stype = SLU_NR, permutation vector of size A->ncol, which
        determines permutation of rows of transpose(A)
        (columns of A) as described above.



        If options->RowPerm = MY_PERMR or
           options->Fact = SamePattern_SameRowPerm, perm_r is an
           input argument.
        otherwise it is an output argument.



L       (output) SuperMatrix*
        The factor L from the factorization
            Pr*A*Pc=L*U              (if A->Stype = SLU_NC) or
            Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
        Uses compressed row subscripts storage for supernodes, i.e.,
        L has types: Stype = SLU_SC, Dtype = SLU_C, Mtype = SLU_TRLU.



U       (output) SuperMatrix*
  The factor U from the factorization
            Pr*A*Pc=L*U              (if A->Stype = SLU_NC) or
            Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
        Uses column-wise storage scheme, i.e., U has types:
        Stype = SLU_NC, Dtype = SLU_C, Mtype = SLU_TRU.



B       (input/output) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_C, Mtype = SLU_GE.
        On entry, the right hand side matrix.
        On exit, the solution matrix if info = 0;

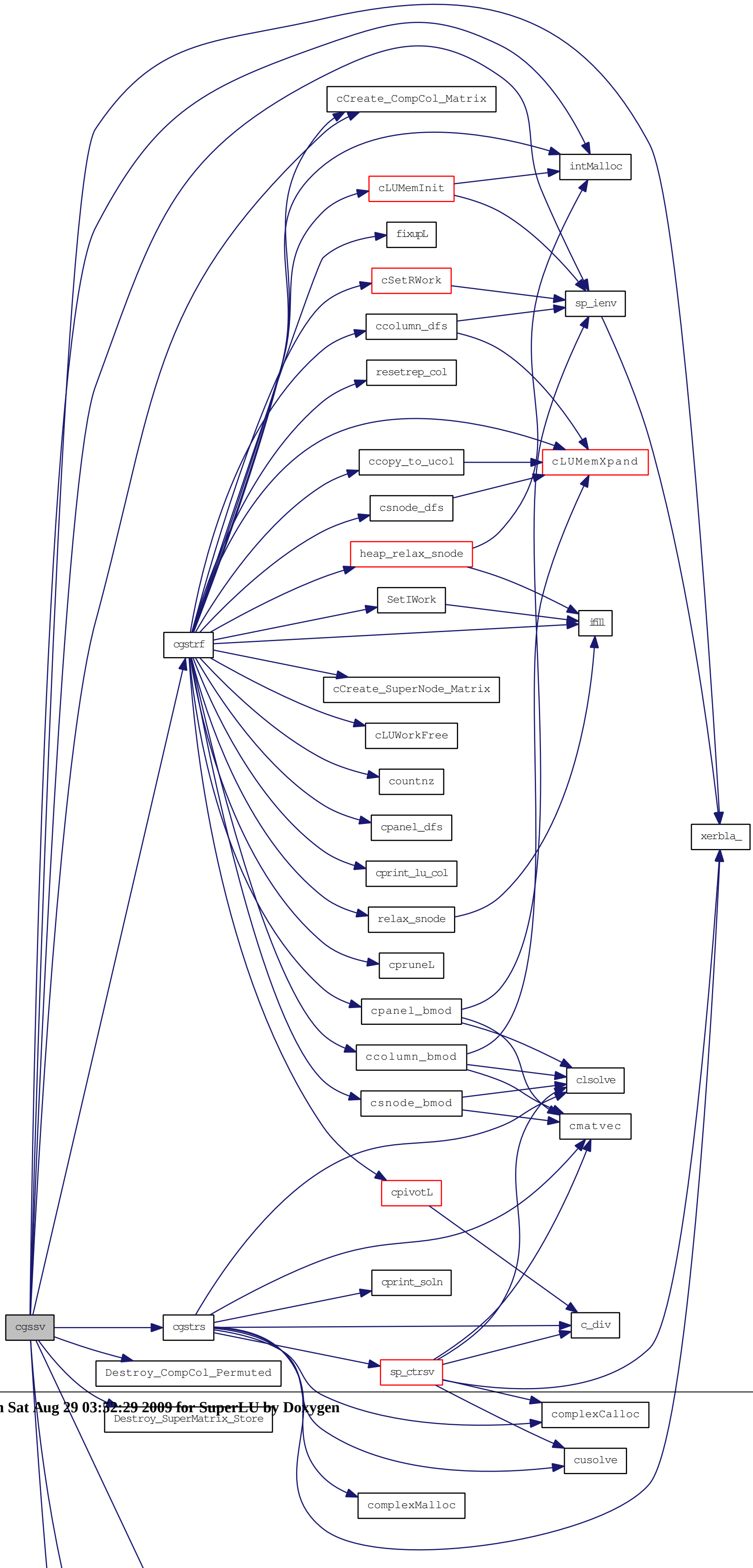


stat   (output) SuperLUStat_t*
       Record the statistics on runtime and floating-point operation count.
       See util.h for the definition of 'SuperLUStat_t'.



info    (output) int*
  = 0: successful exit
        > 0: if info = i, and i is
            <= A->ncol: U(i,i) is exactly zero. The factorization has
               been completed, but the factor U is exactly singular,
               so the solution could not be computed.
            > A->ncol: number of bytes allocated when memory allocation
               failure occurred, plus A->ncol.
```

Here is the call graph for this function:



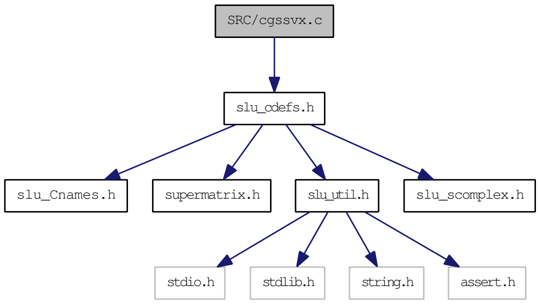
Here is the caller graph for this function:

4.31 SRC/cgssvx.c File Reference

Solves the system of linear equations A∗X=B or A'∗X=B.

`#include "slu_cdefs.h"`

Include dependency graph for cgssvx.c:



Functions

- void cgssvx (superlu_options_t ∗options, SuperMatrix ∗A, int ∗perm_c, int ∗perm_r, int ∗etree, char ∗equed, float ∗R, float ∗C, SuperMatrix ∗L, SuperMatrix ∗U, void ∗work, int lwork, SuperMatrix ∗B, SuperMatrix ∗X, float ∗recip_pivot_growth, float ∗rcond, float ∗ferr, float ∗berr, mem_usage_t ∗mem_usage, SuperLUStat_t ∗stat, int ∗info)

4.31.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003
```

4.31.2 Function Documentation

4.31.2.1 void cgssvx (superlu_options_t ∗ *options*, SuperMatrix ∗ *A*, int ∗ *perm_c*, int ∗ *perm_r*, int ∗ *etree*, char ∗ *equed*, float ∗ *R*, float ∗ *C*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, void ∗ *work*, int *lwork*, SuperMatrix ∗ *B*, SuperMatrix ∗ *X*, float ∗ *recip_pivot_growth*, float ∗ *rcond*, float ∗ *ferr*, float ∗ *berr*, mem_usage_t ∗ *mem_usage*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======


CGSSVX solves the system of linear equations A*X=B or A'*X=B, using
the LU factorization from cgstrf(). Error bounds on the solution and
a condition estimate are also provided. It performs the following steps:


  1. If A is stored column-wise (A->Stype = SLU_NC):
```

```
    1.1. If options->Equil = YES, scaling factors are computed to
         equilibrate the system:
         options->Trans = NOTRANS:
             diag(R)*A*diag(C) *inv(diag(C))*X = diag(R)*B
         options->Trans = TRANS:
             (diag(R)*A*diag(C))**T *inv(diag(R))*X = diag(C)*B
         options->Trans = CONJ:
             (diag(R)*A*diag(C))**H *inv(diag(R))*X = diag(C)*B
         Whether or not the system will be equilibrated depends on the
         scaling of the matrix A, but if equilibration is used, A is
         overwritten by diag(R)*A*diag(C) and B by diag(R)*B
         (if options->Trans=NOTRANS) or diag(C)*B (if options->Trans
         = TRANS or CONJ).

    1.2. Permute columns of A, forming A*Pc, where Pc is a permutation
         matrix that usually preserves sparsity.
         For more details of this step, see sp_preorder.c.

    1.3. If options->Fact != FACTORED, the LU decomposition is used to
         factor the matrix A (after equilibration if options->Equil = YES)
         as Pr*A*Pc = L*U, with Pr determined by partial pivoting.

    1.4. Compute the reciprocal pivot growth factor.

    1.5. If some U(i,i) = 0, so that U is exactly singular, then the
         routine returns with info = i. Otherwise, the factored form of
         A is used to estimate the condition number of the matrix A. If
         the reciprocal of the condition number is less than machine
         precision, info = A->ncol+1 is returned as a warning, but the
         routine still goes on to solve for X and computes error bounds
         as described below.

    1.6. The system of equations is solved for X using the factored form
         of A.

    1.7. If options->IterRefine != NOREFINE, iterative refinement is
         applied to improve the computed solution matrix and calculate
         error bounds and backward error estimates for it.

    1.8. If equilibration was used, the matrix X is premultiplied by
         diag(C) (if options->Trans = NOTRANS) or diag(R)
         (if options->Trans = TRANS or CONJ) so that it solves the
         original system before equilibration.

 2. If A is stored row-wise (A->Stype = SLU_NR), apply the above algorithm
    to the transpose of A:

    2.1. If options->Equil = YES, scaling factors are computed to
         equilibrate the system:
         options->Trans = NOTRANS:
             diag(R)*A*diag(C) *inv(diag(C))*X = diag(R)*B
         options->Trans = TRANS:
             (diag(R)*A*diag(C))**T *inv(diag(R))*X = diag(C)*B
         options->Trans = CONJ:
             (diag(R)*A*diag(C))**H *inv(diag(R))*X = diag(C)*B
```

```
          Whether or not the system will be equilibrated depends on the
          scaling of the matrix A, but if equilibration is used, A' is
          overwritten by diag(R)*A'*diag(C) and B by diag(R)*B
          (if trans='N') or diag(C)*B (if trans = 'T' or 'C').

     2.2. Permute columns of transpose(A) (rows of A),
          forming transpose(A)*Pc, where Pc is a permutation matrix that
          usually preserves sparsity.
          For more details of this step, see sp_preorder.c.

     2.3. If options->Fact != FACTORED, the LU decomposition is used to
          factor the transpose(A) (after equilibration if
          options->Fact = YES) as Pr*transpose(A)*Pc = L*U with the
          permutation Pr determined by partial pivoting.

     2.4. Compute the reciprocal pivot growth factor.

     2.5. If some U(i,i) = 0, so that U is exactly singular, then the
          routine returns with info = i. Otherwise, the factored form
          of transpose(A) is used to estimate the condition number of the
          matrix A. If the reciprocal of the condition number
          is less than machine precision, info = A->nrow+1 is returned as
          a warning, but the routine still goes on to solve for X and
          computes error bounds as described below.

     2.6. The system of equations is solved for X using the factored form
          of transpose(A).

     2.7. If options->IterRefine != NOREFINE, iterative refinement is
          applied to improve the computed solution matrix and calculate
          error bounds and backward error estimates for it.

     2.8. If equilibration was used, the matrix X is premultiplied by
          diag(C) (if options->Trans = NOTRANS) or diag(R)
          (if options->Trans = TRANS or CONJ) so that it solves the
          original system before equilibration.

  See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

options (input) superlu_options_t*
        The structure defines the input parameters to control
        how the LU decomposition will be performed and how the
        system will be solved.

A       (input/output) SuperMatrix*
        Matrix A in A*X=B, of dimension (A->nrow, A->ncol). The number
        of the linear equations is A->nrow. Currently, the type of A can be:
        Stype = SLU_NC or SLU_NR, Dtype = SLU_D, Mtype = SLU_GE.
        In the future, more general A may be handled.
```

```
        On entry, If options->Fact = FACTORED and equed is not 'N',
        then A must have been equilibrated by the scaling factors in
        R and/or C.
        On exit, A is not modified if options->Equil = NO, or if
        options->Equil = YES but equed = 'N' on exit.
        Otherwise, if options->Equil = YES and equed is not 'N',
        A is scaled as follows:
        If A->Stype = SLU_NC:
          equed = 'R':  A := diag(R) * A
          equed = 'C':  A := A * diag(C)
          equed = 'B':  A := diag(R) * A * diag(C).
        If A->Stype = SLU_NR:
          equed = 'R':  transpose(A) := diag(R) * transpose(A)
          equed = 'C':  transpose(A) := transpose(A) * diag(C)
          equed = 'B':  transpose(A) := diag(R) * transpose(A) * diag(C).

perm_c  (input/output) int*
  If A->Stype = SLU_NC, Column permutation vector of size A->ncol,
        which defines the permutation matrix Pc; perm_c[i] = j means
        column i of A is in position j in A*Pc.
        On exit, perm_c may be overwritten by the product of the input
        perm_c and a permutation that postorders the elimination tree
        of Pc'*A'*A*Pc; perm_c is not changed if the elimination tree
        is already in postorder.

        If A->Stype = SLU_NR, column permutation vector of size A->nrow,
        which describes permutation of columns of transpose(A)
        (rows of A) as described above.

perm_r  (input/output) int*
        If A->Stype = SLU_NC, row permutation vector of size A->nrow,
        which defines the permutation matrix Pr, and is determined
        by partial pivoting.  perm_r[i] = j means row i of A is in
        position j in Pr*A.

        If A->Stype = SLU_NR, permutation vector of size A->ncol, which
        determines permutation of rows of transpose(A)
        (columns of A) as described above.

        If options->Fact = SamePattern_SameRowPerm, the pivoting routine
        will try to use the input perm_r, unless a certain threshold
        criterion is violated. In that case, perm_r is overwritten by a
        new permutation determined by partial pivoting or diagonal
        threshold pivoting.
        Otherwise, perm_r is output argument.

etree   (input/output) int*,  dimension (A->ncol)
        Elimination tree of Pc'*A'*A*Pc.
        If options->Fact != FACTORED and options->Fact != DOFACT,
        etree is an input argument, otherwise it is an output argument.
        Note: etree is a vector of parent pointers for a forest whose
        vertices are the integers 0 to A->ncol-1; etree[root]==A->ncol.

equed   (input/output) char*
        Specifies the form of equilibration that was done.
        = 'N': No equilibration.
```

```
         = 'R': Row equilibration, i.e., A was premultiplied by diag(R).
         = 'C': Column equilibration, i.e., A was postmultiplied by diag(C).
         = 'B': Both row and column equilibration, i.e., A was replaced
                by diag(R)*A*diag(C).
         If options->Fact = FACTORED, equed is an input argument,
         otherwise it is an output argument.

R        (input/output) float*, dimension (A->nrow)
         The row scale factors for A or transpose(A).
         If equed = 'R' or 'B', A (if A->Stype = SLU_NC) or transpose(A)
             (if A->Stype = SLU_NR) is multiplied on the left by diag(R).
         If equed = 'N' or 'C', R is not accessed.
         If options->Fact = FACTORED, R is an input argument,
             otherwise, R is output.
         If options->zFact = FACTORED and equed = 'R' or 'B', each element
             of R must be positive.

C        (input/output) float*, dimension (A->ncol)
         The column scale factors for A or transpose(A).
         If equed = 'C' or 'B', A (if A->Stype = SLU_NC) or transpose(A)
             (if A->Stype = SLU_NR) is multiplied on the right by diag(C).
         If equed = 'N' or 'R', C is not accessed.
         If options->Fact = FACTORED, C is an input argument,
             otherwise, C is output.
         If options->Fact = FACTORED and equed = 'C' or 'B', each element
             of C must be positive.

L        (output) SuperMatrix*
   The factor L from the factorization
             Pr*A*Pc=L*U              (if A->Stype SLU_= NC) or
             Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
         Uses compressed row subscripts storage for supernodes, i.e.,
         L has types: Stype = SLU_SC, Dtype = SLU_C, Mtype = SLU_TRLU.

U        (output) SuperMatrix*
   The factor U from the factorization
             Pr*A*Pc=L*U              (if A->Stype = SLU_NC) or
             Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
         Uses column-wise storage scheme, i.e., U has types:
         Stype = SLU_NC, Dtype = SLU_C, Mtype = SLU_TRU.

work     (workspace/output) void*, size (lwork) (in bytes)
         User supplied workspace, should be large enough
         to hold data structures for factors L and U.
         On exit, if fact is not 'F', L and U point to this array.

lwork    (input) int
         Specifies the size of work array in bytes.
         = 0:  allocate space internally by system malloc;
         > 0:  use user-supplied work array of length lwork in bytes,
               returns error if space runs out.
         = -1: the routine guesses the amount of space needed without
               performing the factorization, and returns it in
               mem_usage->total_needed; no other side effects.

         See argument 'mem_usage' for memory usage statistics.
```

```
B       (input/output) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_C, Mtype = SLU_GE.
        On entry, the right hand side matrix.
        If B->ncol = 0, only LU decomposition is performed, the triangular
                        solve is skipped.
        On exit,
           if equed = 'N', B is not modified; otherwise
           if A->Stype = SLU_NC:
              if options->Trans = NOTRANS and equed = 'R' or 'B',
                 B is overwritten by diag(R)*B;
              if options->Trans = TRANS or CONJ and equed = 'C' of 'B',
                 B is overwritten by diag(C)*B;
           if A->Stype = SLU_NR:
              if options->Trans = NOTRANS and equed = 'C' or 'B',
                 B is overwritten by diag(C)*B;
              if options->Trans = TRANS or CONJ and equed = 'R' of 'B',
                 B is overwritten by diag(R)*B.

X       (output) SuperMatrix*
        X has types: Stype = SLU_DN, Dtype = SLU_C, Mtype = SLU_GE.
        If info = 0 or info = A->ncol+1, X contains the solution matrix
        to the original system of equations. Note that A and B are modified
        on exit if equed is not 'N', and the solution to the equilibrated
        system is inv(diag(C))*X if options->Trans = NOTRANS and
        equed = 'C' or 'B', or inv(diag(R))*X if options->Trans = 'T' or 'C'
        and equed = 'R' or 'B'.

recip_pivot_growth (output) float*
        The reciprocal pivot growth factor max_j( norm(A_j)/norm(U_j) ).
        The infinity norm is used. If recip_pivot_growth is much less
        than 1, the stability of the LU factorization could be poor.

rcond   (output) float*
        The estimate of the reciprocal condition number of the matrix A
        after equilibration (if done). If rcond is less than the machine
        precision (in particular, if rcond = 0), the matrix is singular
        to working precision. This condition is indicated by a return
        code of info > 0.

FERR    (output) float*, dimension (B->ncol)
        The estimated forward error bound for each solution vector
        X(j) (the j-th column of the solution matrix X).
        If XTRUE is the true solution corresponding to X(j), FERR(j)
        is an estimated upper bound for the magnitude of the largest
        element in (X(j) - XTRUE) divided by the magnitude of the
        largest element in X(j).  The estimate is as reliable as
        the estimate for RCOND, and is almost always a slight
        overestimate of the true error.
        If options->IterRefine = NOREFINE, ferr = 1.0.

BERR    (output) float*, dimension (B->ncol)
        The componentwise relative backward error of each solution
        vector X(j) (i.e., the smallest relative change in
        any element of A or B that makes X(j) an exact solution).
        If options->IterRefine = NOREFINE, berr = 1.0.

mem_usage (output) mem_usage_t*
```

```
Record the memory usage statistics, consisting of following fields:
```

- for_lu (float)

```
The amount of space used in bytes for L data structures.
```

- total_needed (float)

```
The amount of space needed in bytes to perform factorization.
```

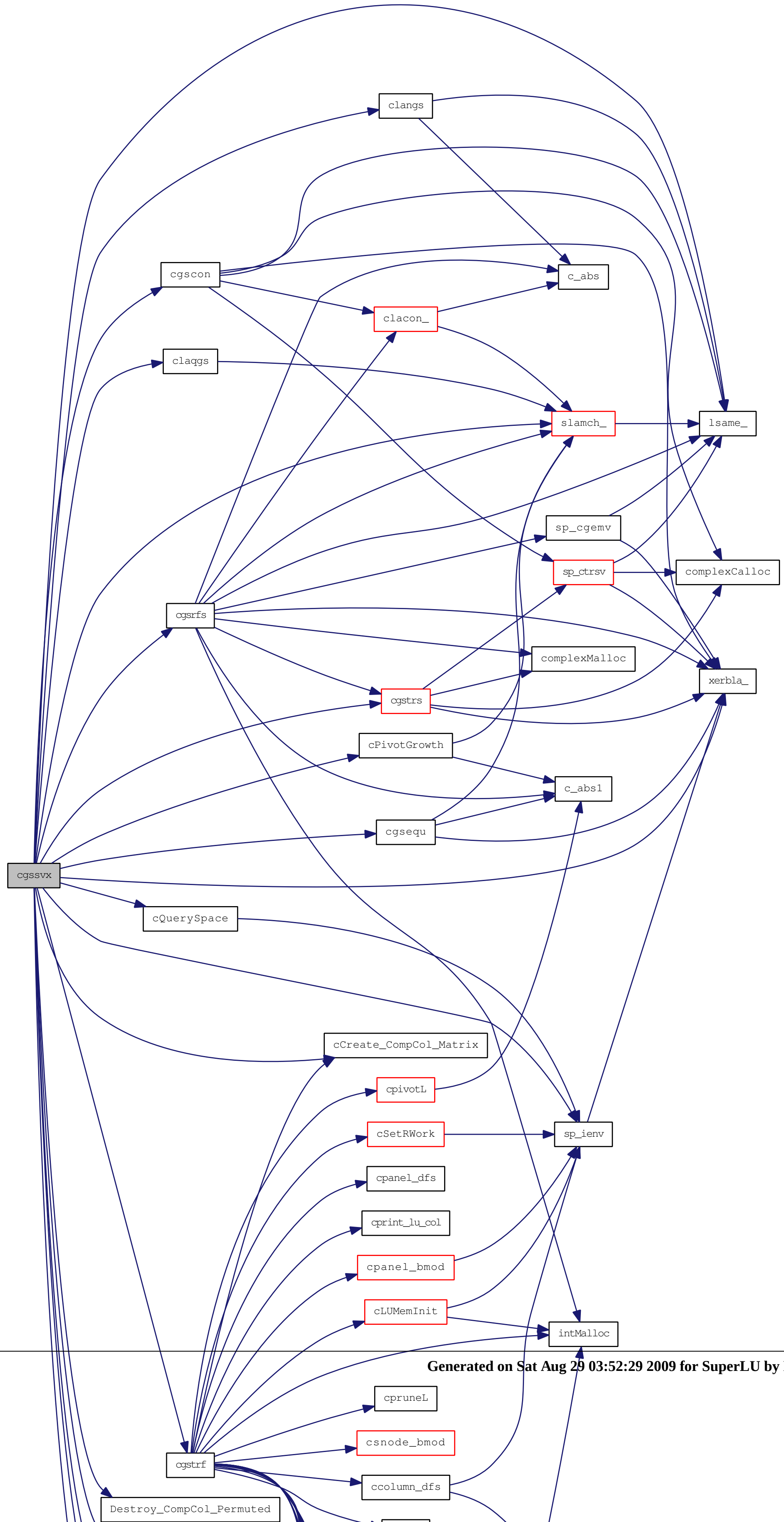
- expansions (int)

```
The number of memory expansions during the LU factorization.
```

```
stat   (output) SuperLUStat_t*
       Record the statistics on runtime and floating-point operation count.
       See util.h for the definition of 'SuperLUStat_t'.
```

```
info    (output) int*
        = 0: successful exit
        < 0: if info = -i, the i-th argument had an illegal value
        > 0: if info = i, and i is
             <= A->ncol: U(i,i) is exactly zero. The factorization has
                   been completed, but the factor U is exactly
                   singular, so the solution and error bounds
                   could not be computed.
             = A->ncol+1: U is nonsingular, but RCOND is less than machine
                   precision, meaning that the matrix is singular to
                   working precision. Nevertheless, the solution and
                   error bounds are computed because there are a number
                   of situations where the computed solution can be more
                   accurate than the value of RCOND would suggest.
             > A->ncol+1: number of bytes allocated when memory allocation
                   failure occurred, plus A->ncol.
```

Here is the call graph for this function:



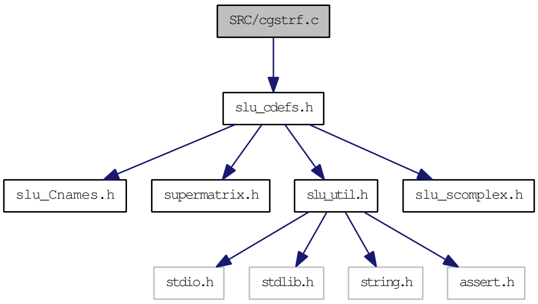
Here is the caller graph for this function:

4.32 SRC/cgstrf.c File Reference

Computes an LU factorization of a general sparse matrix.

`#include "slu_cdefs.h"`

Include dependency graph for cgstrf.c:



Functions

- void cgstrf (superlu_options_t ∗options, SuperMatrix ∗A, float drop_tol, int relax, int panel_size, int ∗etree, void ∗work, int lwork, int ∗perm_c, int ∗perm_r, SuperMatrix ∗L, SuperMatrix ∗U, SuperLUStat_t ∗stat, int ∗info)

4.32.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.32.2 Function Documentation

4.32.2.1 void cgstrf (superlu_options_t ∗ *options*, SuperMatrix ∗ *A*, float *drop_tol*, int *relax*, int *panel_size*, int ∗ *etree*, void ∗ *work*, int *lwork*, int ∗ *perm_c*, int ∗ *perm_r*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======
```

```
CGSTRF computes an LU factorization of a general sparse m-by-n
matrix A using partial pivoting with row interchanges.
The factorization has the form
    Pr * A = L * U
where Pr is a row permutation matrix, L is lower triangular with unit
diagonal elements (lower trapezoidal if A->nrow > A->ncol), and U is upper
triangular (upper trapezoidal if A->nrow < A->ncol).

See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

options (input) superlu_options_t*
        The structure defines the input parameters to control
        how the LU decomposition will be performed.

A        (input) SuperMatrix*
   Original matrix A, permuted by columns, of dimension
         (A->nrow, A->ncol). The type of A can be:
         Stype = SLU_NCP; Dtype = SLU_C; Mtype = SLU_GE.

drop_tol (input) float (NOT IMPLEMENTED)
   Drop tolerance parameter. At step j of the Gaussian elimination,
         if abs(A_ij)/(max_i abs(A_ij)) < drop_tol, drop entry A_ij.
         0 <= drop_tol <= 1. The default value of drop_tol is 0.

relax    (input) int
         To control degree of relaxing supernodes. If the number
         of nodes (columns) in a subtree of the elimination tree is less
         than relax, this subtree is considered as one supernode,
         regardless of the row structures of those columns.

panel_size (input) int
         A panel consists of at most panel_size consecutive columns.

etree    (input) int*, dimension (A->ncol)
         Elimination tree of A'*A.
         Note: etree is a vector of parent pointers for a forest whose
         vertices are the integers 0 to A->ncol-1; etree[root]==A->ncol.
         On input, the columns of A should be permuted so that the
         etree is in a certain postorder.

work     (input/output) void*, size (lwork) (in bytes)
         User-supplied work space and space for the output data structures.
         Not referenced if lwork = 0;

lwork   (input) int
        Specifies the size of work array in bytes.
        = 0:  allocate space internally by system malloc;
        > 0:  use user-supplied work array of length lwork in bytes,
              returns error if space runs out.
        = -1: the routine guesses the amount of space needed without
              performing the factorization, and returns it in
              *info; no other side effects.
```

```
perm_c   (input) int*, dimension (A->ncol)
   Column permutation vector, which defines the
         permutation matrix Pc; perm_c[i] = j means column i of A is
         in position j in A*Pc.
         When searching for diagonal, perm_c[*] is applied to the
         row subscripts of A, so that diagonal threshold pivoting
         can find the diagonal of A, rather than that of A*Pc.


perm_r   (input/output) int*, dimension (A->nrow)
         Row permutation vector which defines the permutation matrix Pr,
         perm_r[i] = j means row i of A is in position j in Pr*A.
         If options->Fact = SamePattern_SameRowPerm, the pivoting routine
            will try to use the input perm_r, unless a certain threshold
            criterion is violated. In that case, perm_r is overwritten by
            a new permutation determined by partial pivoting or diagonal
            threshold pivoting.
         Otherwise, perm_r is output argument;


L        (output) SuperMatrix*
         The factor L from the factorization Pr*A=L*U; use compressed row
         subscripts storage for supernodes, i.e., L has type:
         Stype = SLU_SC, Dtype = SLU_C, Mtype = SLU_TRLU.


U        (output) SuperMatrix*
   The factor U from the factorization Pr*A*Pc=L*U. Use column-wise
         storage scheme, i.e., U has types: Stype = SLU_NC,
         Dtype = SLU_C, Mtype = SLU_TRU.


stat     (output) SuperLUStat_t*
         Record the statistics on runtime and floating-point operation count.
         See util.h for the definition of 'SuperLUStat_t'.


info     (output) int*
         = 0: successful exit
         < 0: if info = -i, the i-th argument had an illegal value
         > 0: if info = i, and i is
            <= A->ncol: U(i,i) is exactly zero. The factorization has
               been completed, but the factor U is exactly singular,
               and division by zero will occur if it is used to solve a
               system of equations.
            > A->ncol: number of bytes allocated when memory allocation
               failure occurred, plus A->ncol. If lwork = -1, it is
               the estimated amount of space needed, plus A->ncol.

======================================================================

Local Working Arrays:
======================
  m = number of rows in the matrix
  n = number of columns in the matrix


  xprune[0:n-1]: xprune[*] points to locations in subscript
vector lsub[*]. For column i, xprune[i] denotes the point where
structural pruning begins. I.e. only xlsub[i],..,xprune[i]-1 need
to be traversed for symbolic factorization.
```

```
   marker[0:3*m-1]: marker[i] = j means that node i has been
reached when working on column j.
Storage: relative to original row subscripts
NOTE: There are 3 of them: marker/marker1 are used for panel dfs,
      see cpanel_dfs.c; marker2 is used for inner-factorization,
            see ccolumn_dfs.c.
```

```
   parent[0:m-1]: parent vector used during dfs
      Storage: relative to new row subscripts
```

```
   xplore[0:m-1]: xplore[i] gives the location of the next (dfs)
unexplored neighbor of i in lsub[*]
```

```
   segrep[0:nseg-1]: contains the list of supernodal representatives
in topological order of the dfs. A supernode representative is the
last column of a supernode.
      The maximum size of segrep[] is n.
```

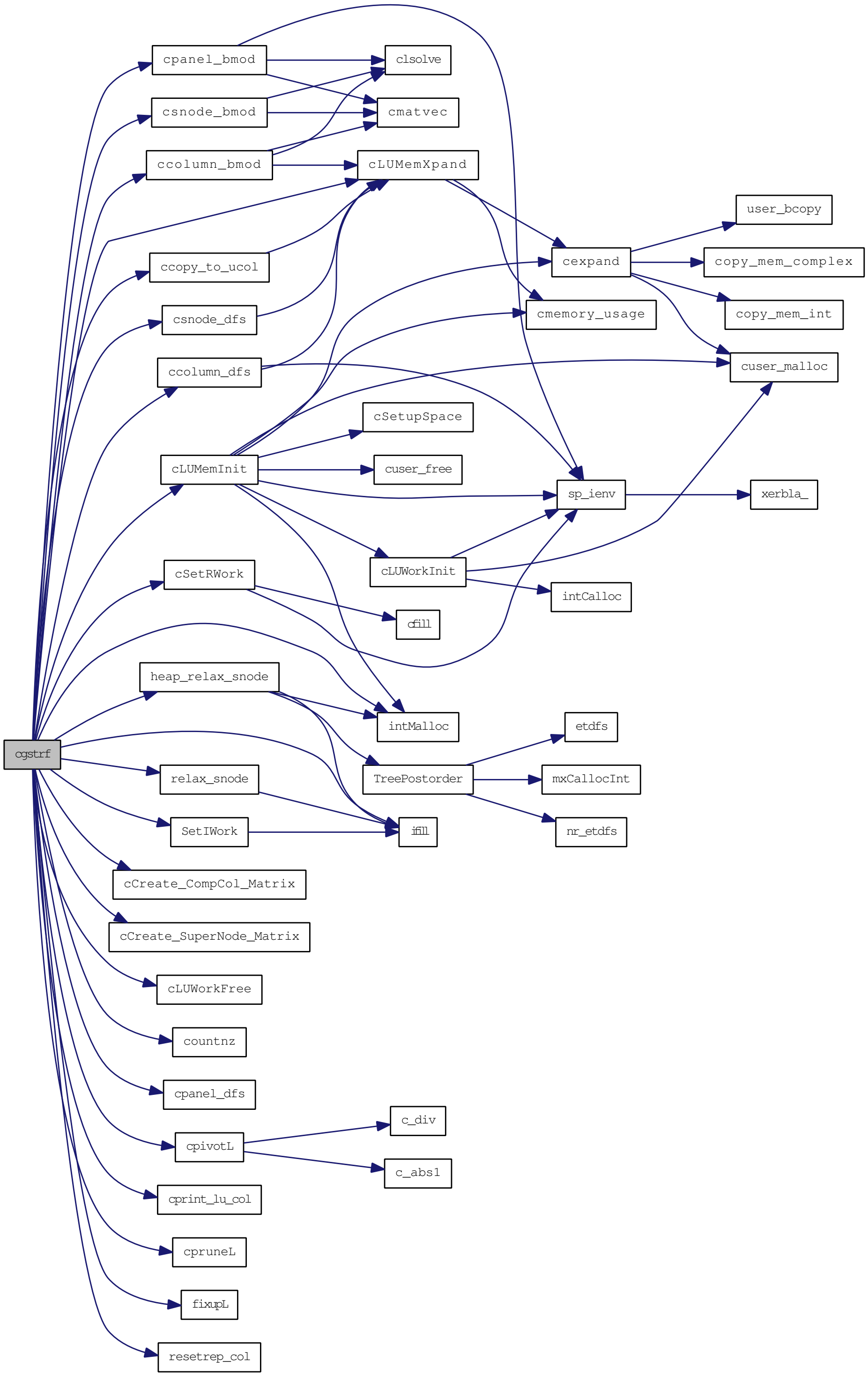
```
   repfnz[0:W*m-1]: for a nonzero segment U[*,j] that ends at a
supernodal representative r, repfnz[r] is the location of the first
nonzero in this segment.  It is also used during the dfs: repfnz[r]>0
indicates the supernode r has been explored.
NOTE: There are W of them, each used for one column of a panel.
```

```
   panel_lsub[0:W*m-1]: temporary for the nonzeros row indices below
      the panel diagonal. These are filled in during cpanel_dfs(), and are
      used later in the inner LU factorization within the panel.
panel_lsub[]/dense[] pair forms the SPA data structure.
NOTE: There are W of them.
```

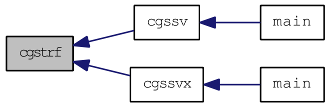
```
   dense[0:W*m-1]: sparse accumulating (SPA) vector for intermediate values;
        NOTE: there are W of them.
```

```
   tempv[0:*]: real temporary used for dense numeric kernels;
The size of this array is defined by NUM_TEMPV() in slu_cdefs.h.
```

Here is the call graph for this function:



Here is the caller graph for this function:

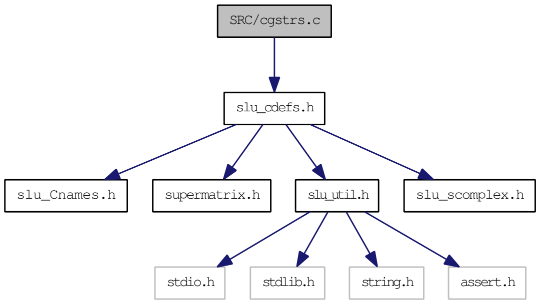


4.33 SRC/cgstrs.c File Reference

Solves a system using LU factorization.

`#include "slu_cdefs.h"`

Include dependency graph for cgstrs.c:



Functions

- void cusolve (int, int, complex ∗, complex ∗)

  Solves a dense upper triangular system.

- void clsolve (int, int, complex ∗, complex ∗)

  Solves a dense UNIT lower triangular system.

- void cmatvec (int, int, int, complex ∗, complex ∗, complex ∗)

  Performs a dense matrix-vector multiply: Mxvec = Mxvec + M ∗ vec.

- void cgstrs (trans_t trans, SuperMatrix ∗L, SuperMatrix ∗U, int ∗perm_c, int ∗perm_r, SuperMatrix ∗B, SuperLUStat_t ∗stat, int ∗info)
- void cprint_soln (int n, int nrhs, complex ∗soln)

4.33.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.33.2 Function Documentation

4.33.2.1 void cgstrs (trans_t *trans*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, int ∗ *perm_c*, int ∗ *perm_r*, SuperMatrix ∗ *B*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======

CGSTRS solves a system of linear equations A*X=B or A'*X=B
with A sparse and B dense, using the LU factorization computed by
CGSTRF.

See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

trans   (input) trans_t
         Specifies the form of the system of equations:
         = NOTRANS: A * X = B  (No transpose)
         = TRANS:   A'* X = B  (Transpose)
         = CONJ:    A**H * X = B  (Conjugate transpose)

L       (input) SuperMatrix*
        The factor L from the factorization Pr*A*Pc=L*U as computed by
        cgstrf(). Use compressed row subscripts storage for supernodes,
        i.e., L has types: Stype = SLU_SC, Dtype = SLU_C, Mtype = SLU_TRLU.

U       (input) SuperMatrix*
        The factor U from the factorization Pr*A*Pc=L*U as computed by
        cgstrf(). Use column-wise storage scheme, i.e., U has types:
        Stype = SLU_NC, Dtype = SLU_C, Mtype = SLU_TRU.

perm_c  (input) int*, dimension (L->ncol)
  Column permutation vector, which defines the
        permutation matrix Pc; perm_c[i] = j means column i of A is
        in position j in A*Pc.

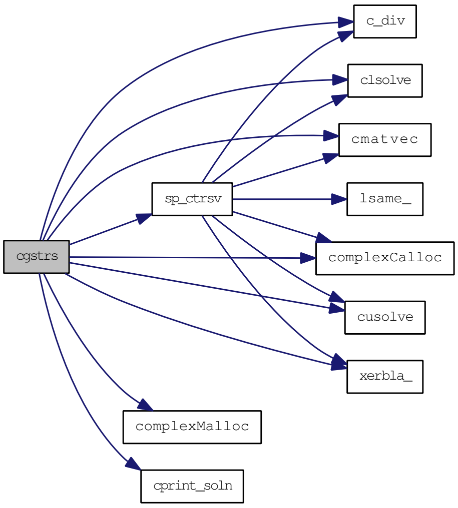
perm_r  (input) int*, dimension (L->nrow)
        Row permutation vector, which defines the permutation matrix Pr;
        perm_r[i] = j means row i of A is in position j in Pr*A.

B       (input/output) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_C, Mtype = SLU_GE.
        On entry, the right hand side matrix.
        On exit, the solution matrix if info = 0;

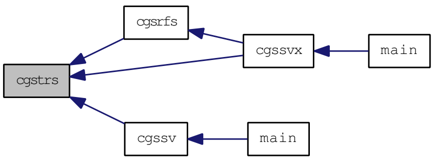
stat     (output) SuperLUStat_t*
         Record the statistics on runtime and floating-point operation count.
         See util.h for the definition of 'SuperLUStat_t'.

info    (output) int*
    = 0: successful exit
  < 0: if info = -i, the i-th argument had an illegal value
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.33.2.2 void clsolve (int *ldm*, int *ncol*, complex ∗ *M*, complex ∗ *rhs*)

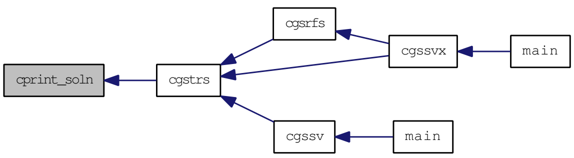
The unit lower triangular matrix is stored in a 2D array M(1:nrow,1:ncol). The solution will be returned in the rhs vector.

4.33.2.3 void cmatvec (int *ldm*, int *nrow*, int *ncol*, complex ∗ *M*, complex ∗ *vec*, complex ∗ *Mxvec*)

The input matrix is M(1:nrow,1:ncol); The product is returned in Mxvec[].

4.33.2.4 void cprint_soln (int *n*, int *nrhs*, complex ∗ *soln*)

Here is the caller graph for this function:



4.33.2.5 void cusolve (int *ldm*, int *ncol*, complex ∗ *M*, complex ∗ *rhs*)

The upper triangular matrix is stored in a 2-dim array M(1:ldm,1:ncol). The solution will be returned in the rhs vector.

4.34 SRC/clacon.c File Reference

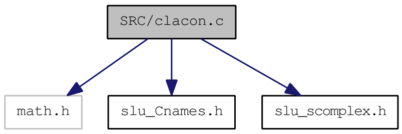
Estimates the 1-norm.

`#include <math.h>`

`#include "slu_Cnames.h"`

`#include "slu_scomplex.h"`

Include dependency graph for clacon.c:



Functions

- int clacon_ (int ∗n, complex ∗v, complex ∗x, float ∗est, int ∗kase)

4.34.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997
```

4.34.2 Function Documentation

4.34.2.1 int clacon_ (int ∗ *n*, complex ∗ *v*, complex ∗ *x*, float ∗ *est*, int ∗ *kase*)

```
Purpose
=======

CLACON estimates the 1-norm of a square matrix A.
Reverse communication is used for evaluating matrix-vector products.


Arguments
=========

N       (input) INT
        The order of the matrix.  N >= 1.

V       (workspace) COMPLEX PRECISION array, dimension (N)
        On the final return, V = A*W,  where  EST = norm(V)/norm(W)
        (W is not returned).

X       (input/output) COMPLEX PRECISION array, dimension (N)
        On an intermediate return, X should be overwritten by
```

```
             A * X,    if KASE=1,
             A' * X,   if KASE=2,
       where A' is the conjugate transpose of A,
      and CLACON must be re-called with all the other parameters
       unchanged.

EST     (output) FLOAT PRECISION
        An estimate (a lower bound) for norm(A).

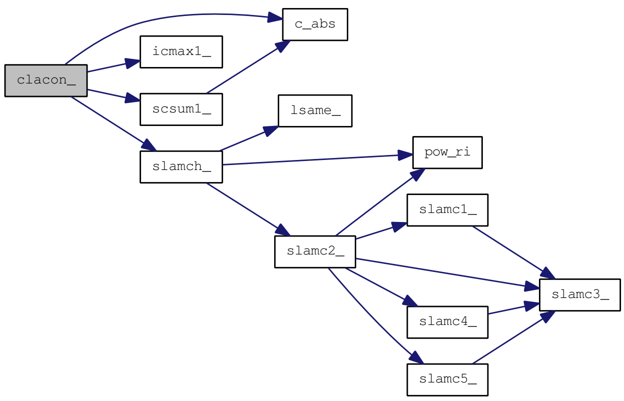
KASE    (input/output) INT
        On the initial call to CLACON, KASE should be 0.
        On an intermediate return, KASE will be 1 or 2, indicating
        whether X should be overwritten by A * X  or A' * X.
        On the final return from CLACON, KASE will again be 0.

Further Details
======= =======

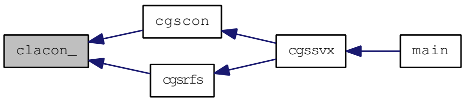
Contributed by Nick Higham, University of Manchester.
Originally named CONEST, dated March 16, 1988.

Reference: N.J. Higham, "FORTRAN codes for estimating the one-norm of
a real or complex matrix, with applications to condition estimation",
ACM Trans. Math. Soft., vol. 14, no. 4, pp. 381-396, December 1988.
=====================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:



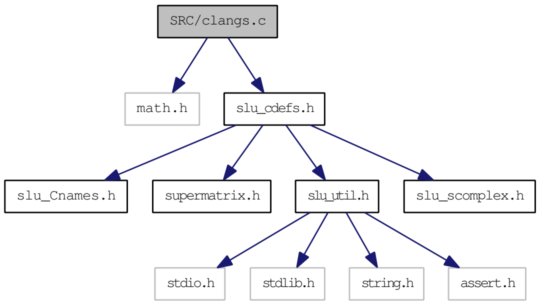
4.35 SRC/clangs.c File Reference

Returns the value of the one norm.

`#include <math.h>`

`#include "slu_cdefs.h"`

Include dependency graph for clangs.c:



Functions

- float clangs (char ∗norm, SuperMatrix ∗A)

4.35.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997

Modified from lapack routine CLANGE
```

4.35.2 Function Documentation

4.35.2.1 float clangs (char ∗ *norm*, SuperMatrix ∗ *A*)

```
Purpose
  =======

  CLANGS returns the value of the one norm, or the Frobenius norm, or
  the infinity norm, or the element of largest absolute value of a
  real matrix A.

  Description
  ===========

  CLANGE returns the value
```

```
    CLANGE = ( max(abs(A(i,j))), NORM = 'M' or 'm'
             (
             ( norm1(A),         NORM = '1', 'O' or 'o'
             (
             ( normI(A),         NORM = 'I' or 'i'
             (
             ( normF(A),         NORM = 'F', 'f', 'E' or 'e'

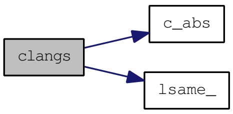
  where  norm1  denotes the  one norm of a matrix (maximum column sum),
  normI  denotes the  infinity norm  of a matrix  (maximum row sum) and
  normF  denotes the  Frobenius norm of a matrix (square root of sum of
  squares).  Note that  max(abs(A(i,j)))  is not a  matrix norm.

  Arguments
  =========

  NORM    (input) CHARACTER*1
          Specifies the value to be returned in CLANGE as described above.
  A       (input) SuperMatrix*
          The M by N sparse matrix A.

 =====================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:

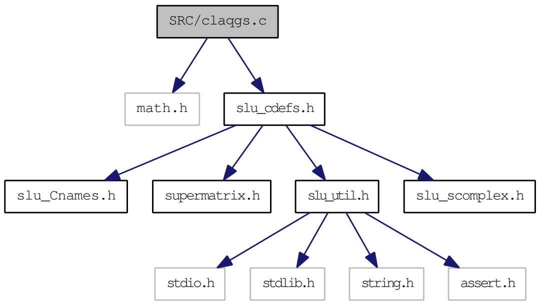
4.36 SRC/claqgs.c File Reference

Equlibrates a general sprase matrix.

`#include <math.h>`

`#include "slu_cdefs.h"`

Include dependency graph for claqgs.c:



Defines

- #define THRESH (0.1)

Functions

- void claqgs (SuperMatrix *A, float *r, float *c, float rowcnd, float colcnd, float amax, char *equed)

4.36.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997

Modified from LAPACK routine CLAQGE
```

4.36.2 Define Documentation

4.36.2.1 #define THRESH (0.1)

4.36.3 Function Documentation

4.36.3.1 void claqgs (SuperMatrix * *A*, float * *r*, float * *c*, float *rowcnd*, float *colcnd*, float *amax*, char * *equed*)

```
Purpose
=======
```

```
CLAQGS equilibrates a general sparse M by N matrix A using the row and
scaling factors in the vectors R and C.

See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

A       (input/output) SuperMatrix*
        On exit, the equilibrated matrix.  See EQUED for the form of
        the equilibrated matrix. The type of A can be:
 Stype = NC; Dtype = SLU_C; Mtype = GE.

R       (input) float*, dimension (A->nrow)
        The row scale factors for A.

C       (input) float*, dimension (A->ncol)
        The column scale factors for A.

ROWCND  (input) float
        Ratio of the smallest R(i) to the largest R(i).

COLCND  (input) float
        Ratio of the smallest C(i) to the largest C(i).

AMAX    (input) float
        Absolute value of largest matrix entry.

EQUED   (output) char*
        Specifies the form of equilibration that was done.
        = 'N':  No equilibration
        = 'R':  Row equilibration, i.e., A has been premultiplied by
                diag(R).
        = 'C':  Column equilibration, i.e., A has been postmultiplied
                by diag(C).
        = 'B':  Both row and column equilibration, i.e., A has been
                replaced by diag(R) * A * diag(C).

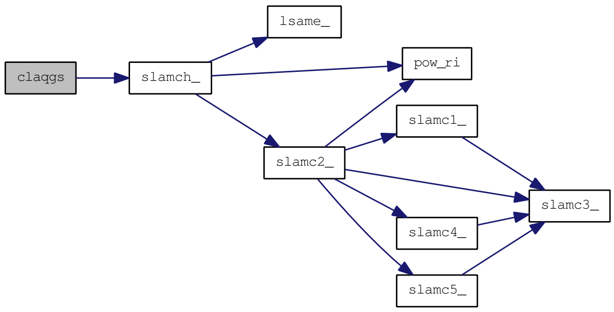
Internal Parameters
===================

THRESH is a threshold value used to decide if row or column scaling
should be done based on the ratio of the row or column scaling
factors.  If ROWCND < THRESH, row scaling is done, and if
COLCND < THRESH, column scaling is done.

LARGE and SMALL are threshold values used to decide if row scaling
should be done based on the absolute size of the largest matrix
element.  If AMAX > LARGE or AMAX < SMALL, row scaling is done.

=====================================================================
```

Here is the call graph for this function:



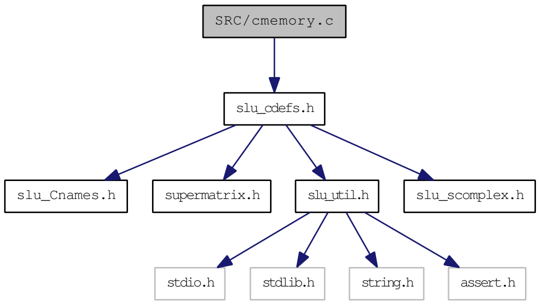
Here is the caller graph for this function:

4.37 SRC/cmemory.c File Reference

Memory details.

`#include "slu_cdefs.h"`

Include dependency graph for cmemory.c:



Data Structures

- struct e_node

    Headers for 4 types of dynamatically managed memory.

- struct LU_stack_t

Defines

- #define NO_MEMTYPE 4
- #define GluIntArray(n) (5 ∗ (n) + 5)
- #define StackFull(x) (x + stack.used >= stack.size)
- #define NotDoubleAlign(addr) ((long int)addr & 7)
- #define DoubleAlign(addr) (((long int)addr + 7) & ∼7L)
- #define TempSpace(m, w)
- #define Reduce(alpha) ((alpha + 1) / 2)

Typedefs

- typedef struct e_node ExpHeader

    Headers for 4 types of dynamatically managed memory.

Functions

- void ∗ cexpand (int ∗prev_len,MemType type,int len_to_copy,int keep_prev,GlobalLU_t ∗Glu)

    Expand the existing storage to accommodate more fill-ins.

- int cLUWorkInit (int m, int n, int panel_size, int ∗∗iworkptr, complex ∗∗dworkptr, LU_space_t MemModel)

Allocate known working storage. Returns 0 if success, otherwise returns the number of bytes allocated so far when failure occurred.

- void copy_mem_complex (int, void ∗, void ∗)
- void cStackCompress (GlobalLU_t ∗Glu)

  Compress the work[] array to remove fragmentation.

- void cSetupSpace (void ∗work, int lwork, LU_space_t ∗MemModel)

  Setup the memory model to be used for factorization.

- void ∗ cuser_malloc (int, int)
- void cuser_free (int, int)
- void copy_mem_int (int, void ∗, void ∗)
- void user_bcopy (char ∗, char ∗, int)
- int cQuerySpace (SuperMatrix ∗L, SuperMatrix ∗U, mem_usage_t ∗mem_usage)
- int cLUMemInit (fact_t fact, void ∗work, int lwork, int m, int n, int annz, int panel_size, SuperMatrix ∗L, SuperMatrix ∗U, GlobalLU_t ∗Glu, int ∗∗iwork, complex ∗∗dwork)

  Allocate storage for the data structures common to all factor routines.

- void cSetRWork (int m, int panel_size, complex ∗dworkptr, complex ∗∗dense, complex ∗∗tempv)

  Set up pointers for real working arrays.

- void cLUWorkFree (int ∗iwork, complex ∗dwork, GlobalLU_t ∗Glu)

  Free the working storage used by factor routines.

- int cLUMemXpand (int jcol, int next, MemType mem_type, int ∗maxlen, GlobalLU_t ∗Glu)

  Expand the data structures for L and U during the factorization.

- void callocateA (int n, int nnz, complex ∗∗a, int ∗∗asub, int ∗∗xa)

  Allocate storage for original matrix A.

- complex ∗ complexMalloc (int n)
- complex ∗ complexCalloc (int n)
- int cmemory_usage (const int nzlmax, const int nzumax, const int nzlumax, const int n)

Variables

- static ExpHeader ∗ expanders = 0
- static LU_stack_t stack
- static int no_expand

4.37.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003
```

4.37.2 Define Documentation

4.37.2.1 #define DoubleAlign(addr) (((long int)addr + 7) & ∼7L)

4.37.2.2 #define GluIntArray(n) (5 ∗ (n) + 5)

4.37.2.3 #define NO_MEMTYPE 4

4.37.2.4 #define NotDoubleAlign(addr) ((long int)addr & 7)

4.37.2.5 #define Reduce(alpha) ((alpha + 1) / 2)

4.37.2.6 #define StackFull(x) (x + stack.used >= stack.size)

4.37.2.7 #define TempSpace(m, w)

Value:

```
( (2*w + 4 + NO_MARKER) * m * sizeof(int) + \
            (w + 1) * m * sizeof(complex) )
```

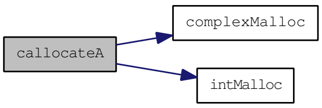
4.37.3 Typedef Documentation

4.37.3.1 typedef struct e_node ExpHeader

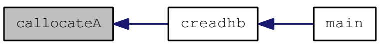
4.37.4 Function Documentation

4.37.4.1 void callocateA (int *n*, int *nnz*, complex ∗∗ *a*, int ∗∗ *asub*, int ∗∗ *xa*)

Here is the call graph for this function:

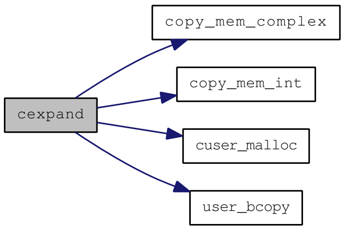


Here is the caller graph for this function:

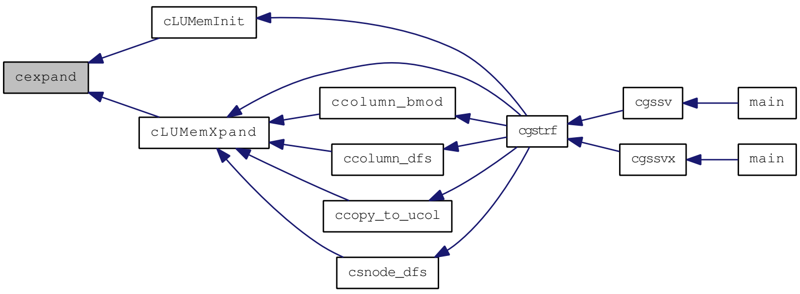


4.37.4.2 void ∗ cexpand (int ∗ *prev_len*, MemType *type*, int *len_to_copy*, int *keep_prev*, GlobalLU_t ∗ *Glu*)

Here is the call graph for this function:



Here is the caller graph for this function:

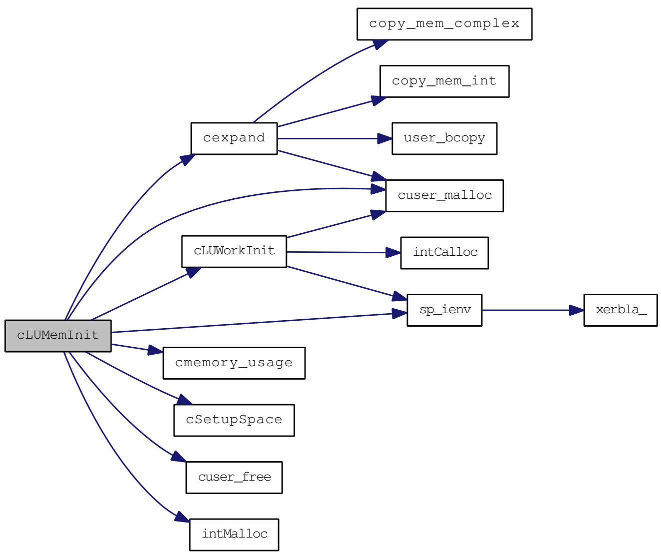


4.37.4.3 int cLUMemInit (fact_t *fact*, void ∗ *work*, int *lwork*, int *m*, int *n*, int *annz*, int *panel_size*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, GlobalLU_t ∗ *Glu*, int ∗∗ *iwork*, complex ∗∗ *dwork*)

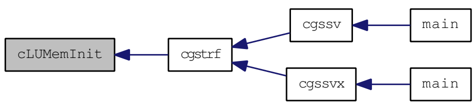
Memory-related.

```
For those unpredictable size, make a guess as FILL * nnz(A).
Return value:
    If lwork = -1, return the estimated amount of space required, plus n;
    otherwise, return the amount of space actually allocated when
    memory allocation failure occurred.
```

Here is the call graph for this function:



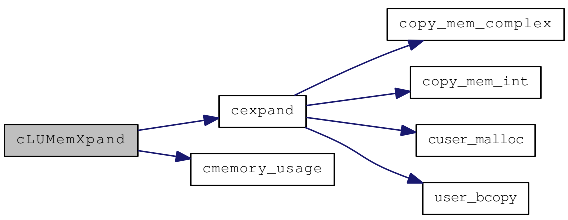
Here is the caller graph for this function:



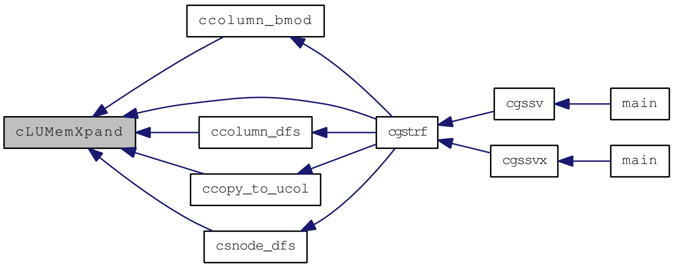
4.37.4.4 int cLUMemXpand (int *jcol*, int *next*, MemType *mem_type*, int ∗ *maxlen*, GlobalLU_t ∗ *Glu*)

```
Return value:   0 - successful return
              > 0 - number of bytes allocated when run out of space
```

Here is the call graph for this function:

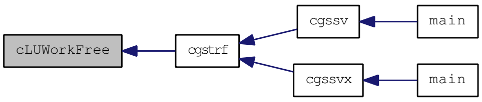


Here is the caller graph for this function:



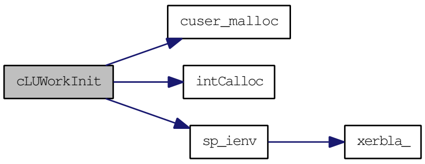
4.37.4.5 void cLUWorkFree (int ∗ *iwork*, complex ∗ *dwork*, GlobalLU_t ∗ *Glu*)

Here is the caller graph for this function:

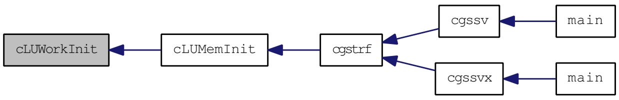


4.37.4.6 int cLUWorkInit (int *m*, int *n*, int *panel_size*, int ∗∗ *iworkptr*, complex ∗∗ *dworkptr*, LU_space_t *MemModel*)

Here is the call graph for this function:

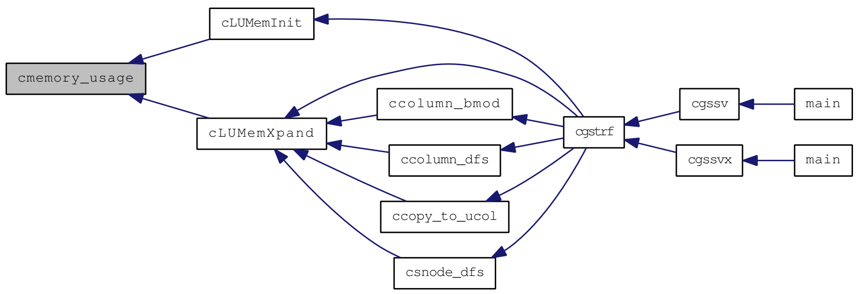


Here is the caller graph for this function:



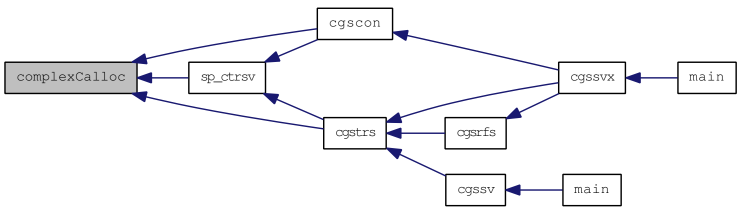
4.37.4.7 int cmemory_usage (const int *nzlmax*, const int *nzumax*, const int *nzlumax*, const int *n*)

Here is the caller graph for this function:



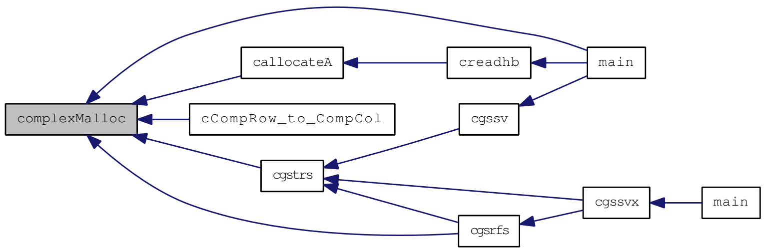
4.37.4.8 complex∗ complexCalloc (int *n*)

Here is the caller graph for this function:



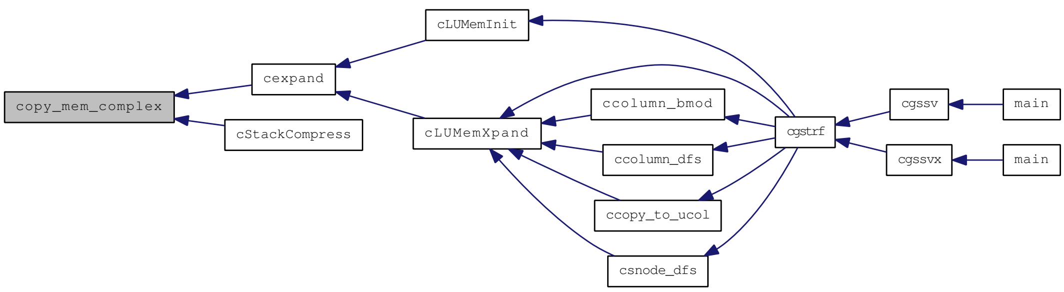
4.37.4.9 complex∗ complexMalloc (int *n*)

Here is the caller graph for this function:



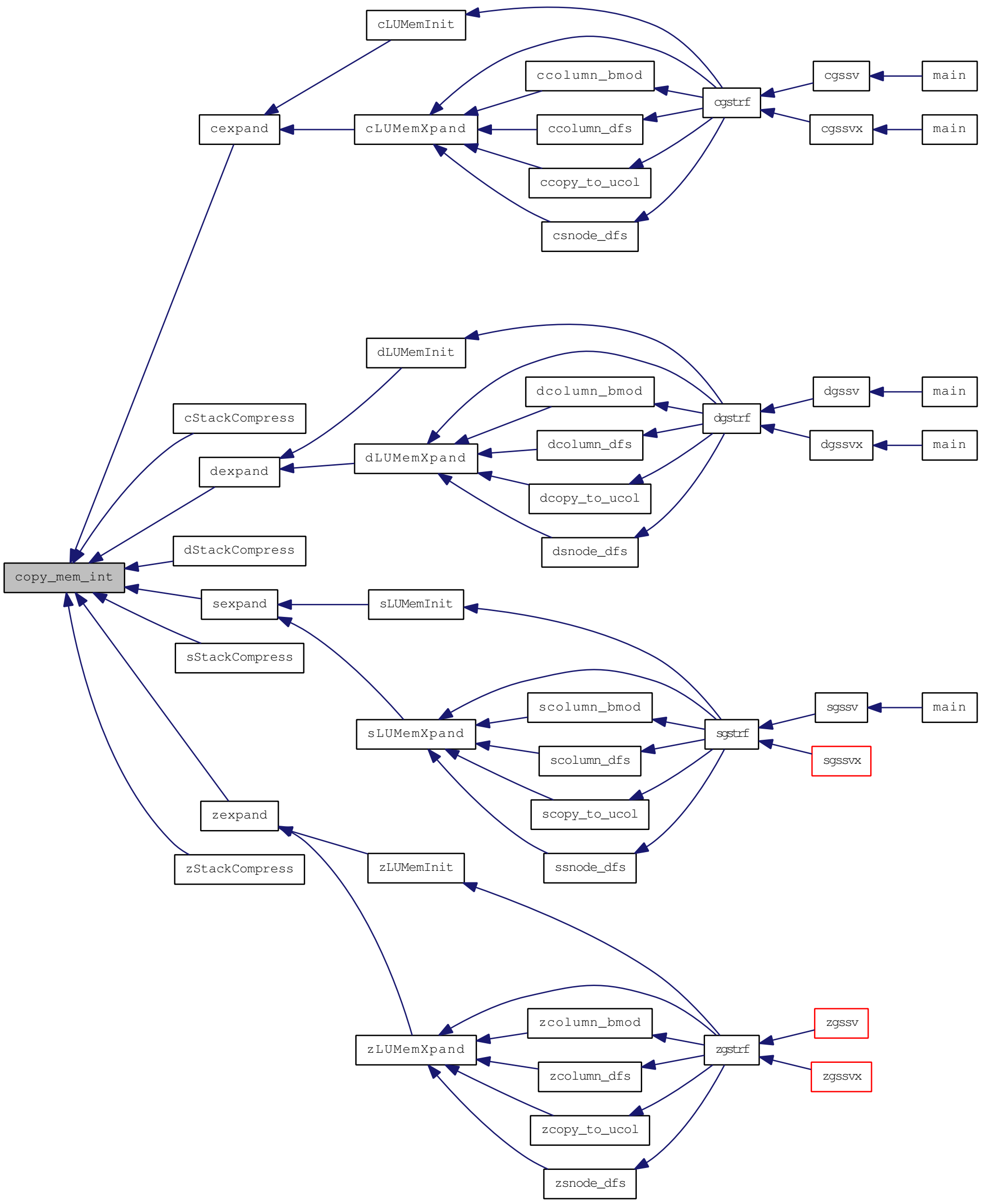
4.37.4.10 void copy_mem_complex (int *howmany*, void * *old*, void * *new*)

Here is the caller graph for this function:



4.37.4.11 void copy_mem_int (int, void *, void *)

Here is the caller graph for this function:



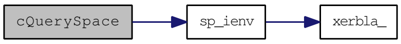
4.37.4.12 int cQuerySpace (SuperMatrix * *L*, SuperMatrix * *U*, mem_usage_t * *mem_usage*)

```
mem_usage consists of the following fields:
```

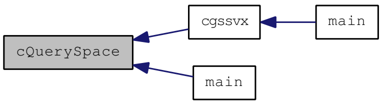
- ```
  for_lu (float)
       The amount of space used in bytes for the L data structures.
  ```
  ```

- `total_needed (float)`
  `The amount of space needed in bytes to perform factorization.`
- `expansions (int)`
  `Number of memory expansions during the LU factorization.`

Here is the call graph for this function:

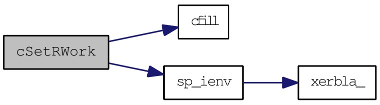


Here is the caller graph for this function:

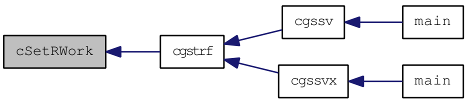


4.37.4.13 void cSetRWork (int *m*, int *panel_size*, complex ∗ *dworkptr*, complex ∗∗ *dense*, complex ∗∗ *tempv*)

Here is the call graph for this function:



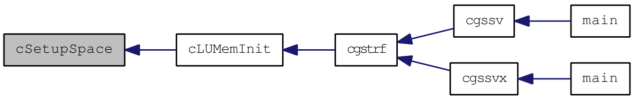
Here is the caller graph for this function:



4.37.4.14 void cSetupSpace (void ∗ *work*, int *lwork*, LU_space_t ∗ *MemModel*)

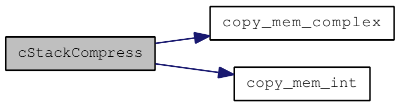
lwork = 0: use system malloc; lwork > 0: use user-supplied work[] space.

Here is the caller graph for this function:



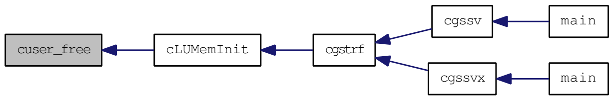
4.37.4.15 void cStackCompress (GlobalLU_t * *Glu*)

Here is the call graph for this function:



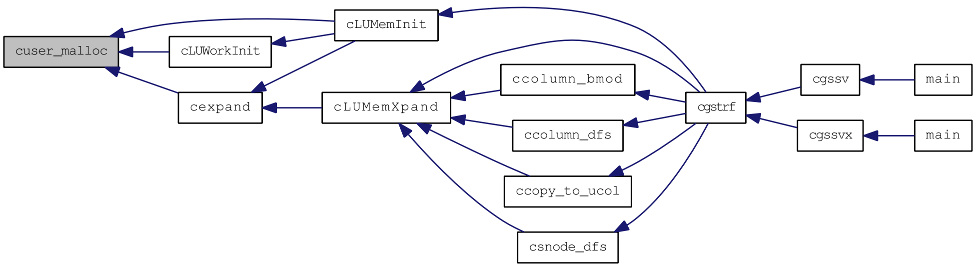
4.37.4.16 void cuser_free (int *bytes*, int *which_end*)

Here is the caller graph for this function:



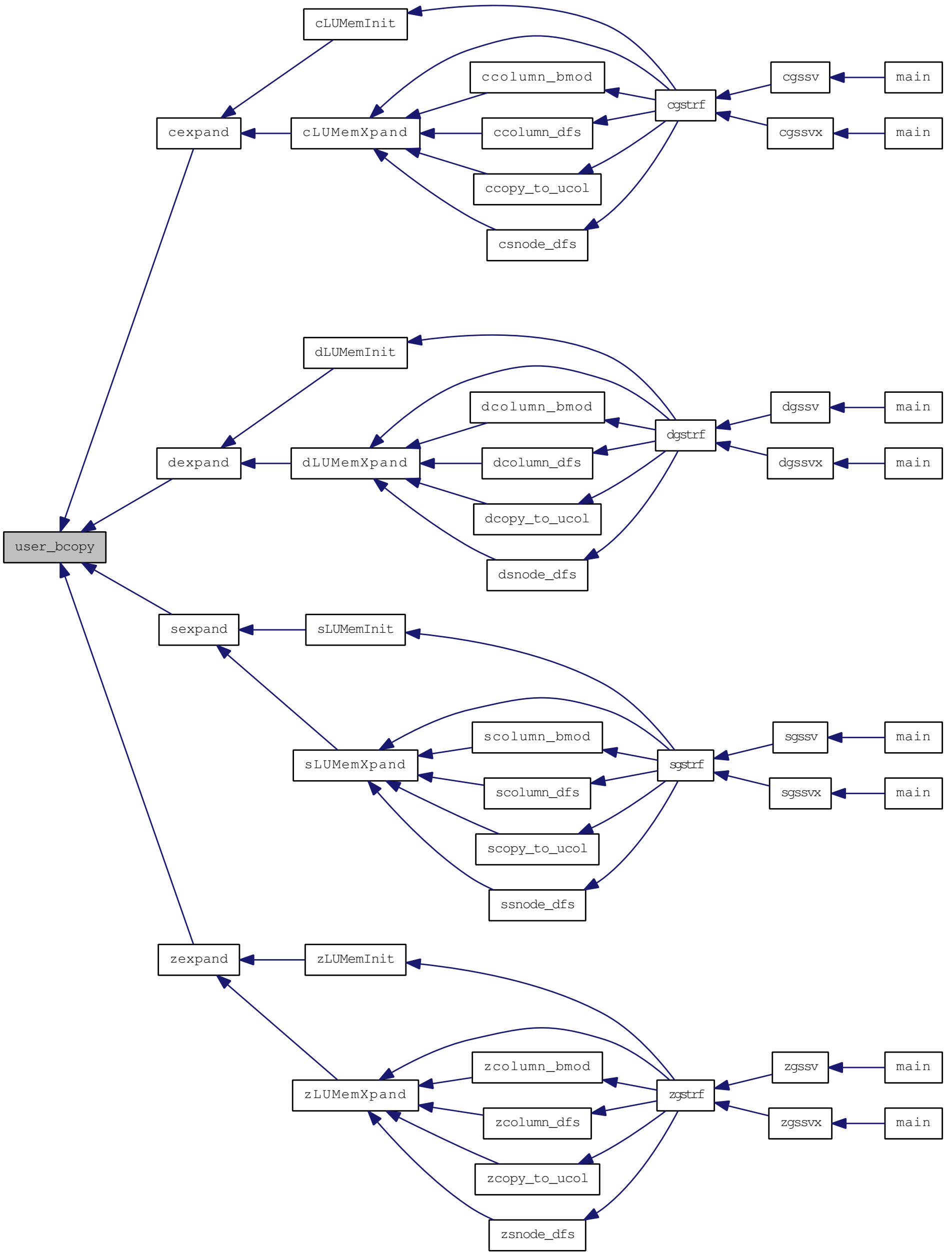
4.37.4.17 void * cuser_malloc (int *bytes*, int *which_end*)

Here is the caller graph for this function:



4.37.4.18 void user_bcopy (char ∗, char ∗, int)

Here is the caller graph for this function:



4.37.5 Variable Documentation

4.37.5.1 ExpHeader∗ expanders = 0 [static]

4.37.5.2 int no_expand [static]

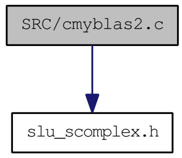
4.37.5.3 LU_stack_t stack [static]

4.38 SRC/cmyblas2.c File Reference

Level 2 Blas operations.

`#include "slu_scomplex.h"`

Include dependency graph for cmyblas2.c:



Functions

- void clsolve (int ldm, int ncol, complex ∗M, complex ∗rhs)

  Solves a dense UNIT lower triangular system.

- void cusolve (int ldm, int ncol, complex ∗M, complex ∗rhs)

  Solves a dense upper triangular system.

- void cmatvec (int ldm, int nrow, int ncol, complex ∗M, complex ∗vec, complex ∗Mxvec)

  Performs a dense matrix-vector multiply: $Mxvec = Mxvec + M * vec$.

4.38.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997
```

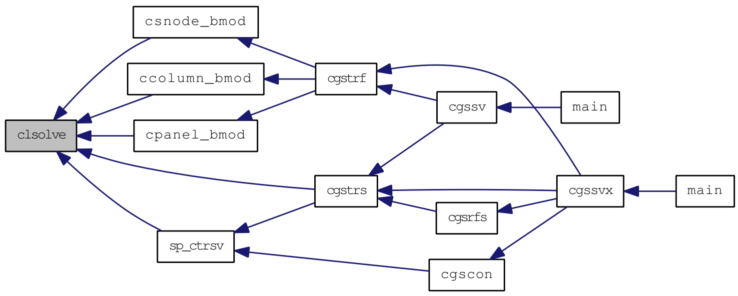
Purpose: Level 2 BLAS operations: solves and matvec, written in C. Note: This is only used when the system lacks an efficient BLAS library.

4.38.2 Function Documentation

4.38.2.1 void clsolve (int *ldm*, int *ncol*, complex ∗ *M*, complex ∗ *rhs*)

The unit lower triangular matrix is stored in a 2D array M(1:nrow,1:ncol). The solution will be returned in the rhs vector.

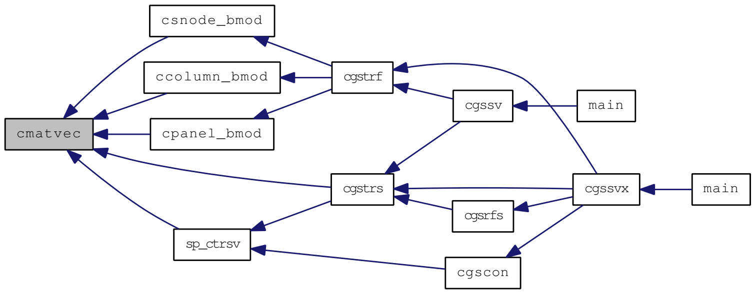
Here is the caller graph for this function:



4.38.2.2 void cmatvec (int *ldm*, int *nrow*, int *ncol*, complex ∗ *M*, complex ∗ *vec*, complex ∗ *Mxvec*)

The input matrix is M(1:nrow,1:ncol); The product is returned in Mxvec[].

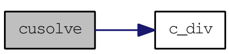
Here is the caller graph for this function:



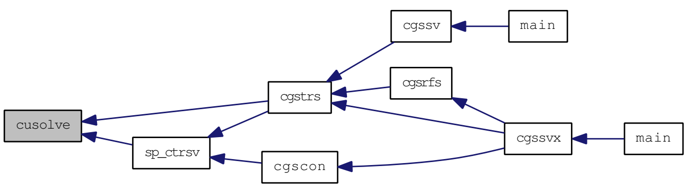
4.38.2.3 void cusolve (int *ldm*, int *ncol*, complex ∗ *M*, complex ∗ *rhs*)

The upper triangular matrix is stored in a 2-dim array M(1:ldm,1:ncol). The solution will be returned in the rhs vector.

Here is the call graph for this function:



Here is the caller graph for this function:

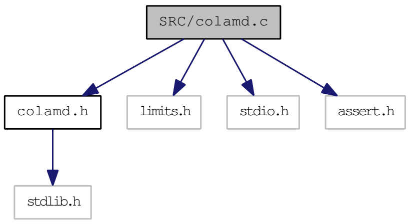


4.39 SRC/colamd.c File Reference

A sparse matrix column ordering algorithm.

```
#include "colamd.h"
#include <limits.h>
#include <stdio.h>
#include <assert.h>
```

Include dependency graph for colamd.c:



Defines

- #define PUBLIC
- #define PRIVATE static
- #define MAX(a, b) (((a) > (b)) ? (a) : (b))
- #define MIN(a, b) (((a) < (b)) ? (a) : (b))
- #define ONES_COMPLEMENT(r) (-(r)-1)
- #define TRUE (1)
- #define FALSE (0)
- #define EMPTY (-1)
- #define ALIVE (0)
- #define DEAD (-1)
- #define DEAD_PRINCIPAL (-1)
- #define DEAD_NON_PRINCIPAL (-2)
- #define ROW_IS_DEAD(r) ROW_IS_MARKED_DEAD (Row[r].shared2.mark)
- #define ROW_IS_MARKED_DEAD(row_mark) (row_mark < ALIVE)
- #define ROW_IS_ALIVE(r) (Row [r].shared2.mark >= ALIVE)
- #define COL_IS_DEAD(c) (Col [c].start < ALIVE)
- #define COL_IS_ALIVE(c) (Col [c].start >= ALIVE)
- #define COL_IS_DEAD_PRINCIPAL(c) (Col [c].start == DEAD_PRINCIPAL)
- #define KILL_ROW(r) { Row [r].shared2.mark = DEAD ; }
- #define KILL_PRINCIPAL_COL(c) { Col [c].start = DEAD_PRINCIPAL ; }
- #define KILL_NON_PRINCIPAL_COL(c) { Col [c].start = DEAD_NON_PRINCIPAL ; }
- #define PRINTF printf
- #define INDEX(i) (i)
- #define DEBUG0(params) ;
- #define DEBUG1(params) ;
- #define DEBUG2(params) ;
- #define DEBUG3(params) ;
- #define DEBUG4(params) ;
- #define ASSERT(expression) ((void) 0)

Functions

- PRIVATE int init_rows_cols (int n_row, int n_col, Colamd_Row Row[ ], Colamd_Col Col[ ], int A[ ], int p[ ], int stats[COLAMD_STATS])
- PRIVATE void init_scoring (int n_row, int n_col, Colamd_Row Row[ ], Colamd_Col Col[ ], int A[ ], int head[ ], double knobs[COLAMD_KNOBS], int ∗p_n_row2, int ∗p_n_col2, int ∗p_max_deg)
- PRIVATE int find_ordering (int n_row, int n_col, int Alen, Colamd_Row Row[ ], Colamd_Col Col[ ], int A[ ], int head[ ], int n_col2, int max_deg, int pfree)
- PRIVATE void order_children (int n_col, Colamd_Col Col[ ], int p[ ])
- PRIVATE void detect_super_cols (Colamd_Col Col[ ], int A[ ], int head[ ], int row_start, int row_length)
- PRIVATE int garbage_collection (int n_row, int n_col, Colamd_Row Row[ ], Colamd_Col Col[ ], int A[ ], int ∗pfree)
- PRIVATE int clear_mark (int n_row, Colamd_Row Row[ ])
- PRIVATE void print_report (char ∗method, int stats[COLAMD_STATS])
- PUBLIC int colamd_recommended (int nnz, int n_row, int n_col)
- PUBLIC void colamd_set_defaults (double knobs[COLAMD_KNOBS])
- PUBLIC int symamd (int n, int A[ ], int p[ ], int perm[ ], double knobs[COLAMD_KNOBS], int stats[COLAMD_STATS], void ∗(∗allocate)(size_t, size_t), void(∗release)(void ∗))
- PUBLIC int colamd (int n_row, int n_col, int Alen, int A[ ], int p[ ], double knobs[COLAMD_KNOBS], int stats[COLAMD_STATS])
- PUBLIC void colamd_report (int stats[COLAMD_STATS])
- PUBLIC void symamd_report (int stats[COLAMD_STATS])

4.39.1 Detailed Description

```
    ==========================================================================
    === colamd/symamd - a sparse matrix column ordering algorithm ============
    ==========================================================================


    colamd:  an approximate minimum degree column ordering algorithm,
     for LU factorization of symmetric or unsymmetric matrices,
QR factorization, least squares, interior point methods for
linear programming problems, and other related problems.


    symamd:  an approximate minimum degree ordering algorithm for Cholesky
     factorization of symmetric matrices.


    Purpose:


Colamd computes a permutation Q such that the Cholesky factorization of
(AQ)'(AQ) has less fill-in and requires fewer floating point operations
than A'A.  This also provides a good ordering for sparse partial
pivoting methods, P(AQ) = LU, where Q is computed prior to numerical
factorization, and P is computed during numerical factorization via
conventional partial pivoting with row interchanges.  Colamd is the
column ordering method used in SuperLU, part of the ScaLAPACK library.
It is also available as built-in function in MATLAB Version 6,
available from MathWorks, Inc. ( http://www.mathworks.com).  This
routine can be used in place of colmmd in MATLAB.
```

```
     Symamd computes a permutation P of a symmetric matrix A such that the
Cholesky factorization of PAP' has less fill-in and requires fewer
floating point operations than A.  Symamd constructs a matrix M such
that M'M has the same nonzero pattern of A, and then orders the columns
of M using colmmd.  The column ordering of M is then returned as the
row and column ordering P of A.

    Authors:

The authors of the code itself are Stefan I. Larimore and Timothy A.
Davis ( davis@cise.ufl.edu), University of Florida.  The algorithm was
developed in collaboration with John Gilbert, Xerox PARC, and Esmond
Ng, Oak Ridge National Laboratory.

    Date:

September 8, 2003.  Version 2.3.

    Acknowledgements:

This work was supported by the National Science Foundation, under
grants DMS-9504974 and DMS-9803599.

    Copyright and License:

Copyright (c) 1998-2003 by the University of Florida.
All Rights Reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

Permission is hereby granted to use, copy, modify, and/or distribute
this program, provided that the Copyright, this License, and the
Availability of the original version is retained on all copies and made
accessible to the end-user of any code or package that includes COLAMD
or any modified version of COLAMD.

    Availability:

The colamd/symamd library is available at

     http://www.cise.ufl.edu/research/sparse/colamd/

This is the  http://www.cise.ufl.edu/research/sparse/colamd/colamd.c
file.  It requires the colamd.h file.  It is required by the colamdmex.c
and symamdmex.c files, for the MATLAB interface to colamd and symamd.

    See the ChangeLog file for changes since Version 1.0.

    ==========================================================================
    === Description of user-callable routines ================================
    ==========================================================================
```

```
    ----------------------------------------------------------------------------
    colamd_recommended:
    ----------------------------------------------------------------------------

C syntax:

    include "colamd.h"
    int colamd_recommended (int nnz, int n_row, int n_col) ;

    or as a C macro

    include "colamd.h"
    Alen = COLAMD_RECOMMENDED (int nnz, int n_row, int n_col) ;

Purpose:

    Returns recommended value of Alen for use by colamd.  Returns -1
    if any input argument is negative.  The use of this routine
    or macro is optional.  Note that the macro uses its arguments
    more than once, so be careful for side effects, if you pass
    expressions as arguments to COLAMD_RECOMMENDED.  Not needed for
    symamd, which dynamically allocates its own memory.

Arguments (all input arguments):

    int nnz ; Number of nonzeros in the matrix A.  This must
be the same value as p [n_col] in the call to
colamd - otherwise you will get a wrong value
of the recommended memory to use.

    int n_row ; Number of rows in the matrix A.

    int n_col ; Number of columns in the matrix A.

    ----------------------------------------------------------------------------
    colamd_set_defaults:
    ----------------------------------------------------------------------------

C syntax:

    include "colamd.h"
    colamd_set_defaults (double knobs [COLAMD_KNOBS]) ;

Purpose:

    Sets the default parameters.  The use of this routine is optional.

Arguments:

    double knobs [COLAMD_KNOBS] ; Output only.
```

```
Colamd: rows with more than (knobs [COLAMD_DENSE_ROW] * n_col)
entries are removed prior to ordering.  Columns with more than
(knobs [COLAMD_DENSE_COL] * n_row) entries are removed prior to
ordering, and placed last in the output column ordering.

Symamd: uses only knobs [COLAMD_DENSE_ROW], which is knobs [0].
Rows and columns with more than (knobs [COLAMD_DENSE_ROW] * n)
entries are removed prior to ordering, and placed last in the
output ordering.

COLAMD_DENSE_ROW and COLAMD_DENSE_COL are defined as 0 and 1,
respectively, in colamd.h.  Default values of these two knobs
are both 0.5.  Currently, only knobs [0] and knobs [1] are
used, but future versions may use more knobs.  If so, they will
be properly set to their defaults by the future version of
colamd_set_defaults, so that the code that calls colamd will
not need to change, assuming that you either use
colamd_set_defaults, or pass a (double *) NULL pointer as the
knobs array to colamd or symamd.

    ----------------------------------------------------------------------------
    colamd:
    ----------------------------------------------------------------------------

C syntax:

    include "colamd.h"
    int colamd (int n_row, int n_col, int Alen, int *A, int *p,
     double knobs [COLAMD_KNOBS], int stats [COLAMD_STATS]) ;

Purpose:

    Computes a column ordering (Q) of A such that P(AQ)=LU or
    (AQ)'AQ=LL' have less fill-in and require fewer floating point
    operations than factorizing the unpermuted matrix A or A'A,
    respectively.

Returns:

    TRUE (1) if successful, FALSE (0) otherwise.

Arguments:

    int n_row ; Input argument.

Number of rows in the matrix A.
Restriction:  n_row >= 0.
Colamd returns FALSE if n_row is negative.

    int n_col ; Input argument.

Number of columns in the matrix A.
Restriction:  n_col >= 0.
Colamd returns FALSE if n_col is negative.
```

```
    int Alen ; Input argument.

Restriction (see note):
Alen >= 2*nnz + 6*(n_col+1) + 4*(n_row+1) + n_col
Colamd returns FALSE if these conditions are not met.

Note:  this restriction makes an modest assumption regarding
the size of the two typedef's structures in colamd.h.
We do, however, guarantee that

Alen >= colamd_recommended (nnz, n_row, n_col)

or equivalently as a C preprocessor macro:

Alen >= COLAMD_RECOMMENDED (nnz, n_row, n_col)

will be sufficient.

    int A [Alen] ; Input argument, undefined on output.

A is an integer array of size Alen.  Alen must be at least as
large as the bare minimum value given above, but this is very
low, and can result in excessive run time.  For best
performance, we recommend that Alen be greater than or equal to
colamd_recommended (nnz, n_row, n_col), which adds
nnz/5 to the bare minimum value given above.

On input, the row indices of the entries in column c of the
matrix are held in A [(p [c]) ... (p [c+1]-1)].  The row indices
in a given column c need not be in ascending order, and
duplicate row indices may be be present.  However, colamd will
work a little faster if both of these conditions are met
(Colamd puts the matrix into this format, if it finds that the
the conditions are not met).

The matrix is 0-based.  That is, rows are in the range 0 to
n_row-1, and columns are in the range 0 to n_col-1.  Colamd
returns FALSE if any row index is out of range.

The contents of A are modified during ordering, and are
undefined on output.

    int p [n_col+1] ; Both input and output argument.

p is an integer array of size n_col+1.  On input, it holds the
"pointers" for the column form of the matrix A.  Column c of
the matrix A is held in A [(p [c]) ... (p [c+1]-1)].  The first
entry, p [0], must be zero, and p [c] <= p [c+1] must hold
for all c in the range 0 to n_col-1.  The value p [n_col] is
thus the total number of entries in the pattern of the matrix A.
Colamd returns FALSE if these conditions are not met.
```

```
On output, if colamd returns TRUE, the array p holds the column
permutation (Q, for P(AQ)=LU or (AQ)'(AQ)=LL'), where p [0] is
the first column index in the new ordering, and p [n_col-1] is
the last.  That is, p [k] = j means that column j of A is the
kth pivot column, in AQ, where k is in the range 0 to n_col-1
(p [0] = j means that column j of A is the first column in AQ).

If colamd returns FALSE, then no permutation is returned, and
p is undefined on output.

    double knobs [COLAMD_KNOBS] ; Input argument.

See colamd_set_defaults for a description.

    int stats [COLAMD_STATS] ; Output argument.

Statistics on the ordering, and error status.
See colamd.h for related definitions.
Colamd returns FALSE if stats is not present.

stats [0]:  number of dense or empty rows ignored.

stats [1]:  number of dense or empty columns ignored (and
ordered last in the output permutation p)
Note that a row can become "empty" if it
contains only "dense" and/or "empty" columns,
and similarly a column can become "empty" if it
only contains "dense" and/or "empty" rows.

stats [2]:  number of garbage collections performed.
This can be excessively high if Alen is close
to the minimum required value.

stats [3]:  status code.  < 0 is an error code.
    > 1 is a warning or notice.

0 OK.  Each column of the input matrix contained
row indices in increasing order, with no
duplicates.

1 OK, but columns of input matrix were jumbled
(unsorted columns or duplicate entries).  Colamd
had to do some extra work to sort the matrix
first and remove duplicate entries, but it
still was able to return a valid permutation
(return value of colamd was TRUE).

stats [4]: highest numbered column that
is unsorted or has duplicate
entries.
stats [5]: last seen duplicate or
unsorted row index.
stats [6]: number of duplicate or
unsorted row indices.
```

```
-1 A is a null pointer

-2 p is a null pointer

-3  n_row is negative

stats [4]: n_row

-4 n_col is negative

stats [4]: n_col

-5 number of nonzeros in matrix is negative

stats [4]: number of nonzeros, p [n_col]

-6 p [0] is nonzero

stats [4]: p [0]

-7 A is too small

stats [4]: required size
stats [5]: actual size (Alen)

-8 a column has a negative number of entries

stats [4]: column with < 0 entries
stats [5]: number of entries in col

-9 a row index is out of bounds

stats [4]: column with bad row index
stats [5]: bad row index
stats [6]: n_row, # of rows of matrx

-10 (unused; see symamd.c)

-999 (unused; see symamd.c)

Future versions may return more statistics in the stats array.

Example:

    See  http://www.cise.ufl.edu/research/sparse/colamd/example.c
    for a complete example.

    To order the columns of a 5-by-4 matrix with 11 nonzero entries in
    the following nonzero pattern
```

```
     x 0 x 0
x 0 x x
0 x x 0
0 0 x x
x x 0 0

    with default knobs and no output statistics, do the following:

include "colamd.h"
define ALEN COLAMD_RECOMMENDED (11, 5, 4)
int A [ALEN] = {1, 2, 5, 3, 5, 1, 2, 3, 4, 2, 4} ;
int p [ ] = {0, 3, 5, 9, 11} ;
int stats [COLAMD_STATS] ;
colamd (5, 4, ALEN, A, p, (double *) NULL, stats) ;

    The permutation is returned in the array p, and A is destroyed.

    ----------------------------------------------------------------------------
    symamd:
    ----------------------------------------------------------------------------

C syntax:

    include "colamd.h"
    int symamd (int n, int *A, int *p, int *perm,
     double knobs [COLAMD_KNOBS], int stats [COLAMD_STATS],
void (*allocate) (size_t, size_t), void (*release) (void *)) ;

Purpose:

         The symamd routine computes an ordering P of a symmetric sparse
    matrix A such that the Cholesky factorization PAP' = LL' remains
    sparse.  It is based on a column ordering of a matrix M constructed
    so that the nonzero pattern of M'M is the same as A.  The matrix A
    is assumed to be symmetric; only the strictly lower triangular part
    is accessed.  You must pass your selected memory allocator (usually
    calloc/free or mxCalloc/mxFree) to symamd, for it to allocate
    memory for the temporary matrix M.

Returns:

    TRUE (1) if successful, FALSE (0) otherwise.

Arguments:

    int n ; Input argument.

     Number of rows and columns in the symmetrix matrix A.
Restriction:  n >= 0.
Symamd returns FALSE if n is negative.

    int A [nnz] ; Input argument.

     A is an integer array of size nnz, where nnz = p [n].
```

```
The row indices of the entries in column c of the matrix are
held in A [(p [c]) ... (p [c+1]-1)].  The row indices in a
given column c need not be in ascending order, and duplicate
row indices may be present.  However, symamd will run faster
if the columns are in sorted order with no duplicate entries.

The matrix is 0-based.  That is, rows are in the range 0 to
n-1, and columns are in the range 0 to n-1.  Symamd
returns FALSE if any row index is out of range.

The contents of A are not modified.

    int p [n+1] ;    Input argument.

p is an integer array of size n+1.  On input, it holds the
"pointers" for the column form of the matrix A.  Column c of
the matrix A is held in A [(p [c]) ... (p [c+1]-1)].  The first
entry, p [0], must be zero, and p [c] <= p [c+1] must hold
for all c in the range 0 to n-1.  The value p [n] is
thus the total number of entries in the pattern of the matrix A.
Symamd returns FALSE if these conditions are not met.

The contents of p are not modified.

    int perm [n+1] ;    Output argument.

On output, if symamd returns TRUE, the array perm holds the
permutation P, where perm [0] is the first index in the new
ordering, and perm [n-1] is the last.  That is, perm [k] = j
means that row and column j of A is the kth column in PAP',
where k is in the range 0 to n-1 (perm [0] = j means
that row and column j of A are the first row and column in
PAP').  The array is used as a workspace during the ordering,
which is why it must be of length n+1, not just n.

    double knobs [COLAMD_KNOBS] ; Input argument.

See colamd_set_defaults for a description.

    int stats [COLAMD_STATS] ; Output argument.

Statistics on the ordering, and error status.
See colamd.h for related definitions.
Symamd returns FALSE if stats is not present.

stats [0]:  number of dense or empty row and columns ignored
(and ordered last in the output permutation
perm).  Note that a row/column can become
"empty" if it contains only "dense" and/or
"empty" columns/rows.

stats [1]:  (same as stats [0])

stats [2]:  number of garbage collections performed.
```

```
stats [3]:  status code.  < 0 is an error code.
    > 1 is a warning or notice.

0 OK.  Each column of the input matrix contained
row indices in increasing order, with no
duplicates.

1 OK, but columns of input matrix were jumbled
(unsorted columns or duplicate entries).  Symamd
had to do some extra work to sort the matrix
first and remove duplicate entries, but it
still was able to return a valid permutation
(return value of symamd was TRUE).

stats [4]: highest numbered column that
is unsorted or has duplicate
entries.
stats [5]: last seen duplicate or
unsorted row index.
stats [6]: number of duplicate or
unsorted row indices.

-1 A is a null pointer

-2 p is a null pointer

-3 (unused, see colamd.c)

-4  n is negative

stats [4]: n

-5 number of nonzeros in matrix is negative

stats [4]: # of nonzeros (p [n]).

-6 p [0] is nonzero

stats [4]: p [0]

-7 (unused)

-8 a column has a negative number of entries

stats [4]: column with < 0 entries
stats [5]: number of entries in col

-9 a row index is out of bounds

stats [4]: column with bad row index
stats [5]: bad row index
stats [6]: n_row, # of rows of matrx
```

```
-10 out of memory (unable to allocate temporary
workspace for M or count arrays using the
"allocate" routine passed into symamd).

-999 internal error.  colamd failed to order the
matrix M, when it should have succeeded.  This
indicates a bug.  If this (and *only* this)
error code occurs, please contact the authors.
Don't contact the authors if you get any other
error code.

Future versions may return more statistics in the stats array.

    void * (*allocate) (size_t, size_t)

     A pointer to a function providing memory allocation.  The
allocated memory must be returned initialized to zero.  For a
C application, this argument should normally be a pointer to
calloc.  For a MATLAB mexFunction, the routine mxCalloc is
passed instead.

    void (*release) (size_t, size_t)

     A pointer to a function that frees memory allocated by the
memory allocation routine above.  For a C application, this
argument should normally be a pointer to free.  For a MATLAB
mexFunction, the routine mxFree is passed instead.

    ----------------------------------------------------------------------------
    colamd_report:
    ----------------------------------------------------------------------------

C syntax:

    include "colamd.h"
    colamd_report (int stats [COLAMD_STATS]) ;

Purpose:

    Prints the error status and statistics recorded in the stats
    array on the standard error output (for a standard C routine)
    or on the MATLAB output (for a mexFunction).

Arguments:

    int stats [COLAMD_STATS] ; Input only.  Statistics from colamd.

    ----------------------------------------------------------------------------
    symamd_report:
    ----------------------------------------------------------------------------

C syntax:
```

```
include "colamd.h"
symamd_report (int stats [COLAMD_STATS]) ;
```

Purpose:

```
Prints the error status and statistics recorded in the stats
array on the standard error output (for a standard C routine)
or on the MATLAB output (for a mexFunction).
```

Arguments:

```
int stats [COLAMD_STATS] ; Input only.  Statistics from symamd.
```

4.39.2 Define Documentation

4.39.2.1 #define ALIVE (0)

4.39.2.2 #define ASSERT(expression) ((void) 0)

4.39.2.3 #define COL_IS_ALIVE(c) (Col [c].start >= ALIVE)

4.39.2.4 #define COL_IS_DEAD(c) (Col [c].start < ALIVE)

4.39.2.5 #define COL_IS_DEAD_PRINCIPAL(c) (Col [c].start == DEAD_PRINCIPAL)

4.39.2.6 #define DEAD (-1)

4.39.2.7 #define DEAD_NON_PRINCIPAL (-2)

4.39.2.8 #define DEAD_PRINCIPAL (-1)

4.39.2.9 #define DEBUG0(params) ;

4.39.2.10 #define DEBUG1(params) ;

4.39.2.11 #define DEBUG2(params) ;

4.39.2.12 #define DEBUG3(params) ;

4.39.2.13 #define DEBUG4(params) ;

4.39.2.14 #define EMPTY (-1)

4.39.2.15 #define FALSE (0)

4.39.2.16 #define INDEX(i) (i)

4.39.2.17 #define KILL_NON_PRINCIPAL_COL(c) { Col [c].start = DEAD_NON_PRINCIPAL ; }

4.39.2.18 #define KILL_PRINCIPAL_COL(c) { Col [c].start = DEAD_PRINCIPAL ; }

4.39.2.19 #define KILL_ROW(r) { Row [r].shared2.mark = DEAD ; }

4.39.2.20 #define MAX(a, b) (((a) > (b)) ? (a) : (b))

4.39.2.21 #define MIN(a, b) (((a) < (b)) ? (a) : (b))

4.39.2.22 #define ONES_COMPLEMENT(r) (-(r)-1)

4.39.2.23 #define PRINTF printf

4.39.2.24 #define PRIVATE static

4.39.2.25 #define PUBLIC

4.39.2.26 #define ROW_IS_ALIVE(r) (Row [r].shared2.mark >= ALIVE)

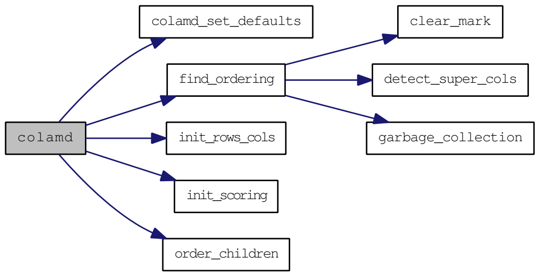
4.39.2.27 #define ROW_IS_DEAD(r) ROW_IS_MARKED_DEAD (Row[r].shared2.mark)

4.39.2.28 #define ROW_IS_MARKED_DEAD(row_mark) (row_mark < ALIVE)

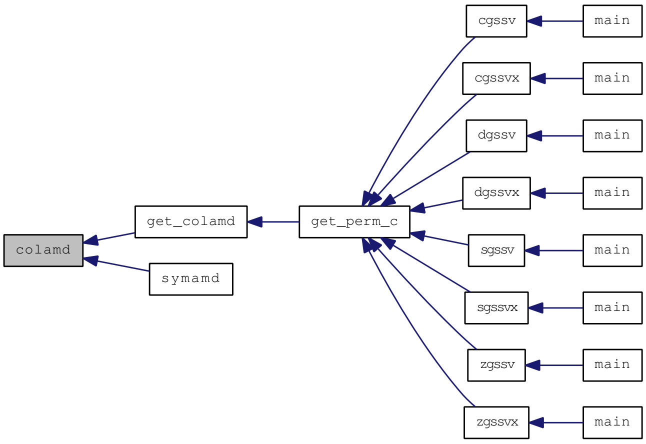
4.39.2.29 #define TRUE (1)

4.39.3.2 PUBLIC int colamd (int *n_row*, int *n_col*, int *Alen*, int *A*[ ], int *p*[ ], double *knobs*[COLAMD_KNOBS], int *stats*[COLAMD_STATS])

Here is the call graph for this function:

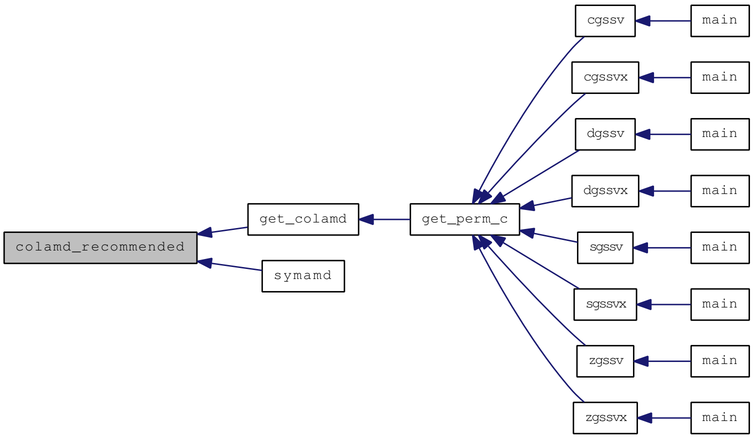


Here is the caller graph for this function:



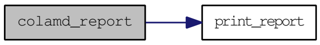
4.39.3.3 PUBLIC int colamd_recommended (int *nnz*, int *n_row*, int *n_col*)

Here is the caller graph for this function:



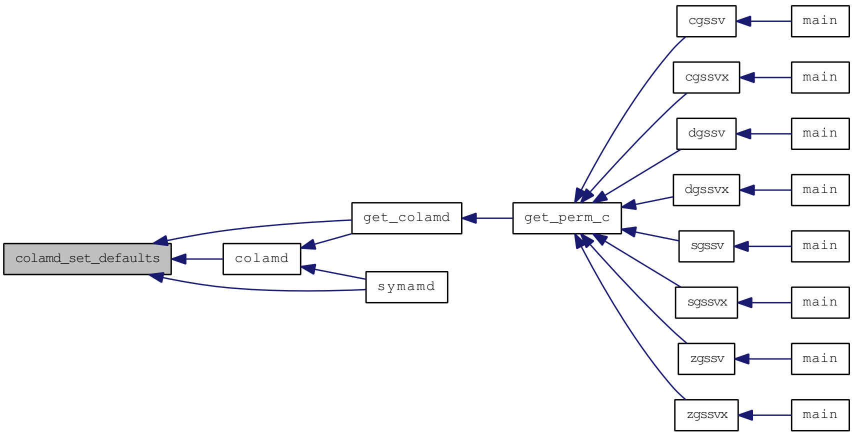
4.39.3.4 PUBLIC void colamd_report (int *stats*[COLAMD_STATS])

Here is the call graph for this function:



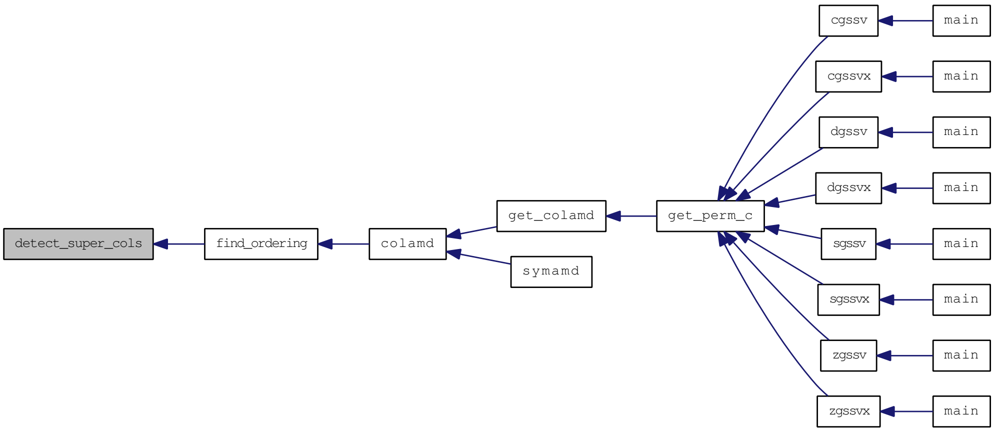
4.39.3.5 PUBLIC void colamd_set_defaults (double *knobs*[COLAMD_KNOBS])

Here is the caller graph for this function:



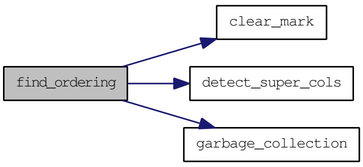
4.39.3.6 PRIVATE void detect_super_cols (Colamd_Col *Col*[ ], int *A*[ ], int *head*[ ], int *row_start*, int *row_length*)

Here is the caller graph for this function:

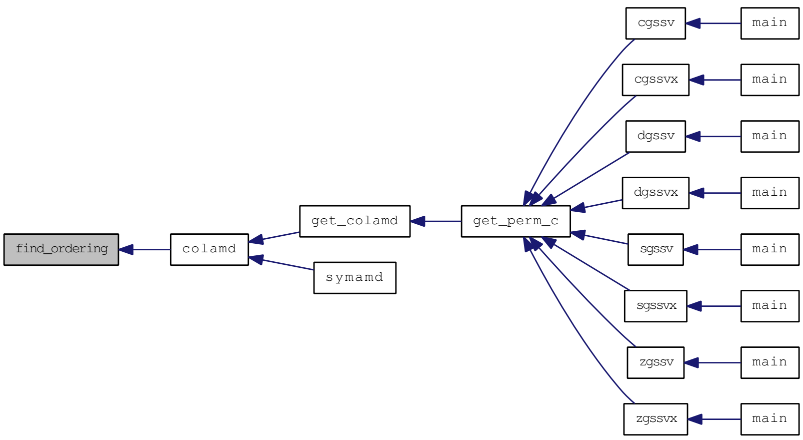


4.39.3.7 PRIVATE int find_ordering (int *n_row*, int *n_col*, int *Alen*, Colamd_Row *Row*[ ], Colamd_Col *Col*[ ], int *A*[ ], int *head*[ ], int *n_col2*, int *max_deg*, int *pfree*)

Here is the call graph for this function:

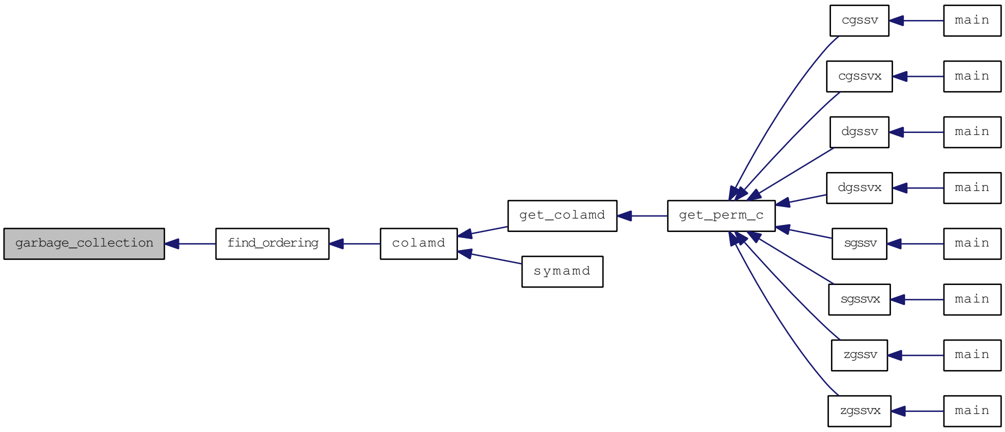


Here is the caller graph for this function:



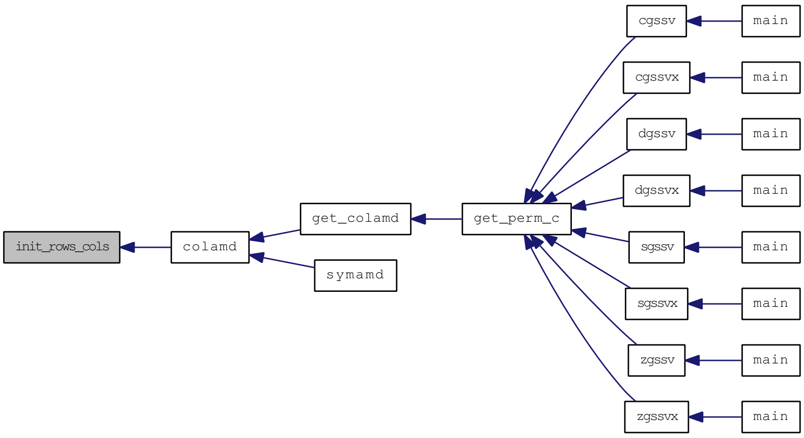
4.39.3.8 PRIVATE int garbage_collection (int *n_row*, int *n_col*, Colamd_Row *Row*[ ], Colamd_Col *Col*[ ], int *A*[ ], int ∗ *pfree*)

Here is the caller graph for this function:



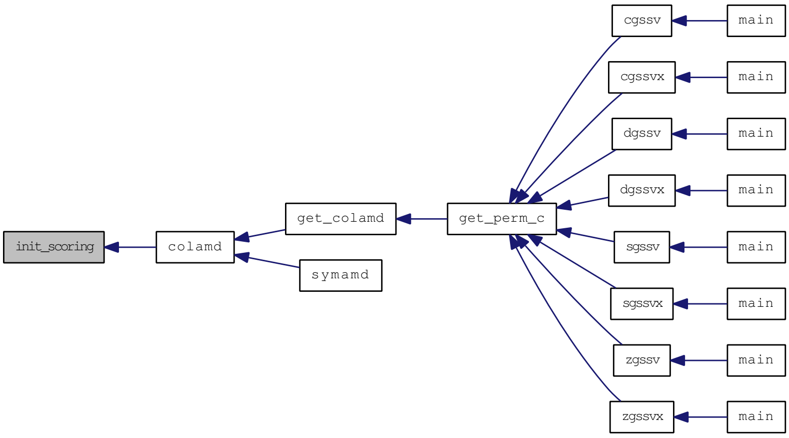
4.39.3.9 PRIVATE int init_rows_cols (int *n_row*, int *n_col*, Colamd_Row *Row*[ ], Colamd_Col *Col*[ ], int *A*[ ], int *p*[ ], int *stats*[COLAMD_STATS])

Here is the caller graph for this function:



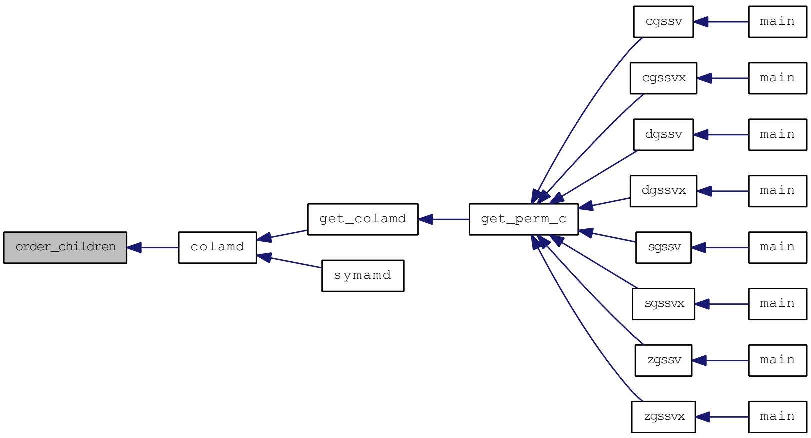
4.39.3.10 PRIVATE void init_scoring (int *n_row*, int *n_col*, Colamd_Row *Row*[ ], Colamd_Col *Col*[ ], int *A*[ ], int *head*[ ], double *knobs*[COLAMD_KNOBS], int ∗ *p_n_row2*, int ∗ *p_n_col2*, int ∗ *p_max_deg*)

Here is the caller graph for this function:



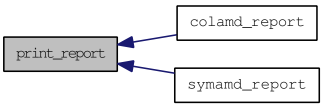
4.39.3.11 PRIVATE void order_children (int *n_col*, Colamd_Col *Col*[ ], int *p*[ ])

Here is the caller graph for this function:



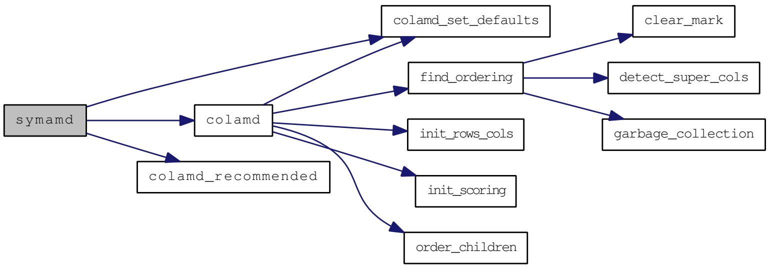
4.39.3.12 PRIVATE void print_report (char ∗ *method*, int *stats*[COLAMD_STATS])

Here is the caller graph for this function:



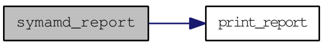
4.39.3.13 PUBLIC int symamd (int *n*, int *A*[ ], int *p*[ ], int *perm*[ ], double *knobs*[COLAMD_KNOBS], int *stats*[COLAMD_STATS], void ∗(∗)(size_t, size_t) *allocate*, void(∗)(void ∗) *release*)

Here is the call graph for this function:



4.39.3.14 PUBLIC void symamd_report (int *stats*[COLAMD_STATS])

Here is the call graph for this function:

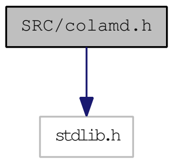


4.40 SRC/colamd.h File Reference

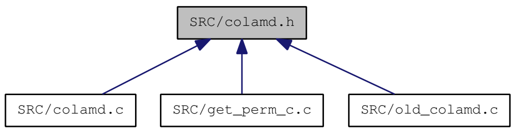
Colamd prototypes and definitions.

`#include <stdlib.h>`

Include dependency graph for colamd.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct Colamd_Col_struct
- struct Colamd_Row_struct

Defines

- #define COLAMD_KNOBS 20
- #define COLAMD_STATS 20
- #define COLAMD_DENSE_ROW 0
- #define COLAMD_DENSE_COL 1
- #define COLAMD_DEFRAG_COUNT 2
- #define COLAMD_STATUS 3
- #define COLAMD_INFO1 4
- #define COLAMD_INFO2 5
- #define COLAMD_INFO3 6
- #define COLAMD_OK (0)
- #define COLAMD_OK_BUT_JUMBLED (1)
- #define COLAMD_ERROR_A_not_present (-1)
- #define COLAMD_ERROR_p_not_present (-2)
- #define COLAMD_ERROR_nrow_negative (-3)
- #define COLAMD_ERROR_ncol_negative (-4)
- #define COLAMD_ERROR_nnz_negative (-5)
- #define COLAMD_ERROR_p0_nonzero (-6)
- #define COLAMD_ERROR_A_too_small (-7)
- #define COLAMD_ERROR_col_length_negative (-8)
- #define COLAMD_ERROR_row_index_out_of_bounds (-9)

- #define COLAMD_ERROR_out_of_memory (-10)
- #define COLAMD_ERROR_internal_error (-999)
- #define COLAMD_C(n_col) ((int) (((n_col) + 1) ∗ sizeof (Colamd_Col) / sizeof (int)))
- #define COLAMD_R(n_row) ((int) (((n_row) + 1) ∗ sizeof (Colamd_Row) / sizeof (int)))
- #define COLAMD_RECOMMENDED(nnz, n_row, n_col)

Typedefs

- typedef struct Colamd_Col_struct Colamd_Col
- typedef struct Colamd_Row_struct Colamd_Row

Functions

- int colamd_recommended (int nnz, int n_row, int n_col)
- void colamd_set_defaults (double knobs[COLAMD_KNOBS])
- int colamd (int n_row, int n_col, int Alen, int A[ ], int p[ ], double knobs[COLAMD_KNOBS], int stats[COLAMD_STATS])
- int symamd (int n, int A[ ], int p[ ], int perm[ ], double knobs[COLAMD_KNOBS], int stats[COLAMD_STATS], void ∗(∗allocate)(size_t, size_t), void(∗release)(void ∗))
- void colamd_report (int stats[COLAMD_STATS])
- void symamd_report (int stats[COLAMD_STATS])

4.40.1 Detailed Description

```
    ==========================================================================
    === colamd/symamd prototypes and definitions =============================
    ==========================================================================


    You must include this file (colamd.h) in any routine that uses colamd,
    symamd, or the related macros and definitions.


    Authors:


The authors of the code itself are Stefan I. Larimore and Timothy A.
Davis ( davis@cise.ufl.edu), University of Florida.  The algorithm was
developed in collaboration with John Gilbert, Xerox PARC, and Esmond
Ng, Oak Ridge National Laboratory.


    Date:


September 8, 2003.  Version 2.3.


    Acknowledgements:


This work was supported by the National Science Foundation, under
grants DMS-9504974 and DMS-9803599.


    Notice:
```

```
    Availability:
```

```
The colamd/symamd library is available at
```

```
     http://www.cise.ufl.edu/research/sparse/colamd/
```

```
This is the  http://www.cise.ufl.edu/research/sparse/colamd/colamd.h
file.  It is required by the colamd.c, colamdmex.c, and symamdmex.c
files, and by any C code that calls the routines whose prototypes are
listed below, or that uses the colamd/symamd definitions listed below.
```

4.40.2 Define Documentation

4.40.2.1 #define COLAMD_C(n_col) ((int) (((n_col) + 1) ∗ sizeof (Colamd_Col) / sizeof (int)))

4.40.2.2 #define COLAMD_DEFRAG_COUNT 2

4.40.2.3 #define COLAMD_DENSE_COL 1

4.40.2.4 #define COLAMD_DENSE_ROW 0

4.40.2.5 #define COLAMD_ERROR_A_not_present (-1)

4.40.2.6 #define COLAMD_ERROR_A_too_small (-7)

4.40.2.7 #define COLAMD_ERROR_col_length_negative (-8)

4.40.2.8 #define COLAMD_ERROR_internal_error (-999)

4.40.2.9 #define COLAMD_ERROR_ncol_negative (-4)

4.40.2.10 #define COLAMD_ERROR_nnz_negative (-5)

4.40.2.11 #define COLAMD_ERROR_nrow_negative (-3)

4.40.2.12 #define COLAMD_ERROR_out_of_memory (-10)

4.40.2.13 #define COLAMD_ERROR_p0_nonzero (-6)

4.40.2.14 #define COLAMD_ERROR_p_not_present (-2)

4.40.2.15 #define COLAMD_ERROR_row_index_out_of_bounds (-9)

4.40.2.16 #define COLAMD_INFO1 4

4.40.2.17 #define COLAMD_INFO2 5

4.40.2.18 #define COLAMD_INFO3 6

4.40.2.19 #define COLAMD_KNOBS 20

4.40.2.20 #define COLAMD_OK (0)

4.40.2.21 #define COLAMD_OK_BUT_JUMBLED (1)

4.40.2.22 #define COLAMD_R(n_row) ((int) (((n_row) + 1) ∗ sizeof (Colamd_Row) / sizeof (int)))

4.40.2.23 #define COLAMD_RECOMMENDED(nnz, n_row, n_col)

Value:

```
(                                                                       \
((nnz) < 0 || (n_row) < 0 || (n_col) < 0)                               \
?                                                                       \
```

```
    (-1)                                                                    \
:                                                                           \
    (2 * (nnz) + COLAMD_C (n_col) + COLAMD_R (n_row) + (n_col) + ((nnz) / 5)) \
)
```

4.40.2.24 #define COLAMD_STATS 20

4.40.2.25 #define COLAMD_STATUS 3

4.40.3 Typedef Documentation

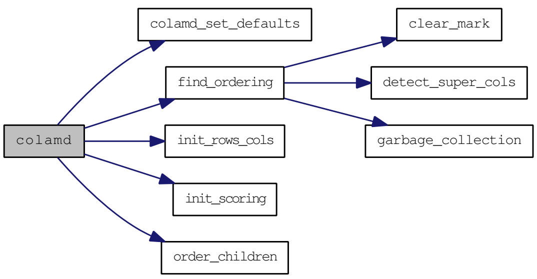
4.40.3.1 typedef struct Colamd_Col_struct Colamd_Col

4.40.3.2 typedef struct Colamd_Row_struct Colamd_Row

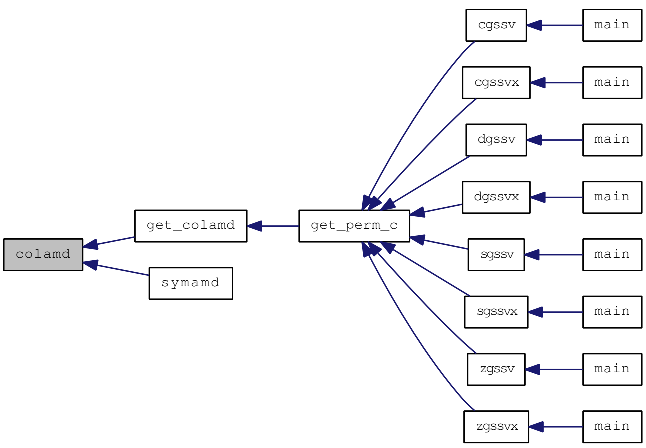
4.40.4 Function Documentation

4.40.4.1 int colamd (int *n_row*, int *n_col*, int *Alen*, int *A*[ ], int *p*[ ], double *knobs*[COLAMD_KNOBS], int *stats*[COLAMD_STATS])

Here is the call graph for this function:



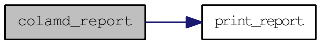
Here is the caller graph for this function:



4.40.4.2 int colamd_recommended (int *nnz*, int *n_row*, int *n_col*)

4.40.4.3 void colamd_report (int *stats*[COLAMD_STATS])

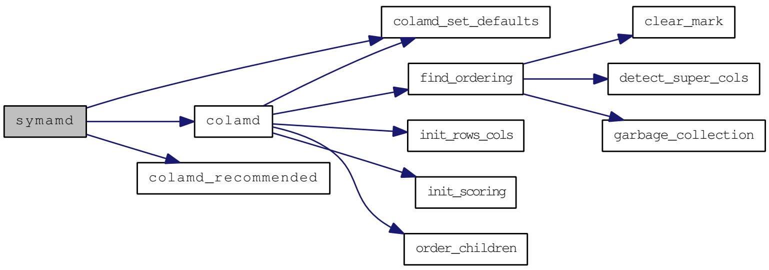
Here is the call graph for this function:



4.40.4.4 void colamd_set_defaults (double *knobs*[COLAMD_KNOBS])

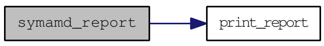
4.40.4.5 int symamd (int *n*, int *A*[ ], int *p*[ ], int *perm*[ ], double *knobs*[COLAMD_KNOBS], int *stats*[COLAMD_STATS], void ∗(∗)(size_t, size_t) *allocate*, void(∗)(void ∗) *release*)

Here is the call graph for this function:



4.40.4.6 void symamd_report (int *stats*[COLAMD_STATS])

Here is the call graph for this function:



4.41 SRC/cpanel_bmod.c File Reference

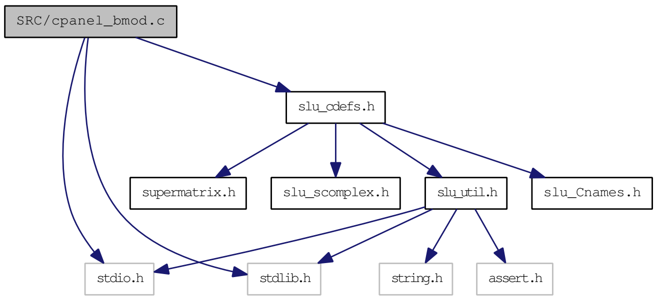
Performs numeric block updates.

`#include <stdio.h>`

`#include <stdlib.h>`

`#include "slu_cdefs.h"`

Include dependency graph for cpanel_bmod.c:



Functions

- void clsolve (int, int, complex ∗, complex ∗)

  Solves a dense UNIT lower triangular system.

- void cmatvec (int, int, int, complex ∗, complex ∗, complex ∗)

  Performs a dense matrix-vector multiply: Mxvec = Mxvec + M ∗ vec.

- void ccheck_tempv ()
- void cpanel_bmod (const int m, const int w, const int jcol, const int nseg, complex ∗dense, complex ∗tempv, int ∗segrep, int ∗repfnz, GlobalLU_t ∗Glu, SuperLUStat_t ∗stat)

4.41.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.41.2 Function Documentation

4.41.2.1 void ccheck_tempv ()

4.41.2.2 void clsolve (int *ldm*, int *ncol*, complex ∗ *M*, complex ∗ *rhs*)

The unit lower triangular matrix is stored in a 2D array M(1:nrow,1:ncol). The solution will be returned in the rhs vector.

4.41.2.3 void cmatvec (int *ldm*, int *nrow*, int *ncol*, complex ∗ *M*, complex ∗ *vec*, complex ∗ *Mxvec*)

The input matrix is M(1:nrow,1:ncol); The product is returned in Mxvec[].

4.41.2.4 void cpanel_bmod (const int *m*, const int *w*, const int *jcol*, const int *nseg*, complex ∗ *dense*, complex ∗ *tempv*, int ∗ *segrep*, int ∗ *repfnz*, GlobalLU_t ∗ *Glu*, SuperLUStat_t ∗ *stat*)

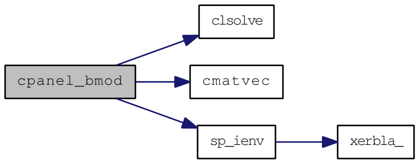
```
 Purpose
 =======

    Performs numeric block updates (sup-panel) in topological order.
    It features: col-col, 2cols-col, 3cols-col, and sup-col updates.
    Special processing on the supernodal portion of L[*,j]

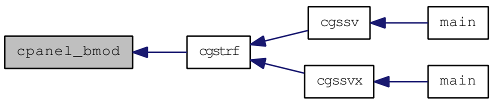
    Before entering this routine, the original nonzeros in the panel
    were already copied into the spa[m,w].

    Updated/Output parameters-
    dense[0:m-1,w]: L[*,j:j+w-1] and U[*,j:j+w-1] are returned
    collectively in the m-by-w vector dense[*].
```

Here is the call graph for this function:



Here is the caller graph for this function:

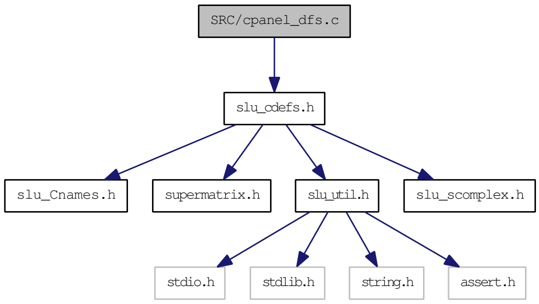


4.42 SRC/cpanel_dfs.c File Reference

Peforms a symbolic factorization on a panel of symbols.

`#include "slu_cdefs.h"`

Include dependency graph for cpanel_dfs.c:



Functions

- void cpanel_dfs (const int m, const int w, const int jcol, SuperMatrix ∗A, int ∗perm_r, int ∗nseg, complex ∗dense, int ∗panel_lsub, int ∗segrep, int ∗repfnz, int ∗xprune, int ∗marker, int ∗parent, int ∗xplore, GlobalLU_t ∗Glu)

4.42.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.42.2 Function Documentation

4.42.2.1 void cpanel_dfs (const int *m*, const int *w*, const int *jcol*, SuperMatrix ∗ *A*, int ∗ *perm_r*, int ∗ *nseg*, complex ∗ *dense*, int ∗ *panel_lsub*, int ∗ *segrep*, int ∗ *repfnz*, int ∗ *xprune*, int ∗ *marker*, int ∗ *parent*, int ∗ *xplore*, GlobalLU_t ∗ *Glu*)

```
Purpose
=======
```

```
Performs a symbolic factorization on a panel of columns [jcol, jcol+w).

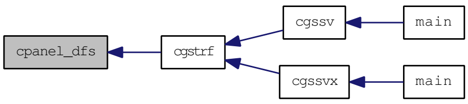
A supernode representative is the last column of a supernode.
The nonzeros in U[*,j] are segments that end at supernodal
representatives.

The routine returns one list of the supernodal representatives
in topological order of the dfs that generates them. This list is
a superset of the topological order of each individual column within
the panel.
The location of the first nonzero in each supernodal segment
(supernodal entry location) is also returned. Each column has a
separate list for this purpose.

Two marker arrays are used for dfs:
  marker[i] == jj, if i was visited during dfs of current column jj;
  marker1[i] >= jcol, if i was visited by earlier columns in this panel;

marker: A-row --> A-row/col (0/1)
repfnz: SuperA-col --> PA-row
parent: SuperA-col --> SuperA-col
xplore: SuperA-col --> index to L-structure
```

Here is the caller graph for this function:



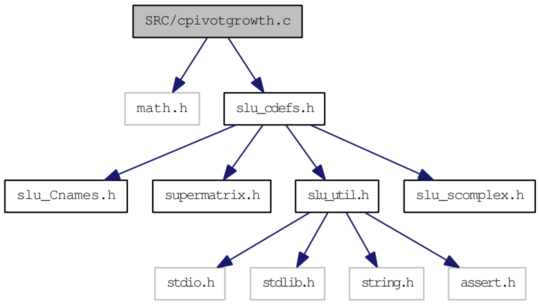
4.43 SRC/cpivotgrowth.c File Reference

Computes the reciprocal pivot growth factor.

`#include <math.h>`

`#include "slu_cdefs.h"`

Include dependency graph for cpivotgrowth.c:



Functions

- float cPivotGrowth (int ncols, SuperMatrix ∗A, int ∗perm_c, SuperMatrix ∗L, SuperMatrix ∗U)

4.43.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997
```

4.43.2 Function Documentation

4.43.2.1 float cPivotGrowth (int *ncols*, SuperMatrix ∗ *A*, int ∗ *perm_c*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*)

```
Purpose
=======

Compute the reciprocal pivot growth factor of the leading ncols columns
of the matrix, using the formula:
    min_j ( max_i(abs(A_ij)) / max_i(abs(U_ij)) )

Arguments
=========

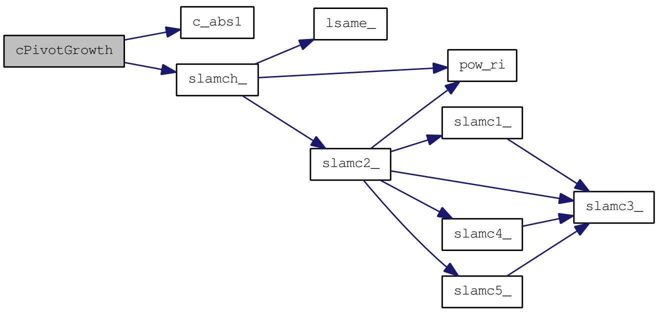
ncols    (input) int
         The number of columns of matrices A, L and U.
```

```
 A         (input) SuperMatrix*
    Original matrix A, permuted by columns, of dimension
          (A->nrow, A->ncol). The type of A can be:
          Stype = NC; Dtype = SLU_C; Mtype = GE.

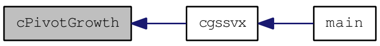
 L         (output) SuperMatrix*
          The factor L from the factorization Pr*A=L*U; use compressed row
          subscripts storage for supernodes, i.e., L has type:
          Stype = SC; Dtype = SLU_C; Mtype = TRLU.

 U         (output) SuperMatrix*
    The factor U from the factorization Pr*A*Pc=L*U. Use column-wise
          storage scheme, i.e., U has types: Stype = NC;
          Dtype = SLU_C; Mtype = TRU.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.44 SRC/cpivotL.c File Reference

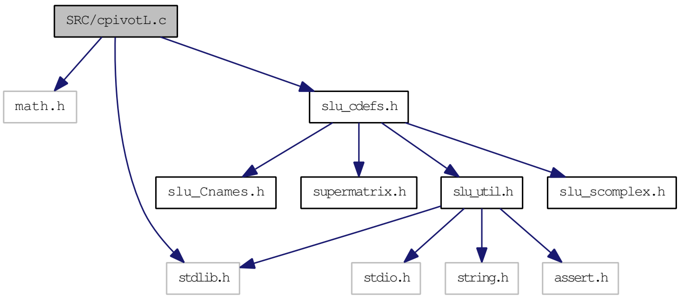
Performs numerical pivoting.

`#include <math.h>`

`#include <stdlib.h>`

`#include "slu_cdefs.h"`

Include dependency graph for cpivotL.c:



Functions

- int cpivotL (const int jcol, const float u, int *usepr, int *perm_r, int *iperm_r, int *iperm_c, int *pivrow, GlobalLU_t *Glu, SuperLUStat_t *stat)

4.44.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.44.2 Function Documentation

4.44.2.1 int cpivotL (const int *jcol*, const float *u*, int * *usepr*, int * *perm_r*, int * *iperm_r*, int * *iperm_c*, int * *pivrow*, GlobalLU_t * *Glu*, SuperLUStat_t * *stat*)

```
Purpose
```

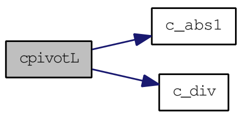
```
=======
  Performs the numerical pivoting on the current column of L,
  and the CDIV operation.

  Pivot policy:
  (1) Compute thresh = u * max_(i>=j) abs(A_ij);
  (2) IF user specifies pivot row k and abs(A_kj) >= thresh THEN
          pivot row = k;
      ELSE IF abs(A_jj) >= thresh THEN
          pivot row = j;
      ELSE
          pivot row = m;

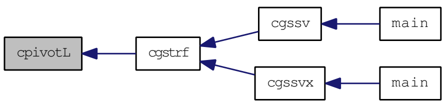
  Note: If you absolutely want to use a given pivot order, then set u=0.0.

  Return value: 0       success;
                i > 0   U(i,i) is exactly zero.
```

Here is the call graph for this function:



Here is the caller graph for this function:

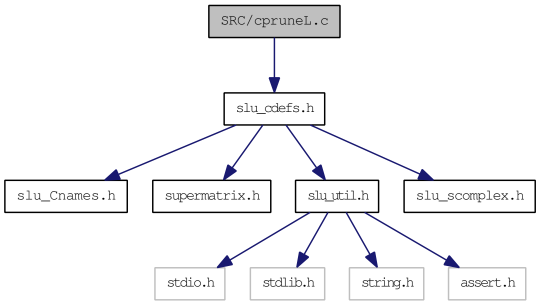


4.45 SRC/cpruneL.c File Reference

Prunes the L-structure.

`#include "slu_cdefs.h"`

Include dependency graph for cpruneL.c:



Functions

- void cpruneL (const int jcol, const int ∗perm_r, const int pivrow, const int nseg, const int ∗segrep, const int ∗repfnz, int ∗xprune, GlobalLU_t ∗Glu)

4.45.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997


Copyright (c) 1994 by Xerox Corporation.  All rights reserved.


THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.


Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
*
```

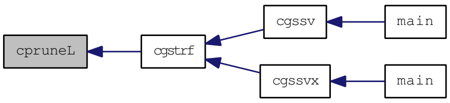
4.45.2 Function Documentation

4.45.2.1 void cpruneL (const int *jcol*, const int ∗ *perm_r*, const int *pivrow*, const int *nseg*, const int ∗ *segrep*, const int ∗ *repfnz*, int ∗ *xprune*, GlobalLU_t ∗ *Glu*)

```
Purpose
=======
  Prunes the L-structure of supernodes whose L-structure
```

```
contains the current pivot row "pivrow"
```

Here is the caller graph for this function:

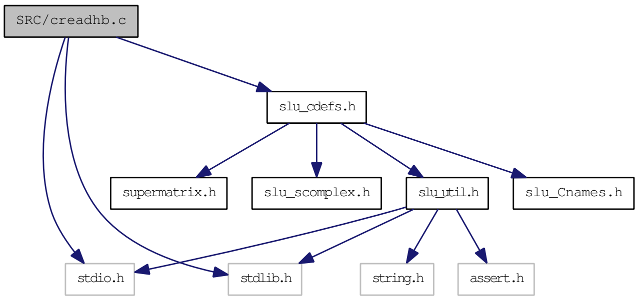


4.46 SRC/creadhb.c File Reference

Read a matrix stored in Harwell-Boeing format.

```
#include <stdio.h>
#include <stdlib.h>
#include "slu_cdefs.h"
```

Include dependency graph for creadhb.c:



Functions

- int cDumpLine (FILE ∗fp)

  Eat up the rest of the current line.

- int cParseIntFormat (char ∗buf, int ∗num, int ∗size)
- int cParseFloatFormat (char ∗buf, int ∗num, int ∗size)
- int cReadVector (FILE ∗fp, int n, int ∗where, int perline, int persize)
- int cReadValues (FILE ∗fp, int n, complex ∗destination, int perline, int persize)

  Read complex numbers as pairs of (real, imaginary).

- void creadhb (int ∗nrow, int ∗ncol, int ∗nonz, complex ∗∗nzval, int ∗∗rowind, int ∗∗colptr)

  Auxiliary routines.

4.46.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997


Purpose
=======


Read a COMPLEX PRECISION matrix stored in Harwell-Boeing format
as described below.
```

```
 Line 1 (A72,A8)
   Col. 1 - 72   Title (TITLE)
Col. 73 - 80  Key (KEY)

 Line 2 (5I14)
  Col. 1 - 14   Total number of lines excluding header (TOTCRD)
  Col. 15 - 28  Number of lines for pointers (PTRCRD)
  Col. 29 - 42  Number of lines for row (or variable) indices (INDCRD)
  Col. 43 - 56  Number of lines for numerical values (VALCRD)
Col. 57 - 70  Number of lines for right-hand sides (RHSCRD)
                    (including starting guesses and solution vectors
       if present)
                  (zero indicates no right-hand side data is present)

 Line 3 (A3, 11X, 4I14)
    Col. 1 - 3    Matrix type (see below) (MXTYPE)
  Col. 15 - 28  Number of rows (or variables) (NROW)
  Col. 29 - 42  Number of columns (or elements) (NCOL)
Col. 43 - 56  Number of row (or variable) indices (NNZERO)
              (equal to number of entries for assembled matrices)
  Col. 57 - 70  Number of elemental matrix entries (NELTVL)
              (zero in the case of assembled matrices)
 Line 4 (2A16, 2A20)
  Col. 1 - 16   Format for pointers (PTRFMT)
Col. 17 - 32  Format for row (or variable) indices (INDFMT)
Col. 33 - 52  Format for numerical values of coefficient matrix (VALFMT)
  Col. 53 - 72 Format for numerical values of right-hand sides (RHSFMT)

 Line 5 (A3, 11X, 2I14) Only present if there are right-hand sides present
     Col. 1        Right-hand side type:
           F for full storage or M for same format as matrix
     Col. 2        G if a starting vector(s) (Guess) is supplied. (RHSTYP)
     Col. 3        X if an exact solution vector(s) is supplied.
Col. 15 - 28  Number of right-hand sides (NRHS)
Col. 29 - 42  Number of row indices (NRHSIX)
                 (ignored in case of unassembled matrices)

 The three character type field on line 3 describes the matrix type.
 The following table lists the permitted values for each of the three
 characters. As an example of the type field, RSA denotes that the matrix
 is real, symmetric, and assembled.

 First Character:
R Real matrix
C Complex matrix
P Pattern only (no numerical values supplied)

 Second Character:
S Symmetric
U Unsymmetric
H Hermitian
Z Skew symmetric
R Rectangular

 Third Character:
A Assembled
E Elemental matrices (unassembled)
```

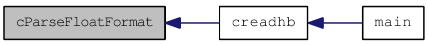
4.46.2 Function Documentation

4.46.2.1 int cDumpLine (FILE * *fp*)

Here is the caller graph for this function:

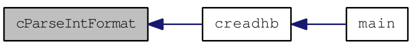
4.46.2.2 int cParseFloatFormat (char * *buf*, int * *num*, int * *size*)

Here is the caller graph for this function:



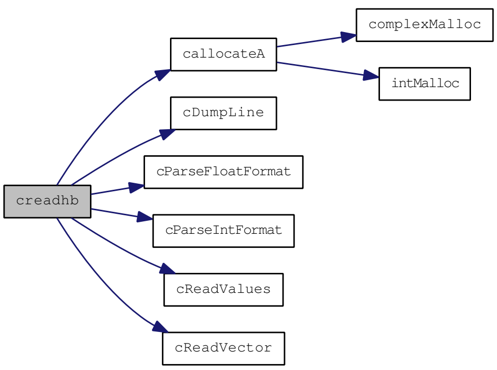
4.46.2.3 int cParseIntFormat (char * *buf*, int * *num*, int * *size*)

Here is the caller graph for this function:

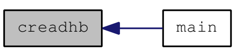


4.46.2.4 void creadhb (int * *nrow*, int * *ncol*, int * *nonz*, complex ** *nzval*, int ** *rowind*, int ** *colptr*)

Here is the call graph for this function:

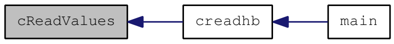


Here is the caller graph for this function:



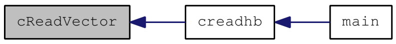
4.46.2.5 int cReadValues (FILE ∗ *fp*, int *n*, complex ∗ *destination*, int *perline*, int *persize*)

Here is the caller graph for this function:



4.46.2.6 int cReadVector (FILE ∗ *fp*, int *n*, int ∗ *where*, int *perline*, int *persize*)

Here is the caller graph for this function:

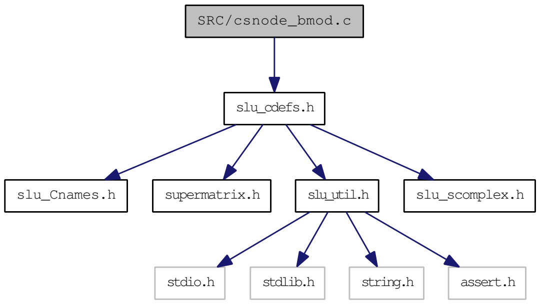


4.47 SRC/csnode_bmod.c File Reference

Performs numeric block updates within the relaxed snode.

`#include "slu_cdefs.h"`

Include dependency graph for csnode_bmod.c:



Functions

- int csnode_bmod (const int jcol, const int jsupno, const int fsupc, complex ∗dense, complex ∗tempv, GlobalLU_t ∗Glu, SuperLUStat_t ∗stat)

  Performs numeric block updates within the relaxed snode.

4.47.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003


Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

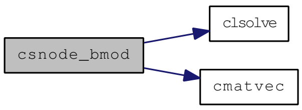

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.


Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

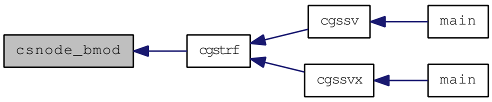
4.47.2 Function Documentation

4.47.2.1 int csnode_bmod (const int *jcol*, const int *jsupno*, const int *fsupc*, complex ∗ *dense*, complex ∗ *tempv*, GlobalLU_t ∗ *Glu*, SuperLUStat_t ∗ *stat*)

Here is the call graph for this function:



Here is the caller graph for this function:

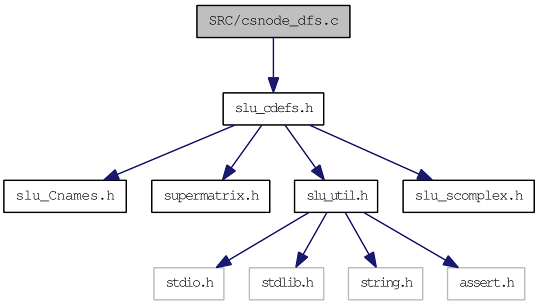


4.48 SRC/csnode_dfs.c File Reference

Determines the union of row structures of columns within the relaxed node.

`#include "slu_cdefs.h"`

Include dependency graph for csnode_dfs.c:



Functions

- int csnode_dfs (const int jcol, const int kcol, const int ∗asub, const int ∗xa_begin, const int ∗xa_end, int ∗xprune, int ∗marker, GlobalLU_t ∗Glu)

4.48.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.48.2 Function Documentation

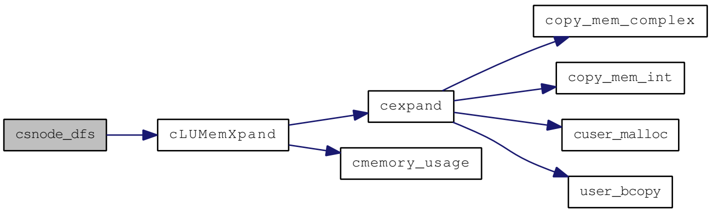
4.48.2.1 int csnode_dfs (const int *jcol*, const int *kcol*, const int ∗ *asub*, const int ∗ *xa_begin*, const int ∗ *xa_end*, int ∗ *xprune*, int ∗ *marker*, GlobalLU_t ∗ *Glu*)

```
Purpose
=======
   csnode_dfs() - Determine the union of the row structures of those
```

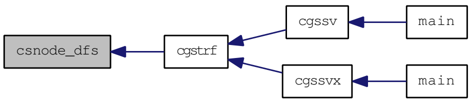
```
    columns within the relaxed snode.
    Note: The relaxed snodes are leaves of the supernodal etree, therefore,
    the portion outside the rectangular supernode must be zero.

 Return value
 ============
     0   success;
    >0   number of bytes allocated when run out of memory.
```

Here is the call graph for this function:



Here is the caller graph for this function:

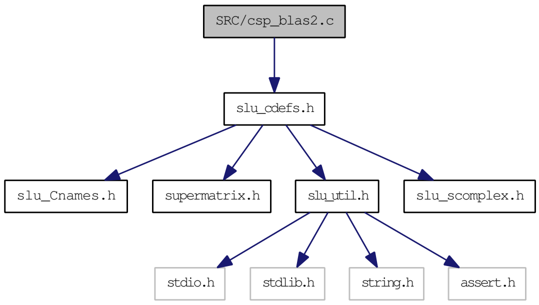


4.49 SRC/csp_blas2.c File Reference

Sparse BLAS 2, using some dense BLAS 2 operations.

`#include "slu_cdefs.h"`

Include dependency graph for csp_blas2.c:



Functions

- void cusolve (int, int, complex ∗, complex ∗)

  Solves a dense upper triangular system.

- void clsolve (int, int, complex ∗, complex ∗)

  Solves a dense UNIT lower triangular system.

- void cmatvec (int, int, int, complex ∗, complex ∗, complex ∗)

  Performs a dense matrix-vector multiply: Mxvec = Mxvec + M ∗ vec.

- int sp_ctrsv (char ∗uplo, char ∗trans, char ∗diag, SuperMatrix ∗L, SuperMatrix ∗U, complex ∗x, SuperLUStat_t ∗stat, int ∗info)

  Solves one of the systems of equations A∗x = b, or A'∗x = b.

- int sp_cgemv (char ∗trans, complex alpha, SuperMatrix ∗A, complex ∗x, int incx, complex beta, complex ∗y, int incy)

  Performs one of the matrix-vector operations y := alpha∗A∗x + beta∗y, or y := alpha∗A'∗x + beta∗y.

4.49.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003
```

4.49.2 Function Documentation

4.49.2.1 void clsolve (int *ldm*, int *ncol*, complex ∗ *M*, complex ∗ *rhs*)

The unit lower triangular matrix is stored in a 2D array M(1:nrow,1:ncol). The solution will be returned in the rhs vector.

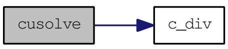
4.49.2.2 void cmatvec (int *ldm*, int *nrow*, int *ncol*, complex ∗ *M*, complex ∗ *vec*, complex ∗ *Mxvec*)

The input matrix is M(1:nrow,1:ncol); The product is returned in Mxvec[].

4.49.2.3 void cusolve (int *ldm*, int *ncol*, complex ∗ *M*, complex ∗ *rhs*)

The upper triangular matrix is stored in a 2-dim array M(1:ldm,1:ncol). The solution will be returned in the rhs vector.

Here is the call graph for this function:



4.49.2.4 int sp_cgemv (char ∗ *trans*, complex *alpha*, SuperMatrix ∗ *A*, complex ∗ *x*, int *incx*, complex *beta*, complex ∗ *y*, int *incy*)

```
Purpose
=======

sp_cgemv()  performs one of the matrix-vector operations
   y := alpha*A*x + beta*y,   or   y := alpha*A'*x + beta*y,
where alpha and beta are scalars, x and y are vectors and A is a
sparse A->nrow by A->ncol matrix.

Parameters
==========

TRANS  - (input) char*
         On entry, TRANS specifies the operation to be performed as
         follows:
            TRANS = 'N' or 'n'   y := alpha*A*x + beta*y.
            TRANS = 'T' or 't'   y := alpha*A'*x + beta*y.
            TRANS = 'C' or 'c'   y := alpha*A'*x + beta*y.

ALPHA  - (input) complex
         On entry, ALPHA specifies the scalar alpha.

A      - (input) SuperMatrix*
         Before entry, the leading m by n part of the array A must
         contain the matrix of coefficients.

X      - (input) complex*, array of DIMENSION at least
         ( 1 + ( n - 1 )*abs( INCX ) ) when TRANS = 'N' or 'n'
```

```
           and at least
            ( 1 + ( m - 1 )*abs( INCX ) ) otherwise.
            Before entry, the incremented array X must contain the
            vector x.

   INCX    - (input) int
            On entry, INCX specifies the increment for the elements of
            X. INCX must not be zero.

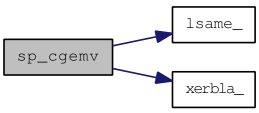
   BETA    - (input) complex
            On entry, BETA specifies the scalar beta. When BETA is
            supplied as zero then Y need not be set on input.

   Y       - (output) complex*,  array of DIMENSION at least
            ( 1 + ( m - 1 )*abs( INCY ) ) when TRANS = 'N' or 'n'
            and at least
            ( 1 + ( n - 1 )*abs( INCY ) ) otherwise.
            Before entry with BETA non-zero, the incremented array Y
            must contain the vector y. On exit, Y is overwritten by the
            updated vector y.

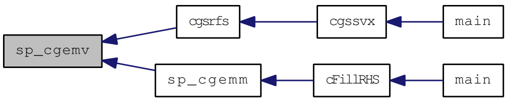
   INCY    - (input) int
            On entry, INCY specifies the increment for the elements of
            Y. INCY must not be zero.

    ==== Sparse Level 2 Blas routine.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.49.2.5 int sp_ctrsv (char ∗ *uplo*, char ∗ *trans*, char ∗ *diag*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, complex ∗ *x*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
   Purpose
   =======

   sp_ctrsv() solves one of the systems of equations
       A*x = b,    or    A'*x = b,
   where b and x are n element vectors and A is a sparse unit , or
   non-unit, upper or lower triangular matrix.
   No test for singularity or near-singularity is included in this
   routine. Such tests must be performed before calling this routine.
```

```
Parameters
==========

uplo   - (input) char*
         On entry, uplo specifies whether the matrix is an upper or
          lower triangular matrix as follows:
             uplo = 'U' or 'u'   A is an upper triangular matrix.
             uplo = 'L' or 'l'   A is a lower triangular matrix.

trans  - (input) char*
          On entry, trans specifies the equations to be solved as
          follows:
             trans = 'N' or 'n'   A*x = b.
             trans = 'T' or 't'   A'*x = b.
             trans = 'C' or 'c'   A^H*x = b.

diag   - (input) char*
          On entry, diag specifies whether or not A is unit
          triangular as follows:
             diag = 'U' or 'u'   A is assumed to be unit triangular.
             diag = 'N' or 'n'   A is not assumed to be unit
                                 triangular.

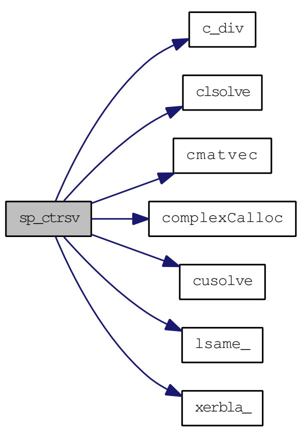
L       - (input) SuperMatrix*
    The factor L from the factorization Pr*A*Pc=L*U. Use
          compressed row subscripts storage for supernodes,
          i.e., L has types: Stype = SC, Dtype = SLU_C, Mtype = TRLU.

U       - (input) SuperMatrix*
     The factor U from the factorization Pr*A*Pc=L*U.
     U has types: Stype = NC, Dtype = SLU_C, Mtype = TRU.

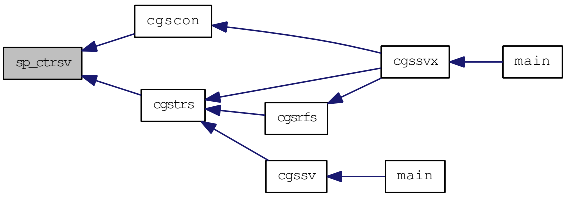
x       - (input/output) complex*
          Before entry, the incremented array X must contain the n
          element right-hand side vector b. On exit, X is overwritten
          with the solution vector x.

info    - (output) int*
          If *info = -i, the i-th argument had an illegal value.
```

Here is the call graph for this function:



Here is the caller graph for this function:

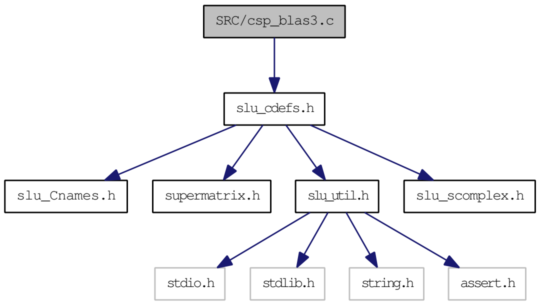


4.50 SRC/csp_blas3.c File Reference

Sparse BLAS3, using some dense BLAS3 operations.

`#include "slu_cdefs.h"`

Include dependency graph for csp_blas3.c:



Functions

- int sp_cgemm (char ∗transa, char ∗transb, int m, int n, int k, complex alpha, SuperMatrix ∗A, complex ∗b, int ldb, complex beta, complex ∗c, int ldc)

4.50.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997
```

4.50.2 Function Documentation

4.50.2.1 int sp_cgemm (char ∗ *transa*, char ∗ *transb*, int *m*, int *n*, int *k*, complex *alpha*, SuperMatrix ∗ *A*, complex ∗ *b*, int *ldb*, complex *beta*, complex ∗ *c*, int *ldc*)

```
Purpose
  =======

  sp_c performs one of the matrix-matrix operations

     C := alpha*op( A )*op( B ) + beta*C,

  where  op( X ) is one of

     op( X ) = X   or   op( X ) = X'   or   op( X ) = conjg( X' ),

  alpha and beta are scalars, and A, B and C are matrices, with op( A )
  an m by k matrix,  op( B )  a  k by n matrix and  C an m by n matrix.
```

```
Parameters
==========

TRANSA - (input) char*
         On entry, TRANSA specifies the form of op( A ) to be used in
         the matrix multiplication as follows:
            TRANSA = 'N' or 'n',  op( A ) = A.
            TRANSA = 'T' or 't',  op( A ) = A'.
            TRANSA = 'C' or 'c',  op( A ) = conjg( A' ).
         Unchanged on exit.

TRANSB - (input) char*
         On entry, TRANSB specifies the form of op( B ) to be used in
         the matrix multiplication as follows:
            TRANSB = 'N' or 'n',  op( B ) = B.
            TRANSB = 'T' or 't',  op( B ) = B'.
            TRANSB = 'C' or 'c',  op( B ) = conjg( B' ).
         Unchanged on exit.

M      - (input) int
         On entry,  M  specifies  the number of rows of the matrix
  op( A ) and of the matrix C.  M must be at least zero.
  Unchanged on exit.

N      - (input) int
         On entry,  N specifies the number of columns of the matrix
  op( B ) and the number of columns of the matrix C. N must be
  at least zero.
  Unchanged on exit.

K      - (input) int
         On entry, K specifies the number of columns of the matrix
  op( A ) and the number of rows of the matrix op( B ). K must
  be at least  zero.
        Unchanged on exit.

ALPHA  - (input) complex
         On entry, ALPHA specifies the scalar alpha.

A      - (input) SuperMatrix*
         Matrix A with a sparse format, of dimension (A->nrow, A->ncol).
         Currently, the type of A can be:
             Stype = NC or NCP; Dtype = SLU_C; Mtype = GE.
         In the future, more general A can be handled.

B      - COMPLEX PRECISION array of DIMENSION ( LDB, kb ), where kb is
         n when TRANSB = 'N' or 'n',  and is  k otherwise.
         Before entry with  TRANSB = 'N' or 'n',  the leading k by n
         part of the array B must contain the matrix B, otherwise
         the leading n by k part of the array B must contain the
         matrix B.
         Unchanged on exit.

LDB    - (input) int
         On entry, LDB specifies the first dimension of B as declared
         in the calling (sub) program. LDB must be at least max( 1, n ).
         Unchanged on exit.
```

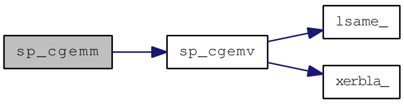
```
BETA    - (input) complex
          On entry, BETA specifies the scalar beta. When BETA is
          supplied as zero then C need not be set on input.

C       - COMPLEX PRECISION array of DIMENSION ( LDC, n ).
          Before entry, the leading m by n part of the array C must
          contain the matrix C,  except when beta is zero, in which
          case C need not be set on entry.
          On exit, the array C is overwritten by the m by n matrix
  ( alpha*op( A )*B + beta*C ).

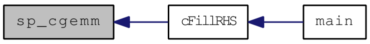
LDC     - (input) int
          On entry, LDC specifies the first dimension of C as declared
          in the calling (sub)program. LDC must be at least max(1,m).
          Unchanged on exit.

==== Sparse Level 3 Blas routine.
```

Here is the call graph for this function:



Here is the caller graph for this function:



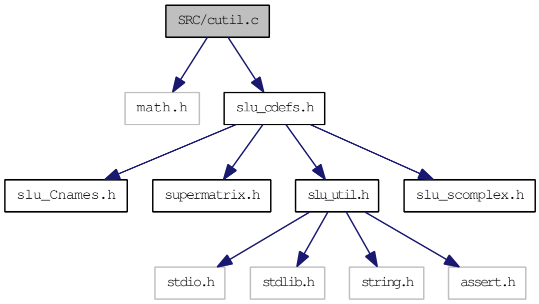
4.51 SRC/cutil.c File Reference

Matrix utility functions.

`#include <math.h>`

`#include "slu_cdefs.h"`

Include dependency graph for cutil.c:



Functions

- void cCreate_CompCol_Matrix (SuperMatrix ∗A, int m, int n, int nnz, complex ∗nzval, int ∗rowind, int ∗colptr, Stype_t stype, Dtype_t dtype, Mtype_t mtype)

  Supernodal LU factor related.

- void cCreate_CompRow_Matrix (SuperMatrix ∗A, int m, int n, int nnz, complex ∗nzval, int ∗colind, int ∗rowptr, Stype_t stype, Dtype_t dtype, Mtype_t mtype)
- void cCopy_CompCol_Matrix (SuperMatrix ∗A, SuperMatrix ∗B)

  Copy matrix A into matrix B.

- void cCreate_Dense_Matrix (SuperMatrix ∗X, int m, int n, complex ∗x, int ldx, Stype_t stype, Dtype_t dtype, Mtype_t mtype)
- void cCopy_Dense_Matrix (int M, int N, complex ∗X, int ldx, complex ∗Y, int ldy)
- void cCreate_SuperNode_Matrix (SuperMatrix ∗L, int m, int n, int nnz, complex ∗nzval, int ∗nzval_colptr, int ∗rowind, int ∗rowind_colptr, int ∗col_to_sup, int ∗sup_to_col, Stype_t stype, Dtype_t dtype, Mtype_t mtype)
- void cCompRow_to_CompCol (int m, int n, int nnz, complex ∗a, int ∗colind, int ∗rowptr, complex ∗∗at, int ∗∗rowind, int ∗∗colptr)

  Convert a row compressed storage into a column compressed storage.

- void cPrint_CompCol_Matrix (char ∗what, SuperMatrix ∗A)

  Routines for debugging.

- void cPrint_SuperNode_Matrix (char ∗what, SuperMatrix ∗A)
- void cPrint_Dense_Matrix (char ∗what, SuperMatrix ∗A)
- void cprint_lu_col (char ∗msg, int jcol, int pivrow, int ∗xprune, GlobalLU_t ∗Glu)

  Diagnostic print of column "jcol" in the U/L factor.

- void ccheck_tempv (int n, complex ∗tempv)

Check whether tempv[] == 0. This should be true before and after calling any numeric routines, i.e., "panel_bmod" and "column_bmod".

- void cGenXtrue (int n, int nrhs, complex ∗x, int ldx)
- void cFillRHS (trans_t trans, int nrhs, complex ∗x, int ldx, SuperMatrix ∗A, SuperMatrix ∗B)

  Let rhs[i] = sum of i-th row of A, so the solution vector is all 1's.

- void cfill (complex ∗a, int alen, complex dval)

  Fills a complex precision array with a given value.

- void cinf_norm_error (int nrhs, SuperMatrix ∗X, complex ∗xtrue)

  Check the inf-norm of the error vector.

- void cPrintPerf (SuperMatrix ∗L, SuperMatrix ∗U, mem_usage_t ∗mem_usage, float rpg, float rcond, float ∗ferr, float ∗berr, char ∗equed, SuperLUStat_t ∗stat)

  Print performance of the code.

- print_complex_vec (char ∗what, int n, complex ∗vec)

4.51.1 Detailed Description

```
-- SuperLU routine (version 3.1) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
August 1, 2008

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

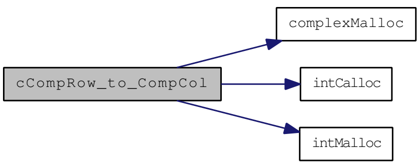
Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.51.2 Function Documentation

4.51.2.1 void ccheck_tempv (int *n*, complex ∗ *tempv*)

4.51.2.2 void cCompRow_to_CompCol (int *m*, int *n*, int *nnz*, complex ∗ *a*, int ∗ *colind*, int ∗ *rowptr*, complex ∗∗ *at*, int ∗∗ *rowind*, int ∗∗ *colptr*)

Here is the call graph for this function:



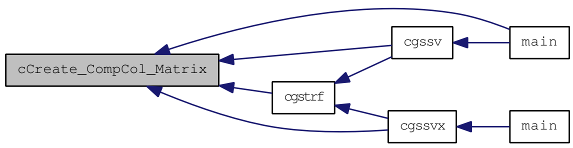
4.51.2.3 void cCopy_CompCol_Matrix (SuperMatrix ∗ *A*, SuperMatrix ∗ *B*)

4.51.2.4 void cCopy_Dense_Matrix (int *M*, int *N*, complex ∗ *X*, int *ldx*, complex ∗ *Y*, int *ldy*)

Copies a two-dimensional matrix X to another matrix Y.

4.51.2.5 void cCreate_CompCol_Matrix (SuperMatrix ∗ *A*, int *m*, int *n*, int *nnz*, complex ∗ *nzval*, int ∗ *rowind*, int ∗ *colptr*, Stype_t *stype*, Dtype_t *dtype*, Mtype_t *mtype*)

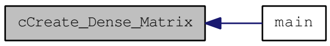
Here is the caller graph for this function:



4.51.2.6 void cCreate_CompRow_Matrix (SuperMatrix ∗ *A*, int *m*, int *n*, int *nnz*, complex ∗ *nzval*, int ∗ *colind*, int ∗ *rowptr*, Stype_t *stype*, Dtype_t *dtype*, Mtype_t *mtype*)

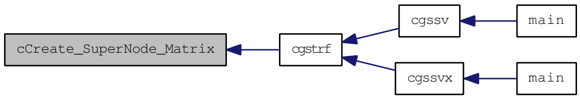
4.51.2.7 void cCreate_Dense_Matrix (SuperMatrix ∗ *X*, int *m*, int *n*, complex ∗ *x*, int *ldx*, Stype_t *stype*, Dtype_t *dtype*, Mtype_t *mtype*)

Here is the caller graph for this function:



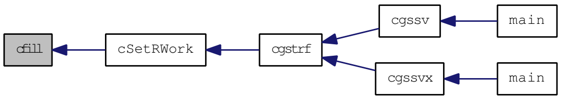
4.51.2.8 void cCreate_SuperNode_Matrix (SuperMatrix ∗ *L*, int *m*, int *n*, int *nnz*, complex ∗ *nzval*, int ∗ *nzval_colptr*, int ∗ *rowind*, int ∗ *rowind_colptr*, int ∗ *col_to_sup*, int ∗ *sup_to_col*, Stype_t *stype*, Dtype_t *dtype*, Mtype_t *mtype*)

Here is the caller graph for this function:



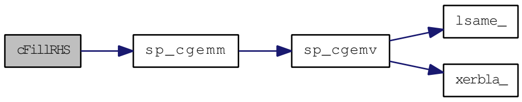
4.51.2.9 void cfill (complex ∗ *a*, int *alen*, complex *dval*)

Here is the caller graph for this function:

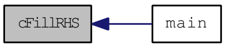


4.51.2.10 void cFillRHS (trans_t *trans*, int *nrhs*, complex ∗ *x*, int *ldx*, SuperMatrix ∗ *A*, SuperMatrix ∗ *B*)

Here is the call graph for this function:

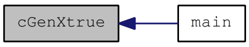


Here is the caller graph for this function:



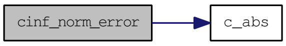
4.51.2.11 void cGenXtrue (int *n*, int *nrhs*, complex ∗ *x*, int *ldx*)

Here is the caller graph for this function:

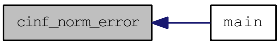


4.51.2.12 void cinf_norm_error (int *nrhs*, SuperMatrix ∗ *X*, complex ∗ *xtrue*)

Here is the call graph for this function:



Here is the caller graph for this function:

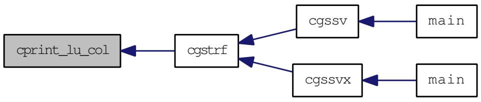


4.51.2.13 void cPrint_CompCol_Matrix (char ∗ *what*, SuperMatrix ∗ *A*)

4.51.2.14 void cPrint_Dense_Matrix (char ∗ *what*, SuperMatrix ∗ *A*)

4.51.2.15 void cprint_lu_col (char ∗ *msg*, int *jcol*, int *pivrow*, int ∗ *xprune*, GlobalLU_t ∗ *Glu*)

Here is the caller graph for this function:



4.51.2.16 void cPrint_SuperNode_Matrix (char * *what*, SuperMatrix * *A*)

4.51.2.17 void cPrintPerf (SuperMatrix * *L*, SuperMatrix * *U*, mem_usage_t * *mem_usage*, float *rpg*, float *rcond*, float * *ferr*, float * *berr*, char * *equed*, SuperLUStat_t * *stat*)

4.51.2.18 print_complex_vec (char * *what*, int *n*, complex * *vec*)

4.52 SRC/dcolumn_bmod.c File Reference

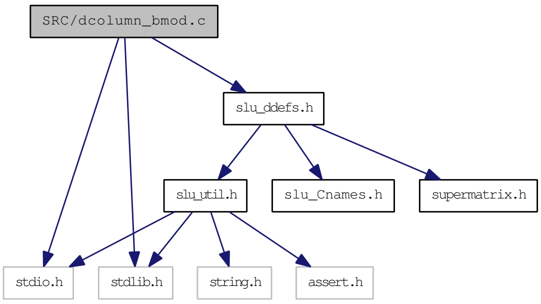
performs numeric block updates

#include <stdio.h>

#include <stdlib.h>

#include "slu_ddefs.h"

Include dependency graph for dcolumn_bmod.c:



Functions

- void dusolve (int, int, double ∗, double ∗)

  Solves a dense upper triangular system.

- void dlsolve (int, int, double ∗, double ∗)

  Solves a dense UNIT lower triangular system.

- void dmatvec (int, int, int, double ∗, double ∗, double ∗)

  Performs a dense matrix-vector multiply: Mxvec = Mxvec + M ∗ vec.

- int dcolumn_bmod (const int jcol, const int nseg, double ∗dense, double ∗tempv, int ∗segrep, int ∗repfnz, int fpanelc, GlobalLU_t ∗Glu, SuperLUStat_t ∗stat)

4.52.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

 Permission is hereby granted to use or copy this program for any
 purpose, provided the above notices are retained on all copies.
 Permission to modify the code and to distribute modified code is
```

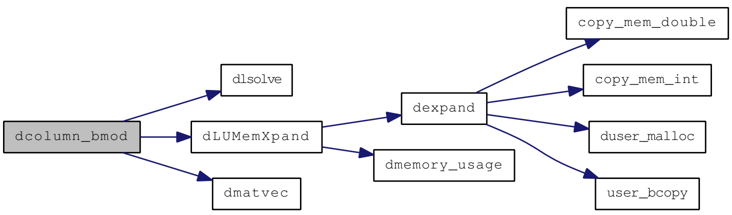
```
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.52.2 Function Documentation

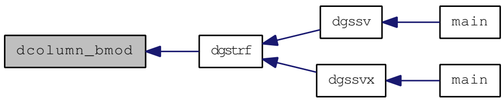
4.52.2.1 int dcolumn_bmod (const int *jcol*, const int *nseg*, double ∗ *dense*, double ∗ *tempv*, int ∗ *segrep*, int ∗ *repfnz*, int *fpanelc*, GlobalLU_t ∗ *Glu*, SuperLUStat_t ∗ *stat*)

```
Purpose:
========
Performs numeric block updates (sup-col) in topological order.
It features: col-col, 2cols-col, 3cols-col, and sup-col updates.
Special processing on the supernodal portion of L[*,j]
Return value:   0 - successful return
              > 0 - number of bytes allocated when run out of space
```

Here is the call graph for this function:



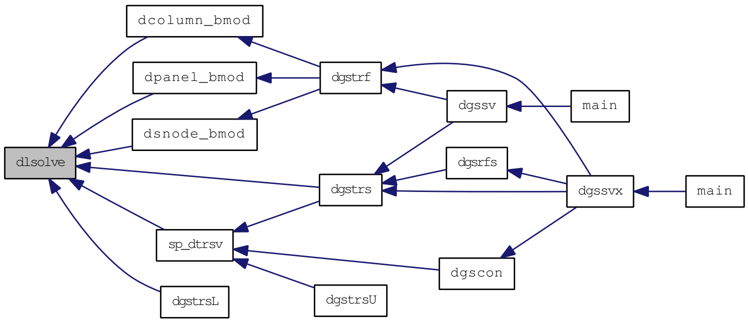
Here is the caller graph for this function:



4.52.2.2 void dlsolve (int *ldm*, int *ncol*, double ∗ *M*, double ∗ *rhs*)

The unit lower triangular matrix is stored in a 2D array M(1:nrow,1:ncol). The solution will be returned in the rhs vector.

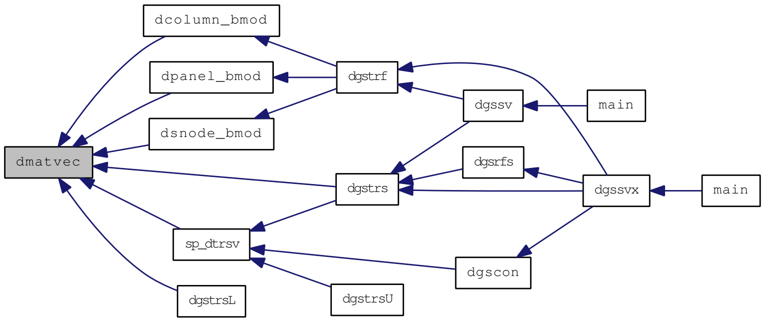
Here is the caller graph for this function:



4.52.2.3 void dmatvec (int *ldm*, int *nrow*, int *ncol*, double ∗ *M*, double ∗ *vec*, double ∗ *Mxvec*)

The input matrix is M(1:nrow,1:ncol); The product is returned in Mxvec[].

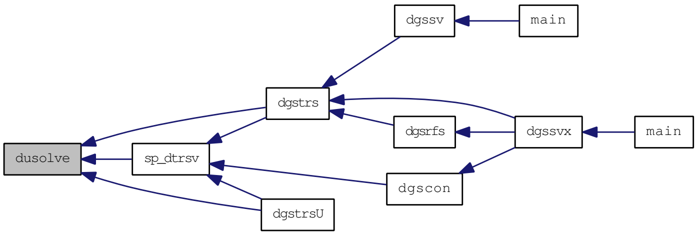
Here is the caller graph for this function:



4.52.2.4 void dusolve (int *ldm*, int *ncol*, double ∗ *M*, double ∗ *rhs*)

The upper triangular matrix is stored in a 2-dim array M(1:ldm,1:ncol). The solution will be returned in the rhs vector.

Here is the caller graph for this function:

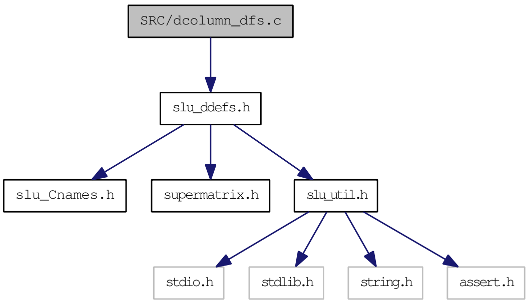


4.53 SRC/dcolumn_dfs.c File Reference

Performs a symbolic factorization.

`#include "slu_ddefs.h"`

Include dependency graph for dcolumn_dfs.c:



Defines

- #define T2_SUPER

  What type of supernodes we want.

Functions

- int dcolumn_dfs (const int m, const int jcol, int ∗perm_r, int ∗nseg, int ∗lsub_col, int ∗segrep, int ∗repfnz, int ∗xprune, int ∗marker, int ∗parent, int ∗xplore, GlobalLU_t ∗Glu)

4.53.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003


Copyright (c) 1994 by Xerox Corporation.  All rights reserved.


THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.


Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.53.2 Define Documentation

4.53.2.1 #define T2_SUPER

4.53.3 Function Documentation

4.53.3.1 int dcolumn_dfs (const int *m*, const int *jcol*, int ∗ *perm_r*, int ∗ *nseg*, int ∗ *lsub_col*, int ∗ *segrep*, int ∗ *repfnz*, int ∗ *xprune*, int ∗ *marker*, int ∗ *parent*, int ∗ *xplore*, GlobalLU_t ∗ *Glu*)

```
 Purpose
 =======
   "column_dfs" performs a symbolic factorization on column jcol, and
   decide the supernode boundary.


   This routine does not use numeric values, but only use the RHS
   row indices to start the dfs.

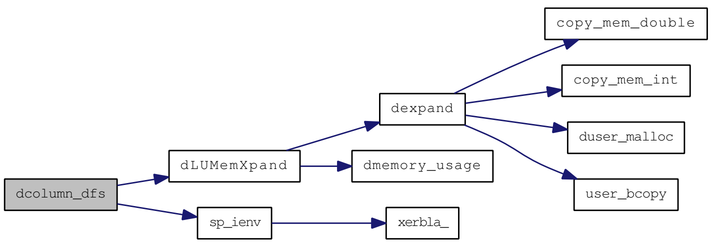

   A supernode representative is the last column of a supernode.
   The nonzeros in U[*,j] are segments that end at supernodal
   representatives. The routine returns a list of such supernodal
   representatives in topological order of the dfs that generates them.
   The location of the first nonzero in each such supernodal segment
   (supernodal entry location) is also returned.


 Local parameters
 ================
   nseg: no of segments in current U[*,j]
   jsuper: jsuper=EMPTY if column j does not belong to the same
supernode as j-1. Otherwise, jsuper=nsuper.

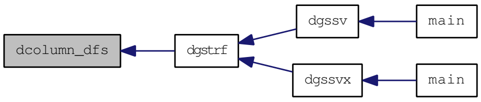

   marker2: A-row --> A-row/col (0/1)
   repfnz: SuperA-col --> PA-row
   parent: SuperA-col --> SuperA-col
   xplore: SuperA-col --> index to L-structure


 Return value
 ============
     0  success;
   > 0  number of bytes allocated when run out of space.
```

Here is the call graph for this function:



Here is the caller graph for this function:

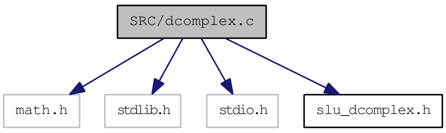


4.54 SRC/dcomplex.c File Reference

Common arithmetic for complex type.

```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include "slu_dcomplex.h"
```

Include dependency graph for dcomplex.c:



Functions

- void z_div (doublecomplex ∗c, doublecomplex ∗a, doublecomplex ∗b)

  Complex Division c = a/b.

- double z_abs (doublecomplex ∗z)

  Returns sqrt(z.r^2 + z.i^2).

- double z_abs1 (doublecomplex ∗z)

  Approximates the abs. Returns abs(z.r) + abs(z.i).

- void z_exp (doublecomplex ∗r, doublecomplex ∗z)

  Return the exponentiation.

- void d_cnjg (doublecomplex ∗r, doublecomplex ∗z)

  Return the complex conjugate.

- double d_imag (doublecomplex ∗z)

  Return the imaginary part.

4.54.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997


This file defines common arithmetic operations for complex type.
```

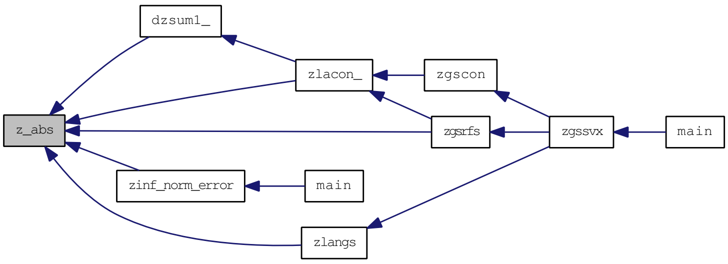
4.54.2 Function Documentation

4.54.2.1 void d_cnjg (doublecomplex ∗ *r*, doublecomplex ∗ *z*)

4.54.2.2 double d_imag (doublecomplex ∗ *z*)

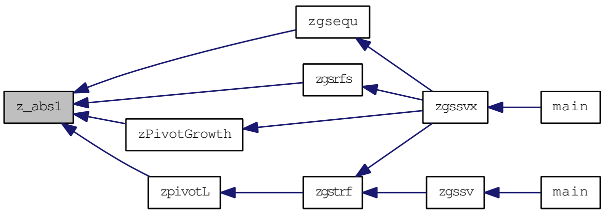
4.54.2.3 double z_abs (doublecomplex ∗ *z*)

Here is the caller graph for this function:



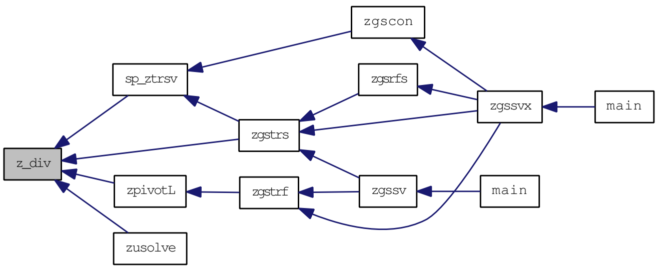
4.54.2.4 double z_abs1 (doublecomplex ∗ *z*)

Here is the caller graph for this function:



4.54.2.5 void z_div (doublecomplex ∗ *c*, doublecomplex ∗ *a*, doublecomplex ∗ *b*)

Here is the caller graph for this function:



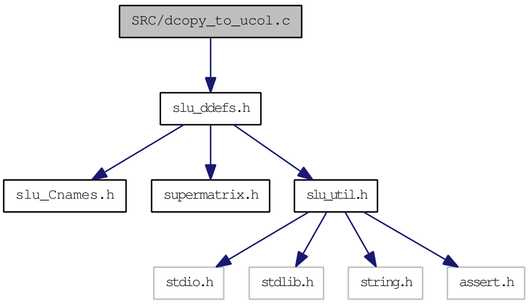
4.54.2.6 void z_exp (doublecomplex ∗ *r*, doublecomplex ∗ *z*)

4.55 SRC/dcopy_to_ucol.c File Reference

Copy a computed column of U to the compressed data structure.

`#include "slu_ddefs.h"`

Include dependency graph for dcopy_to_ucol.c:



Functions

- int dcopy_to_ucol (int jcol, int nseg, int *segrep, int *repfnz, int *perm_r, double *dense, GlobalLU_t *Glu)

4.55.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997
Copyright (c) 1994 by Xerox Corporation.  All rights reserved.
```

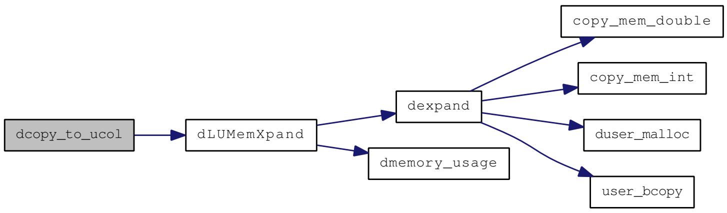
```
THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.
```

```
Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

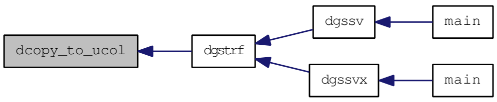
4.55.2 Function Documentation

4.55.2.1 int dcopy_to_ucol (int *jcol*, int *nseg*, int ∗ *segrep*, int ∗ *repfnz*, int ∗ *perm_r*, double ∗ *dense*, GlobalLU_t ∗ *Glu*)

Here is the call graph for this function:



Here is the caller graph for this function:

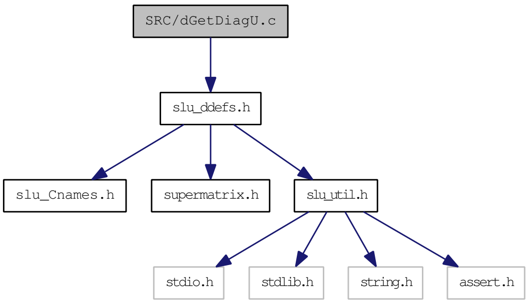


4.56 SRC/dGetDiagU.c File Reference

Extracts main diagonal of matrix.

`#include <slu_ddefs.h>`

Include dependency graph for dGetDiagU.c:



Functions

- void dGetDiagU (SuperMatrix *L, double *diagU)

4.56.1 Detailed Description

```
-- Auxiliary routine in SuperLU (version 2.0) --
Lawrence Berkeley National Lab, Univ. of California Berkeley.
Xiaoye S. Li
September 11, 2003

 Purpose
=======

GetDiagU extracts the main diagonal of matrix U of the LU factorization.

Arguments
=========

L       (input) SuperMatrix*
        The factor L from the factorization Pr*A*Pc=L*U as computed by
        dgstrf(). Use compressed row subscripts storage for supernodes,
        i.e., L has types: Stype = SLU_SC, Dtype = SLU_D, Mtype = SLU_TRLU.

diagU   (output) double*, dimension (n)
        The main diagonal of matrix U.

Note
====
The diagonal blocks of the L and U matrices are stored in the L
data structures.
```

4.56.2 Function Documentation

4.56.2.1 void dGetDiagU (SuperMatrix ∗ *L*, double ∗ *diagU*)

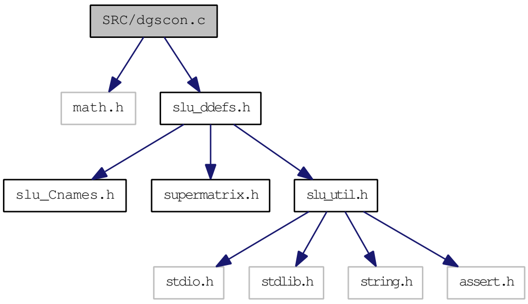
4.57 SRC/dgscon.c File Reference

Estimates reciprocal of the condition number of a general matrix.

`#include <math.h>`

`#include "slu_ddefs.h"`

Include dependency graph for dgscon.c:



Functions

- void dgscon (char ∗norm, SuperMatrix ∗L, SuperMatrix ∗U, double anorm, double ∗rcond, SuperLUStat_t ∗stat, int ∗info)

4.57.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003

Modified from lapack routines DGECON.
```

4.57.2 Function Documentation

4.57.2.1 void dgscon (char ∗ *norm*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, double *anorm*, double ∗ *rcond*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======

DGSCON estimates the reciprocal of the condition number of a general
real matrix A, in either the 1-norm or the infinity-norm, using
the LU factorization computed by DGETRF.   *

An estimate is obtained for norm(inv(A)), and the reciprocal of the
condition number is computed as
   RCOND = 1 / ( norm(A) * norm(inv(A)) ).
```

```
See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

 NORM    (input) char*
         Specifies whether the 1-norm condition number or the
         infinity-norm condition number is required:
         = '1' or 'O':  1-norm;
         = 'I':         Infinity-norm.

 L       (input) SuperMatrix*
         The factor L from the factorization Pr*A*Pc=L*U as computed by
         dgstrf(). Use compressed row subscripts storage for supernodes,
         i.e., L has types: Stype = SLU_SC, Dtype = SLU_D, Mtype = SLU_TRLU.

 U       (input) SuperMatrix*
         The factor U from the factorization Pr*A*Pc=L*U as computed by
         dgstrf(). Use column-wise storage scheme, i.e., U has types:
         Stype = SLU_NC, Dtype = SLU_D, Mtype = SLU_TRU.

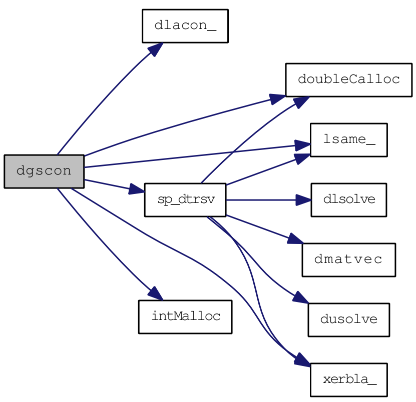
 ANORM   (input) double
         If NORM = '1' or 'O', the 1-norm of the original matrix A.
         If NORM = 'I', the infinity-norm of the original matrix A.

 RCOND   (output) double*
        The reciprocal of the condition number of the matrix A,
        computed as RCOND = 1/(norm(A) * norm(inv(A))).

 INFO    (output) int*
        = 0:  successful exit
        < 0:  if INFO = -i, the i-th argument had an illegal value

 ======================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:

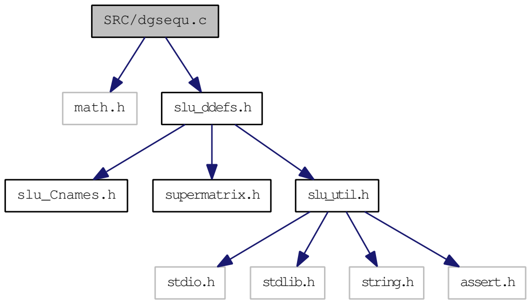
4.58 SRC/dgsequ.c File Reference

Computes row and column scalings.

`#include <math.h>`

`#include "slu_ddefs.h"`

Include dependency graph for dgsequ.c:



Functions

- void dgsequ (SuperMatrix *A, double *r, double *c, double *rowcnd, double *colcnd, double *amax, int *info)

  Driver related.

4.58.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997


Modified from LAPACK routine DGEEQU
```

4.58.2 Function Documentation

4.58.2.1 void dgsequ (SuperMatrix * *A*, double * *r*, double * *c*, double * *rowcnd*, double * *colcnd*, double * *amax*, int * *info*)

```
Purpose
  =======

  DGSEQU computes row and column scalings intended to equilibrate an
  M-by-N sparse matrix A and reduce its condition number. R returns the row
  scale factors and C the column scale factors, chosen to try to make
  the largest element in each row and column of the matrix B with
  elements B(i,j)=R(i)*A(i,j)*C(j) have absolute value 1.
```

```
R(i) and C(j) are restricted to be between SMLNUM = smallest safe
number and BIGNUM = largest safe number.  Use of these scaling
factors is not guaranteed to reduce the condition number of A but
works well in practice.


See supermatrix.h for the definition of 'SuperMatrix' structure.


Arguments
=========


A       (input) SuperMatrix*
        The matrix of dimension (A->nrow, A->ncol) whose equilibration
        factors are to be computed. The type of A can be:
        Stype = SLU_NC; Dtype = SLU_D; Mtype = SLU_GE.


R       (output) double*, size A->nrow
        If INFO = 0 or INFO > M, R contains the row scale factors
        for A.


C       (output) double*, size A->ncol
        If INFO = 0,  C contains the column scale factors for A.


ROWCND  (output) double*
        If INFO = 0 or INFO > M, ROWCND contains the ratio of the
        smallest R(i) to the largest R(i).  If ROWCND >= 0.1 and
        AMAX is neither too large nor too small, it is not worth
        scaling by R.

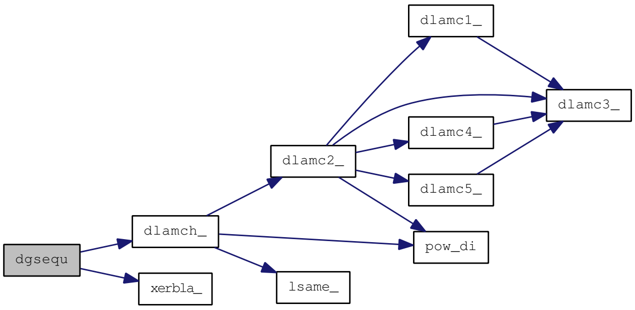

COLCND  (output) double*
        If INFO = 0, COLCND contains the ratio of the smallest
        C(i) to the largest C(i).  If COLCND >= 0.1, it is not
        worth scaling by C.


AMAX    (output) double*
        Absolute value of largest matrix element.  If AMAX is very
        close to overflow or very close to underflow, the matrix
        should be scaled.


INFO    (output) int*
        = 0:  successful exit
        < 0:  if INFO = -i, the i-th argument had an illegal value
        > 0:  if INFO = i,  and i is
              <= A->nrow:  the i-th row of A is exactly zero
              >  A->ncol:  the (i-M)-th column of A is exactly zero


=====================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:

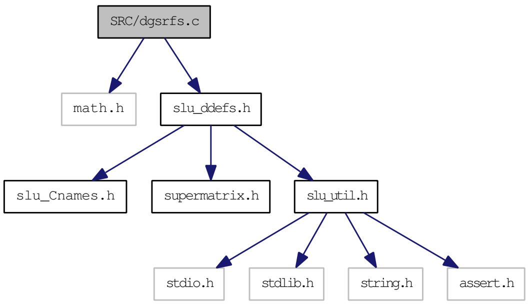
4.59 SRC/dgsrfs.c File Reference

Improves computed solution to a system of inear equations.

`#include <math.h>`

`#include "slu_ddefs.h"`

Include dependency graph for dgsrfs.c:



Defines

- #define ITMAX 5

Functions

- void dgsrfs (trans_t trans, SuperMatrix ∗A, SuperMatrix ∗L, SuperMatrix ∗U, int ∗perm_c, int ∗perm_r, char ∗equed, double ∗R, double ∗C, SuperMatrix ∗B, SuperMatrix ∗X, double ∗ferr, double ∗berr, SuperLUStat_t ∗stat, int ∗info)

4.59.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003
```

```
Modified from lapack routine DGERFS
```

4.59.2 Define Documentation

4.59.2.1 #define ITMAX 5

4.59.3 Function Documentation

4.59.3.1 void dgsrfs (trans_t *trans*, SuperMatrix ∗ *A*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, int ∗ *perm_c*, int ∗ *perm_r*, char ∗ *equed*, double ∗ *R*, double ∗ *C*, SuperMatrix ∗ *B*, SuperMatrix ∗ *X*, double ∗ *ferr*, double ∗ *berr*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
  Purpose
  =======

  DGSRFS improves the computed solution to a system of linear
  equations and provides error bounds and backward error estimates for
  the solution.

  If equilibration was performed, the system becomes:
          (diag(R)*A_original*diag(C)) * X = diag(R)*B_original.

  See supermatrix.h for the definition of 'SuperMatrix' structure.

  Arguments
  =========

 trans   (input) trans_t
          Specifies the form of the system of equations:
          = NOTRANS: A * X = B  (No transpose)
          = TRANS:   A'* X = B  (Transpose)
          = CONJ:    A**H * X = B  (Conjugate transpose)

  A       (input) SuperMatrix*
          The original matrix A in the system, or the scaled A if
          equilibration was done. The type of A can be:
          Stype = SLU_NC, Dtype = SLU_D, Mtype = SLU_GE.

  L       (input) SuperMatrix*
    The factor L from the factorization Pr*A*Pc=L*U. Use
          compressed row subscripts storage for supernodes,
          i.e., L has types: Stype = SLU_SC, Dtype = SLU_D, Mtype = SLU_TRLU.

  U       (input) SuperMatrix*
          The factor U from the factorization Pr*A*Pc=L*U as computed by
          dgstrf(). Use column-wise storage scheme,
          i.e., U has types: Stype = SLU_NC, Dtype = SLU_D, Mtype = SLU_TRU.

  perm_c  (input) int*, dimension (A->ncol)
    Column permutation vector, which defines the
          permutation matrix Pc; perm_c[i] = j means column i of A is
          in position j in A*Pc.

  perm_r  (input) int*, dimension (A->nrow)
          Row permutation vector, which defines the permutation matrix Pr;
          perm_r[i] = j means row i of A is in position j in Pr*A.
```

```
equed   (input) Specifies the form of equilibration that was done.
        = 'N': No equilibration.
        = 'R': Row equilibration, i.e., A was premultiplied by diag(R).
        = 'C': Column equilibration, i.e., A was postmultiplied by
               diag(C).
        = 'B': Both row and column equilibration, i.e., A was replaced
               by diag(R)*A*diag(C).

R       (input) double*, dimension (A->nrow)
        The row scale factors for A.
        If equed = 'R' or 'B', A is premultiplied by diag(R).
        If equed = 'N' or 'C', R is not accessed.

C       (input) double*, dimension (A->ncol)
        The column scale factors for A.
        If equed = 'C' or 'B', A is postmultiplied by diag(C).
        If equed = 'N' or 'R', C is not accessed.

B       (input) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_D, Mtype = SLU_GE.
        The right hand side matrix B.
        if equed = 'R' or 'B', B is premultiplied by diag(R).

X       (input/output) SuperMatrix*
        X has types: Stype = SLU_DN, Dtype = SLU_D, Mtype = SLU_GE.
        On entry, the solution matrix X, as computed by dgstrs().
        On exit, the improved solution matrix X.
        if *equed = 'C' or 'B', X should be premultiplied by diag(C)
            in order to obtain the solution to the original system.

FERR    (output) double*, dimension (B->ncol)
        The estimated forward error bound for each solution vector
        X(j) (the j-th column of the solution matrix X).
        If XTRUE is the true solution corresponding to X(j), FERR(j)
        is an estimated upper bound for the magnitude of the largest
        element in (X(j) - XTRUE) divided by the magnitude of the
        largest element in X(j).  The estimate is as reliable as
        the estimate for RCOND, and is almost always a slight
        overestimate of the true error.

BERR    (output) double*, dimension (B->ncol)
        The componentwise relative backward error of each solution
        vector X(j) (i.e., the smallest relative change in
        any element of A or B that makes X(j) an exact solution).

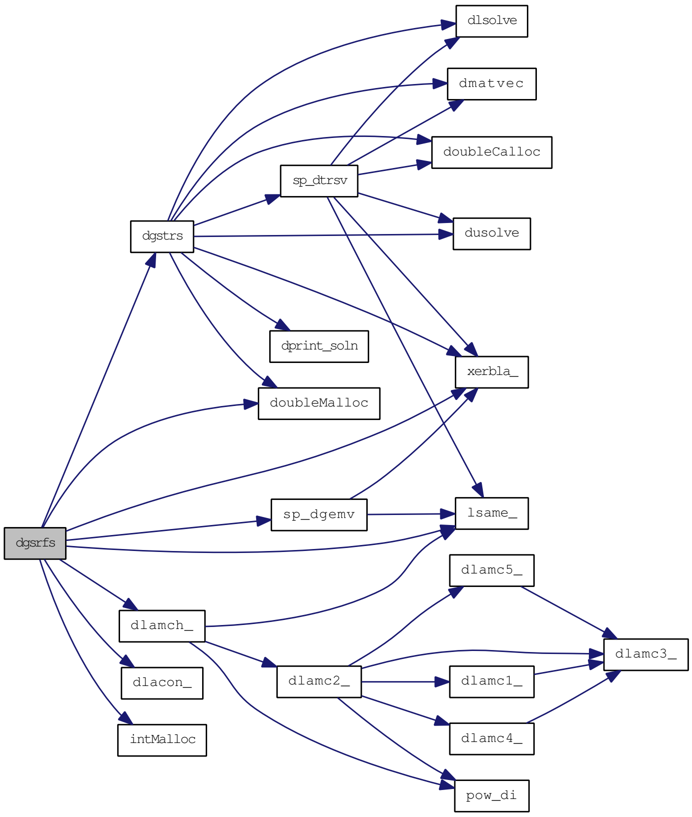
stat     (output) SuperLUStat_t*
         Record the statistics on runtime and floating-point operation count.
         See util.h for the definition of 'SuperLUStat_t'.

info    (output) int*
        = 0:  successful exit
         < 0:  if INFO = -i, the i-th argument had an illegal value

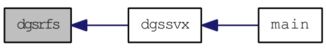
 Internal Parameters
 ===================
```

ITMAX is the maximum number of steps of iterative refinement.

Here is the call graph for this function:



Here is the caller graph for this function:

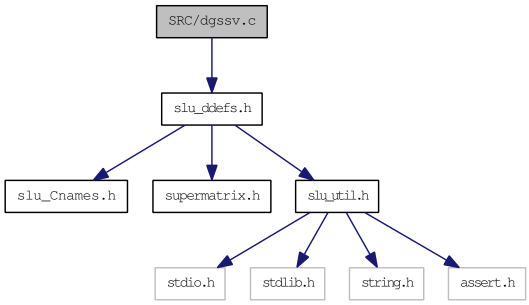


4.60 SRC/dgssv.c File Reference

Solves the system of linear equations A∗X=B.

`#include "slu_ddefs.h"`

Include dependency graph for dgssv.c:



Functions

- void dgssv (superlu_options_t ∗options, SuperMatrix ∗A, int ∗perm_c, int ∗perm_r, SuperMatrix ∗L, SuperMatrix ∗U, SuperMatrix ∗B, SuperLUStat_t ∗stat, int ∗info)

  Driver routines.

4.60.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003
```

4.60.2 Function Documentation

4.60.2.1 void dgssv (superlu_options_t ∗ *options*, SuperMatrix ∗ *A*, int ∗ *perm_c*, int ∗ *perm_r*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, SuperMatrix ∗ *B*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======

DGSSV solves the system of linear equations A*X=B, using the
LU factorization from DGSTRF. It performs the following steps:

  1. If A is stored column-wise (A->Stype = SLU_NC):

     1.1. Permute the columns of A, forming A*Pc, where Pc
          is a permutation matrix. For more details of this step,
          see sp_preorder.c.
```

```
    1.2. Factor A as Pr*A*Pc=L*U with the permutation Pr determined
         by Gaussian elimination with partial pivoting.
         L is unit lower triangular with offdiagonal entries
         bounded by 1 in magnitude, and U is upper triangular.

    1.3. Solve the system of equations A*X=B using the factored
         form of A.

 2. If A is stored row-wise (A->Stype = SLU_NR), apply the
    above algorithm to the transpose of A:

    2.1. Permute columns of transpose(A) (rows of A),
         forming transpose(A)*Pc, where Pc is a permutation matrix.
         For more details of this step, see sp_preorder.c.

    2.2. Factor A as Pr*transpose(A)*Pc=L*U with the permutation Pr
         determined by Gaussian elimination with partial pivoting.
         L is unit lower triangular with offdiagonal entries
         bounded by 1 in magnitude, and U is upper triangular.

    2.3. Solve the system of equations A*X=B using the factored
         form of A.

 See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

options (input) superlu_options_t*
        The structure defines the input parameters to control
        how the LU decomposition will be performed and how the
        system will be solved.

A       (input) SuperMatrix*
        Matrix A in A*X=B, of dimension (A->nrow, A->ncol). The number
        of linear equations is A->nrow. Currently, the type of A can be:
        Stype = SLU_NC or SLU_NR; Dtype = SLU_D; Mtype = SLU_GE.
        In the future, more general A may be handled.

perm_c  (input/output) int*
        If A->Stype = SLU_NC, column permutation vector of size A->ncol
        which defines the permutation matrix Pc; perm_c[i] = j means
        column i of A is in position j in A*Pc.
        If A->Stype = SLU_NR, column permutation vector of size A->nrow
        which describes permutation of columns of transpose(A)
        (rows of A) as described above.

        If options->ColPerm = MY_PERMC or options->Fact = SamePattern or
           options->Fact = SamePattern_SameRowPerm, it is an input argument.
           On exit, perm_c may be overwritten by the product of the input
           perm_c and a permutation that postorders the elimination tree
           of Pc'*A'*A*Pc; perm_c is not changed if the elimination tree
           is already in postorder.
        Otherwise, it is an output argument.
```

```
perm_r  (input/output) int*
        If A->Stype = SLU_NC, row permutation vector of size A->nrow,
        which defines the permutation matrix Pr, and is determined
        by partial pivoting.  perm_r[i] = j means row i of A is in
        position j in Pr*A.
        If A->Stype = SLU_NR, permutation vector of size A->ncol, which
        determines permutation of rows of transpose(A)
        (columns of A) as described above.



        If options->RowPerm = MY_PERMR or
           options->Fact = SamePattern_SameRowPerm, perm_r is an
           input argument.
        otherwise it is an output argument.



L       (output) SuperMatrix*
        The factor L from the factorization
            Pr*A*Pc=L*U              (if A->Stype = SLU_NC) or
            Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
        Uses compressed row subscripts storage for supernodes, i.e.,
        L has types: Stype = SLU_SC, Dtype = SLU_D, Mtype = SLU_TRLU.



U       (output) SuperMatrix*
  The factor U from the factorization
            Pr*A*Pc=L*U              (if A->Stype = SLU_NC) or
            Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
        Uses column-wise storage scheme, i.e., U has types:
        Stype = SLU_NC, Dtype = SLU_D, Mtype = SLU_TRU.



B       (input/output) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_D, Mtype = SLU_GE.
        On entry, the right hand side matrix.
        On exit, the solution matrix if info = 0;

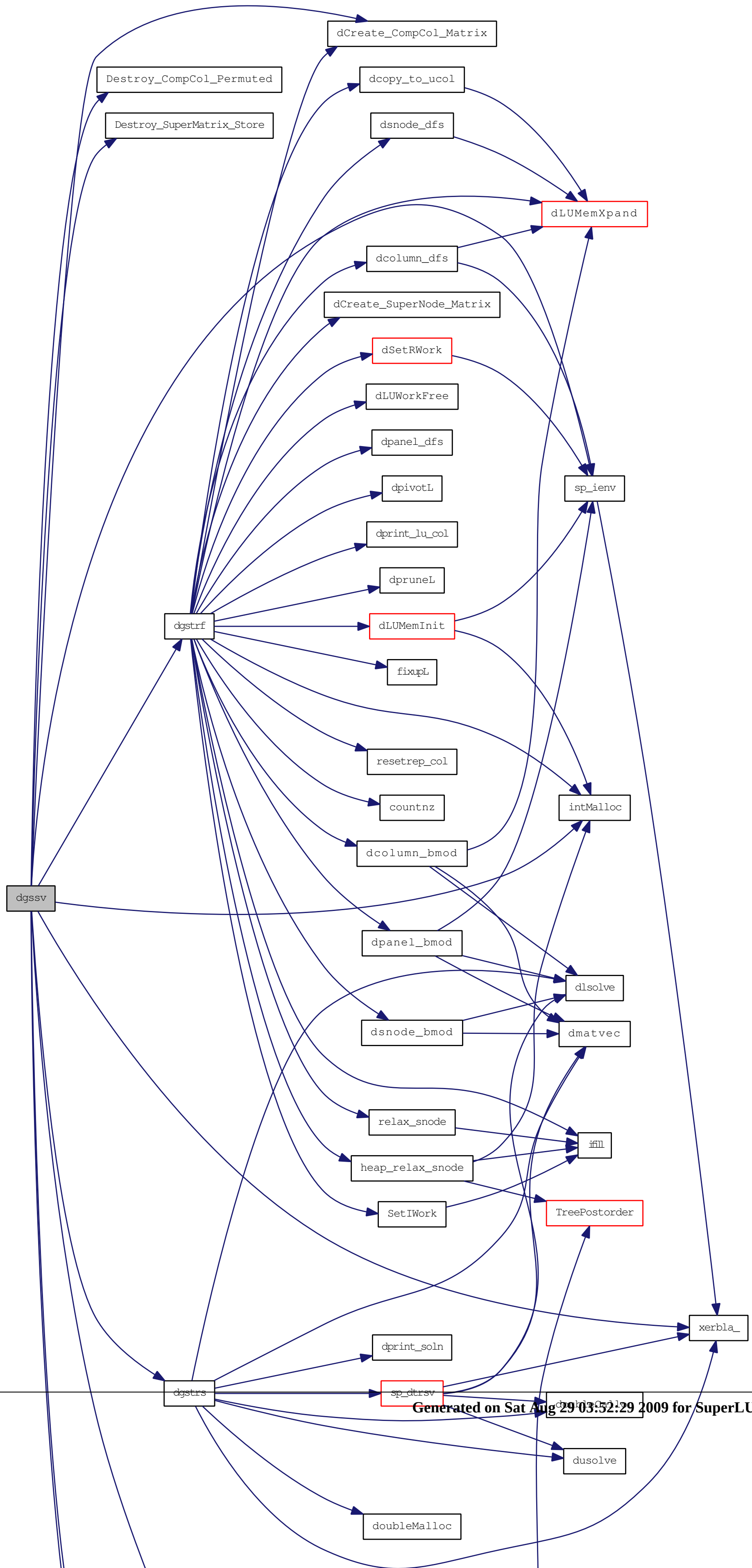


stat   (output) SuperLUStat_t*
       Record the statistics on runtime and floating-point operation count.
       See util.h for the definition of 'SuperLUStat_t'.



info    (output) int*
  = 0: successful exit
        > 0: if info = i, and i is
            <= A->ncol: U(i,i) is exactly zero. The factorization has
               been completed, but the factor U is exactly singular,
               so the solution could not be computed.
            > A->ncol: number of bytes allocated when memory allocation
               failure occurred, plus A->ncol.
```

Here is the call graph for this function:



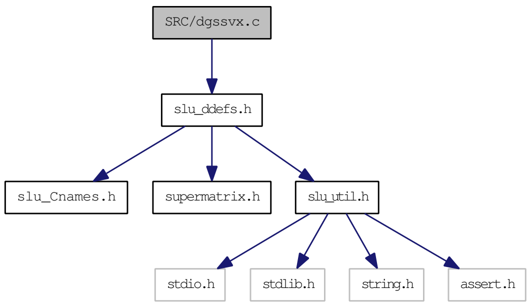
Here is the caller graph for this function:

4.61 SRC/dgssvx.c File Reference

Solves the system of linear equations A∗X=B or A'∗X=B.

`#include "slu_ddefs.h"`

Include dependency graph for dgssvx.c:



Functions

- void dgssvx (superlu_options_t ∗options, SuperMatrix ∗A, int ∗perm_c, int ∗perm_r, int ∗etree, char ∗equed, double ∗R, double ∗C, SuperMatrix ∗L, SuperMatrix ∗U, void ∗work, int lwork, SuperMatrix ∗B, SuperMatrix ∗X, double ∗recip_pivot_growth, double ∗rcond, double ∗ferr, double ∗berr, mem_usage_t ∗mem_usage, SuperLUStat_t ∗stat, int ∗info)

4.61.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003
```

4.61.2 Function Documentation

4.61.2.1 void dgssvx (superlu_options_t ∗ *options*, SuperMatrix ∗ *A*, int ∗ *perm_c*, int ∗ *perm_r*, int ∗ *etree*, char ∗ *equed*, double ∗ *R*, double ∗ *C*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, void ∗ *work*, int *lwork*, SuperMatrix ∗ *B*, SuperMatrix ∗ *X*, double ∗ *recip_pivot_growth*, double ∗ *rcond*, double ∗ *ferr*, double ∗ *berr*, mem_usage_t ∗ *mem_usage*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======

DGSSVX solves the system of linear equations A*X=B or A'*X=B, using
the LU factorization from dgstrf(). Error bounds on the solution and
a condition estimate are also provided. It performs the following steps:

  1. If A is stored column-wise (A->Stype = SLU_NC):
```

```
      1.1. If options->Equil = YES, scaling factors are computed to
           equilibrate the system:
           options->Trans = NOTRANS:
               diag(R)*A*diag(C) *inv(diag(C))*X = diag(R)*B
           options->Trans = TRANS:
               (diag(R)*A*diag(C))**T *inv(diag(R))*X = diag(C)*B
           options->Trans = CONJ:
               (diag(R)*A*diag(C))**H *inv(diag(R))*X = diag(C)*B
           Whether or not the system will be equilibrated depends on the
           scaling of the matrix A, but if equilibration is used, A is
           overwritten by diag(R)*A*diag(C) and B by diag(R)*B
           (if options->Trans=NOTRANS) or diag(C)*B (if options->Trans
           = TRANS or CONJ).

      1.2. Permute columns of A, forming A*Pc, where Pc is a permutation
           matrix that usually preserves sparsity.
           For more details of this step, see sp_preorder.c.

      1.3. If options->Fact != FACTORED, the LU decomposition is used to
           factor the matrix A (after equilibration if options->Equil = YES)
           as Pr*A*Pc = L*U, with Pr determined by partial pivoting.

      1.4. Compute the reciprocal pivot growth factor.

      1.5. If some U(i,i) = 0, so that U is exactly singular, then the
           routine returns with info = i. Otherwise, the factored form of
           A is used to estimate the condition number of the matrix A. If
           the reciprocal of the condition number is less than machine
           precision, info = A->ncol+1 is returned as a warning, but the
           routine still goes on to solve for X and computes error bounds
           as described below.

      1.6. The system of equations is solved for X using the factored form
           of A.

      1.7. If options->IterRefine != NOREFINE, iterative refinement is
           applied to improve the computed solution matrix and calculate
           error bounds and backward error estimates for it.

      1.8. If equilibration was used, the matrix X is premultiplied by
           diag(C) (if options->Trans = NOTRANS) or diag(R)
           (if options->Trans = TRANS or CONJ) so that it solves the
           original system before equilibration.

   2. If A is stored row-wise (A->Stype = SLU_NR), apply the above algorithm
      to the transpose of A:

      2.1. If options->Equil = YES, scaling factors are computed to
           equilibrate the system:
           options->Trans = NOTRANS:
               diag(R)*A*diag(C) *inv(diag(C))*X = diag(R)*B
           options->Trans = TRANS:
               (diag(R)*A*diag(C))**T *inv(diag(R))*X = diag(C)*B
           options->Trans = CONJ:
               (diag(R)*A*diag(C))**H *inv(diag(R))*X = diag(C)*B
```

```
        Whether or not the system will be equilibrated depends on the
        scaling of the matrix A, but if equilibration is used, A' is
        overwritten by diag(R)*A'*diag(C) and B by diag(R)*B
        (if trans='N') or diag(C)*B (if trans = 'T' or 'C').

   2.2. Permute columns of transpose(A) (rows of A),
        forming transpose(A)*Pc, where Pc is a permutation matrix that
        usually preserves sparsity.
        For more details of this step, see sp_preorder.c.

   2.3. If options->Fact != FACTORED, the LU decomposition is used to
        factor the transpose(A) (after equilibration if
        options->Fact = YES) as Pr*transpose(A)*Pc = L*U with the
        permutation Pr determined by partial pivoting.

   2.4. Compute the reciprocal pivot growth factor.

   2.5. If some U(i,i) = 0, so that U is exactly singular, then the
        routine returns with info = i. Otherwise, the factored form
        of transpose(A) is used to estimate the condition number of the
        matrix A. If the reciprocal of the condition number
        is less than machine precision, info = A->nrow+1 is returned as
        a warning, but the routine still goes on to solve for X and
        computes error bounds as described below.

   2.6. The system of equations is solved for X using the factored form
        of transpose(A).

   2.7. If options->IterRefine != NOREFINE, iterative refinement is
        applied to improve the computed solution matrix and calculate
        error bounds and backward error estimates for it.

   2.8. If equilibration was used, the matrix X is premultiplied by
        diag(C) (if options->Trans = NOTRANS) or diag(R)
        (if options->Trans = TRANS or CONJ) so that it solves the
        original system before equilibration.

  See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

options (input) superlu_options_t*
        The structure defines the input parameters to control
        how the LU decomposition will be performed and how the
        system will be solved.

A       (input/output) SuperMatrix*
        Matrix A in A*X=B, of dimension (A->nrow, A->ncol). The number
        of the linear equations is A->nrow. Currently, the type of A can be:
        Stype = SLU_NC or SLU_NR, Dtype = SLU_D, Mtype = SLU_GE.
        In the future, more general A may be handled.
```

```
        On entry, If options->Fact = FACTORED and equed is not 'N',
        then A must have been equilibrated by the scaling factors in
        R and/or C.
        On exit, A is not modified if options->Equil = NO, or if
        options->Equil = YES but equed = 'N' on exit.
        Otherwise, if options->Equil = YES and equed is not 'N',
        A is scaled as follows:
        If A->Stype = SLU_NC:
          equed = 'R':  A := diag(R) * A
          equed = 'C':  A := A * diag(C)
          equed = 'B':  A := diag(R) * A * diag(C).
        If A->Stype = SLU_NR:
          equed = 'R':  transpose(A) := diag(R) * transpose(A)
          equed = 'C':  transpose(A) := transpose(A) * diag(C)
          equed = 'B':  transpose(A) := diag(R) * transpose(A) * diag(C).

perm_c  (input/output) int*
  If A->Stype = SLU_NC, Column permutation vector of size A->ncol,
        which defines the permutation matrix Pc; perm_c[i] = j means
        column i of A is in position j in A*Pc.
        On exit, perm_c may be overwritten by the product of the input
        perm_c and a permutation that postorders the elimination tree
        of Pc'*A'*A*Pc; perm_c is not changed if the elimination tree
        is already in postorder.

        If A->Stype = SLU_NR, column permutation vector of size A->nrow,
        which describes permutation of columns of transpose(A)
        (rows of A) as described above.

perm_r  (input/output) int*
        If A->Stype = SLU_NC, row permutation vector of size A->nrow,
        which defines the permutation matrix Pr, and is determined
        by partial pivoting.  perm_r[i] = j means row i of A is in
        position j in Pr*A.

        If A->Stype = SLU_NR, permutation vector of size A->ncol, which
        determines permutation of rows of transpose(A)
        (columns of A) as described above.

        If options->Fact = SamePattern_SameRowPerm, the pivoting routine
        will try to use the input perm_r, unless a certain threshold
        criterion is violated. In that case, perm_r is overwritten by a
        new permutation determined by partial pivoting or diagonal
        threshold pivoting.
        Otherwise, perm_r is output argument.

etree   (input/output) int*,  dimension (A->ncol)
        Elimination tree of Pc'*A'*A*Pc.
        If options->Fact != FACTORED and options->Fact != DOFACT,
        etree is an input argument, otherwise it is an output argument.
        Note: etree is a vector of parent pointers for a forest whose
        vertices are the integers 0 to A->ncol-1; etree[root]==A->ncol.

equed   (input/output) char*
        Specifies the form of equilibration that was done.
        = 'N': No equilibration.
```

```
        = 'R': Row equilibration, i.e., A was premultiplied by diag(R).
        = 'C': Column equilibration, i.e., A was postmultiplied by diag(C).
        = 'B': Both row and column equilibration, i.e., A was replaced
               by diag(R)*A*diag(C).
        If options->Fact = FACTORED, equed is an input argument,
        otherwise it is an output argument.

R       (input/output) double*, dimension (A->nrow)
        The row scale factors for A or transpose(A).
        If equed = 'R' or 'B', A (if A->Stype = SLU_NC) or transpose(A)
            (if A->Stype = SLU_NR) is multiplied on the left by diag(R).
        If equed = 'N' or 'C', R is not accessed.
        If options->Fact = FACTORED, R is an input argument,
            otherwise, R is output.
        If options->zFact = FACTORED and equed = 'R' or 'B', each element
            of R must be positive.

C       (input/output) double*, dimension (A->ncol)
        The column scale factors for A or transpose(A).
        If equed = 'C' or 'B', A (if A->Stype = SLU_NC) or transpose(A)
            (if A->Stype = SLU_NR) is multiplied on the right by diag(C).
        If equed = 'N' or 'R', C is not accessed.
        If options->Fact = FACTORED, C is an input argument,
            otherwise, C is output.
        If options->Fact = FACTORED and equed = 'C' or 'B', each element
            of C must be positive.

L       (output) SuperMatrix*
  The factor L from the factorization
            Pr*A*Pc=L*U              (if A->Stype SLU_= NC) or
            Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
        Uses compressed row subscripts storage for supernodes, i.e.,
        L has types: Stype = SLU_SC, Dtype = SLU_D, Mtype = SLU_TRLU.

U       (output) SuperMatrix*
  The factor U from the factorization
            Pr*A*Pc=L*U              (if A->Stype = SLU_NC) or
            Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
        Uses column-wise storage scheme, i.e., U has types:
        Stype = SLU_NC, Dtype = SLU_D, Mtype = SLU_TRU.

work    (workspace/output) void*, size (lwork) (in bytes)
        User supplied workspace, should be large enough
        to hold data structures for factors L and U.
        On exit, if fact is not 'F', L and U point to this array.

lwork   (input) int
        Specifies the size of work array in bytes.
        = 0:  allocate space internally by system malloc;
        > 0:  use user-supplied work array of length lwork in bytes,
              returns error if space runs out.
        = -1: the routine guesses the amount of space needed without
              performing the factorization, and returns it in
              mem_usage->total_needed; no other side effects.

        See argument 'mem_usage' for memory usage statistics.
```

```
B       (input/output) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_D, Mtype = SLU_GE.
        On entry, the right hand side matrix.
        If B->ncol = 0, only LU decomposition is performed, the triangular
                        solve is skipped.
        On exit,
           if equed = 'N', B is not modified; otherwise
           if A->Stype = SLU_NC:
              if options->Trans = NOTRANS and equed = 'R' or 'B',
                 B is overwritten by diag(R)*B;
              if options->Trans = TRANS or CONJ and equed = 'C' of 'B',
                 B is overwritten by diag(C)*B;
           if A->Stype = SLU_NR:
              if options->Trans = NOTRANS and equed = 'C' or 'B',
                 B is overwritten by diag(C)*B;
              if options->Trans = TRANS or CONJ and equed = 'R' of 'B',
                 B is overwritten by diag(R)*B.

X       (output) SuperMatrix*
        X has types: Stype = SLU_DN, Dtype = SLU_D, Mtype = SLU_GE.
        If info = 0 or info = A->ncol+1, X contains the solution matrix
        to the original system of equations. Note that A and B are modified
        on exit if equed is not 'N', and the solution to the equilibrated
        system is inv(diag(C))*X if options->Trans = NOTRANS and
        equed = 'C' or 'B', or inv(diag(R))*X if options->Trans = 'T' or 'C'
        and equed = 'R' or 'B'.

recip_pivot_growth (output) double*
        The reciprocal pivot growth factor max_j( norm(A_j)/norm(U_j) ).
        The infinity norm is used. If recip_pivot_growth is much less
        than 1, the stability of the LU factorization could be poor.

rcond   (output) double*
        The estimate of the reciprocal condition number of the matrix A
        after equilibration (if done). If rcond is less than the machine
        precision (in particular, if rcond = 0), the matrix is singular
        to working precision. This condition is indicated by a return
        code of info > 0.

FERR    (output) double*, dimension (B->ncol)
        The estimated forward error bound for each solution vector
        X(j) (the j-th column of the solution matrix X).
        If XTRUE is the true solution corresponding to X(j), FERR(j)
        is an estimated upper bound for the magnitude of the largest
        element in (X(j) - XTRUE) divided by the magnitude of the
        largest element in X(j).  The estimate is as reliable as
        the estimate for RCOND, and is almost always a slight
        overestimate of the true error.
        If options->IterRefine = NOREFINE, ferr = 1.0.

BERR    (output) double*, dimension (B->ncol)
        The componentwise relative backward error of each solution
        vector X(j) (i.e., the smallest relative change in
        any element of A or B that makes X(j) an exact solution).
        If options->IterRefine = NOREFINE, berr = 1.0.

mem_usage (output) mem_usage_t*
```

```
Record the memory usage statistics, consisting of following fields:
```

- for_lu (float)

```
The amount of space used in bytes for L data structures.
```

- total_needed (float)

```
The amount of space needed in bytes to perform factorization.
```

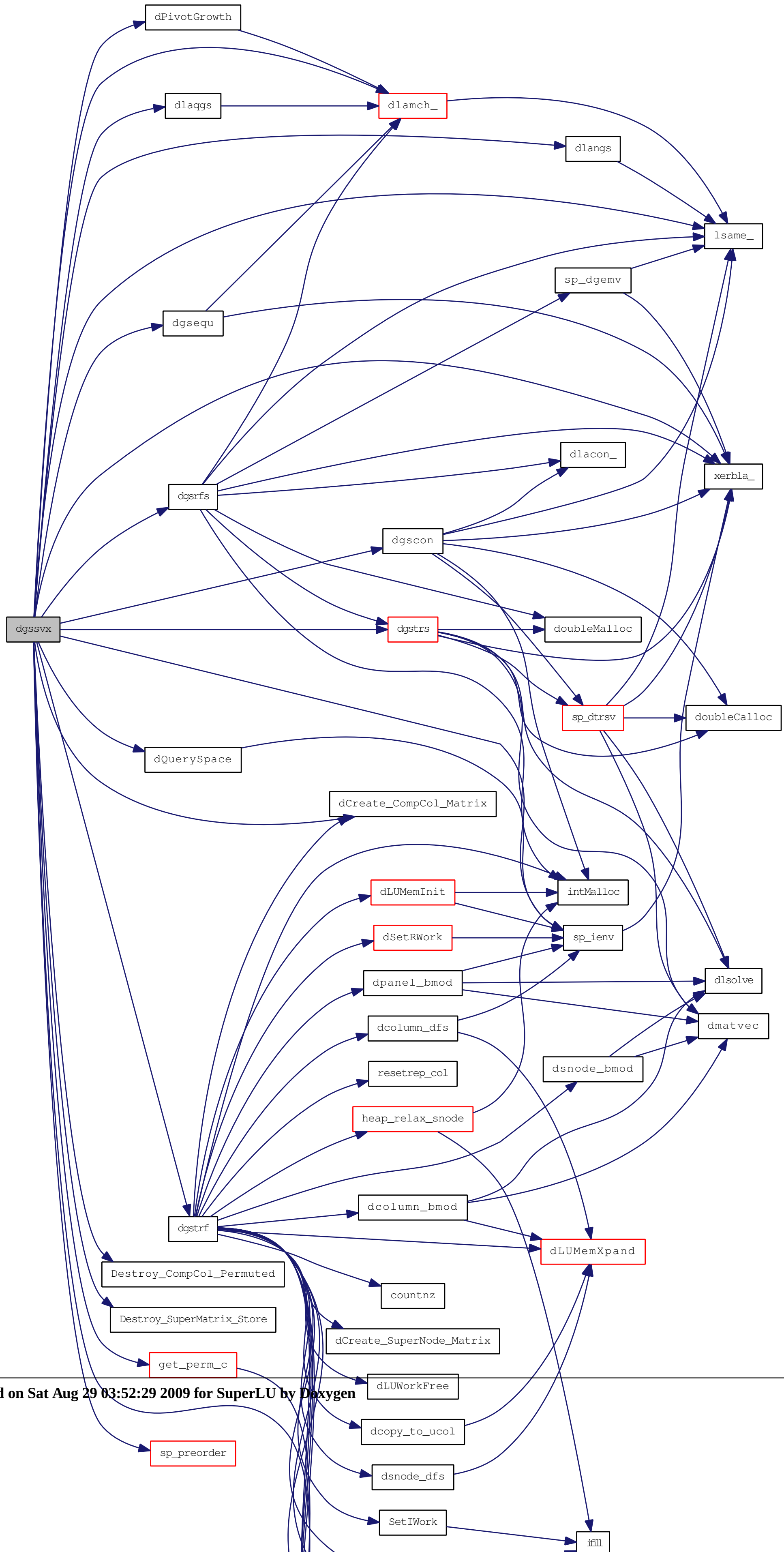
- expansions (int)

```
The number of memory expansions during the LU factorization.
```

```
stat   (output) SuperLUStat_t*
       Record the statistics on runtime and floating-point operation count.
       See util.h for the definition of 'SuperLUStat_t'.
```

```
info    (output) int*
        = 0: successful exit
        < 0: if info = -i, the i-th argument had an illegal value
        > 0: if info = i, and i is
             <= A->ncol: U(i,i) is exactly zero. The factorization has
                   been completed, but the factor U is exactly
                   singular, so the solution and error bounds
                   could not be computed.
             = A->ncol+1: U is nonsingular, but RCOND is less than machine
                   precision, meaning that the matrix is singular to
                   working precision. Nevertheless, the solution and
                   error bounds are computed because there are a number
                   of situations where the computed solution can be more
                   accurate than the value of RCOND would suggest.
             > A->ncol+1: number of bytes allocated when memory allocation
                   failure occurred, plus A->ncol.
```

Here is the call graph for this function:



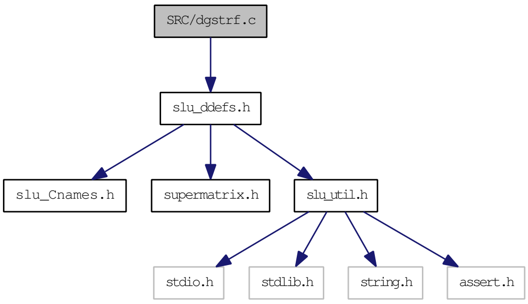
Here is the caller graph for this function:

4.62 SRC/dgstrf.c File Reference

Computes an LU factorization of a general sparse matrix.

`#include "slu_ddefs.h"`

Include dependency graph for dgstrf.c:



Functions

- void dgstrf (superlu_options_t *options, SuperMatrix *A, double drop_tol, int relax, int panel_size, int *etree, void *work, int lwork, int *perm_c, int *perm_r, SuperMatrix *L, SuperMatrix *U, SuperLUStat_t *stat, int *info)

4.62.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.62.2 Function Documentation

4.62.2.1 void dgstrf (superlu_options_t * *options*, SuperMatrix * *A*, double *drop_tol*, int *relax*, int *panel_size*, int * *etree*, void * *work*, int *lwork*, int * *perm_c*, int * *perm_r*, SuperMatrix * *L*, SuperMatrix * *U*, SuperLUStat_t * *stat*, int * *info*)

```
Purpose
=======
```

```
DGSTRF computes an LU factorization of a general sparse m-by-n
matrix A using partial pivoting with row interchanges.
The factorization has the form
    Pr * A = L * U
where Pr is a row permutation matrix, L is lower triangular with unit
diagonal elements (lower trapezoidal if A->nrow > A->ncol), and U is upper
triangular (upper trapezoidal if A->nrow < A->ncol).

See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

options (input) superlu_options_t*
        The structure defines the input parameters to control
        how the LU decomposition will be performed.

A        (input) SuperMatrix*
   Original matrix A, permuted by columns, of dimension
         (A->nrow, A->ncol). The type of A can be:
         Stype = SLU_NCP; Dtype = SLU_D; Mtype = SLU_GE.

drop_tol (input) double (NOT IMPLEMENTED)
   Drop tolerance parameter. At step j of the Gaussian elimination,
         if abs(A_ij)/(max_i abs(A_ij)) < drop_tol, drop entry A_ij.
         0 <= drop_tol <= 1. The default value of drop_tol is 0.

relax    (input) int
         To control degree of relaxing supernodes. If the number
         of nodes (columns) in a subtree of the elimination tree is less
         than relax, this subtree is considered as one supernode,
         regardless of the row structures of those columns.

panel_size (input) int
         A panel consists of at most panel_size consecutive columns.

etree    (input) int*, dimension (A->ncol)
         Elimination tree of A'*A.
         Note: etree is a vector of parent pointers for a forest whose
         vertices are the integers 0 to A->ncol-1; etree[root]==A->ncol.
         On input, the columns of A should be permuted so that the
         etree is in a certain postorder.

work     (input/output) void*, size (lwork) (in bytes)
         User-supplied work space and space for the output data structures.
         Not referenced if lwork = 0;

lwork   (input) int
        Specifies the size of work array in bytes.
        = 0:  allocate space internally by system malloc;
        > 0:  use user-supplied work array of length lwork in bytes,
              returns error if space runs out.
        = -1: the routine guesses the amount of space needed without
              performing the factorization, and returns it in
              *info; no other side effects.
```

```
perm_c   (input) int*, dimension (A->ncol)
   Column permutation vector, which defines the
         permutation matrix Pc; perm_c[i] = j means column i of A is
         in position j in A*Pc.
         When searching for diagonal, perm_c[*] is applied to the
         row subscripts of A, so that diagonal threshold pivoting
         can find the diagonal of A, rather than that of A*Pc.


perm_r   (input/output) int*, dimension (A->nrow)
         Row permutation vector which defines the permutation matrix Pr,
         perm_r[i] = j means row i of A is in position j in Pr*A.
         If options->Fact = SamePattern_SameRowPerm, the pivoting routine
            will try to use the input perm_r, unless a certain threshold
            criterion is violated. In that case, perm_r is overwritten by
            a new permutation determined by partial pivoting or diagonal
            threshold pivoting.
         Otherwise, perm_r is output argument;


L        (output) SuperMatrix*
         The factor L from the factorization Pr*A=L*U; use compressed row
         subscripts storage for supernodes, i.e., L has type:
         Stype = SLU_SC, Dtype = SLU_D, Mtype = SLU_TRLU.


U        (output) SuperMatrix*
   The factor U from the factorization Pr*A*Pc=L*U. Use column-wise
         storage scheme, i.e., U has types: Stype = SLU_NC,
         Dtype = SLU_D, Mtype = SLU_TRU.


stat     (output) SuperLUStat_t*
         Record the statistics on runtime and floating-point operation count.
         See util.h for the definition of 'SuperLUStat_t'.


info     (output) int*
         = 0: successful exit
         < 0: if info = -i, the i-th argument had an illegal value
         > 0: if info = i, and i is
            <= A->ncol: U(i,i) is exactly zero. The factorization has
               been completed, but the factor U is exactly singular,
               and division by zero will occur if it is used to solve a
               system of equations.
            > A->ncol: number of bytes allocated when memory allocation
               failure occurred, plus A->ncol. If lwork = -1, it is
               the estimated amount of space needed, plus A->ncol.


======================================================================


Local Working Arrays:
======================
  m = number of rows in the matrix
  n = number of columns in the matrix


  xprune[0:n-1]: xprune[*] points to locations in subscript
vector lsub[*]. For column i, xprune[i] denotes the point where
structural pruning begins. I.e. only xlsub[i],..,xprune[i]-1 need
to be traversed for symbolic factorization.
```

```
   marker[0:3*m-1]: marker[i] = j means that node i has been
reached when working on column j.
Storage: relative to original row subscripts
NOTE: There are 3 of them: marker/marker1 are used for panel dfs,
      see dpanel_dfs.c; marker2 is used for inner-factorization,
            see dcolumn_dfs.c.
```

```
   parent[0:m-1]: parent vector used during dfs
      Storage: relative to new row subscripts
```

```
   xplore[0:m-1]: xplore[i] gives the location of the next (dfs)
unexplored neighbor of i in lsub[*]
```

```
   segrep[0:nseg-1]: contains the list of supernodal representatives
in topological order of the dfs. A supernode representative is the
last column of a supernode.
      The maximum size of segrep[] is n.
```

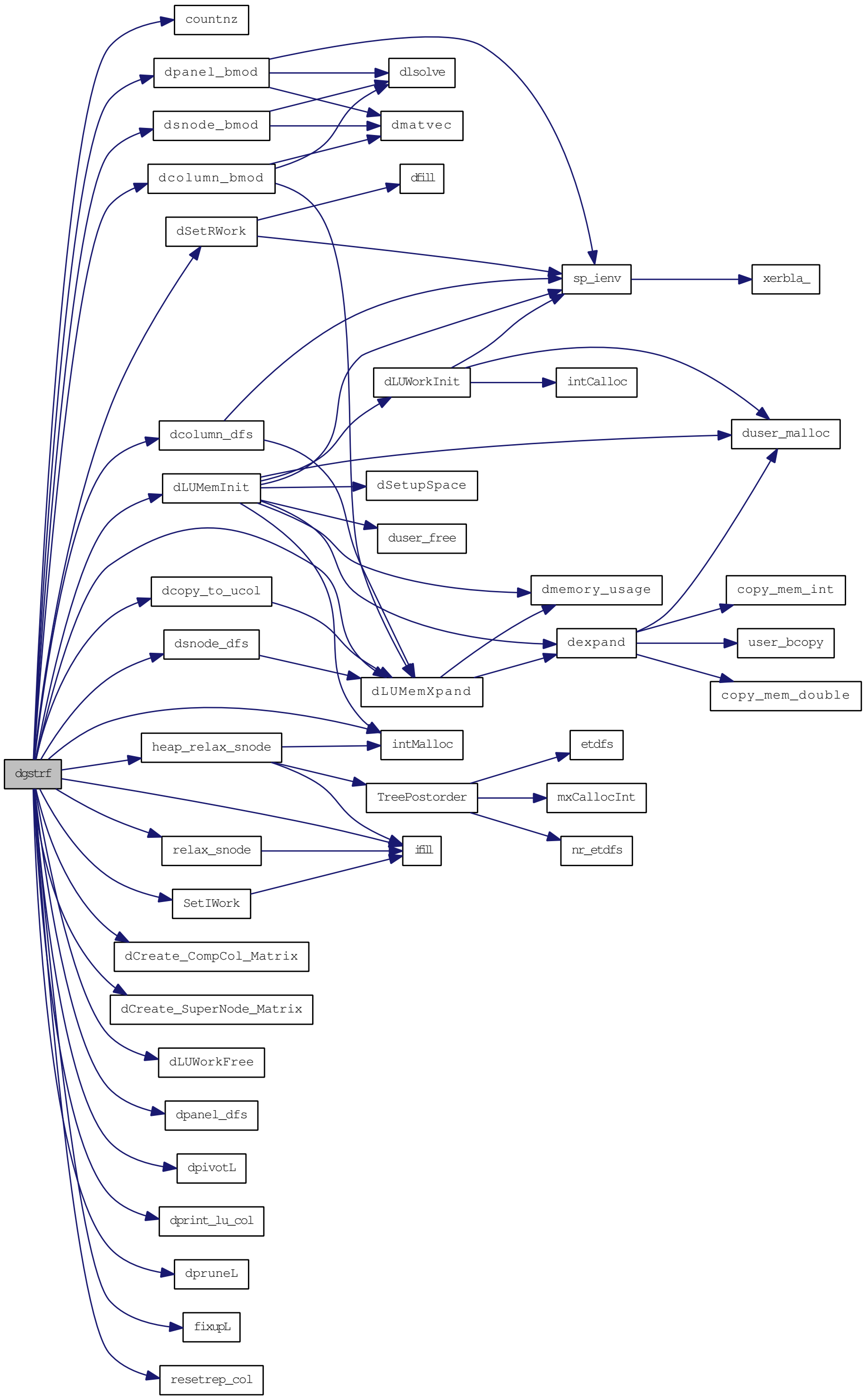
```
   repfnz[0:W*m-1]: for a nonzero segment U[*,j] that ends at a
supernodal representative r, repfnz[r] is the location of the first
nonzero in this segment.  It is also used during the dfs: repfnz[r]>0
indicates the supernode r has been explored.
NOTE: There are W of them, each used for one column of a panel.
```

```
   panel_lsub[0:W*m-1]: temporary for the nonzeros row indices below
      the panel diagonal. These are filled in during dpanel_dfs(), and are
      used later in the inner LU factorization within the panel.
panel_lsub[]/dense[] pair forms the SPA data structure.
NOTE: There are W of them.
```

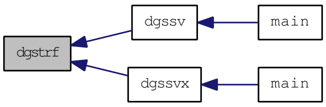
```
   dense[0:W*m-1]: sparse accumulating (SPA) vector for intermediate values;
        NOTE: there are W of them.
```

```
   tempv[0:*]: real temporary used for dense numeric kernels;
The size of this array is defined by NUM_TEMPV() in slu_ddefs.h.
```

Here is the call graph for this function:



Here is the caller graph for this function:

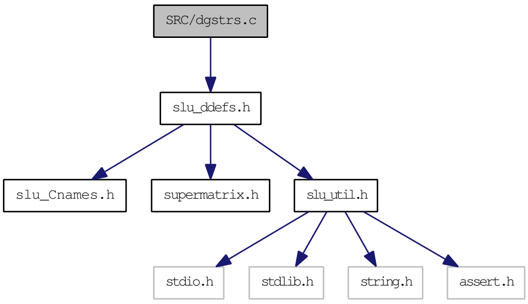


4.63 SRC/dgstrs.c File Reference

Solves a system using LU factorization.

`#include "slu_ddefs.h"`

Include dependency graph for dgstrs.c:



Functions

- void dusolve (int, int, double ∗, double ∗)

  Solves a dense upper triangular system.

- void dlsolve (int, int, double ∗, double ∗)

  Solves a dense UNIT lower triangular system.

- void dmatvec (int, int, int, double ∗, double ∗, double ∗)

  Performs a dense matrix-vector multiply: Mxvec = Mxvec + M ∗ vec.

- void dgstrs (trans_t trans, SuperMatrix ∗L, SuperMatrix ∗U, int ∗perm_c, int ∗perm_r, SuperMatrix ∗B, SuperLUStat_t ∗stat, int ∗info)
- void dprint_soln (int n, int nrhs, double ∗soln)

4.63.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.63.2 Function Documentation

4.63.2.1 void dgstrs (trans_t *trans*, SuperMatrix * *L*, SuperMatrix * *U*, int * *perm_c*, int * *perm_r*, SuperMatrix * *B*, SuperLUStat_t * *stat*, int * *info*)

```
Purpose
=======

DGSTRS solves a system of linear equations A*X=B or A'*X=B
with A sparse and B dense, using the LU factorization computed by
DGSTRF.

See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

trans   (input) trans_t
         Specifies the form of the system of equations:
         = NOTRANS: A * X = B  (No transpose)
         = TRANS:   A'* X = B  (Transpose)
         = CONJ:    A**H * X = B  (Conjugate transpose)

L       (input) SuperMatrix*
        The factor L from the factorization Pr*A*Pc=L*U as computed by
        dgstrf(). Use compressed row subscripts storage for supernodes,
        i.e., L has types: Stype = SLU_SC, Dtype = SLU_D, Mtype = SLU_TRLU.

U       (input) SuperMatrix*
        The factor U from the factorization Pr*A*Pc=L*U as computed by
        dgstrf(). Use column-wise storage scheme, i.e., U has types:
        Stype = SLU_NC, Dtype = SLU_D, Mtype = SLU_TRU.

perm_c  (input) int*, dimension (L->ncol)
  Column permutation vector, which defines the
        permutation matrix Pc; perm_c[i] = j means column i of A is
        in position j in A*Pc.

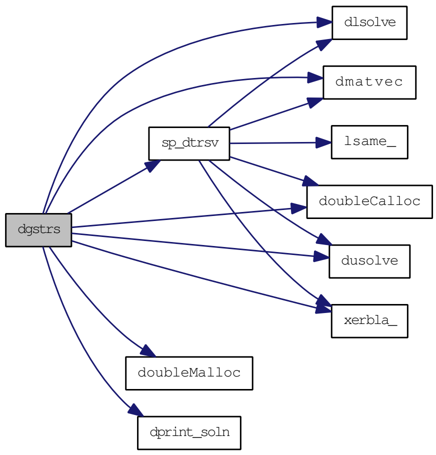
perm_r  (input) int*, dimension (L->nrow)
        Row permutation vector, which defines the permutation matrix Pr;
        perm_r[i] = j means row i of A is in position j in Pr*A.

B       (input/output) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_D, Mtype = SLU_GE.
        On entry, the right hand side matrix.
        On exit, the solution matrix if info = 0;

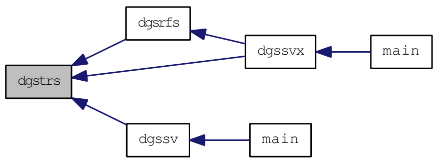
stat     (output) SuperLUStat_t*
         Record the statistics on runtime and floating-point operation count.
         See util.h for the definition of 'SuperLUStat_t'.

info    (output) int*
    = 0: successful exit
  < 0: if info = -i, the i-th argument had an illegal value
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.63.2.2 void dlsolve (int *ldm*, int *ncol*, double ∗ *M*, double ∗ *rhs*)

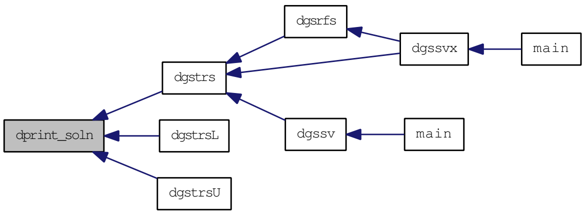
The unit lower triangular matrix is stored in a 2D array M(1:nrow,1:ncol). The solution will be returned in the rhs vector.

4.63.2.3 void dmatvec (int *ldm*, int *nrow*, int *ncol*, double ∗ *M*, double ∗ *vec*, double ∗ *Mxvec*)

The input matrix is M(1:nrow,1:ncol); The product is returned in Mxvec[].

4.63.2.4 void dprint_soln (int *n*, int *nrhs*, double ∗ *soln*)

Here is the caller graph for this function:



4.63.2.5 void dusolve (int *ldm*, int *ncol*, double * *M*, double * *rhs*)

The upper triangular matrix is stored in a 2-dim array M(1:ldm,1:ncol). The solution will be returned in the rhs vector.

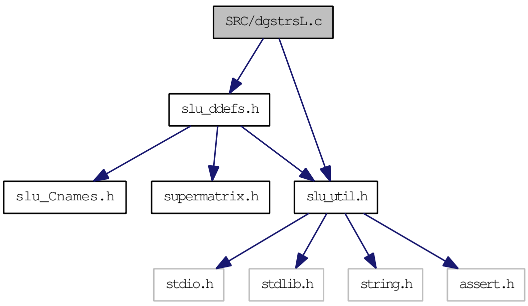
4.64 SRC/dgstrsL.c File Reference

Performs the L-solve using the LU factorization computed by DGSTRF.

`#include "slu_ddefs.h"`

`#include "slu_util.h"`

Include dependency graph for dgstrsL.c:



Functions

- void dusolve (int, int, double ∗, double ∗)

  Solves a dense upper triangular system.

- void dlsolve (int, int, double ∗, double ∗)

  Solves a dense UNIT lower triangular system.

- void dmatvec (int, int, int, double ∗, double ∗, double ∗)

  Performs a dense matrix-vector multiply: Mxvec = Mxvec + M ∗ vec.

- void dgstrsL (char ∗trans, SuperMatrix ∗L, int ∗perm_r, SuperMatrix ∗B, int ∗info)
- void dprint_soln (int n, int nrhs, double ∗soln)

4.64.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
September 15, 2003

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.64.2 Function Documentation

4.64.2.1 void dgstrsL (char ∗ *trans*, SuperMatrix ∗ *L*, int ∗ *perm_r*, SuperMatrix ∗ *B*, int ∗ *info*)

```
Purpose
=======


dgstrsL only performs the L-solve using the LU factorization computed
by DGSTRF.


See supermatrix.h for the definition of 'SuperMatrix' structure.


Arguments
=========


trans   (input) char*
         Specifies the form of the system of equations:
         = 'N':  A * X = B  (No transpose)
         = 'T':  A'* X = B  (Transpose)
         = 'C':  A**H * X = B  (Conjugate transpose)


L       (input) SuperMatrix*
        The factor L from the factorization Pr*A*Pc=L*U as computed by
        dgstrf(). Use compressed row subscripts storage for supernodes,
        i.e., L has types: Stype = SLU_SC, Dtype = SLU_D, Mtype = SLU_TRLU.

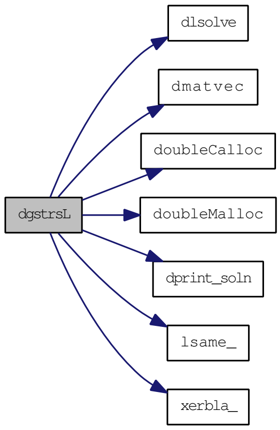

U       (input) SuperMatrix*
        The factor U from the factorization Pr*A*Pc=L*U as computed by
        dgstrf(). Use column-wise storage scheme, i.e., U has types:
        Stype = SLU_NC, Dtype = SLU_D, Mtype = SLU_TRU.


perm_r  (input) int*, dimension (L->nrow)
        Row permutation vector, which defines the permutation matrix Pr;
        perm_r[i] = j means row i of A is in position j in Pr*A.


B       (input/output) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_D, Mtype = SLU_GE.
        On entry, the right hand side matrix.
        On exit, the solution matrix if info = 0;


info    (output) int*
    = 0: successful exit
  < 0: if info = -i, the i-th argument had an illegal value
```

Here is the call graph for this function:



4.64.2.2 void dlsolve (int *ldm*, int *ncol*, double ∗ *M*, double ∗ *rhs*)

The unit lower triangular matrix is stored in a 2D array M(1:nrow,1:ncol). The solution will be returned in the rhs vector.

4.64.2.3 void dmatvec (int *ldm*, int *nrow*, int *ncol*, double ∗ *M*, double ∗ *vec*, double ∗ *Mxvec*)

The input matrix is M(1:nrow,1:ncol); The product is returned in Mxvec[].

4.64.2.4 void dprint_soln (int *n*, int *nrhs*, double ∗ *soln*)

4.64.2.5 void dusolve (int *ldm*, int *ncol*, double ∗ *M*, double ∗ *rhs*)

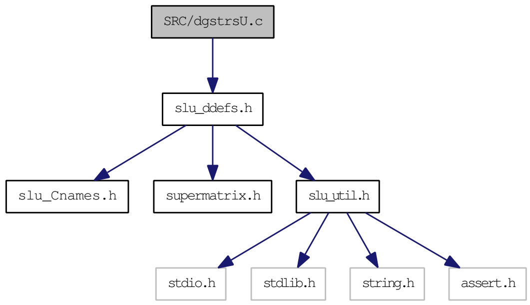
The upper triangular matrix is stored in a 2-dim array M(1:ldm,1:ncol). The solution will be returned in the rhs vector.

4.65 SRC/dgstrsU.c File Reference

Performs the U-solve using the LU factorization computed by DGSTRF.

`#include "slu_ddefs.h"`

Include dependency graph for dgstrsU.c:



Functions

- void dusolve (int, int, double ∗, double ∗)

  Solves a dense upper triangular system.

- void dlsolve (int, int, double ∗, double ∗)

  Solves a dense UNIT lower triangular system.

- void dmatvec (int, int, int, double ∗, double ∗, double ∗)

  Performs a dense matrix-vector multiply: Mxvec = Mxvec + M ∗ vec.

- void dgstrsU (trans_t trans, SuperMatrix ∗L, SuperMatrix ∗U, int ∗perm_c, int ∗perm_r, SuperMatrix ∗B, SuperLUStat_t ∗stat, int ∗info)

4.65.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.65.2 Function Documentation

4.65.2.1 void dgstrsU (trans_t *trans*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, int ∗ *perm_c*, int ∗ *perm_r*, SuperMatrix ∗ *B*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======

dgstrsU only performs the U-solve using the LU factorization computed
by DGSTRF.

See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

trans   (input) trans_t
         Specifies the form of the system of equations:
         = NOTRANS: A * X = B  (No transpose)
         = TRANS:   A'* X = B  (Transpose)
         = CONJ:    A**H * X = B  (Conjugate transpose)

L       (input) SuperMatrix*
        The factor L from the factorization Pr*A*Pc=L*U as computed by
        dgstrf(). Use compressed row subscripts storage for supernodes,
        i.e., L has types: Stype = SLU_SC, Dtype = SLU_D, Mtype = SLU_TRLU.

U       (input) SuperMatrix*
        The factor U from the factorization Pr*A*Pc=L*U as computed by
        dgstrf(). Use column-wise storage scheme, i.e., U has types:
        Stype = SLU_NC, Dtype = SLU_D, Mtype = SLU_TRU.

perm_c  (input) int*, dimension (L->ncol)
  Column permutation vector, which defines the
        permutation matrix Pc; perm_c[i] = j means column i of A is
        in position j in A*Pc.

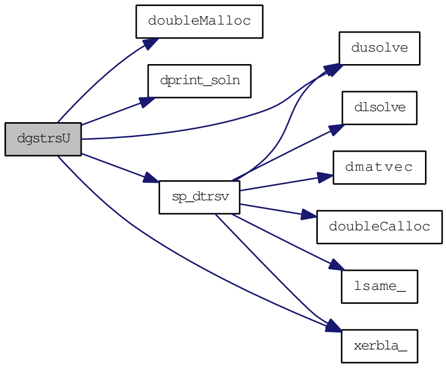
perm_r  (input) int*, dimension (L->nrow)
        Row permutation vector, which defines the permutation matrix Pr;
        perm_r[i] = j means row i of A is in position j in Pr*A.

B       (input/output) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_D, Mtype = SLU_GE.
        On entry, the right hand side matrix.
        On exit, the solution matrix if info = 0;

stat     (output) SuperLUStat_t*
         Record the statistics on runtime and floating-point operation count.
         See util.h for the definition of 'SuperLUStat_t'.

info    (output) int*
    = 0: successful exit
  < 0: if info = -i, the i-th argument had an illegal value
```

Here is the call graph for this function:



4.65.2.2 void dlsolve (int *ldm*, int *ncol*, double ∗ *M*, double ∗ *rhs*)

The unit lower triangular matrix is stored in a 2D array M(1:nrow,1:ncol). The solution will be returned in the rhs vector.

4.65.2.3 void dmatvec (int *ldm*, int *nrow*, int *ncol*, double ∗ *M*, double ∗ *vec*, double ∗ *Mxvec*)

The input matrix is M(1:nrow,1:ncol); The product is returned in Mxvec[].

4.65.2.4 void dusolve (int *ldm*, int *ncol*, double ∗ *M*, double ∗ *rhs*)

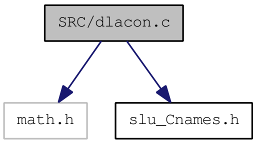
The upper triangular matrix is stored in a 2-dim array M(1:ldm,1:ncol). The solution will be returned in the rhs vector.

4.66 SRC/dlacon.c File Reference

Estimates the 1-norm.

```
#include <math.h>
#include "slu_Cnames.h"
```

Include dependency graph for dlacon.c:



Defines

- #define d_sign(a, b) (b >= 0 ? fabs(a) : -fabs(a))
- #define i_dnnt(a) (a>=0 ? floor(a+.5) : -floor(.5-a))

Functions

- int dlacon_ (int *n, double *v, double *x, int *isgn, double *est, int *kase)

4.66.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997
```

4.66.2 Define Documentation

4.66.2.1 #define d_sign(a, b) (b >= 0 ? fabs(a) : -fabs(a))

4.66.2.2 #define i_dnnt(a) (a>=0 ? floor(a+.5) : -floor(.5-a))

4.66.3 Function Documentation

4.66.3.1 int dlacon_ (int * *n*, double * *v*, double * *x*, int * *isgn*, double * *est*, int * *kase*)

```
Purpose
=======

DLACON estimates the 1-norm of a square matrix A.
Reverse communication is used for evaluating matrix-vector products.

Arguments
=========
```

```
N       (input) INT
        The order of the matrix.  N >= 1.

V       (workspace) DOUBLE PRECISION array, dimension (N)
        On the final return, V = A*W,  where  EST = norm(V)/norm(W)
        (W is not returned).

X       (input/output) DOUBLE PRECISION array, dimension (N)
        On an intermediate return, X should be overwritten by
              A * X,   if KASE=1,
              A' * X,  if KASE=2,
       and DLACON must be re-called with all the other parameters
        unchanged.

ISGN    (workspace) INT array, dimension (N)

EST     (output) DOUBLE PRECISION
        An estimate (a lower bound) for norm(A).

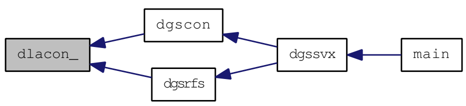
KASE    (input/output) INT
        On the initial call to DLACON, KASE should be 0.
        On an intermediate return, KASE will be 1 or 2, indicating
        whether X should be overwritten by A * X  or A' * X.
        On the final return from DLACON, KASE will again be 0.

Further Details
======= =======

Contributed by Nick Higham, University of Manchester.
Originally named CONEST, dated March 16, 1988.

Reference: N.J. Higham, "FORTRAN codes for estimating the one-norm of
a real or complex matrix, with applications to condition estimation",
ACM Trans. Math. Soft., vol. 14, no. 4, pp. 381-396, December 1988.
=====================================================================
```

Here is the caller graph for this function:



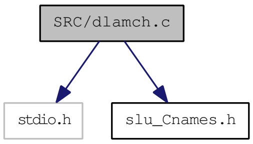
4.67 SRC/dlamch.c File Reference

Determines double precision machine parameters.

`#include <stdio.h>`

`#include "slu_Cnames.h"`

Include dependency graph for dlamch.c:



Defines

- #define TRUE_ (1)
- #define FALSE_ (0)
- #define abs(x) ((x) >= 0 ? (x) : -(x))
- #define min(a, b) ((a) <= (b) ? (a) : (b))
- #define max(a, b) ((a) >= (b) ? (a) : (b))

Functions

- double dlamch_ (char ∗cmach)
- int dlamc1_ (int ∗beta, int ∗t, int ∗rnd, int ∗ieee1)
- int dlamc2_ (int ∗beta, int ∗t, int ∗rnd, double ∗eps, int ∗emin, double ∗rmin, int ∗emax, double ∗rmax)
- double dlamc3_ (double ∗a, double ∗b)
- int dlamc4_ (int ∗emin, double ∗start, int ∗base)
- int dlamc5_ (int ∗beta, int ∗p, int ∗emin, int ∗ieee, int ∗emax, double ∗rmax)
- double pow_di (double ∗ap, int ∗bp)

4.67.1 Detailed Description

```
-- LAPACK auxiliary routine (version 2.0) --
Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
Courant Institute, Argonne National Lab, and Rice University
October 31, 1992
```

4.67.2 Define Documentation

4.67.2.1 #define abs(x) ((x) >= 0 ? (x) : -(x))

4.67.2.2 #define FALSE_ (0)

4.67.2.3 #define max(a, b) ((a) >= (b) ? (a) : (b))

4.67.2.4 #define min(a, b) ((a) <= (b) ? (a) : (b))

4.67.2.5 #define TRUE_ (1)

4.67.3 Function Documentation

4.67.3.1 int dlamc1_ (int * *beta*, int * *t*, int * *rnd*, int * *ieee1*)

```
Purpose
   =======

   DLAMC1 determines the machine parameters given by BETA, T, RND, and
   IEEE1.

   Arguments
   =========

   BETA    (output) INT
           The base of the machine.

   T       (output) INT
           The number of ( BETA ) digits in the mantissa.

   RND     (output) INT
           Specifies whether proper rounding  ( RND = .TRUE. )  or
           chopping  ( RND = .FALSE. )  occurs in addition. This may not

           be a reliable guide to the way in which the machine performs

           its arithmetic.

   IEEE1   (output) INT
           Specifies whether rounding appears to be done in the IEEE
           'round to nearest' style.

   Further Details
   ===============

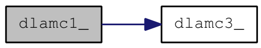
   The routine is based on the routine  ENVRON  by Malcolm and
   incorporates suggestions by Gentleman and Marovich. See

      Malcolm M. A. (1972) Algorithms to reveal properties of
         floating-point arithmetic. Comms. of the ACM, 15, 949-951.
```

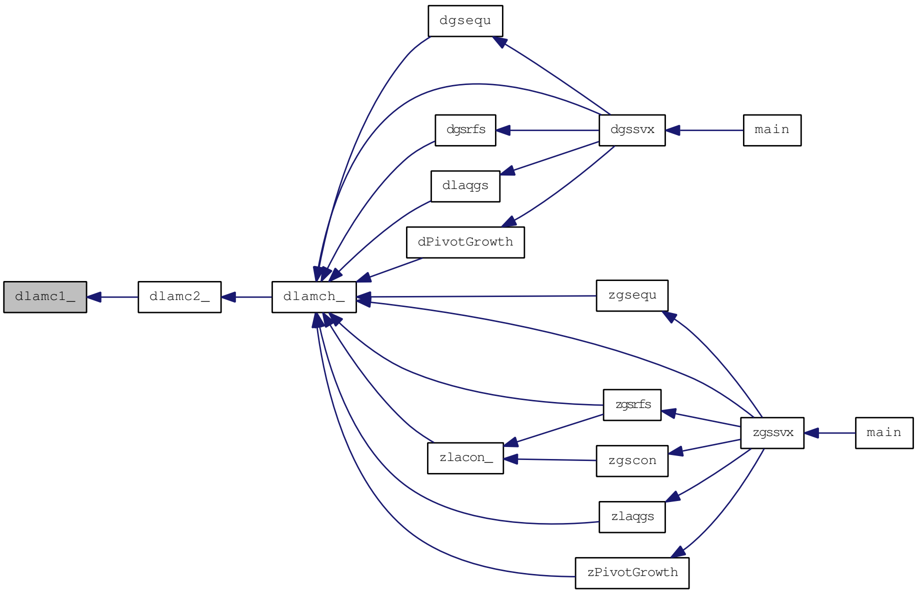
```
       Gentleman W. M. and Marovich S. B. (1974) More on algorithms
          that reveal properties of floating point arithmetic units.
          Comms. of the ACM, 17, 276-277.

   =====================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.67.3.2 int dlamc2_ (int ∗ *beta*, int ∗ *t*, int ∗ *rnd*, double ∗ *eps*, int ∗ *emin*, double ∗ *rmin*, int ∗ *emax*, double ∗ *rmax*)

```
Purpose
=======

DLAMC2 determines the machine parameters specified in its argument
list.

Arguments
=========

BETA    (output) INT
        The base of the machine.

T       (output) INT
        The number of ( BETA ) digits in the mantissa.
```

```
    RND     (output) INT
            Specifies whether proper rounding  ( RND = .TRUE. )  or
            chopping  ( RND = .FALSE. )  occurs in addition. This may not

            be a reliable guide to the way in which the machine performs

            its arithmetic.

    EPS     (output) DOUBLE PRECISION
            The smallest positive number such that

               fl( 1.0 - EPS ) .LT. 1.0,

            where fl denotes the computed value.

    EMIN    (output) INT
            The minimum exponent before (gradual) underflow occurs.

    RMIN    (output) DOUBLE PRECISION
            The smallest normalized number for the machine, given by
            BASE**( EMIN - 1 ), where  BASE  is the floating point value

            of BETA.

    EMAX    (output) INT
            The maximum exponent before overflow occurs.

    RMAX    (output) DOUBLE PRECISION
            The largest positive number for the machine, given by
            BASE**EMAX * ( 1 - EPS ), where  BASE  is the floating point

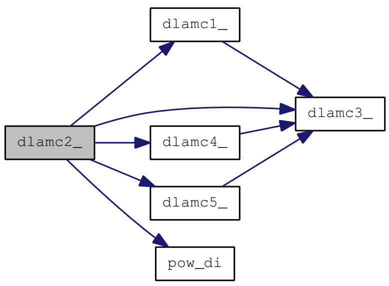
            value of BETA.

    Further Details
    ===============

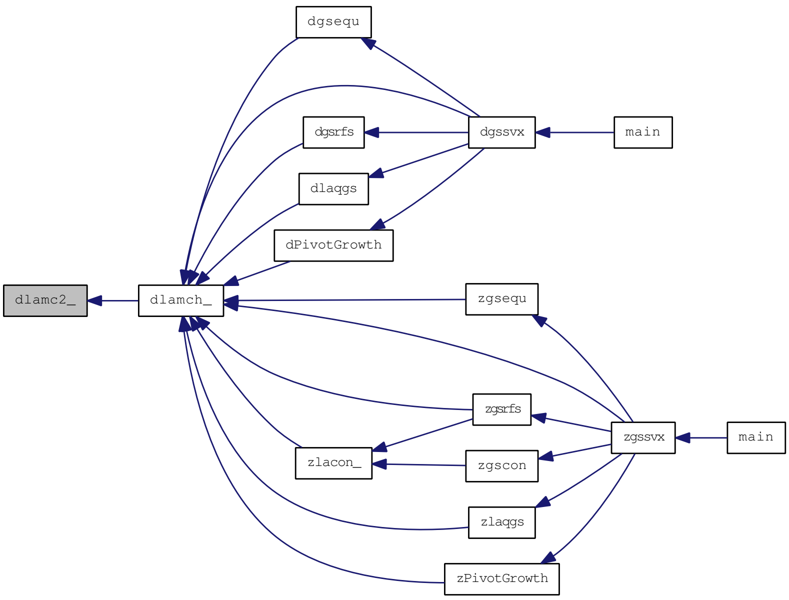
    The computation of  EPS  is based on a routine PARANOIA by
    W. Kahan of the University of California at Berkeley.

   =====================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.67.3.3 double dlamc3_ (double * *a*, double * *b*)

```
Purpose
=======


DLAMC3  is intended to force  A  and  B  to be stored prior to doing


the addition of  A  and  B ,  for use in situations where optimizers

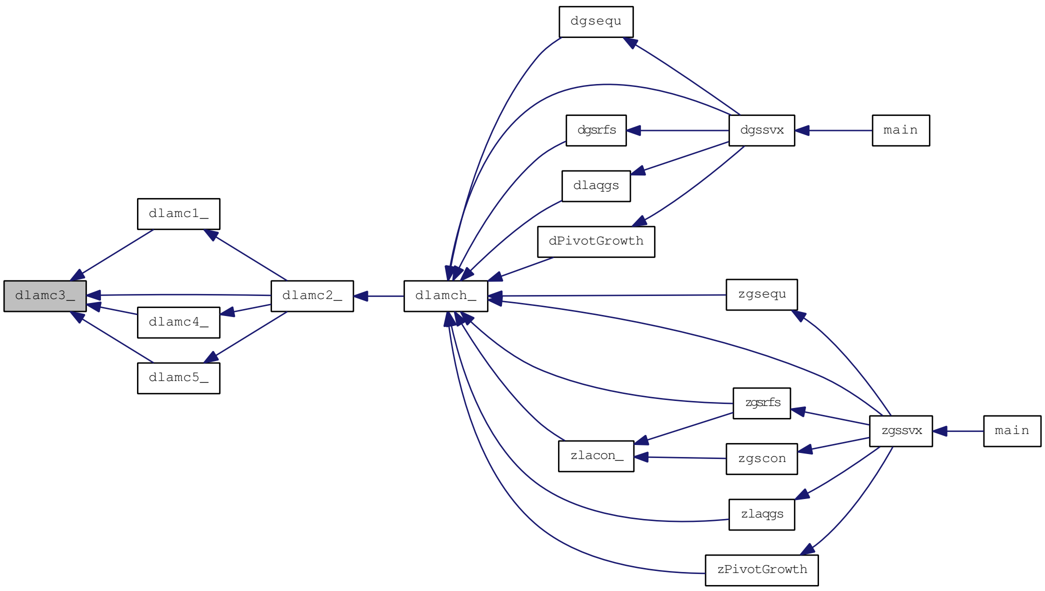

might hold one of these in a register.


Arguments
=========


A, B    (input) DOUBLE PRECISION
        The values A and B.


 =====================================================================
```

Here is the caller graph for this function:



4.67.3.4 int dlamc4_ (int * *emin*, double * *start*, int * *base*)

```
Purpose
=======

DLAMC4 is a service routine for DLAMC2.

Arguments
=========

EMIN    (output) EMIN
        The minimum exponent before (gradual) underflow, computed by

        setting A = START and dividing by BASE until the previous A
        can not be recovered.

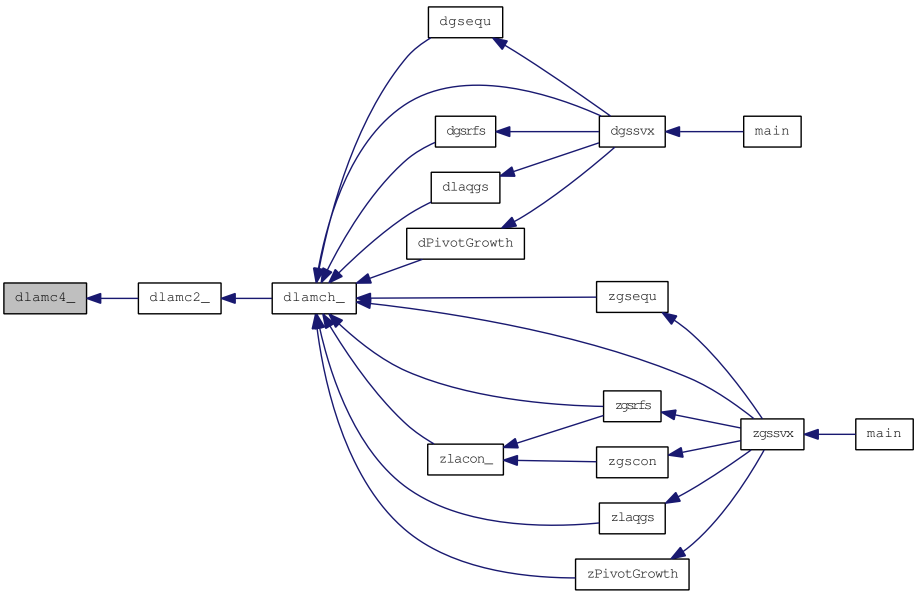
START   (input) DOUBLE PRECISION
        The starting point for determining EMIN.

BASE    (input) INT
        The base of the machine.

 =====================================================================
```

Here is the call graph for this function:

Here is the caller graph for this function:



4.67.3.5 int dlamc5_ (int * *beta*, int * *p*, int * *emin*, int * *ieee*, int * *emax*, double * *rmax*)

```
Purpose
=======

DLAMC5 attempts to compute RMAX, the largest machine floating-point
number, without overflow.  It assumes that EMAX + abs(EMIN) sum
approximately to a power of 2.  It will fail on machines where this
assumption does not hold, for example, the Cyber 205 (EMIN = -28625,

EMAX = 28718).  It will also fail if the value supplied for EMIN is
too large (i.e. too close to zero), probably with overflow.

Arguments
=========

BETA    (input) INT
        The base of floating-point arithmetic.

P       (input) INT
        The number of base BETA digits in the mantissa of a
        floating-point value.

EMIN    (input) INT
        The minimum exponent before (gradual) underflow.

IEEE    (input) INT
        A int flag specifying whether or not the arithmetic
        system is thought to comply with the IEEE standard.
```

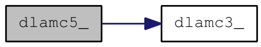
```
EMAX     (output) INT
         The largest exponent before overflow

RMAX     (output) DOUBLE PRECISION
         The largest machine floating-point number.

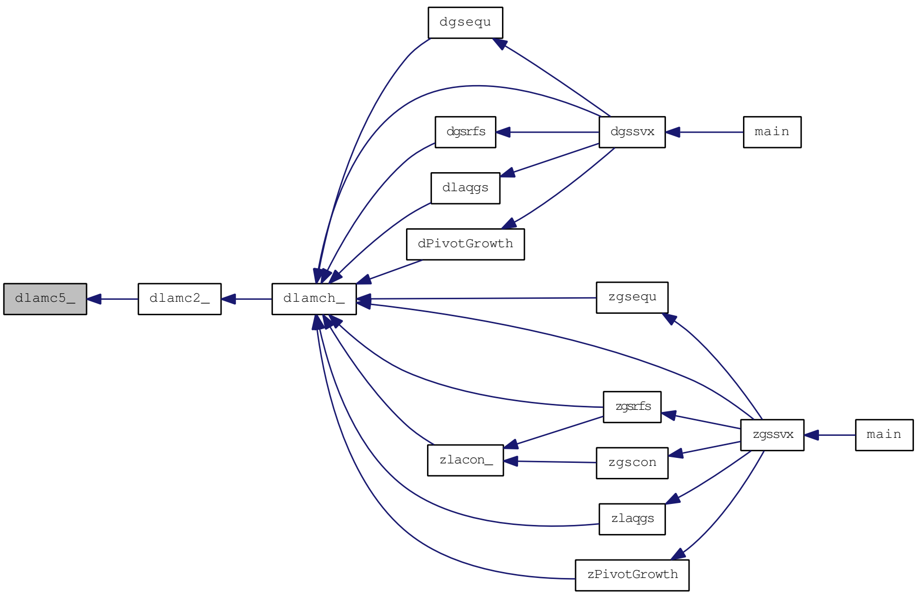
   =====================================================================

       First compute LEXP and UEXP, two powers of 2 that bound
       abs(EMIN). We then assume that EMAX + abs(EMIN) will sum
       approximately to the bound that is closest to abs(EMIN).
       (EMAX is the exponent of the required number RMAX).
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.67.3.6 double dlamch_ (char ∗ *cmach*)

```
Purpose
=======

DLAMCH determines double precision machine parameters.

Arguments
=========
```

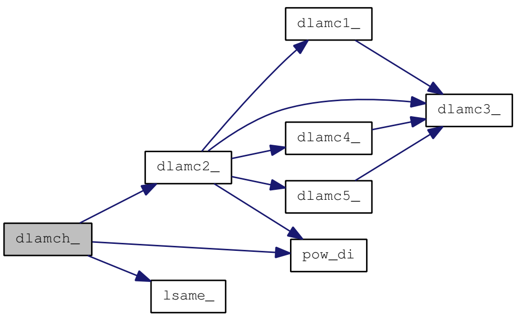
```
CMACH   (input) CHARACTER*1
        Specifies the value to be returned by DLAMCH:
        = 'E' or 'e',   DLAMCH := eps
        = 'S' or 's ,   DLAMCH := sfmin
        = 'B' or 'b',   DLAMCH := base
        = 'P' or 'p',   DLAMCH := eps*base
        = 'N' or 'n',   DLAMCH := t
        = 'R' or 'r',   DLAMCH := rnd
        = 'M' or 'm',   DLAMCH := emin
        = 'U' or 'u',   DLAMCH := rmin
        = 'L' or 'l',   DLAMCH := emax
        = 'O' or 'o',   DLAMCH := rmax

        where

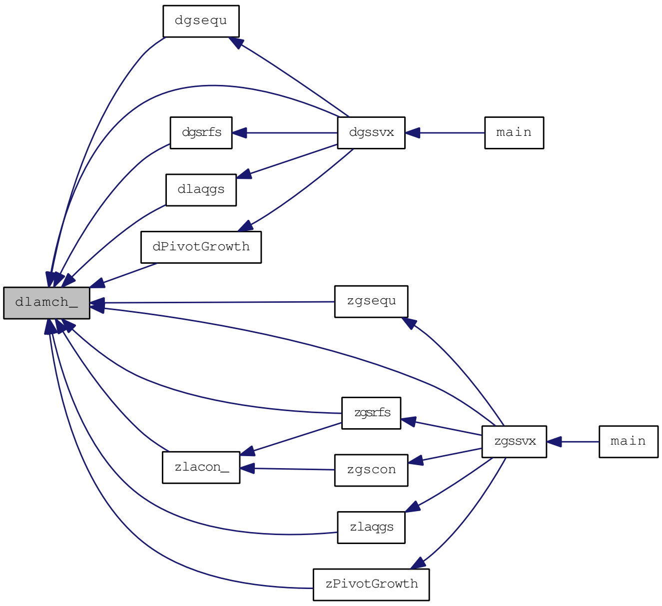
        eps   = relative machine precision
        sfmin = safe minimum, such that 1/sfmin does not overflow
        base  = base of the machine
        prec  = eps*base
        t     = number of (base) digits in the mantissa
        rnd   = 1.0 when rounding occurs in addition, 0.0 otherwise
        emin  = minimum exponent before (gradual) underflow
        rmin  = underflow threshold - base**(emin-1)
        emax  = largest exponent before overflow
        rmax  = overflow threshold  - (base**emax)*(1-eps)

  =====================================================================
```

Here is the call graph for this function:

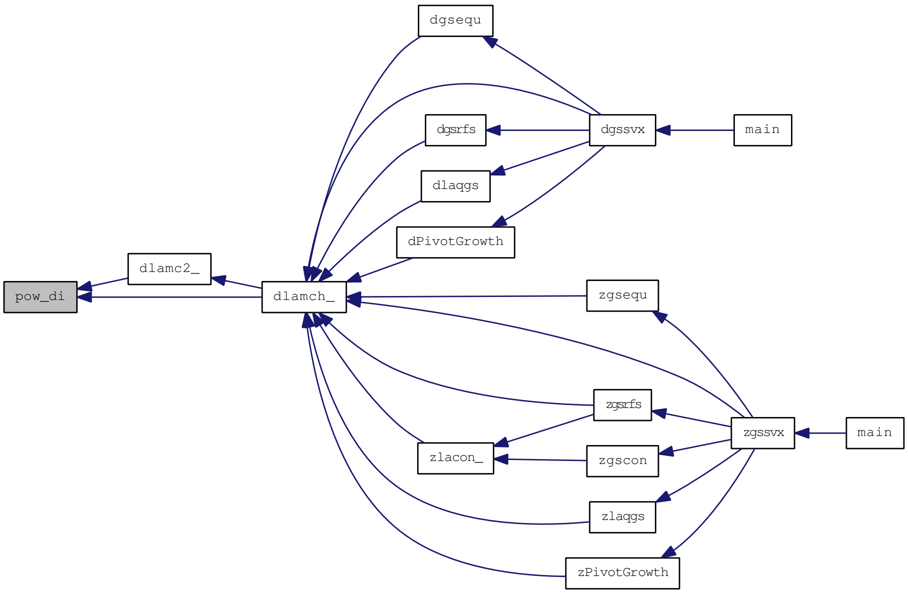


Here is the caller graph for this function:



4.67.3.7 double pow_di (double ∗ *ap*, int ∗ *bp*)

Here is the caller graph for this function:



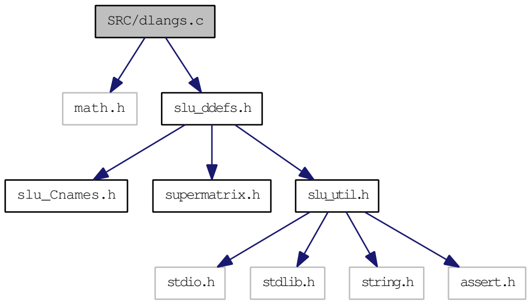
4.68 SRC/dlangs.c File Reference

Returns the value of the one norm.

`#include <math.h>`

`#include "slu_ddefs.h"`

Include dependency graph for dlangs.c:



Functions

- double dlangs (char ∗norm, SuperMatrix ∗A)

4.68.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997


Modified from lapack routine DLANGE
```

4.68.2 Function Documentation

4.68.2.1 double dlangs (char ∗ *norm*, SuperMatrix ∗ *A*)

```
Purpose
  =======

  DLANGS returns the value of the one norm, or the Frobenius norm, or
  the infinity norm, or the element of largest absolute value of a
  real matrix A.

  Description
  ===========

  DLANGE returns the value
```

```
   DLANGE = ( max(abs(A(i,j))), NORM = 'M' or 'm'
            (
            ( norm1(A),         NORM = '1', 'O' or 'o'
            (
            ( normI(A),         NORM = 'I' or 'i'
            (
            ( normF(A),         NORM = 'F', 'f', 'E' or 'e'

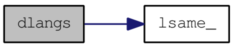
where  norm1  denotes the  one norm of a matrix (maximum column sum),
normI  denotes the  infinity norm  of a matrix  (maximum row sum) and
normF  denotes the  Frobenius norm of a matrix (square root of sum of
squares).  Note that  max(abs(A(i,j)))  is not a  matrix norm.

Arguments
=========

NORM    (input) CHARACTER*1
        Specifies the value to be returned in DLANGE as described above.
A       (input) SuperMatrix*
        The M by N sparse matrix A.

=====================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:

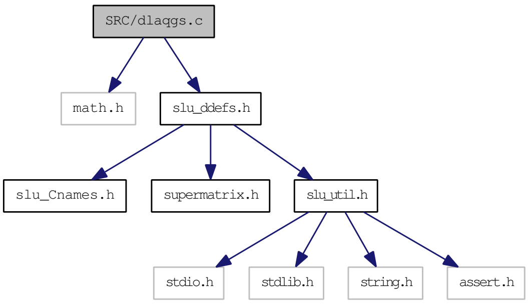
4.69 SRC/dlaqgs.c File Reference

Equlibrates a general sprase matrix.

`#include <math.h>`

`#include "slu_ddefs.h"`

Include dependency graph for dlaqgs.c:



Defines

- #define THRESH (0.1)

Functions

- void dlaqgs (SuperMatrix ∗A, double ∗r, double ∗c, double rowcnd, double colcnd, double amax, char ∗equed)

4.69.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997

Modified from LAPACK routine DLAQGE
```

4.69.2 Define Documentation

4.69.2.1 #define THRESH (0.1)

4.69.3 Function Documentation

4.69.3.1 void dlaqgs (SuperMatrix ∗ *A*, double ∗ *r*, double ∗ *c*, double *rowcnd*, double *colcnd*, double *amax*, char ∗ *equed*)

```
Purpose
=======
```

```
DLAQGS equilibrates a general sparse M by N matrix A using the row and
scaling factors in the vectors R and C.

See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

A       (input/output) SuperMatrix*
        On exit, the equilibrated matrix.  See EQUED for the form of
        the equilibrated matrix. The type of A can be:
 Stype = NC; Dtype = SLU_D; Mtype = GE.

R       (input) double*, dimension (A->nrow)
        The row scale factors for A.

C       (input) double*, dimension (A->ncol)
        The column scale factors for A.

ROWCND  (input) double
        Ratio of the smallest R(i) to the largest R(i).

COLCND  (input) double
        Ratio of the smallest C(i) to the largest C(i).

AMAX    (input) double
        Absolute value of largest matrix entry.

EQUED   (output) char*
        Specifies the form of equilibration that was done.
        = 'N':  No equilibration
        = 'R':  Row equilibration, i.e., A has been premultiplied by
                diag(R).
        = 'C':  Column equilibration, i.e., A has been postmultiplied
                by diag(C).
        = 'B':  Both row and column equilibration, i.e., A has been
                replaced by diag(R) * A * diag(C).

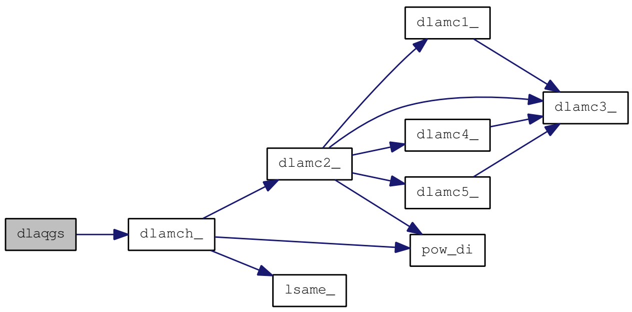
Internal Parameters
===================

THRESH is a threshold value used to decide if row or column scaling
should be done based on the ratio of the row or column scaling
factors.  If ROWCND < THRESH, row scaling is done, and if
COLCND < THRESH, column scaling is done.

LARGE and SMALL are threshold values used to decide if row scaling
should be done based on the absolute size of the largest matrix
element.  If AMAX > LARGE or AMAX < SMALL, row scaling is done.

=====================================================================
```

Here is the call graph for this function:



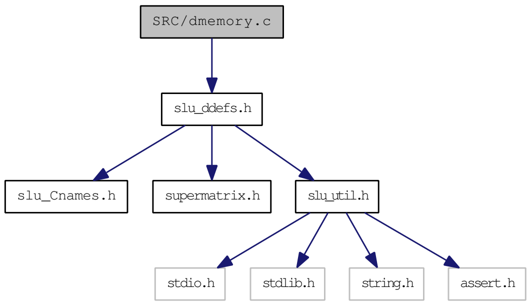
Here is the caller graph for this function:

4.70 SRC/dmemory.c File Reference

Memory details.

`#include "slu_ddefs.h"`

Include dependency graph for dmemory.c:



Data Structures

- struct e_node

  Headers for 4 types of dynamatically managed memory.

- struct LU_stack_t

Defines

- #define NO_MEMTYPE 4
- #define GluIntArray(n) (5 ∗ (n) + 5)
- #define StackFull(x) (x + stack.used >= stack.size)
- #define NotDoubleAlign(addr) ((long int)addr & 7)
- #define DoubleAlign(addr) (((long int)addr + 7) & ∼7L)
- #define TempSpace(m, w)
- #define Reduce(alpha) ((alpha + 1) / 2)

Typedefs

- typedef struct e_node ExpHeader

  Headers for 4 types of dynamatically managed memory.

Functions

- void ∗ dexpand (int ∗prev_len,MemType type,int len_to_copy,int keep_prev,GlobalLU_t ∗Glu)

  Expand the existing storage to accommodate more fill-ins.

- int dLUWorkInit (int m, int n, int panel_size, int ∗∗iworkptr, double ∗∗dworkptr, LU_space_t MemModel)

Allocate known working storage. Returns 0 if success, otherwise returns the number of bytes allocated so far when failure occurred.

- void copy_mem_double (int, void ∗, void ∗)
- void dStackCompress (GlobalLU_t ∗Glu)

  Compress the work[] array to remove fragmentation.

- void dSetupSpace (void ∗work, int lwork, LU_space_t ∗MemModel)

  Setup the memory model to be used for factorization.

- void ∗ duser_malloc (int, int)
- void duser_free (int, int)
- void copy_mem_int (int, void ∗, void ∗)
- void user_bcopy (char ∗, char ∗, int)
- int dQuerySpace (SuperMatrix ∗L, SuperMatrix ∗U, mem_usage_t ∗mem_usage)
- int dLUMemInit (fact_t fact, void ∗work, int lwork, int m, int n, int annz, int panel_size, SuperMatrix ∗L, SuperMatrix ∗U, GlobalLU_t ∗Glu, int ∗∗iwork, double ∗∗dwork)

  Allocate storage for the data structures common to all factor routines.

- void dSetRWork (int m, int panel_size, double ∗dworkptr, double ∗∗dense, double ∗∗tempv)

  Set up pointers for real working arrays.

- void dLUWorkFree (int ∗iwork, double ∗dwork, GlobalLU_t ∗Glu)

  Free the working storage used by factor routines.

- int dLUMemXpand (int jcol, int next, MemType mem_type, int ∗maxlen, GlobalLU_t ∗Glu)

  Expand the data structures for L and U during the factorization.

- void dallocateA (int n, int nnz, double ∗∗a, int ∗∗asub, int ∗∗xa)

  Allocate storage for original matrix A.

- double ∗ doubleMalloc (int n)
- double ∗ doubleCalloc (int n)
- int dmemory_usage (const int nzlmax, const int nzumax, const int nzlumax, const int n)

Variables

- static ExpHeader ∗ expanders = 0
- static LU_stack_t stack
- static int no_expand

4.70.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003
```

4.70.2 Define Documentation

4.70.2.1 #define DoubleAlign(addr) (((long int)addr + 7) & ∼7L)

4.70.2.2 #define GluIntArray(n) (5 ∗ (n) + 5)

4.70.2.3 #define NO_MEMTYPE 4

4.70.2.4 #define NotDoubleAlign(addr) ((long int)addr & 7)

4.70.2.5 #define Reduce(alpha) ((alpha + 1) / 2)

4.70.2.6 #define StackFull(x) (x + stack.used >= stack.size)

4.70.2.7 #define TempSpace(m, w)

Value:

```
( (2*w + 4 + NO_MARKER) * m * sizeof(int) + \
            (w + 1) * m * sizeof(double) )
```

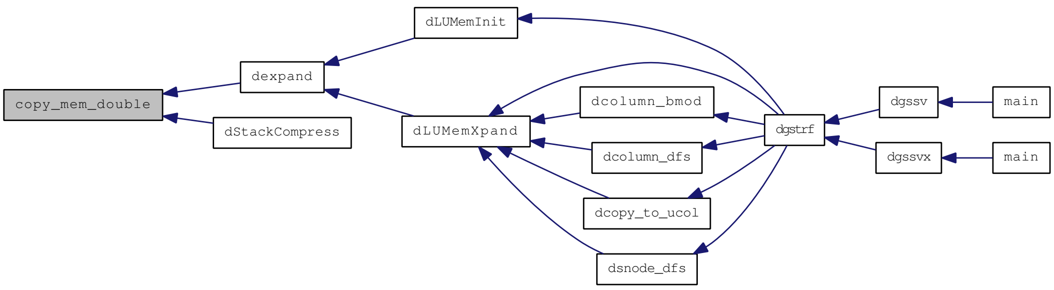
4.70.3 Typedef Documentation

4.70.3.1 typedef struct e_node ExpHeader

4.70.4 Function Documentation

4.70.4.1 void copy_mem_double (int *howmany*, void ∗ *old*, void ∗ *new*)

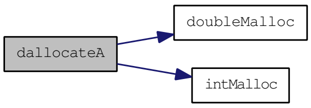
Here is the caller graph for this function:



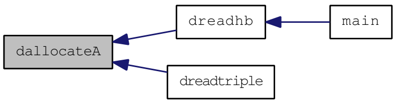
4.70.4.2 void copy_mem_int (int, void ∗, void ∗)

4.70.4.3 void dallocateA (int *n*, int *nnz*, double ∗∗ *a*, int ∗∗ *asub*, int ∗∗ *xa*)

Here is the call graph for this function:

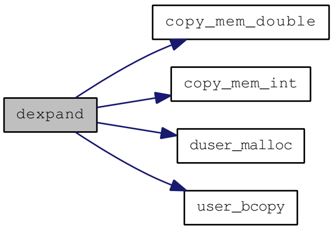


Here is the caller graph for this function:

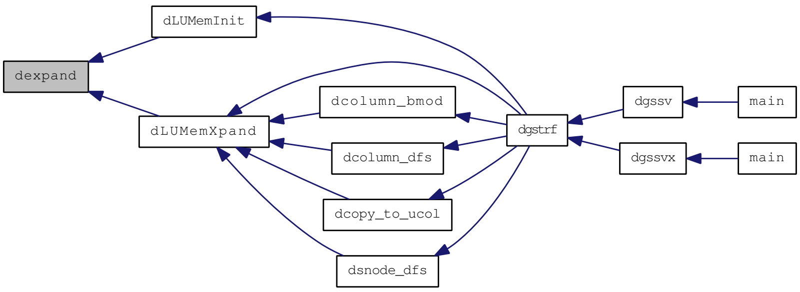


4.70.4.4 void ∗ dexpand (int ∗ *prev_len*, MemType *type*, int *len_to_copy*, int *keep_prev*, GlobalLU_t ∗ *Glu*)

Here is the call graph for this function:



Here is the caller graph for this function:

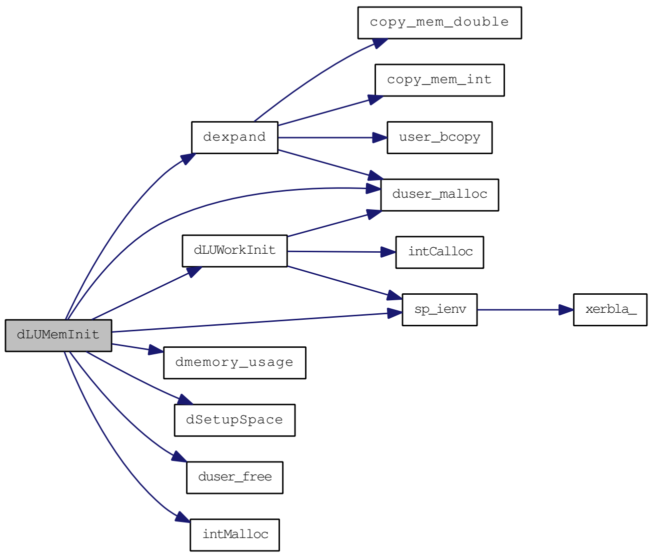


4.70.4.5 int dLUMemInit (fact_t *fact*, void ∗ *work*, int *lwork*, int *m*, int *n*, int *annz*, int *panel_size*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, GlobalLU_t ∗ *Glu*, int ∗∗ *iwork*, double ∗∗ *dwork*)

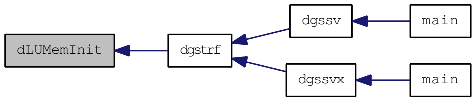
Memory-related.

```
For those unpredictable size, make a guess as FILL * nnz(A).
Return value:
    If lwork = -1, return the estimated amount of space required, plus n;
    otherwise, return the amount of space actually allocated when
    memory allocation failure occurred.
```

Here is the call graph for this function:



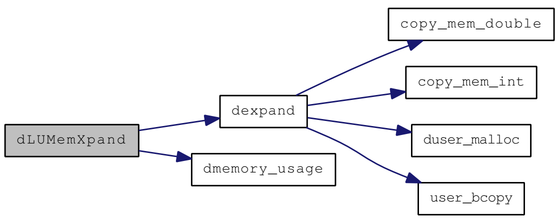
Here is the caller graph for this function:



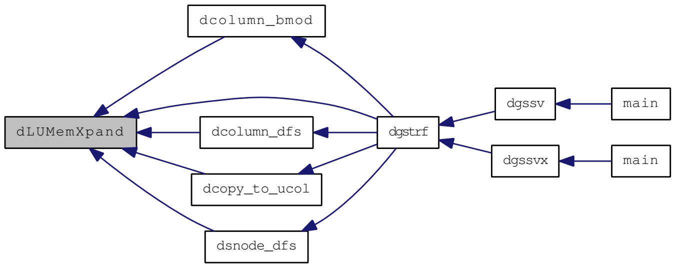
4.70.4.6 int dLUMemXpand (int *jcol*, int *next*, MemType *mem_type*, int ∗ *maxlen*, GlobalLU_t ∗ *Glu*)

```
Return value:   0 - successful return
              > 0 - number of bytes allocated when run out of space
```

Here is the call graph for this function:

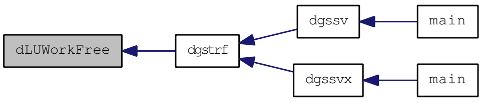


Here is the caller graph for this function:



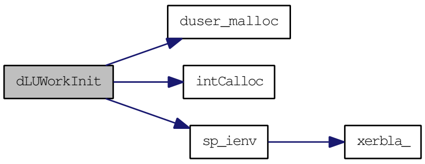
4.70.4.7 void dLUWorkFree (int ∗ *iwork*, double ∗ *dwork*, GlobalLU_t ∗ *Glu*)

Here is the caller graph for this function:

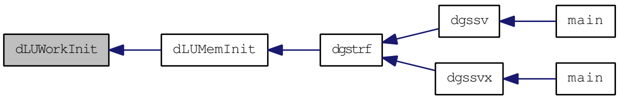


4.70.4.8 int dLUWorkInit (int *m*, int *n*, int *panel_size*, int ∗∗ *iworkptr*, double ∗∗ *dworkptr*, LU_space_t *MemModel*)

Here is the call graph for this function:

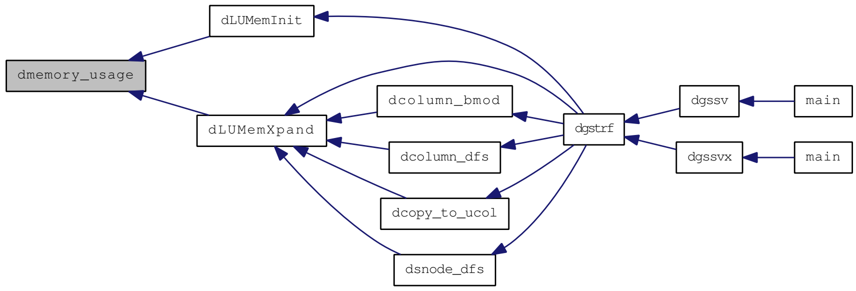


Here is the caller graph for this function:



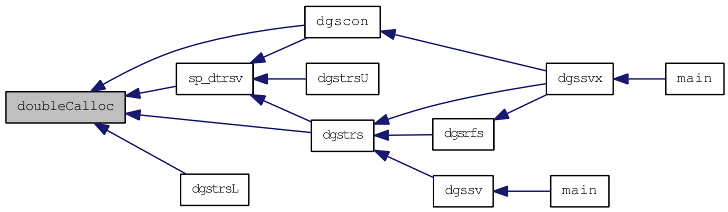
4.70.4.9 int dmemory_usage (const int *nzlmax*, const int *nzumax*, const int *nzlumax*, const int *n*)

Here is the caller graph for this function:



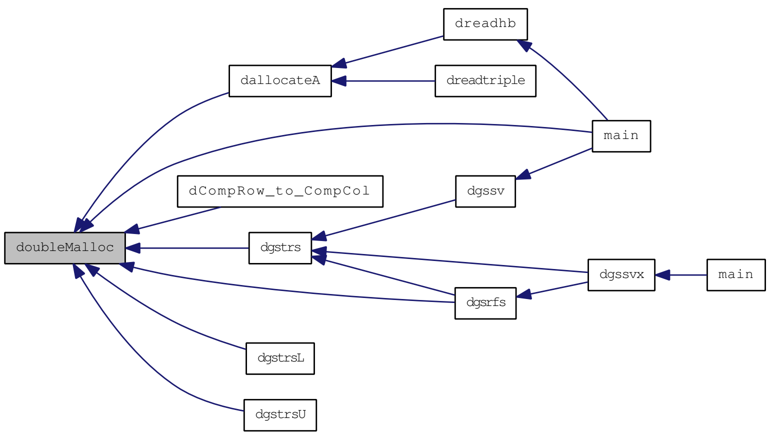
4.70.4.10 double* doubleCalloc (int *n*)

Here is the caller graph for this function:



4.70.4.11 double* doubleMalloc (int *n*)

Here is the caller graph for this function:

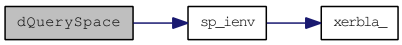


4.70.4.12 int dQuerySpace (SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, mem_usage_t ∗ *mem_usage*)

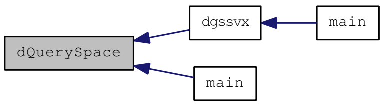
```
mem_usage consists of the following fields:
```

- `for_lu (float)`
  `The amount of space used in bytes for the L data structures.`
- `total_needed (float)`
  `The amount of space needed in bytes to perform factorization.`
- `expansions (int)`
  `Number of memory expansions during the LU factorization.`

Here is the call graph for this function:

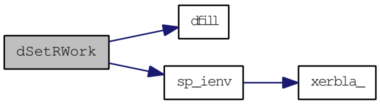


Here is the caller graph for this function:

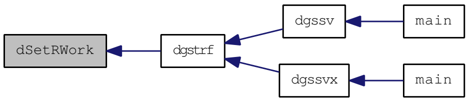


4.70.4.13 void dSetRWork (int *m*, int *panel_size*, double ∗ *dworkptr*, double ∗∗ *dense*, double ∗∗ *tempv*)

Here is the call graph for this function:



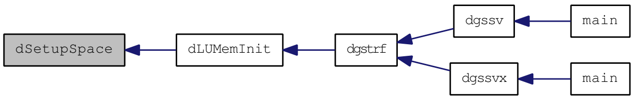
Here is the caller graph for this function:



4.70.4.14 void dSetupSpace (void ∗ *work*, int *lwork*, LU_space_t ∗ *MemModel*)

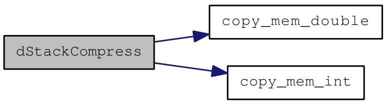
lwork = 0: use system malloc; lwork > 0: use user-supplied work[] space.

Here is the caller graph for this function:



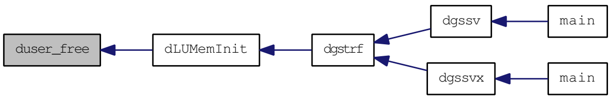
4.70.4.15 void dStackCompress (GlobalLU_t * *Glu*)

Here is the call graph for this function:



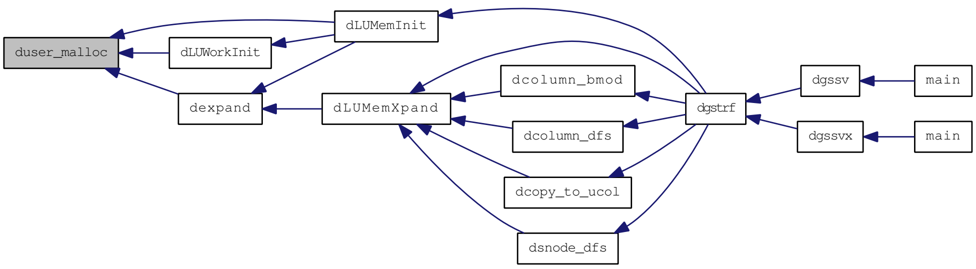
4.70.4.16 void duser_free (int *bytes*, int *which_end*)

Here is the caller graph for this function:



4.70.4.17 void * duser_malloc (int *bytes*, int *which_end*)

Here is the caller graph for this function:



4.70.4.18 void user_bcopy (char *, char *, int)

4.70.5 Variable Documentation

4.70.5.1 ExpHeader* expanders = 0 `[static]`

4.70.5.2 int no_expand `[static]`

4.70.5.3 LU_stack_t stack `[static]`

4.71 SRC/dmyblas2.c File Reference

Level 2 Blas operations.

Functions

- void dlsolve (int ldm, int ncol, double ∗M, double ∗rhs)

  Solves a dense UNIT lower triangular system.

- void dusolve (int ldm, int ncol, double ∗M, double ∗rhs)

  Solves a dense upper triangular system.

- void dmatvec (int ldm, int nrow, int ncol, double ∗M, double ∗vec, double ∗Mxvec)

  Performs a dense matrix-vector multiply: Mxvec = Mxvec + M ∗ vec.

4.71.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997
```

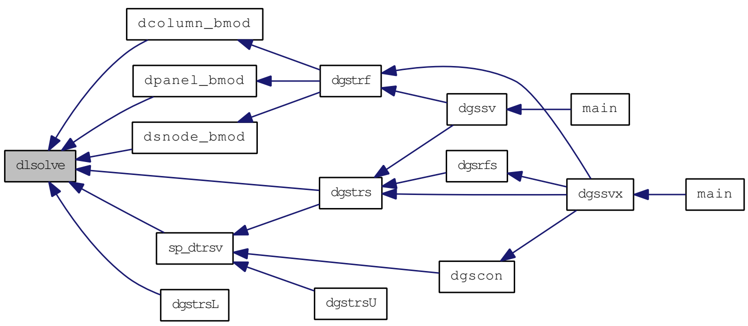
Purpose: Level 2 BLAS operations: solves and matvec, written in C. Note: This is only used when the system lacks an efficient BLAS library.

4.71.2 Function Documentation

4.71.2.1 void dlsolve (int *ldm*, int *ncol*, double ∗ *M*, double ∗ *rhs*)

The unit lower triangular matrix is stored in a 2D array M(1:nrow,1:ncol). The solution will be returned in the rhs vector.

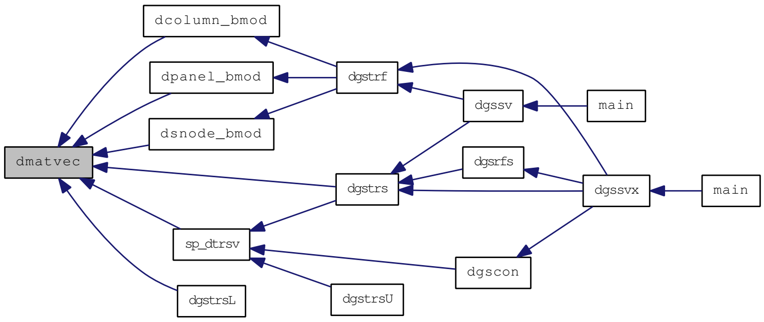
Here is the caller graph for this function:



4.71.2.2 void dmatvec (int *ldm*, int *nrow*, int *ncol*, double ∗ *M*, double ∗ *vec*, double ∗ *Mxvec*)

The input matrix is M(1:nrow,1:ncol); The product is returned in Mxvec[].

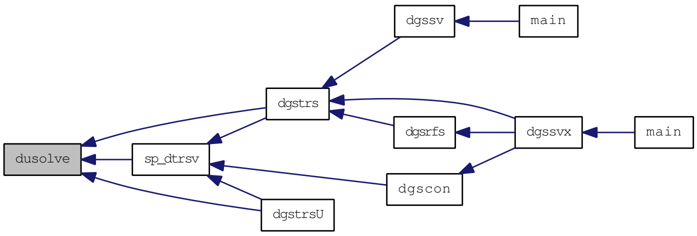
Here is the caller graph for this function:



4.71.2.3 void dusolve (int *ldm*, int *ncol*, double ∗ *M*, double ∗ *rhs*)

The upper triangular matrix is stored in a 2-dim array M(1:ldm,1:ncol). The solution will be returned in the rhs vector.

Here is the caller graph for this function:



4.72 SRC/dpanel_bmod.c File Reference

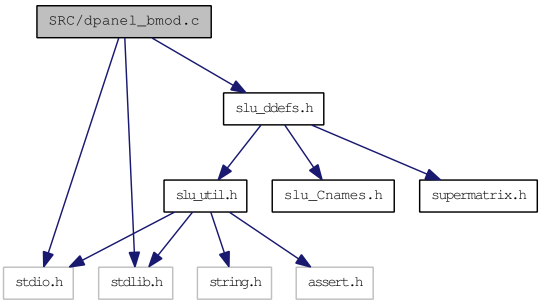
Performs numeric block updates.

`#include <stdio.h>`

`#include <stdlib.h>`

`#include "slu_ddefs.h"`

Include dependency graph for dpanel_bmod.c:



Functions

- void dlsolve (int, int, double ∗, double ∗)

  Solves a dense UNIT lower triangular system.

- void dmatvec (int, int, int, double ∗, double ∗, double ∗)

  Performs a dense matrix-vector multiply: Mxvec = Mxvec + M ∗ vec.

- void dcheck_tempv ()
- void dpanel_bmod (const int m, const int w, const int jcol, const int nseg, double ∗dense, double ∗tempv, int ∗segrep, int ∗repfnz, GlobalLU_t ∗Glu, SuperLUStat_t ∗stat)

4.72.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.72.2 Function Documentation

4.72.2.1 void dcheck_tempv ()

4.72.2.2 void dlsolve (int *ldm*, int *ncol*, double ∗ *M*, double ∗ *rhs*)

The unit lower triangular matrix is stored in a 2D array M(1:nrow,1:ncol). The solution will be returned in the rhs vector.

4.72.2.3 void dmatvec (int *ldm*, int *nrow*, int *ncol*, double ∗ *M*, double ∗ *vec*, double ∗ *Mxvec*)

The input matrix is M(1:nrow,1:ncol); The product is returned in Mxvec[].

4.72.2.4 void dpanel_bmod (const int *m*, const int *w*, const int *jcol*, const int *nseg*, double ∗ *dense*, double ∗ *tempv*, int ∗ *segrep*, int ∗ *repfnz*, GlobalLU_t ∗ *Glu*, SuperLUStat_t ∗ *stat*)

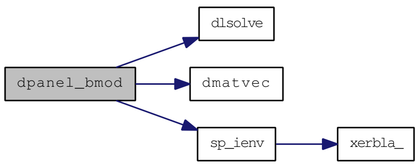
```
Purpose
=======

   Performs numeric block updates (sup-panel) in topological order.
   It features: col-col, 2cols-col, 3cols-col, and sup-col updates.
   Special processing on the supernodal portion of L[*,j]

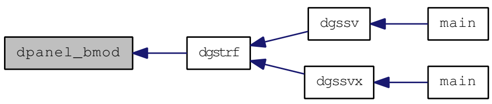
   Before entering this routine, the original nonzeros in the panel
   were already copied into the spa[m,w].

   Updated/Output parameters-
   dense[0:m-1,w]: L[*,j:j+w-1] and U[*,j:j+w-1] are returned
   collectively in the m-by-w vector dense[*].
```

Here is the call graph for this function:



Here is the caller graph for this function:

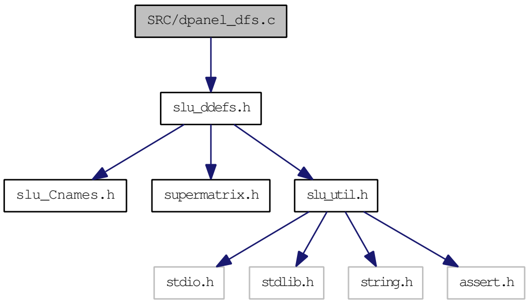


4.73 SRC/dpanel_dfs.c File Reference

Peforms a symbolic factorization on a panel of symbols.

`#include "slu_ddefs.h"`

Include dependency graph for dpanel_dfs.c:



Functions

- void dpanel_dfs (const int m, const int w, const int jcol, SuperMatrix ∗A, int ∗perm_r, int ∗nseg, double ∗dense, int ∗panel_lsub, int ∗segrep, int ∗repfnz, int ∗xprune, int ∗marker, int ∗parent, int ∗xplore, GlobalLU_t ∗Glu)

4.73.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.73.2 Function Documentation

4.73.2.1 void dpanel_dfs (const int *m*, const int *w*, const int *jcol*, SuperMatrix ∗ *A*, int ∗ *perm_r*, int ∗ *nseg*, double ∗ *dense*, int ∗ *panel_lsub*, int ∗ *segrep*, int ∗ *repfnz*, int ∗ *xprune*, int ∗ *marker*, int ∗ *parent*, int ∗ *xplore*, GlobalLU_t ∗ *Glu*)

```
Purpose
=======
```

```
Performs a symbolic factorization on a panel of columns [jcol, jcol+w).

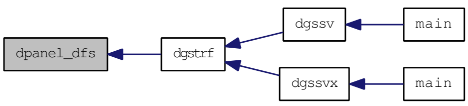
A supernode representative is the last column of a supernode.
The nonzeros in U[*,j] are segments that end at supernodal
representatives.

The routine returns one list of the supernodal representatives
in topological order of the dfs that generates them. This list is
a superset of the topological order of each individual column within
the panel.
The location of the first nonzero in each supernodal segment
(supernodal entry location) is also returned. Each column has a
separate list for this purpose.

Two marker arrays are used for dfs:
  marker[i] == jj, if i was visited during dfs of current column jj;
  marker1[i] >= jcol, if i was visited by earlier columns in this panel;

marker: A-row --> A-row/col (0/1)
repfnz: SuperA-col --> PA-row
parent: SuperA-col --> SuperA-col
xplore: SuperA-col --> index to L-structure
```

Here is the caller graph for this function:



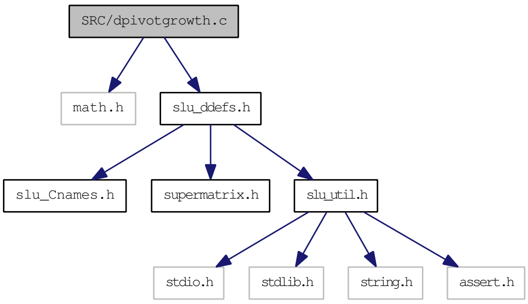
4.74 SRC/dpivotgrowth.c File Reference

Computes the reciprocal pivot growth factor.

`#include <math.h>`

`#include "slu_ddefs.h"`

Include dependency graph for dpivotgrowth.c:



Functions

- double dPivotGrowth (int ncols, SuperMatrix *A, int *perm_c, SuperMatrix *L, SuperMatrix *U)

4.74.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997
```

4.74.2 Function Documentation

4.74.2.1 double dPivotGrowth (int *ncols*, SuperMatrix * *A*, int * *perm_c*, SuperMatrix * *L*, SuperMatrix * *U*)

```
Purpose
=======

Compute the reciprocal pivot growth factor of the leading ncols columns
of the matrix, using the formula:
    min_j ( max_i(abs(A_ij)) / max_i(abs(U_ij)) )

Arguments
=========

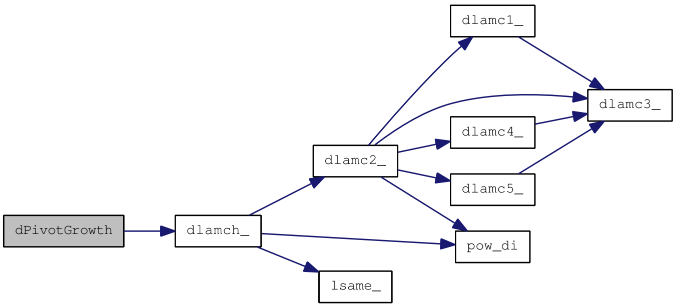
ncols    (input) int
         The number of columns of matrices A, L and U.
```

```
 A       (input) SuperMatrix*
    Original matrix A, permuted by columns, of dimension
         (A->nrow, A->ncol). The type of A can be:
         Stype = NC; Dtype = SLU_D; Mtype = GE.

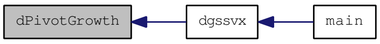
 L       (output) SuperMatrix*
         The factor L from the factorization Pr*A=L*U; use compressed row
         subscripts storage for supernodes, i.e., L has type:
         Stype = SC; Dtype = SLU_D; Mtype = TRLU.

 U       (output) SuperMatrix*
    The factor U from the factorization Pr*A*Pc=L*U. Use column-wise
         storage scheme, i.e., U has types: Stype = NC;
         Dtype = SLU_D; Mtype = TRU.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.75 SRC/dpivotL.c File Reference

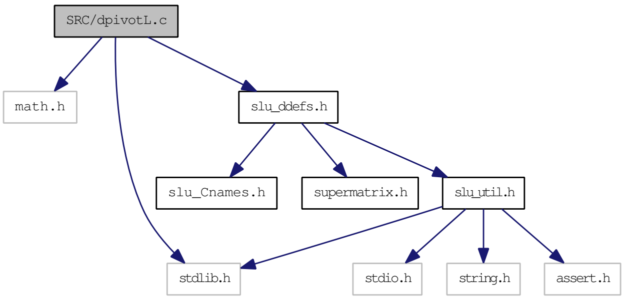
Performs numerical pivoting.

`#include <math.h>`

`#include <stdlib.h>`

`#include "slu_ddefs.h"`

Include dependency graph for dpivotL.c:



Functions

- int dpivotL (const int jcol, const double u, int ∗usepr, int ∗perm_r, int ∗iperm_r, int ∗iperm_c, int ∗pivrow, GlobalLU_t ∗Glu, SuperLUStat_t ∗stat)

4.75.1 Detailed Description

```
 -- SuperLU routine (version 3.0) --
 Univ. of California Berkeley, Xerox Palo Alto Research Center,
 and Lawrence Berkeley National Lab.
 October 15, 2003

 Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

 THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
 EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

 Permission is hereby granted to use or copy this program for any
 purpose, provided the above notices are retained on all copies.
 Permission to modify the code and to distribute modified code is
 granted, provided the above notices are retained, and a notice that
 the code was modified is included with the above copyright notice.
```

4.75.2 Function Documentation

4.75.2.1 int dpivotL (const int *jcol*, const double *u*, int ∗ *usepr*, int ∗ *perm_r*, int ∗ *iperm_r*, int ∗ *iperm_c*, int ∗ *pivrow*, GlobalLU_t ∗ *Glu*, SuperLUStat_t ∗ *stat*)

```
 Purpose
```

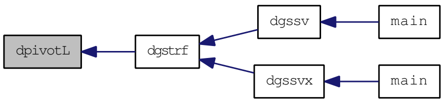
```
=======
  Performs the numerical pivoting on the current column of L,
  and the CDIV operation.

  Pivot policy:
  (1) Compute thresh = u * max_(i>=j) abs(A_ij);
  (2) IF user specifies pivot row k and abs(A_kj) >= thresh THEN
          pivot row = k;
      ELSE IF abs(A_jj) >= thresh THEN
          pivot row = j;
      ELSE
          pivot row = m;

  Note: If you absolutely want to use a given pivot order, then set u=0.0.

  Return value: 0      success;
                i > 0  U(i,i) is exactly zero.
```

Here is the caller graph for this function:

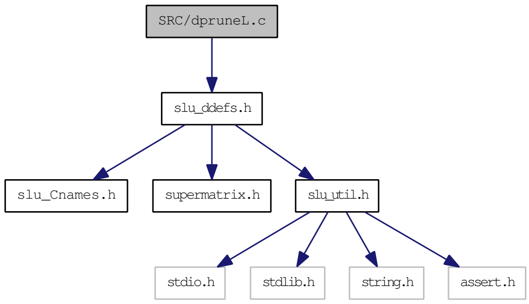


4.76 SRC/dpruneL.c File Reference

Prunes the L-structure.

`#include "slu_ddefs.h"`

Include dependency graph for dpruneL.c:



Functions

- void dpruneL (const int jcol, const int ∗perm_r, const int pivrow, const int nseg, const int ∗segrep, const int ∗repfnz, int ∗xprune, GlobalLU_t ∗Glu)

4.76.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997


Copyright (c) 1994 by Xerox Corporation.  All rights reserved.


THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.


Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
*
```

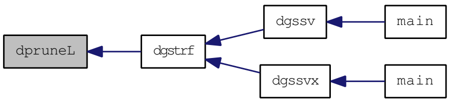
4.76.2 Function Documentation

4.76.2.1 void dpruneL (const int *jcol*, const int ∗ *perm_r*, const int *pivrow*, const int *nseg*, const int ∗ *segrep*, const int ∗ *repfnz*, int ∗ *xprune*, GlobalLU_t ∗ *Glu*)

```
Purpose
=======
  Prunes the L-structure of supernodes whose L-structure
```

```
contains the current pivot row "pivrow"
```

Here is the caller graph for this function:



4.77 SRC/dreadhb.c File Reference

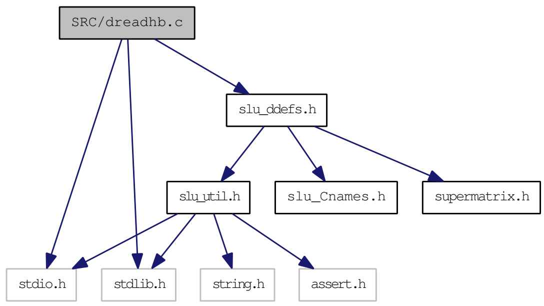
Read a matrix stored in Harwell-Boeing format.

`#include <stdio.h>`

`#include <stdlib.h>`

`#include "slu_ddefs.h"`

Include dependency graph for dreadhb.c:



Functions

- int dDumpLine (FILE ∗fp)

  Eat up the rest of the current line.

- int dParseIntFormat (char ∗buf, int ∗num, int ∗size)
- int dParseFloatFormat (char ∗buf, int ∗num, int ∗size)
- int dReadVector (FILE ∗fp, int n, int ∗where, int perline, int persize)
- int dReadValues (FILE ∗fp, int n, double ∗destination, int perline, int persize)
- void dreadhb (int ∗nrow, int ∗ncol, int ∗nonz, double ∗∗nzval, int ∗∗rowind, int ∗∗colptr)

  Auxiliary routines.

4.77.1 Detailed Description

```
 -- SuperLU routine (version 2.0) --
 Univ. of California Berkeley, Xerox Palo Alto Research Center,
 and Lawrence Berkeley National Lab.
 November 15, 1997

 Purpose
 =======

 Read a DOUBLE PRECISION matrix stored in Harwell-Boeing format
 as described below.

 Line 1 (A72,A8)
   Col. 1 - 72   Title (TITLE)
Col. 73 - 80  Key (KEY)
```

```
 Line 2 (5I14)
  Col. 1 - 14   Total number of lines excluding header (TOTCRD)
  Col. 15 - 28  Number of lines for pointers (PTRCRD)
  Col. 29 - 42  Number of lines for row (or variable) indices (INDCRD)
  Col. 43 - 56  Number of lines for numerical values (VALCRD)
Col. 57 - 70  Number of lines for right-hand sides (RHSCRD)
                    (including starting guesses and solution vectors
       if present)
                  (zero indicates no right-hand side data is present)


 Line 3 (A3, 11X, 4I14)
    Col. 1 - 3    Matrix type (see below) (MXTYPE)
  Col. 15 - 28  Number of rows (or variables) (NROW)
  Col. 29 - 42  Number of columns (or elements) (NCOL)
Col. 43 - 56  Number of row (or variable) indices (NNZERO)
              (equal to number of entries for assembled matrices)
  Col. 57 - 70  Number of elemental matrix entries (NELTVL)
              (zero in the case of assembled matrices)
 Line 4 (2A16, 2A20)
  Col. 1 - 16   Format for pointers (PTRFMT)
Col. 17 - 32  Format for row (or variable) indices (INDFMT)
Col. 33 - 52  Format for numerical values of coefficient matrix (VALFMT)
  Col. 53 - 72 Format for numerical values of right-hand sides (RHSFMT)


 Line 5 (A3, 11X, 2I14) Only present if there are right-hand sides present
     Col. 1        Right-hand side type:
            F for full storage or M for same format as matrix
     Col. 2        G if a starting vector(s) (Guess) is supplied. (RHSTYP)
     Col. 3        X if an exact solution vector(s) is supplied.
Col. 15 - 28  Number of right-hand sides (NRHS)
Col. 29 - 42  Number of row indices (NRHSIX)
                 (ignored in case of unassembled matrices)


 The three character type field on line 3 describes the matrix type.
 The following table lists the permitted values for each of the three
 characters. As an example of the type field, RSA denotes that the matrix
 is real, symmetric, and assembled.


 First Character:
R Real matrix
C Complex matrix
P Pattern only (no numerical values supplied)


 Second Character:
S Symmetric
U Unsymmetric
H Hermitian
Z Skew symmetric
R Rectangular


 Third Character:
A Assembled
E Elemental matrices (unassembled)
```

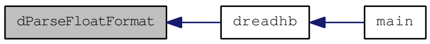
4.77.2 Function Documentation

4.77.2.1 int dDumpLine (FILE * *fp*)

Here is the caller graph for this function:

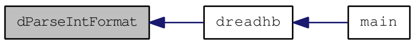
4.77.2.2 int dParseFloatFormat (char * *buf*, int * *num*, int * *size*)

Here is the caller graph for this function:



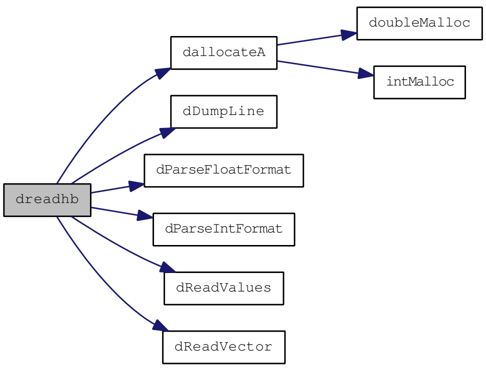
4.77.2.3 int dParseIntFormat (char * *buf*, int * *num*, int * *size*)

Here is the caller graph for this function:

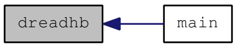


4.77.2.4 void dreadhb (int * *nrow*, int * *ncol*, int * *nonz*, double ** *nzval*, int ** *rowind*, int ** *colptr*)

Here is the call graph for this function:

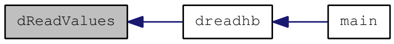


Here is the caller graph for this function:



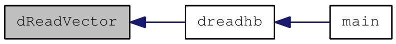
4.77.2.5 int dReadValues (FILE * *fp*, int *n*, double * *destination*, int *perline*, int *persize*)

Here is the caller graph for this function:



4.77.2.6 int dReadVector (FILE * *fp*, int *n*, int * *where*, int *perline*, int *persize*)

Here is the caller graph for this function:

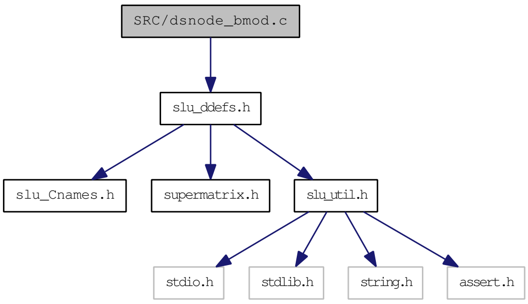


4.78 SRC/dsnode_bmod.c File Reference

Performs numeric block updates within the relaxed snode.

`#include "slu_ddefs.h"`

Include dependency graph for dsnode_bmod.c:



Functions

- int dsnode_bmod (const int jcol, const int jsupno, const int fsupc, double ∗dense, double ∗tempv, GlobalLU_t ∗Glu, SuperLUStat_t ∗stat)

  Performs numeric block updates within the relaxed snode.

4.78.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003


Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

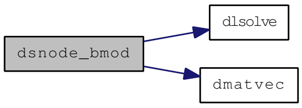

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.


Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

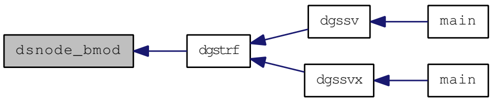
4.78.2 Function Documentation

4.78.2.1 int dsnode_bmod (const int *jcol*, const int *jsupno*, const int *fsupc*, double ∗ *dense*, double ∗ *tempv*, GlobalLU_t ∗ *Glu*, SuperLUStat_t ∗ *stat*)

Here is the call graph for this function:



Here is the caller graph for this function:

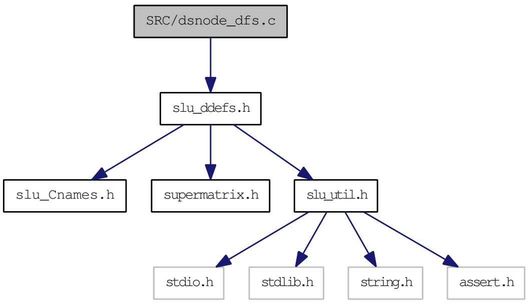


4.79 SRC/dsnode_dfs.c File Reference

Determines the union of row structures of columns within the relaxed node.

```
#include "slu_ddefs.h"
```

Include dependency graph for dsnode_dfs.c:



Functions

- int dsnode_dfs (const int jcol, const int kcol, const int ∗asub, const int ∗xa_begin, const int ∗xa_end, int ∗xprune, int ∗marker, GlobalLU_t ∗Glu)

4.79.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997
```


4.79.2 Function Documentation

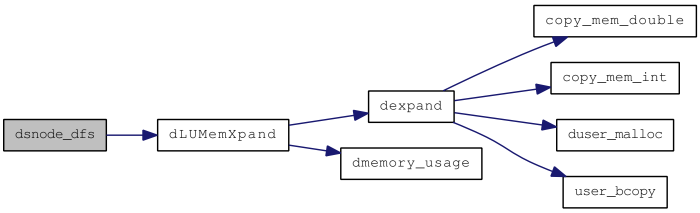
4.79.2.1 int dsnode_dfs (const int *jcol*, const int *kcol*, const int ∗ *asub*, const int ∗ *xa_begin*, const int ∗ *xa_end*, int ∗ *xprune*, int ∗ *marker*, GlobalLU_t ∗ *Glu*)

```
Purpose
=======
   dsnode_dfs() - Determine the union of the row structures of those
```

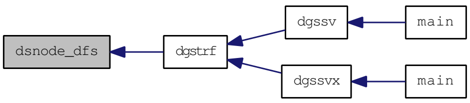
```
    columns within the relaxed snode.
    Note: The relaxed snodes are leaves of the supernodal etree, therefore,
    the portion outside the rectangular supernode must be zero.

 Return value
 ============
     0   success;
    >0   number of bytes allocated when run out of memory.
```

Here is the call graph for this function:



Here is the caller graph for this function:

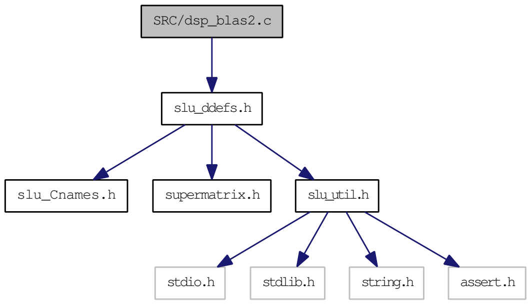


4.80 SRC/dsp_blas2.c File Reference

Sparse BLAS 2, using some dense BLAS 2 operations.

`#include "slu_ddefs.h"`

Include dependency graph for dsp_blas2.c:



Functions

- void dusolve (int, int, double ∗, double ∗)

  Solves a dense upper triangular system.

- void dlsolve (int, int, double ∗, double ∗)

  Solves a dense UNIT lower triangular system.

- void dmatvec (int, int, int, double ∗, double ∗, double ∗)

  Performs a dense matrix-vector multiply: Mxvec = Mxvec + M ∗ vec.

- int sp_dtrsv (char ∗uplo, char ∗trans, char ∗diag, SuperMatrix ∗L, SuperMatrix ∗U, double ∗x, SuperLUStat_t ∗stat, int ∗info)

  Solves one of the systems of equations A∗x = b, or A'∗x = b.

- int sp_dgemv (char ∗trans, double alpha, SuperMatrix ∗A, double ∗x, int incx, double beta, double ∗y, int incy)

  Performs one of the matrix-vector operations y := alpha∗A∗x + beta∗y, or y := alpha∗A'∗x + beta∗y,.

4.80.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003
```

4.80.2 Function Documentation

4.80.2.1 void dlsolve (int *ldm*, int *ncol*, double ∗ *M*, double ∗ *rhs*)

The unit lower triangular matrix is stored in a 2D array M(1:nrow,1:ncol). The solution will be returned in the rhs vector.

4.80.2.2 void dmatvec (int *ldm*, int *nrow*, int *ncol*, double ∗ *M*, double ∗ *vec*, double ∗ *Mxvec*)

The input matrix is M(1:nrow,1:ncol); The product is returned in Mxvec[].

4.80.2.3 void dusolve (int *ldm*, int *ncol*, double ∗ *M*, double ∗ *rhs*)

The upper triangular matrix is stored in a 2-dim array M(1:ldm,1:ncol). The solution will be returned in the rhs vector.

4.80.2.4 int sp_dgemv (char ∗ *trans*, double *alpha*, SuperMatrix ∗ *A*, double ∗ *x*, int *incx*, double *beta*, double ∗ *y*, int *incy*)

```
Purpose
=======

sp_dgemv()  performs one of the matrix-vector operations
   y := alpha*A*x + beta*y,   or   y := alpha*A'*x + beta*y,
where alpha and beta are scalars, x and y are vectors and A is a
sparse A->nrow by A->ncol matrix.

Parameters
==========

TRANS  - (input) char*
         On entry, TRANS specifies the operation to be performed as
         follows:
            TRANS = 'N' or 'n'   y := alpha*A*x + beta*y.
            TRANS = 'T' or 't'   y := alpha*A'*x + beta*y.
            TRANS = 'C' or 'c'   y := alpha*A'*x + beta*y.

ALPHA  - (input) double
         On entry, ALPHA specifies the scalar alpha.

A      - (input) SuperMatrix*
         Matrix A with a sparse format, of dimension (A->nrow, A->ncol).
         Currently, the type of A can be:
             Stype = NC or NCP; Dtype = SLU_D; Mtype = GE.
         In the future, more general A can be handled.

X      - (input) double*, array of DIMENSION at least
         ( 1 + ( n - 1 )*abs( INCX ) ) when TRANS = 'N' or 'n'
         and at least
         ( 1 + ( m - 1 )*abs( INCX ) ) otherwise.
         Before entry, the incremented array X must contain the
         vector x.
```

```
INCX   - (input) int
         On entry, INCX specifies the increment for the elements of
         X. INCX must not be zero.

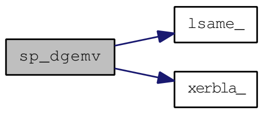
BETA   - (input) double
         On entry, BETA specifies the scalar beta. When BETA is
         supplied as zero then Y need not be set on input.

Y      - (output) double*,  array of DIMENSION at least
         ( 1 + ( m - 1 )*abs( INCY ) ) when TRANS = 'N' or 'n'
         and at least
         ( 1 + ( n - 1 )*abs( INCY ) ) otherwise.
         Before entry with BETA non-zero, the incremented array Y
         must contain the vector y. On exit, Y is overwritten by the
         updated vector y.

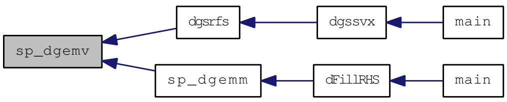
INCY   - (input) int
         On entry, INCY specifies the increment for the elements of
         Y. INCY must not be zero.

==== Sparse Level 2 Blas routine.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.80.2.5 int sp_dtrsv (char ∗ *uplo*, char ∗ *trans*, char ∗ *diag*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, double ∗ *x*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======

sp_dtrsv() solves one of the systems of equations
    A*x = b,    or    A'*x = b,
where b and x are n element vectors and A is a sparse unit , or
non-unit, upper or lower triangular matrix.
No test for singularity or near-singularity is included in this
routine. Such tests must be performed before calling this routine.

Parameters
==========
```

```
uplo   - (input) char*
         On entry, uplo specifies whether the matrix is an upper or
          lower triangular matrix as follows:
             uplo = 'U' or 'u'   A is an upper triangular matrix.
             uplo = 'L' or 'l'   A is a lower triangular matrix.


trans  - (input) char*
          On entry, trans specifies the equations to be solved as
          follows:
             trans = 'N' or 'n'   A*x = b.
             trans = 'T' or 't'   A'*x = b.
             trans = 'C' or 'c'   A'*x = b.


diag   - (input) char*
          On entry, diag specifies whether or not A is unit
          triangular as follows:
             diag = 'U' or 'u'   A is assumed to be unit triangular.
             diag = 'N' or 'n'   A is not assumed to be unit
                                 triangular.

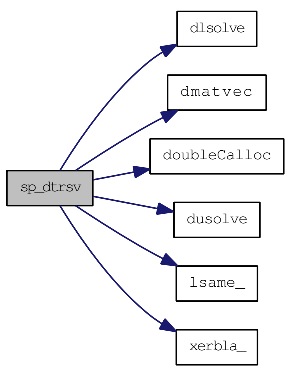

L       - (input) SuperMatrix*
    The factor L from the factorization Pr*A*Pc=L*U. Use
          compressed row subscripts storage for supernodes,
          i.e., L has types: Stype = SC, Dtype = SLU_D, Mtype = TRLU.


U       - (input) SuperMatrix*
     The factor U from the factorization Pr*A*Pc=L*U.
     U has types: Stype = NC, Dtype = SLU_D, Mtype = TRU.

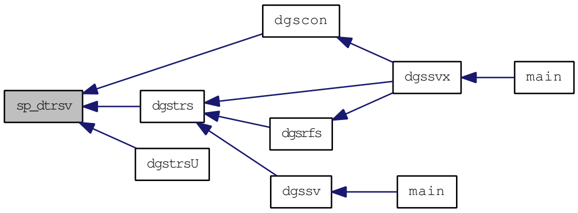

x       - (input/output) double*
          Before entry, the incremented array X must contain the n
          element right-hand side vector b. On exit, X is overwritten
          with the solution vector x.


info    - (output) int*
          If *info = -i, the i-th argument had an illegal value.
```

Here is the call graph for this function:



Here is the caller graph for this function:

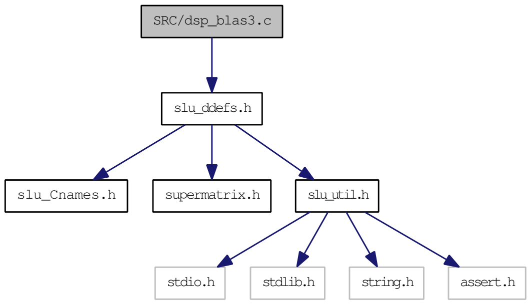


4.81 SRC/dsp_blas3.c File Reference

Sparse BLAS3, using some dense BLAS3 operations.

`#include "slu_ddefs.h"`

Include dependency graph for dsp_blas3.c:



Functions

- int sp_dgemm (char ∗transa, char ∗transb, int m, int n, int k, double alpha, SuperMatrix ∗A, double ∗b, int ldb, double beta, double ∗c, int ldc)

4.81.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997
```

4.81.2 Function Documentation

4.81.2.1 int sp_dgemm (char ∗ *transa*, char ∗ *transb*, int *m*, int *n*, int *k*, double *alpha*, SuperMatrix ∗ *A*, double ∗ *b*, int *ldb*, double *beta*, double ∗ *c*, int *ldc*)

```
Purpose
  =======

  sp_d performs one of the matrix-matrix operations

     C := alpha*op( A )*op( B ) + beta*C,

  where  op( X ) is one of

     op( X ) = X   or   op( X ) = X'   or   op( X ) = conjg( X' ),

  alpha and beta are scalars, and A, B and C are matrices, with op( A )
  an m by k matrix,  op( B )  a  k by n matrix and  C an m by n matrix.
```

```
Parameters
==========

TRANSA - (input) char*
         On entry, TRANSA specifies the form of op( A ) to be used in
         the matrix multiplication as follows:
            TRANSA = 'N' or 'n',  op( A ) = A.
            TRANSA = 'T' or 't',  op( A ) = A'.
            TRANSA = 'C' or 'c',  op( A ) = conjg( A' ).
         Unchanged on exit.

TRANSB - (input) char*
         On entry, TRANSB specifies the form of op( B ) to be used in
         the matrix multiplication as follows:
            TRANSB = 'N' or 'n',  op( B ) = B.
            TRANSB = 'T' or 't',  op( B ) = B'.
            TRANSB = 'C' or 'c',  op( B ) = conjg( B' ).
         Unchanged on exit.

M      - (input) int
         On entry,  M  specifies  the number of rows of the matrix
  op( A ) and of the matrix C.  M must be at least zero.
  Unchanged on exit.

N      - (input) int
         On entry,  N specifies the number of columns of the matrix
  op( B ) and the number of columns of the matrix C. N must be
  at least zero.
  Unchanged on exit.

K      - (input) int
         On entry, K specifies the number of columns of the matrix
  op( A ) and the number of rows of the matrix op( B ). K must
  be at least  zero.
        Unchanged on exit.

ALPHA  - (input) double
         On entry, ALPHA specifies the scalar alpha.

A      - (input) SuperMatrix*
         Matrix A with a sparse format, of dimension (A->nrow, A->ncol).
         Currently, the type of A can be:
             Stype = NC or NCP; Dtype = SLU_D; Mtype = GE.
         In the future, more general A can be handled.

B      - DOUBLE PRECISION array of DIMENSION ( LDB, kb ), where kb is
         n when TRANSB = 'N' or 'n',  and is  k otherwise.
         Before entry with  TRANSB = 'N' or 'n',  the leading k by n
         part of the array B must contain the matrix B, otherwise
         the leading n by k part of the array B must contain the
         matrix B.
         Unchanged on exit.

LDB    - (input) int
         On entry, LDB specifies the first dimension of B as declared
         in the calling (sub) program. LDB must be at least max( 1, n ).
         Unchanged on exit.
```

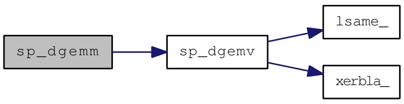
```
BETA   - (input) double
         On entry, BETA specifies the scalar beta. When BETA is
         supplied as zero then C need not be set on input.

C      - DOUBLE PRECISION array of DIMENSION ( LDC, n ).
         Before entry, the leading m by n part of the array C must
         contain the matrix C,  except when beta is zero, in which
         case C need not be set on entry.
         On exit, the array C is overwritten by the m by n matrix
  ( alpha*op( A )*B + beta*C ).

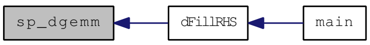
LDC    - (input) int
         On entry, LDC specifies the first dimension of C as declared
         in the calling (sub)program. LDC must be at least max(1,m).
         Unchanged on exit.

==== Sparse Level 3 Blas routine.
```

Here is the call graph for this function:



Here is the caller graph for this function:



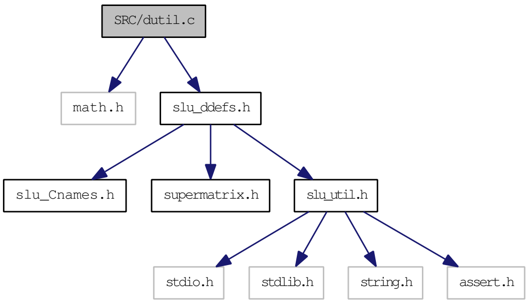
4.82 SRC/dutil.c File Reference

Matrix utility functions.

`#include <math.h>`

`#include "slu_ddefs.h"`

Include dependency graph for dutil.c:



Functions

- void dCreate_CompCol_Matrix (SuperMatrix *A, int m, int n, int nnz, double *nzval, int *rowind, int *colptr, Stype_t stype, Dtype_t dtype, Mtype_t mtype)

    Supernodal LU factor related.

- void dCreate_CompRow_Matrix (SuperMatrix *A, int m, int n, int nnz, double *nzval, int *colind, int *rowptr, Stype_t stype, Dtype_t dtype, Mtype_t mtype)
- void dCopy_CompCol_Matrix (SuperMatrix *A, SuperMatrix *B)

    Copy matrix A into matrix B.

- void dCreate_Dense_Matrix (SuperMatrix *X, int m, int n, double *x, int ldx, Stype_t stype, Dtype_t dtype, Mtype_t mtype)
- void dCopy_Dense_Matrix (int M, int N, double *X, int ldx, double *Y, int ldy)
- void dCreate_SuperNode_Matrix (SuperMatrix *L, int m, int n, int nnz, double *nzval, int *nzval_-colptr, int *rowind, int *rowind_colptr, int *col_to_sup, int *sup_to_col, Stype_t stype, Dtype_t dtype, Mtype_t mtype)
- void dCompRow_to_CompCol (int m, int n, int nnz, double *a, int *colind, int *rowptr, double **at, int **rowind, int **colptr)

    Convert a row compressed storage into a column compressed storage.

- void dPrint_CompCol_Matrix (char *what, SuperMatrix *A)

    Routines for debugging.

- void dPrint_SuperNode_Matrix (char *what, SuperMatrix *A)
- void dPrint_Dense_Matrix (char *what, SuperMatrix *A)
- void dprint_lu_col (char *msg, int jcol, int pivrow, int *xprune, GlobalLU_t *Glu)

    Diagnostic print of column "jcol" in the U/L factor.

- void dcheck_tempv (int n, double *tempv)

Check whether tempv[] == 0. This should be true before and after calling any numeric routines, i.e., "panel_bmod" and "column_bmod".

- void dGenXtrue (int n, int nrhs, double ∗x, int ldx)
- void dFillRHS (trans_t trans, int nrhs, double ∗x, int ldx, SuperMatrix ∗A, SuperMatrix ∗B)

  Let rhs[i] = sum of i-th row of A, so the solution vector is all 1's.

- void dfill (double ∗a, int alen, double dval)

  Fills a double precision array with a given value.

- void dinf_norm_error (int nrhs, SuperMatrix ∗X, double ∗xtrue)

  Check the inf-norm of the error vector.

- void dPrintPerf (SuperMatrix ∗L, SuperMatrix ∗U, mem_usage_t ∗mem_usage, double rpg, double rcond, double ∗ferr, double ∗berr, char ∗equed, SuperLUStat_t ∗stat)

  Print performance of the code.

- print_double_vec (char ∗what, int n, double ∗vec)

4.82.1 Detailed Description

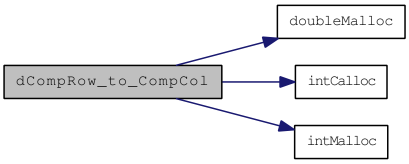
```
-- SuperLU routine (version 3.1) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
August 1, 2008
```

4.82.2 Function Documentation

4.82.2.1 void dcheck_tempv (int *n*, double ∗ *tempv*)

4.82.2.2 void dCompRow_to_CompCol (int *m*, int *n*, int *nnz*, double ∗ *a*, int ∗ *colind*, int ∗ *rowptr*, double ∗∗ *at*, int ∗∗ *rowind*, int ∗∗ *colptr*)

Here is the call graph for this function:



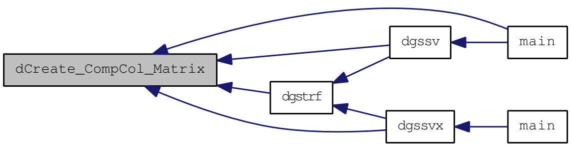
4.82.2.3 void dCopy_CompCol_Matrix (SuperMatrix ∗ *A*, SuperMatrix ∗ *B*)

4.82.2.4 void dCopy_Dense_Matrix (int *M*, int *N*, double ∗ *X*, int *ldx*, double ∗ *Y*, int *ldy*)

Copies a two-dimensional matrix X to another matrix Y.

4.82.2.5 void dCreate_CompCol_Matrix (SuperMatrix ∗ *A*, int *m*, int *n*, int *nnz*, double ∗ *nzval*, int ∗ *rowind*, int ∗ *colptr*, Stype_t *stype*, Dtype_t *dtype*, Mtype_t *mtype*)

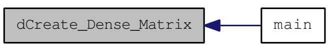
Here is the caller graph for this function:



4.82.2.6 void dCreate_CompRow_Matrix (SuperMatrix ∗ *A*, int *m*, int *n*, int *nnz*, double ∗ *nzval*, int ∗ *colind*, int ∗ *rowptr*, Stype_t *stype*, Dtype_t *dtype*, Mtype_t *mtype*)

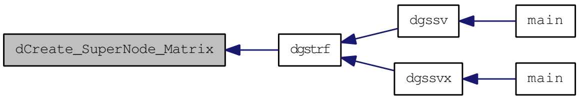
4.82.2.7 void dCreate_Dense_Matrix (SuperMatrix ∗ *X*, int *m*, int *n*, double ∗ *x*, int *ldx*, Stype_t *stype*, Dtype_t *dtype*, Mtype_t *mtype*)

Here is the caller graph for this function:



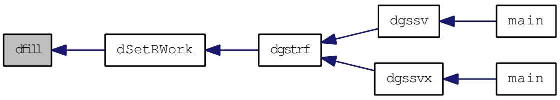
4.82.2.8 void dCreate_SuperNode_Matrix (SuperMatrix ∗ *L*, int *m*, int *n*, int *nnz*, double ∗ *nzval*, int ∗ *nzval_colptr*, int ∗ *rowind*, int ∗ *rowind_colptr*, int ∗ *col_to_sup*, int ∗ *sup_to_col*, Stype_t *stype*, Dtype_t *dtype*, Mtype_t *mtype*)

Here is the caller graph for this function:



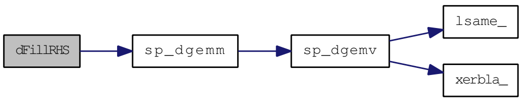
4.82.2.9 void dfill (double ∗ *a*, int *alen*, double *dval*)

Here is the caller graph for this function:

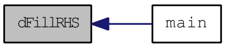


4.82.2.10 void dFillRHS (trans_t *trans*, int *nrhs*, double ∗ *x*, int *ldx*, SuperMatrix ∗ *A*, SuperMatrix ∗ *B*)

Here is the call graph for this function:

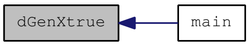


Here is the caller graph for this function:



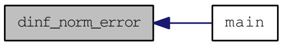
4.82.2.11 void dGenXtrue (int *n*, int *nrhs*, double ∗ *x*, int *ldx*)

Here is the caller graph for this function:



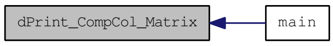
4.82.2.12 void dinf_norm_error (int *nrhs*, SuperMatrix ∗ *X*, double ∗ *xtrue*)

Here is the caller graph for this function:



4.82.2.13 void dPrint_CompCol_Matrix (char ∗ *what*, SuperMatrix ∗ *A*)

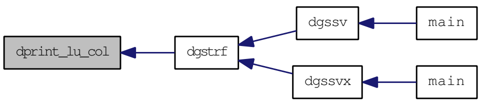
Here is the caller graph for this function:



4.82.2.14 void dPrint_Dense_Matrix (char ∗ *what*, SuperMatrix ∗ *A*)

4.82.2.15 void dprint_lu_col (char ∗ *msg*, int *jcol*, int *pivrow*, int ∗ *xprune*, GlobalLU_t ∗ *Glu*)

Here is the caller graph for this function:



4.82.2.16 void dPrint_SuperNode_Matrix (char ∗ *what*, SuperMatrix ∗ *A*)

Here is the caller graph for this function:

4.82.2.17 void dPrintPerf (SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, mem_usage_t ∗ *mem_usage*, double *rpg*, double *rcond*, double ∗ *ferr*, double ∗ *berr*, char ∗ *equed*, SuperLUStat_t ∗ *stat*)

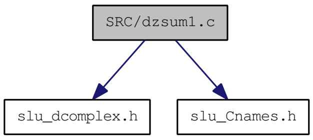
4.82.2.18 print_double_vec (char ∗ *what*, int *n*, double ∗ *vec*)

4.83 SRC/dzsum1.c File Reference

Takes sum of the absolute values of a complex vector and returns a double precision result.

```
#include "slu_dcomplex.h"
#include "slu_Cnames.h"
```

Include dependency graph for dzsum1.c:



Defines

- #define CX(I) cx[(I)-1]

Functions

- double dzsum1_ (int ∗n, doublecomplex ∗cx, int ∗incx)

4.83.1 Detailed Description

```
-- LAPACK auxiliary routine (version 2.0) --
Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
Courant Institute, Argonne National Lab, and Rice University
October 31, 1992
```

4.83.2 Define Documentation

4.83.2.1 #define CX(I) cx[(I)-1]

4.83.3 Function Documentation

4.83.3.1 double dzsum1_ (int ∗ *n*, doublecomplex ∗ *cx*, int ∗ *incx*)

```
Purpose
=======

DZSUM1 takes the sum of the absolute values of a complex
vector and returns a double precision result.

Based on DZASUM from the Level 1 BLAS.
The change is to use the 'genuine' absolute value.

Contributed by Nick Higham for use with ZLACON.
```

```
Arguments
=========

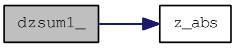
N       (input) INT
        The number of elements in the vector CX.

CX      (input) COMPLEX*16 array, dimension (N)
        The vector whose elements will be summed.

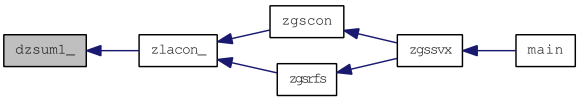
INCX    (input) INT
        The spacing between successive values of CX.  INCX > 0.

=====================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:



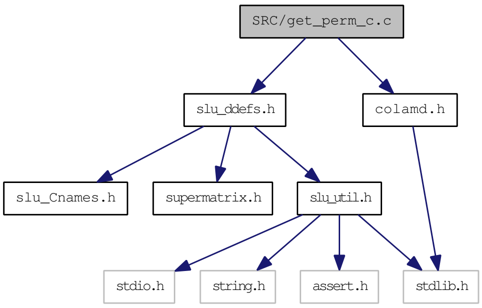
4.84 SRC/get_perm_c.c File Reference

Matrix permutation operations.

`#include "slu_ddefs.h"`

`#include "colamd.h"`

Include dependency graph for get_perm_c.c:



Functions

- int genmmd_ (int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗)
- void get_colamd (const int m, const int n, const int nnz, int ∗colptr, int ∗rowind, int ∗perm_c)
- void getata (const int m, const int n, const int nz, int ∗colptr, int ∗rowind, int ∗atanz, int ∗∗ata_colptr, int ∗∗ata_rowind)
- void at_plus_a (const int n, const int nz, int ∗colptr, int ∗rowind, int ∗bnz, int ∗∗b_colptr, int ∗∗b_rowind)
- void get_perm_c (int ispec, SuperMatrix ∗A, int ∗perm_c)

4.84.1 Detailed Description

```
-- SuperLU routine (version 3.1) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
August 1, 2008
```

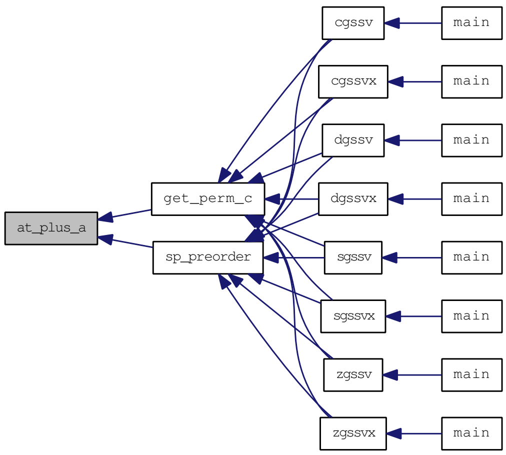
4.84.2 Function Documentation

4.84.2.1 void at_plus_a (const int *n*, const int *nz*, int ∗ *colptr*, int ∗ *rowind*, int ∗ *bnz*, int ∗∗ *b_colptr*, int ∗∗ *b_rowind*)

```
Purpose
=======

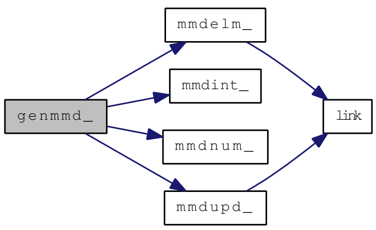
Form the structure of A'+A. A is an n-by-n matrix in column oriented
format represented by (colptr, rowind). The output A'+A is in column
oriented format (symmetrically, also row oriented), represented by
(b_colptr, b_rowind).
```

Here is the caller graph for this function:

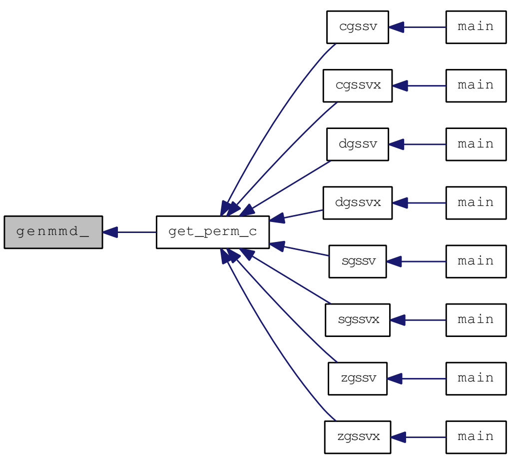


4.84.2.2 int genmmd_ (int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗)

Here is the call graph for this function:

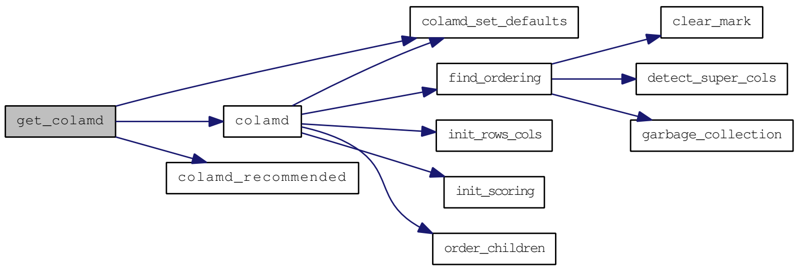


Here is the caller graph for this function:

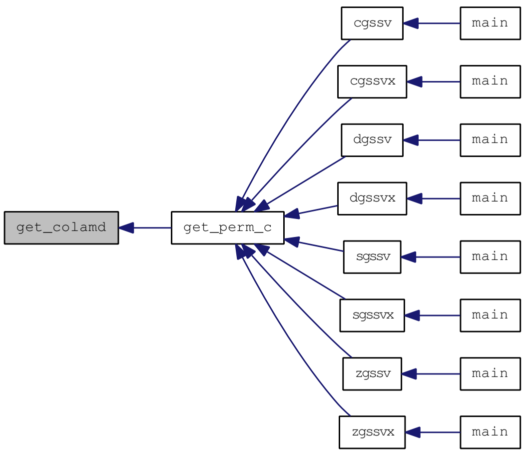


4.84.2.3 void get_colamd (const int *m*, const int *n*, const int *nnz*, int ∗ *colptr*, int ∗ *rowind*, int ∗ *perm_c*)

Here is the call graph for this function:



Here is the caller graph for this function:



4.84.2.4 void get_perm_c (int *ispec*, SuperMatrix ∗ *A*, int ∗ *perm_c*)

```
Purpose
=======

GET_PERM_C obtains a permutation matrix Pc, by applying the multiple
minimum degree ordering code by Joseph Liu to matrix A'*A or A+A'.
or using approximate minimum degree column ordering by Davis et. al.
The LU factorization of A*Pc tends to have less fill than the LU
factorization of A.

Arguments
=========

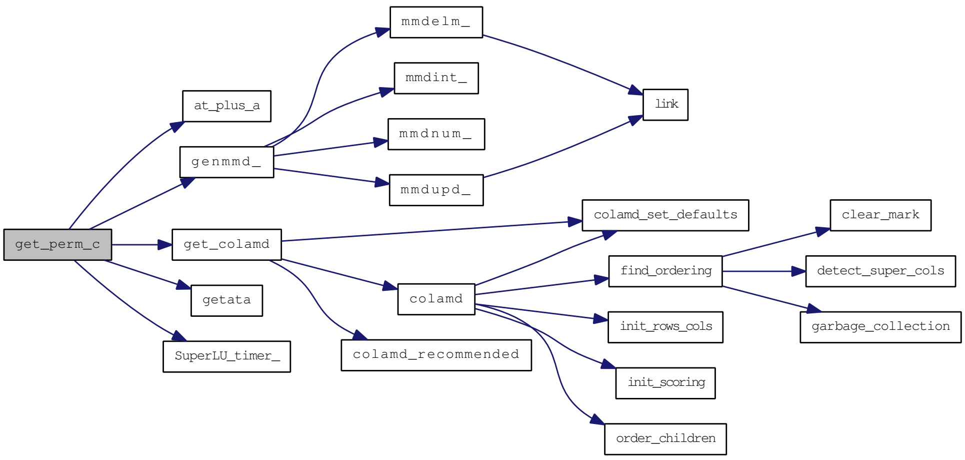
ispec   (input) int
        Specifies the type of column ordering to reduce fill:
        = 1: minimum degree on the structure of A^T * A
        = 2: minimum degree on the structure of A^T + A
```

```
        = 3: approximate minimum degree for unsymmetric matrices
        If ispec == 0, the natural ordering (i.e., Pc = I) is returned.

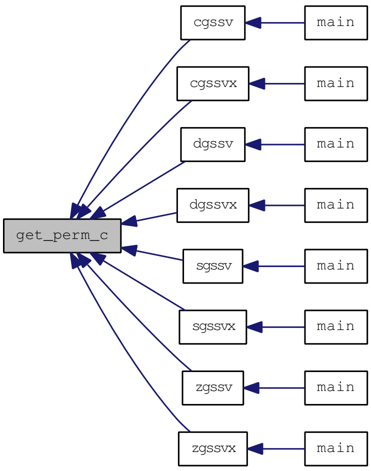

A       (input) SuperMatrix*
        Matrix A in A*X=B, of dimension (A->nrow, A->ncol). The number
        of the linear equations is A->nrow. Currently, the type of A
        can be: Stype = NC; Dtype = _D; Mtype = GE. In the future,
        more general A can be handled.


perm_c  (output) int*
  Column permutation vector of size A->ncol, which defines the
        permutation matrix Pc; perm_c[i] = j means column i of A is
        in position j in A*Pc.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.84.2.5 void getata (const int *m*, const int *n*, const int *nz*, int ∗ *colptr*, int ∗ *rowind*, int ∗ *atanz*, int ∗∗ *ata_colptr*, int ∗∗ *ata_rowind*)

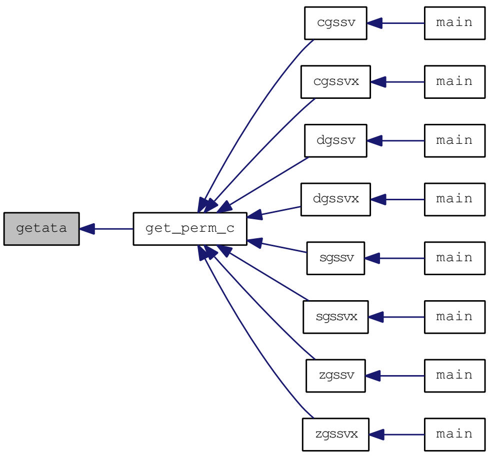
```
Purpose
=======

Form the structure of A'*A. A is an m-by-n matrix in column oriented
format represented by (colptr, rowind). The output A'*A is in column
oriented format (symmetrically, also row oriented), represented by
(ata_colptr, ata_rowind).

This routine is modified from GETATA routine by Tim Davis.
The complexity of this algorithm is: SUM_{i=1,m} r(i)^2,
i.e., the sum of the square of the row counts.

Questions
=========
    o  Do I need to withhold the *dense* rows?
    o  How do I know the number of nonzeros in A'*A?
```

Here is the caller graph for this function:

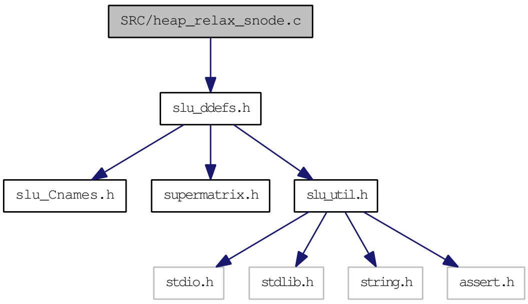


4.85 SRC/heap_relax_snode.c File Reference

Identify the initial relaxed supernodes.

`#include "slu_ddefs.h"`

Include dependency graph for heap_relax_snode.c:



Functions

- void heap_relax_snode (const int n, int ∗et, const int relax_columns, int ∗descendants, int ∗relax_-end)

4.85.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003


Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

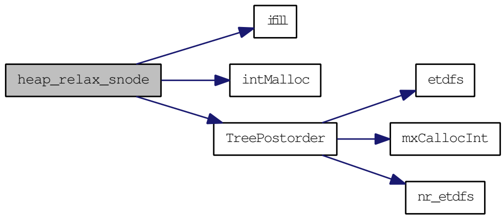
4.85.2 Function Documentation

4.85.2.1 void heap_relax_snode (const int *n*, int ∗ *et*, const int *relax_columns*, int ∗ *descendants*, int ∗ *relax_end*)

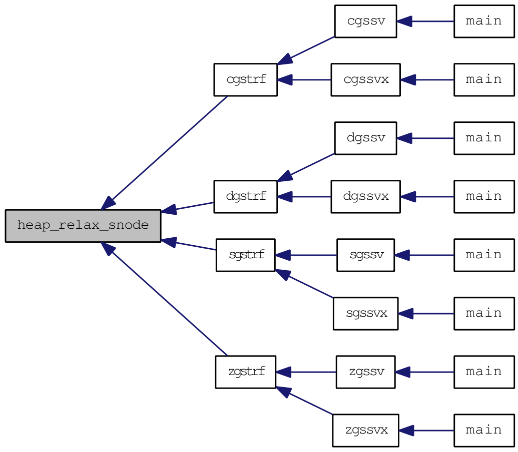
```
Purpose
=======
   relax_snode() - Identify the initial relaxed supernodes, assuming that
```

```
the matrix has been reordered according to the postorder of the etree.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.86 SRC/icmax1.c File Reference

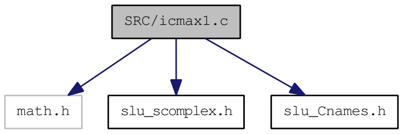
Finds the index of the element whose real part has maximum absolute value.

`#include <math.h>`

`#include "slu_scomplex.h"`

`#include "slu_Cnames.h"`

Include dependency graph for icmax1.c:



Defines

- #define CX(I) cx[(I)-1]

Functions

- int icmax1_ (int ∗n, complex ∗cx, int ∗incx)

4.86.1 Detailed Description

```
-- LAPACK auxiliary routine (version 2.0) --
Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
Courant Institute, Argonne National Lab, and Rice University
October 31, 1992
```

4.86.2 Define Documentation

4.86.2.1 #define CX(I) cx[(I)-1]

4.86.3 Function Documentation

4.86.3.1 int icmax1_ (int ∗ *n*, complex ∗ *cx*, int ∗ *incx*)

```
Purpose
=======

ICMAX1 finds the index of the element whose real part has maximum
absolute value.

Based on ICAMAX from Level 1 BLAS.
The change is to use the 'genuine' absolute value.
```

```
    Contributed by Nick Higham for use with CLACON.

    Arguments
    =========

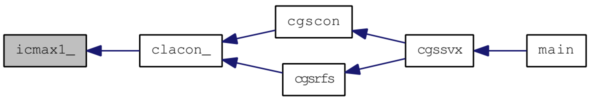
    N       (input) INT
            The number of elements in the vector CX.

    CX      (input) COMPLEX array, dimension (N)
            The vector whose elements will be summed.

    INCX    (input) INT
            The spacing between successive values of CX.  INCX >= 1.

   =====================================================================
```

Here is the caller graph for this function:



4.87 SRC/izmax1.c File Reference

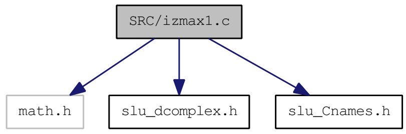
Finds the index of the element whose real part has maximum absolute value.

`#include <math.h>`

`#include "slu_dcomplex.h"`

`#include "slu_Cnames.h"`

Include dependency graph for izmax1.c:



Defines

- #define CX(I) cx[(I)-1]

Functions

- int izmax1_ (int *n, doublecomplex *cx, int *incx)

4.87.1 Detailed Description

```
-- LAPACK auxiliary routine (version 2.0) --
Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
Courant Institute, Argonne National Lab, and Rice University
October 31, 1992
```

4.87.2 Define Documentation

4.87.2.1 #define CX(I) cx[(I)-1]

4.87.3 Function Documentation

4.87.3.1 int izmax1_ (int * *n*, doublecomplex * *cx*, int * *incx*)

```
Purpose
=======

IZMAX1 finds the index of the element whose real part has maximum
absolute value.

Based on IZAMAX from Level 1 BLAS.
The change is to use the 'genuine' absolute value.
```

```
    Contributed by Nick Higham for use with ZLACON.

    Arguments
    =========

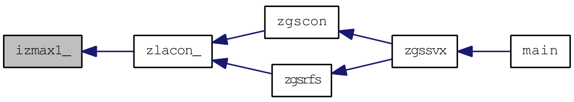
    N       (input) INT
            The number of elements in the vector CX.

    CX      (input) COMPLEX*16 array, dimension (N)
            The vector whose elements will be summed.

    INCX    (input) INT
            The spacing between successive values of CX.  INCX >= 1.

   =====================================================================
```

Here is the caller graph for this function:

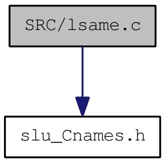


4.88 SRC/lsame.c File Reference

Check if CA is the same letter as CB regardless of case.

`#include "slu_Cnames.h"`

Include dependency graph for lsame.c:



Functions

- int lsame_ (char ∗ca, char ∗cb)

4.88.1 Detailed Description

```
 -- LAPACK auxiliary routine (version 2.0) --
      Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
      Courant Institute, Argonne National Lab, and Rice University
      September 30, 1994
```

4.88.2 Function Documentation

4.88.2.1 int lsame_ (char ∗ *ca*, char ∗ *cb*)

```
    Purpose
    =======

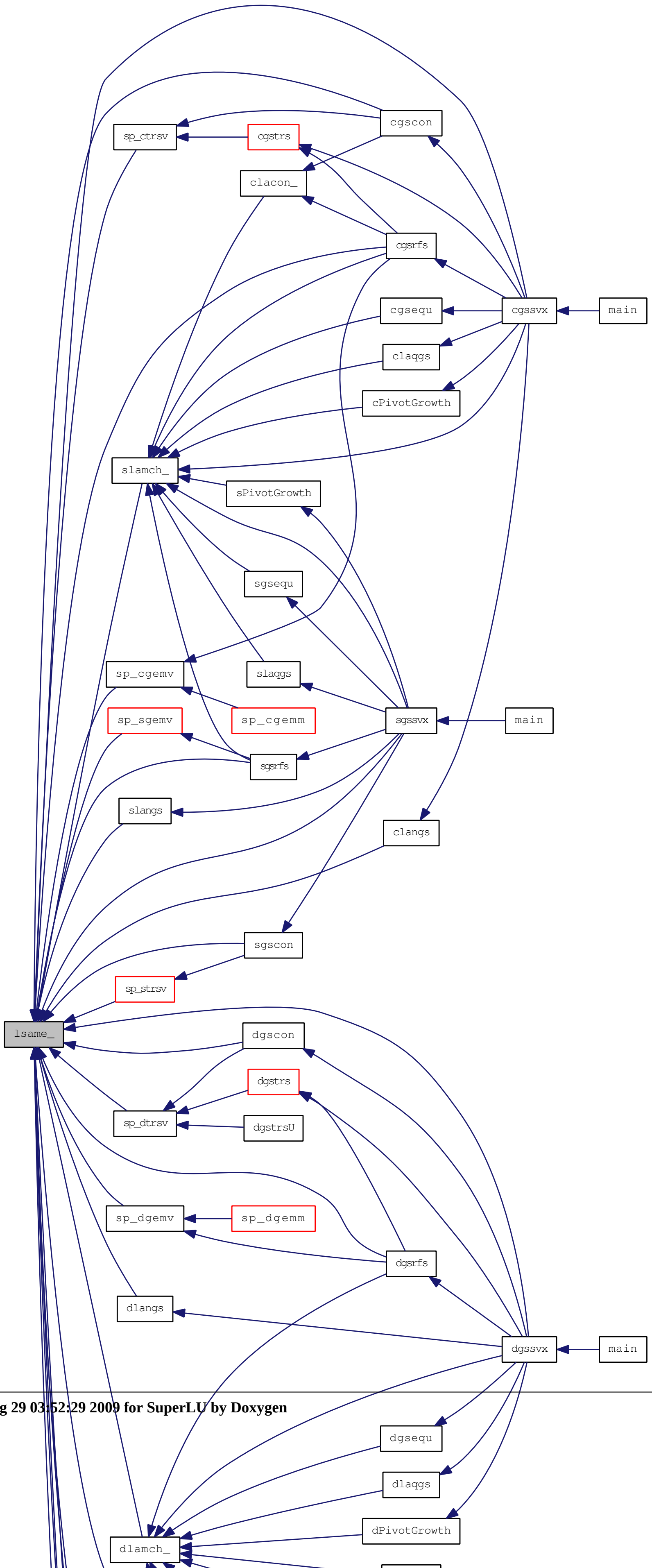

    LSAME returns .TRUE. if CA is the same letter as CB regardless of case.


    Arguments
    =========


    CA      (input) CHARACTER*1
    CB      (input) CHARACTER*1
            CA and CB specify the single characters to be compared.


   =====================================================================
```

Here is the caller graph for this function:

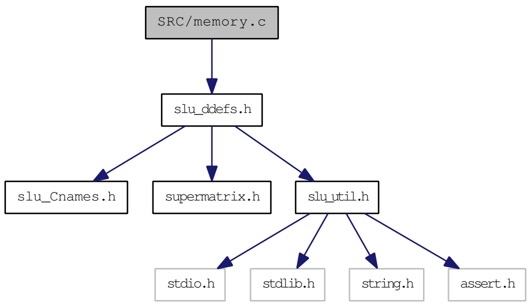


4.89 SRC/memory.c File Reference

Precision-independent memory-related routines.

`#include "slu_ddefs.h"`

Include dependency graph for memory.c:



Functions

- void ∗ superlu_malloc (size_t size)
- void superlu_free (void ∗addr)
- void SetIWork (int m, int n, int panel_size, int ∗iworkptr, int ∗∗segrep, int ∗∗parent, int ∗∗xplore, int ∗∗repfnz, int ∗∗panel_lsub, int ∗∗xprune, int ∗∗marker)

    Set up pointers for integer working arrays.

- void copy_mem_int (int howmany, void ∗old, void ∗new)
- void user_bcopy (char ∗src, char ∗dest, int bytes)
- int ∗ intMalloc (int n)
- int ∗ intCalloc (int n)

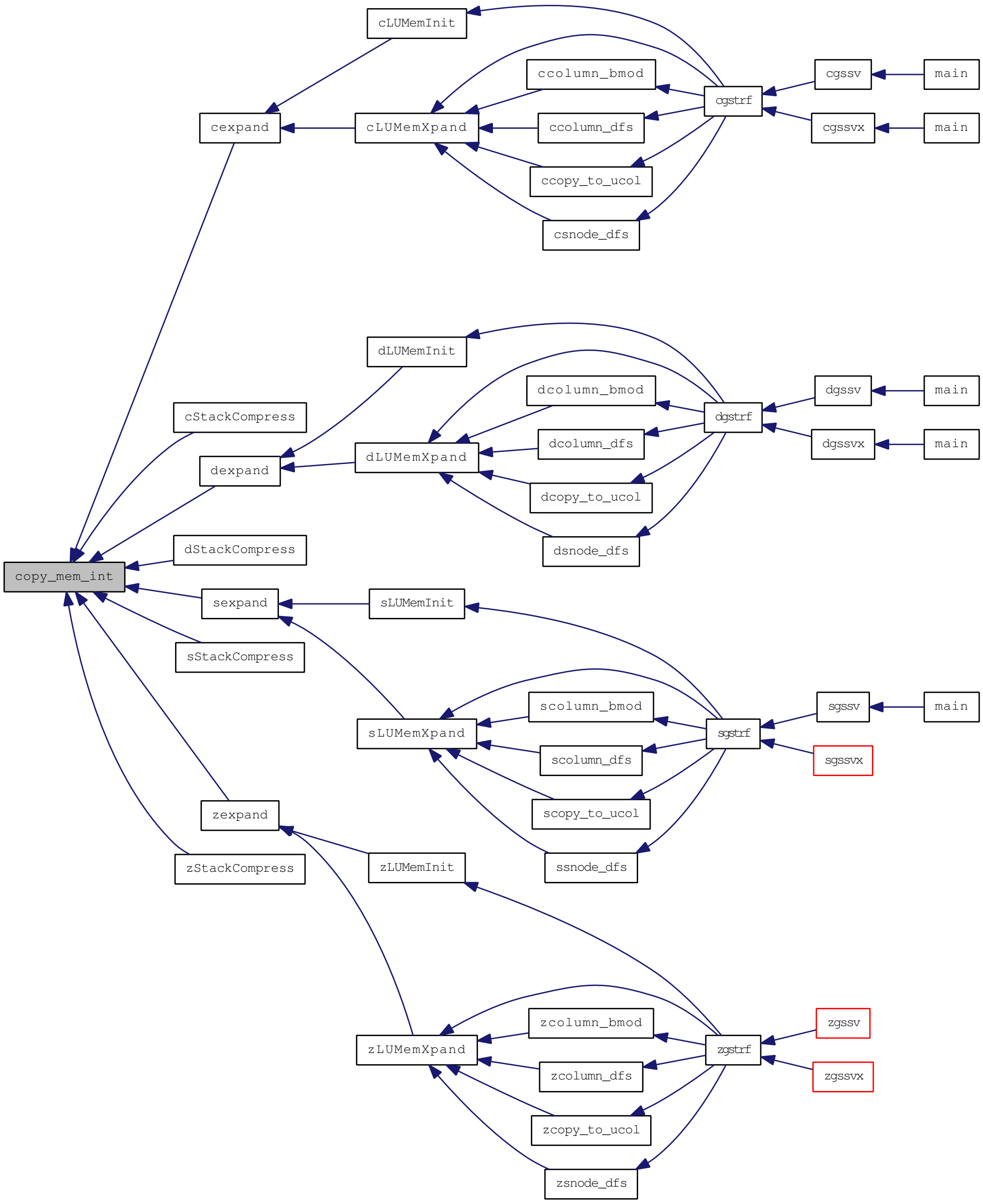
4.89.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997
```

4.89.2 Function Documentation

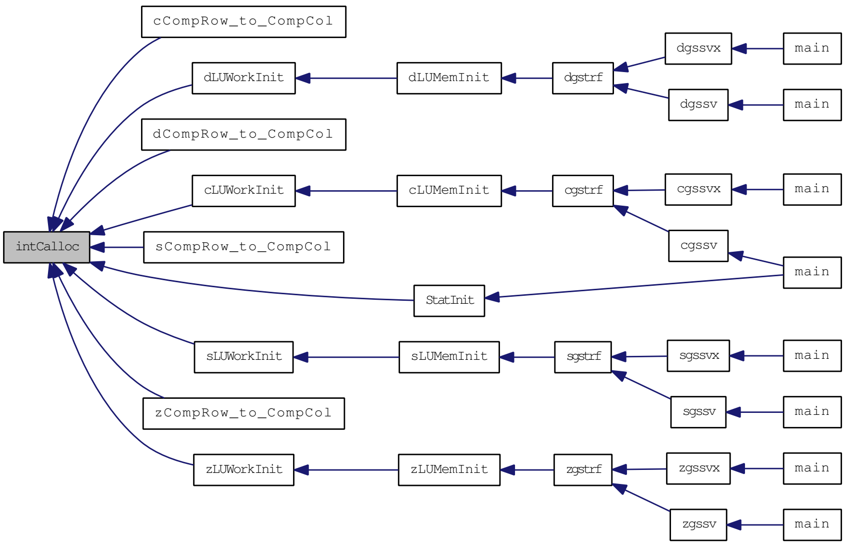
4.89.2.1 void copy_mem_int (int *howmany*, void ∗ *old*, void ∗ *new*)

Here is the caller graph for this function:



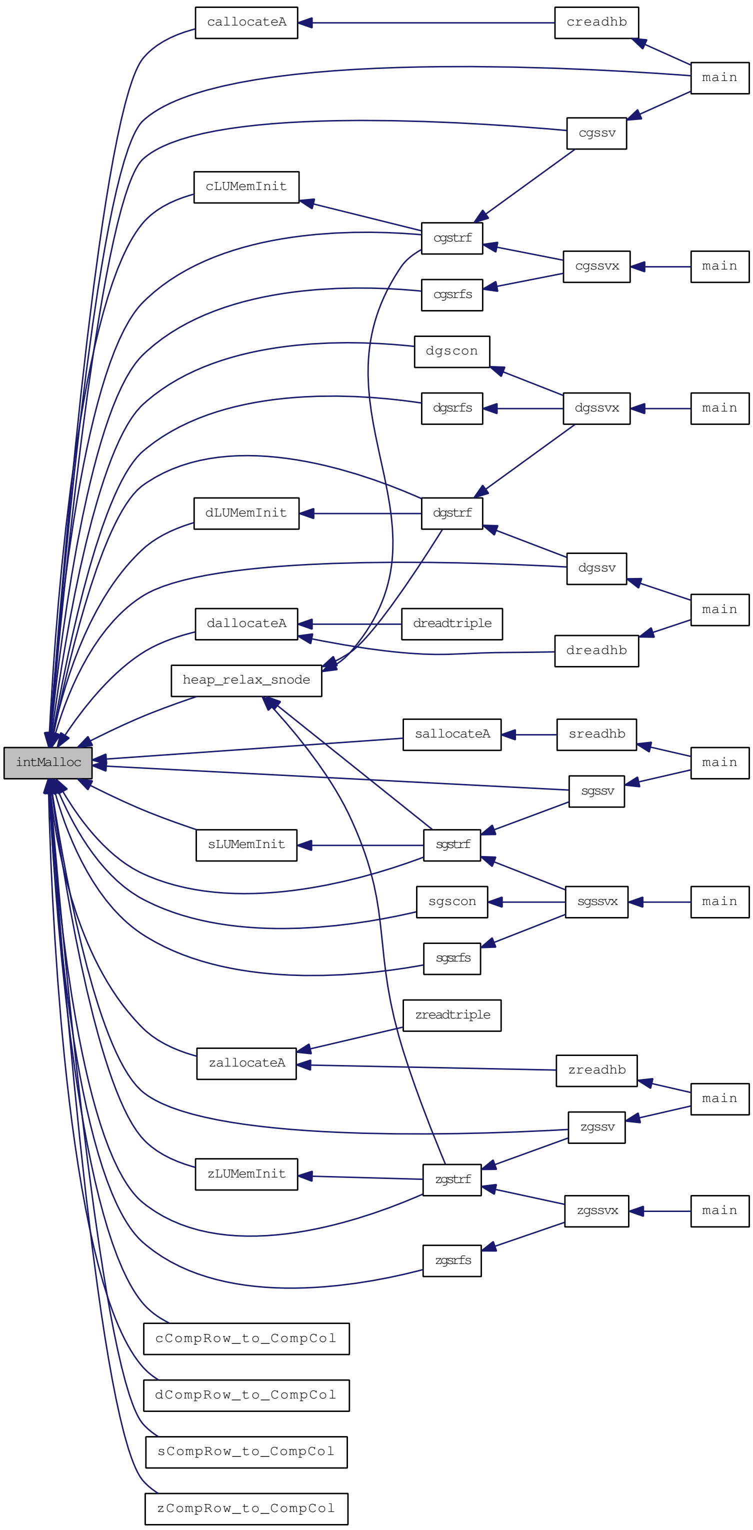
4.89.2.2 int∗ intCalloc (int *n*)

Here is the caller graph for this function:



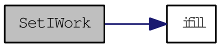
4.89.2.3 int∗ intMalloc (int *n*)

Here is the caller graph for this function:

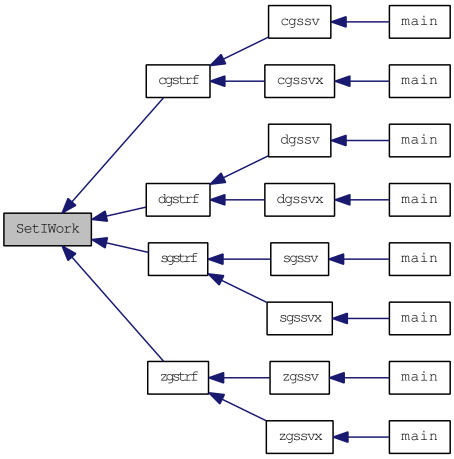


4.89.2.4 void SetIWork (int *m*, int *n*, int *panel_size*, int ∗ *iworkptr*, int ∗∗ *segrep*, int ∗∗ *parent*, int ∗∗ *xplore*, int ∗∗ *repfnz*, int ∗∗ *panel_lsub*, int ∗∗ *xprune*, int ∗∗ *marker*)

Here is the call graph for this function:



Here is the caller graph for this function:



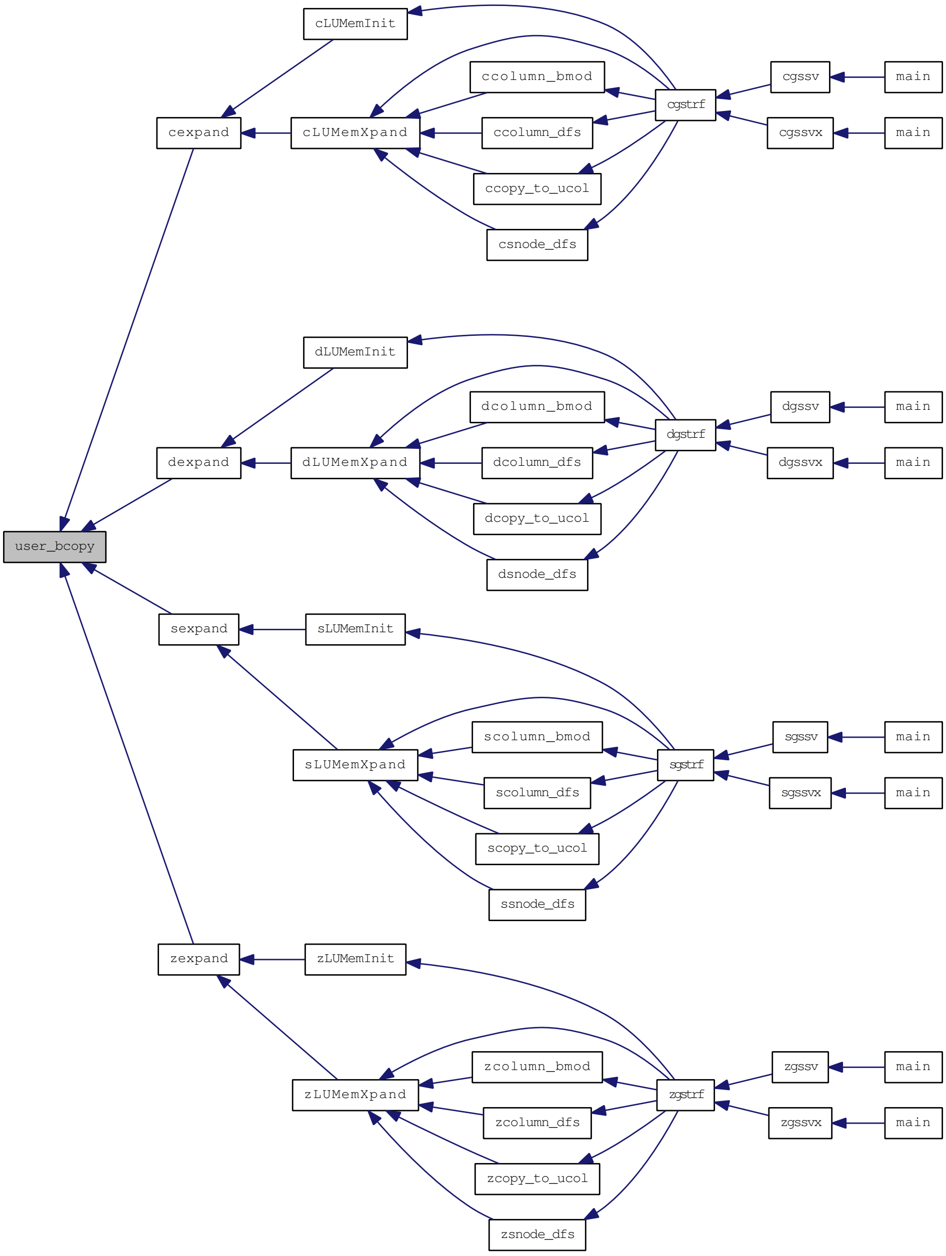
4.89.2.5 void superlu_free (void ∗ *addr*)

4.89.2.6 void∗ superlu_malloc (size_t *size*)

Precision-independent memory-related routines. (Shared by [sdcz]memory.c)

4.89.2.7 void user_bcopy (char ∗ *src*, char ∗ *dest*, int *bytes*)

Here is the caller graph for this function:



4.90 SRC/mmd.c File Reference

Typedefs

- typedef int shortint

Functions

- int genmmd_ (int ∗neqns, int ∗xadj, shortint ∗adjncy, shortint ∗invp, shortint ∗perm, int ∗delta, shortint ∗dhead, shortint ∗qsize, shortint ∗llist, shortint ∗marker, int ∗maxint, int ∗nofsub)
- int mmdint_ (int ∗neqns, int ∗xadj, shortint ∗adjncy, shortint ∗dhead, shortint ∗dforw, shortint ∗dbakw, shortint ∗qsize, shortint ∗llist, shortint ∗marker)
- int mmdelm_ (int ∗mdnode, int ∗xadj, shortint ∗adjncy, shortint ∗dhead, shortint ∗dforw, shortint ∗dbakw, shortint ∗qsize, shortint ∗llist, shortint ∗marker, int ∗maxint, int ∗tag)
- int mmdupd_ (int ∗ehead, int ∗neqns, int ∗xadj, shortint ∗adjncy, int ∗delta, int ∗mdeg, shortint ∗dhead, shortint ∗dforw, shortint ∗dbakw, shortint ∗qsize, shortint ∗llist, shortint ∗marker, int ∗maxint, int ∗tag)
- int mmdnum_ (int ∗neqns, shortint ∗perm, shortint ∗invp, shortint ∗qsize)

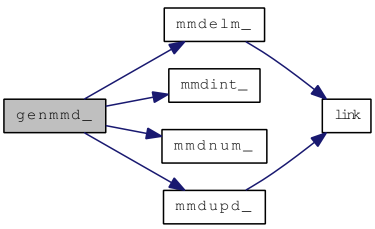
4.90.1 Typedef Documentation

4.90.1.1 typedef int shortint

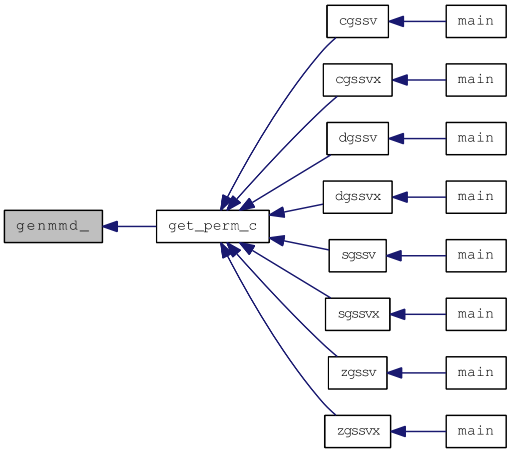
4.90.2 Function Documentation

4.90.2.1 int genmmd_ (int ∗ *neqns*, int ∗ *xadj*, shortint ∗ *adjncy*, shortint ∗ *invp*, shortint ∗ *perm*, int ∗ *delta*, shortint ∗ *dhead*, shortint ∗ *qsize*, shortint ∗ *llist*, shortint ∗ *marker*, int ∗ *maxint*, int ∗ *nofsub*)

Here is the call graph for this function:

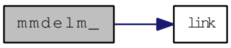


Here is the caller graph for this function:

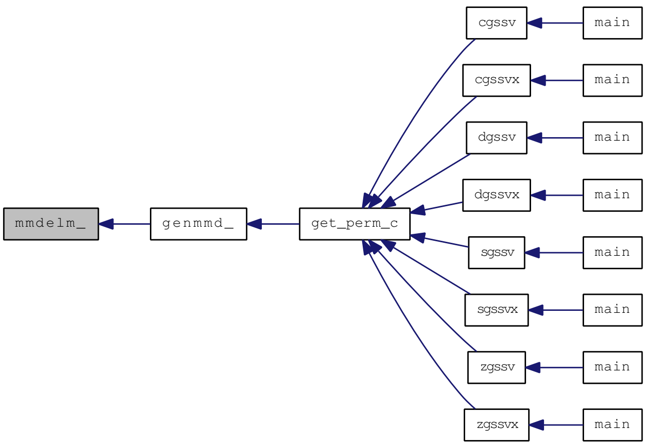


4.90.2.2 int mmdelm_ (int ∗ *mdnode*, int ∗ *xadj*, shortint ∗ *adjncy*, shortint ∗ *dhead*, shortint ∗ *dforw*, shortint ∗ *dbakw*, shortint ∗ *qsize*, shortint ∗ *llist*, shortint ∗ *marker*, int ∗ *maxint*, int ∗ *tag*)

Here is the call graph for this function:

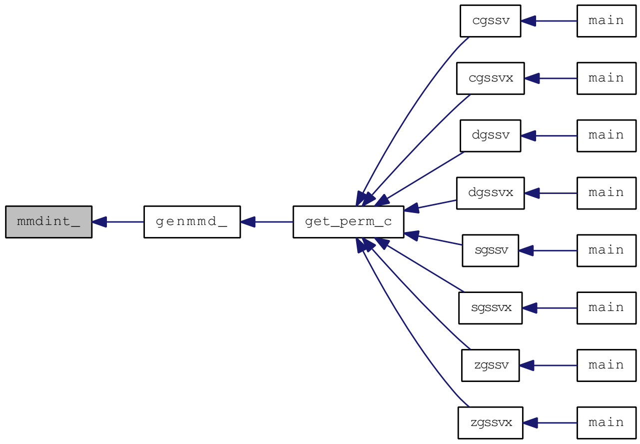


Here is the caller graph for this function:



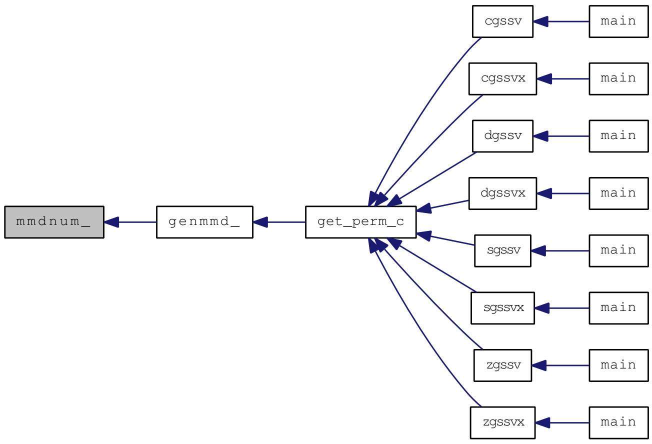
4.90.2.3 int mmdint_ (int * *neqns*, int * *xadj*, shortint * *adjncy*, shortint * *dhead*, shortint * *dforw*, shortint * *dbakw*, shortint * *qsize*, shortint * *llist*, shortint * *marker*)

Here is the caller graph for this function:



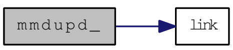
4.90.2.4 int mmdnum_ (int * *neqns*, shortint * *perm*, shortint * *invp*, shortint * *qsize*)

Here is the caller graph for this function:

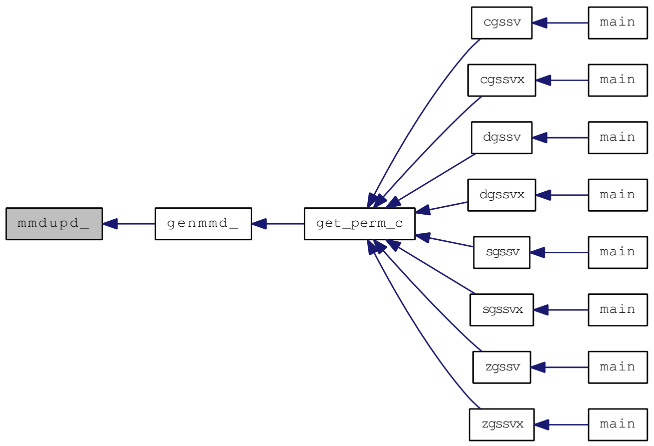


4.90.2.5 int mmdupd_ (int * *ehead*, int * *neqns*, int * *xadj*, shortint * *adjncy*, int * *delta*, int * *mdeg*, shortint * *dhead*, shortint * *dforw*, shortint * *dbakw*, shortint * *qsize*, shortint * *llist*, shortint * *marker*, int * *maxint*, int * *tag*)

Here is the call graph for this function:



Here is the caller graph for this function:



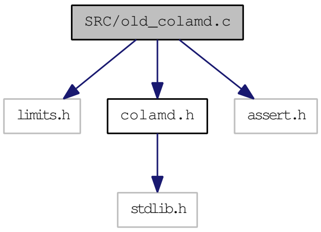
4.91 SRC/old_colamd.c File Reference

#include <limits.h>

#include "colamd.h"

#include <assert.h>

Include dependency graph for old_colamd.c:



Data Structures

- struct ColInfo_struct
- struct RowInfo_struct

Defines

- #define MAX(a, b) (((a) > (b)) ? (a) : (b))
- #define MIN(a, b) (((a) < (b)) ? (a) : (b))
- #define ONES_COMPLEMENT(r) (-(r)-1)
- #define TRUE (1)
- #define FALSE (0)
- #define EMPTY (-1)
- #define ALIVE (0)
- #define DEAD (-1)
- #define DEAD_PRINCIPAL (-1)
- #define DEAD_NON_PRINCIPAL (-2)
- #define ROW_IS_DEAD(r) ROW_IS_MARKED_DEAD (Row[r].shared2.mark)
- #define ROW_IS_MARKED_DEAD(row_mark) (row_mark < ALIVE)
- #define ROW_IS_ALIVE(r) (Row [r].shared2.mark >= ALIVE)
- #define COL_IS_DEAD(c) (Col [c].start < ALIVE)
- #define COL_IS_ALIVE(c) (Col [c].start >= ALIVE)
- #define COL_IS_DEAD_PRINCIPAL(c) (Col [c].start == DEAD_PRINCIPAL)
- #define KILL_ROW(r) { Row [r].shared2.mark = DEAD ; }
- #define KILL_PRINCIPAL_COL(c) { Col [c].start = DEAD_PRINCIPAL ; }
- #define KILL_NON_PRINCIPAL_COL(c) { Col [c].start = DEAD_NON_PRINCIPAL ; }
- #define PUBLIC
- #define PRIVATE static
- #define DEBUG0(params) ;
- #define DEBUG1(params) ;
- #define DEBUG2(params) ;
- #define DEBUG3(params) ;
- #define DEBUG4(params) ;

Typedefs

- typedef struct ColInfo_struct ColInfo
- typedef struct RowInfo_struct RowInfo

Functions

- PRIVATE int init_rows_cols (int n_row, int n_col, RowInfo Row[ ], ColInfo Col[ ], int A[ ], int p[ ])
- PRIVATE void init_scoring (int n_row, int n_col, RowInfo Row[ ], ColInfo Col[ ], int A[ ], int head[ ], double knobs[COLAMD_KNOBS], int ∗p_n_row2, int ∗p_n_col2, int ∗p_max_deg)
- PRIVATE int find_ordering (int n_row, int n_col, int Alen, RowInfo Row[ ], ColInfo Col[ ], int A[ ], int head[ ], int n_col2, int max_deg, int pfree)
- PRIVATE void order_children (int n_col, ColInfo Col[ ], int p[ ])
- PRIVATE void detect_super_cols (ColInfo Col[ ], int A[ ], int head[ ], int row_start, int row_length)
- PRIVATE int garbage_collection (int n_row, int n_col, RowInfo Row[ ], ColInfo Col[ ], int A[ ], int ∗pfree)
- PRIVATE int clear_mark (int n_row, RowInfo Row[ ])
- PUBLIC int colamd_recommended (int nnz, int n_row, int n_col)
- PUBLIC void colamd_set_defaults (double knobs[COLAMD_KNOBS])
- PUBLIC int colamd (int n_row, int n_col, int Alen, int A[ ], int p[ ], double knobs[COLAMD_-KNOBS])

4.91.1 Define Documentation

4.91.1.1 #define ALIVE (0)

4.91.1.2 #define COL_IS_ALIVE(c) (Col [c].start >= ALIVE)

4.91.1.3 #define COL_IS_DEAD(c) (Col [c].start < ALIVE)

4.91.1.4 #define COL_IS_DEAD_PRINCIPAL(c) (Col [c].start == DEAD_PRINCIPAL)

4.91.1.5 #define DEAD (-1)

4.91.1.6 #define DEAD_NON_PRINCIPAL (-2)

4.91.1.7 #define DEAD_PRINCIPAL (-1)

4.91.1.8 #define DEBUG0(params) ;

4.91.1.9 #define DEBUG1(params) ;

4.91.1.10 #define DEBUG2(params) ;

4.91.1.11 #define DEBUG3(params) ;

4.91.1.12 #define DEBUG4(params) ;

4.91.1.13 #define EMPTY (-1)

4.91.1.14 #define FALSE (0)

4.91.1.15 #define KILL_NON_PRINCIPAL_COL(c) { Col [c].start = DEAD_NON_PRINCIPAL ; }

4.91.1.16 #define KILL_PRINCIPAL_COL(c) { Col [c].start = DEAD_PRINCIPAL ; }

4.91.1.17 #define KILL_ROW(r) { Row [r].shared2.mark = DEAD ; }

4.91.1.18 #define MAX(a, b) (((a) > (b)) ? (a) : (b))

4.91.1.19 #define MIN(a, b) (((a) < (b)) ? (a) : (b))

4.91.1.20 #define ONES_COMPLEMENT(r) (-(r)-1)

4.91.1.21 #define PRIVATE static

4.91.1.22 #define PUBLIC

4.91.1.23 #define ROW_IS_ALIVE(r) (Row [r].shared2.mark >= ALIVE)

4.91.1.24 #define ROW_IS_DEAD(r) ROW_IS_MARKED_DEAD (Row[r].shared2.mark)

4.91.1.25 #define ROW_IS_MARKED_DEAD(row_mark) (row_mark < ALIVE)

4.91.1.26 #define TRUE (1)

4.91.2 Typedef Documentation

4.91.2.1 typedef struct ColInfo_struct ColInfo

4.91.2.2 typedef struct RowInfo_struct RowInfo

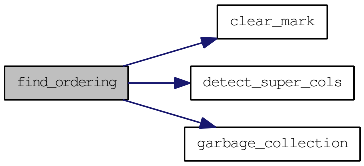
4.91.3.3 PUBLIC int colamd_recommended (int *nnz*, int *n_row*, int *n_col*)

4.91.3.4 PUBLIC void colamd_set_defaults (double *knobs*[COLAMD_KNOBS])

4.91.3.5 PRIVATE void detect_super_cols (ColInfo *Col*[ ], int *A*[ ], int *head*[ ], int *row_start*, int *row_length*)

4.91.3.6 PRIVATE int find_ordering (int *n_row*, int *n_col*, int *Alen*, RowInfo *Row*[ ], ColInfo *Col*[ ], int *A*[ ], int *head*[ ], int *n_col2*, int *max_deg*, int *pfree*)

Here is the call graph for this function:



4.91.3.7 PRIVATE int garbage_collection (int *n_row*, int *n_col*, RowInfo *Row*[ ], ColInfo *Col*[ ], int *A*[ ], int ∗ *pfree*)

4.91.3.8 PRIVATE int init_rows_cols (int *n_row*, int *n_col*, RowInfo *Row*[ ], ColInfo *Col*[ ], int *A*[ ], int *p*[ ])

4.91.3.9 PRIVATE void init_scoring (int *n_row*, int *n_col*, RowInfo *Row*[ ], ColInfo *Col*[ ], int *A*[ ], int *head*[ ], double *knobs*[COLAMD_KNOBS], int ∗ *p_n_row2*, int ∗ *p_n_col2*, int ∗ *p_max_deg*)

4.91.3.10 PRIVATE void order_children (int *n_col*, ColInfo *Col*[ ], int *p*[ ])

4.92 SRC/old_colamd.h File Reference

Defines

- #define COLAMD_KNOBS 20
- #define COLAMD_STATS 20
- #define COLAMD_DENSE_ROW 0
- #define COLAMD_DENSE_COL 1
- #define COLAMD_DEFRAG_COUNT 2
- #define COLAMD_JUMBLED_COLS 3

Functions

- int colamd_recommended (int nnz, int n_row, int n_col)
- void colamd_set_defaults (double knobs[COLAMD_KNOBS])
- int colamd (int n_row, int n_col, int Alen, int A[ ], int p[ ], double knobs[COLAMD_KNOBS])

4.92.1 Define Documentation

4.92.1.1 #define COLAMD_DEFRAG_COUNT 2

4.92.1.2 #define COLAMD_DENSE_COL 1

4.92.1.3 #define COLAMD_DENSE_ROW 0

4.92.1.4 #define COLAMD_JUMBLED_COLS 3

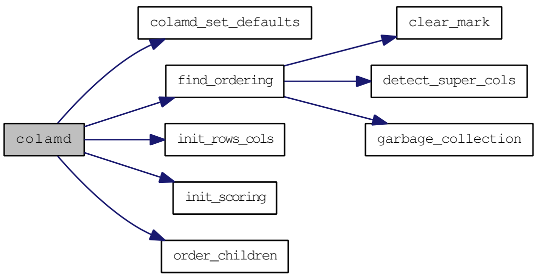
4.92.1.5 #define COLAMD_KNOBS 20

4.92.1.6 #define COLAMD_STATS 20

4.92.2 Function Documentation

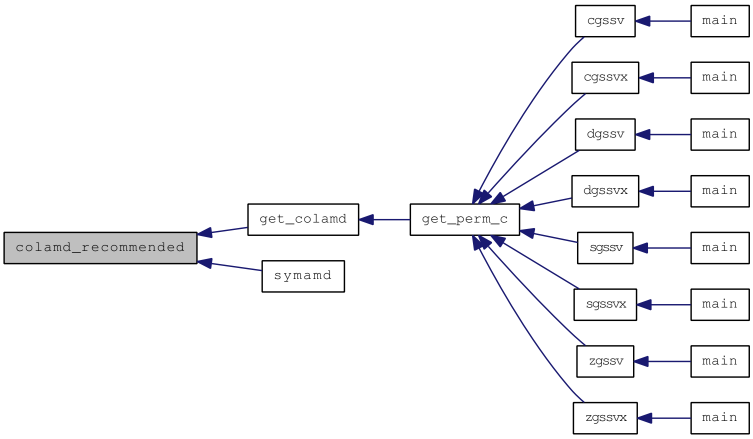
4.92.2.1 int colamd (int *n_row*, int *n_col*, int *Alen*, int *A*[ ], int *p*[ ], double *knobs*[COLAMD_KNOBS])

Here is the call graph for this function:



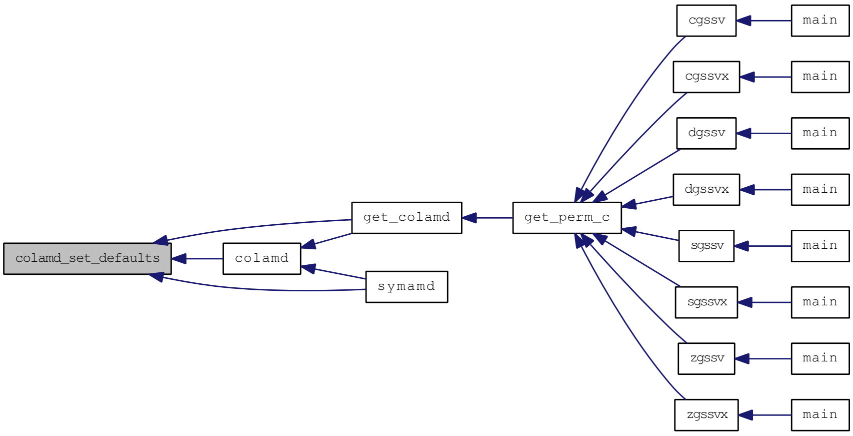
4.92.2.2 int colamd_recommended (int *nnz*, int *n_row*, int *n_col*)

Here is the caller graph for this function:



4.92.2.3 void colamd_set_defaults (double *knobs*[COLAMD_KNOBS])

Here is the caller graph for this function:

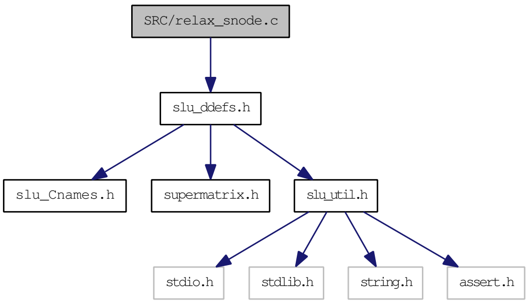


4.93 SRC/relax_snode.c File Reference

Identify initial relaxed supernodes.

`#include "slu_ddefs.h"`

Include dependency graph for relax_snode.c:



Functions

- void relax_snode (const int n, int ∗et, const int relax_columns, int ∗descendants, int ∗relax_end)

4.93.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

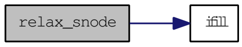
Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.93.2 Function Documentation

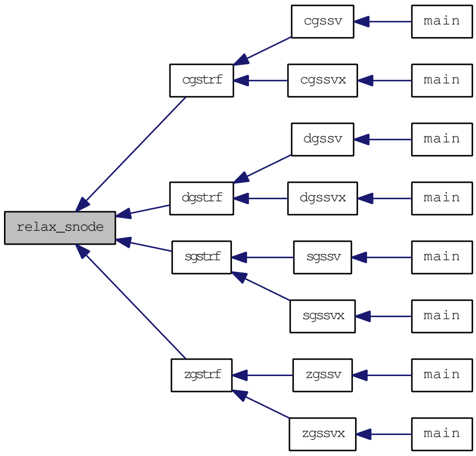
4.93.2.1 void relax_snode (const int *n*, int ∗ *et*, const int *relax_columns*, int ∗ *descendants*, int ∗ *relax_end*)

```
Purpose
=======
   relax_snode() - Identify the initial relaxed supernodes, assuming that
   the matrix has been reordered according to the postorder of the etree.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.94 SRC/scolumn_bmod.c File Reference

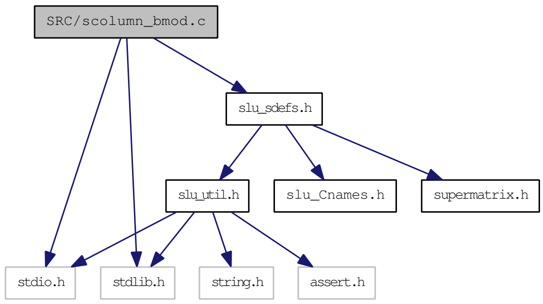
performs numeric block updates

#include <stdio.h>

#include <stdlib.h>

#include "slu_sdefs.h"

Include dependency graph for scolumn_bmod.c:



Functions

- void susolve (int, int, float ∗, float ∗)

  Solves a dense upper triangular system.

- void slsolve (int, int, float ∗, float ∗)

  Solves a dense UNIT lower triangular system.

- void smatvec (int, int, int, float ∗, float ∗, float ∗)

  Performs a dense matrix-vector multiply: Mxvec = Mxvec + M ∗ vec.

- int scolumn_bmod (const int jcol, const int nseg, float ∗dense, float ∗tempv, int ∗segrep, int ∗repfnz, int fpanelc, GlobalLU_t ∗Glu, SuperLUStat_t ∗stat)

4.94.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

 Permission is hereby granted to use or copy this program for any
 purpose, provided the above notices are retained on all copies.
 Permission to modify the code and to distribute modified code is
```

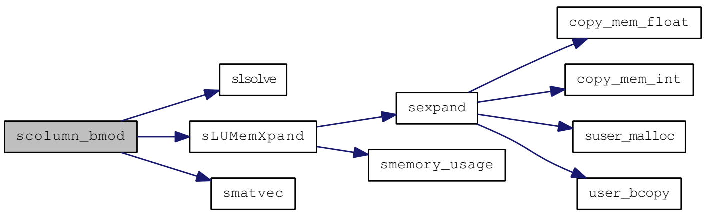
```
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.94.2 Function Documentation

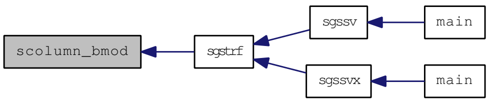
4.94.2.1 int scolumn_bmod (const int *jcol*, const int *nseg*, float ∗ *dense*, float ∗ *tempv*, int ∗ *segrep*, int ∗ *repfnz*, int *fpanelc*, GlobalLU_t ∗ *Glu*, SuperLUStat_t ∗ *stat*)

```
Purpose:
========
Performs numeric block updates (sup-col) in topological order.
It features: col-col, 2cols-col, 3cols-col, and sup-col updates.
Special processing on the supernodal portion of L[*,j]
Return value:   0 - successful return
              > 0 - number of bytes allocated when run out of space
```

Here is the call graph for this function:



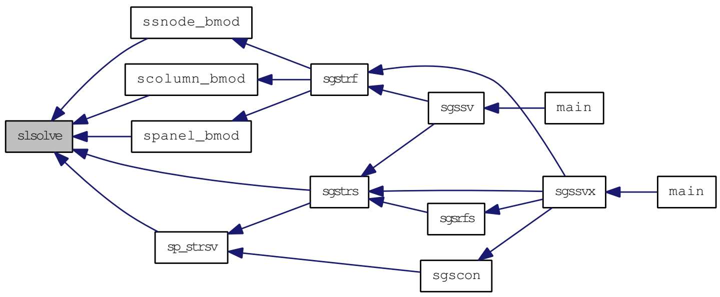
Here is the caller graph for this function:



4.94.2.2 void slsolve (int *ldm*, int *ncol*, float ∗ *M*, float ∗ *rhs*)

The unit lower triangular matrix is stored in a 2D array M(1:nrow,1:ncol). The solution will be returned in the rhs vector.

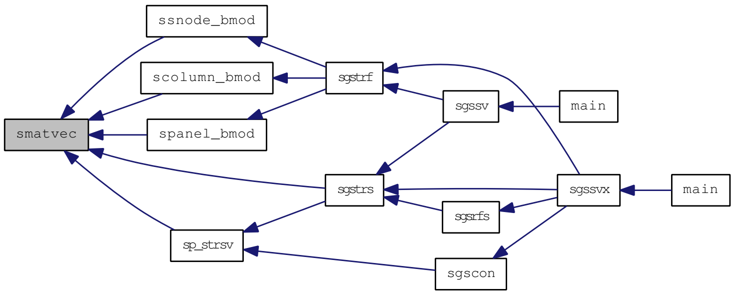
Here is the caller graph for this function:



4.94.2.3 void smatvec (int *ldm*, int *nrow*, int *ncol*, float ∗ *M*, float ∗ *vec*, float ∗ *Mxvec*)

The input matrix is M(1:nrow,1:ncol); The product is returned in Mxvec[].

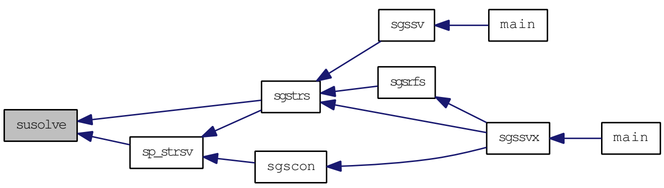
Here is the caller graph for this function:



4.94.2.4 void susolve (int *ldm*, int *ncol*, float ∗ *M*, float ∗ *rhs*)

The upper triangular matrix is stored in a 2-dim array M(1:ldm,1:ncol). The solution will be returned in the rhs vector.

Here is the caller graph for this function:

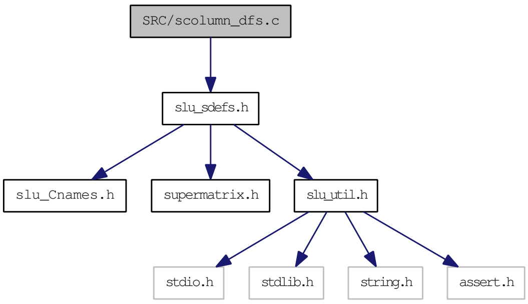


4.95 SRC/scolumn_dfs.c File Reference

Performs a symbolic factorization.

`#include "slu_sdefs.h"`

Include dependency graph for scolumn_dfs.c:



Defines

- #define T2_SUPER

  What type of supernodes we want.

Functions

- int scolumn_dfs (const int m, const int jcol, int ∗perm_r, int ∗nseg, int ∗lsub_col, int ∗segrep, int ∗repfnz, int ∗xprune, int ∗marker, int ∗parent, int ∗xplore, GlobalLU_t ∗Glu)

4.95.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003


Copyright (c) 1994 by Xerox Corporation.  All rights reserved.


THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.


Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.95.2 Define Documentation

4.95.2.1 #define T2_SUPER

4.95.3 Function Documentation

4.95.3.1 int scolumn_dfs (const int *m*, const int *jcol*, int ∗ *perm_r*, int ∗ *nseg*, int ∗ *lsub_col*, int ∗ *segrep*, int ∗ *repfnz*, int ∗ *xprune*, int ∗ *marker*, int ∗ *parent*, int ∗ *xplore*, GlobalLU_t ∗ *Glu*)

```
 Purpose
 =======
   "column_dfs" performs a symbolic factorization on column jcol, and
   decide the supernode boundary.


   This routine does not use numeric values, but only use the RHS
   row indices to start the dfs.

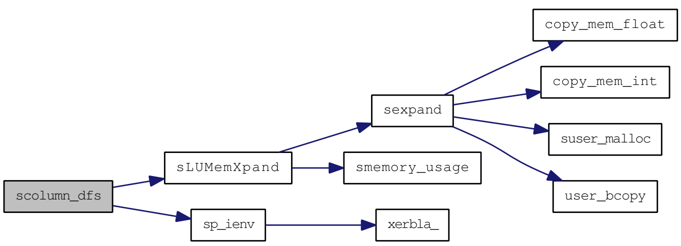

   A supernode representative is the last column of a supernode.
   The nonzeros in U[*,j] are segments that end at supernodal
   representatives. The routine returns a list of such supernodal
   representatives in topological order of the dfs that generates them.
   The location of the first nonzero in each such supernodal segment
   (supernodal entry location) is also returned.


 Local parameters
 ================
   nseg: no of segments in current U[*,j]
   jsuper: jsuper=EMPTY if column j does not belong to the same
supernode as j-1. Otherwise, jsuper=nsuper.

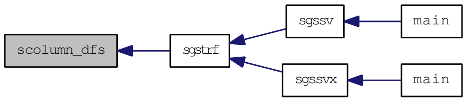

   marker2: A-row --> A-row/col (0/1)
   repfnz: SuperA-col --> PA-row
   parent: SuperA-col --> SuperA-col
   xplore: SuperA-col --> index to L-structure


 Return value
 ============
     0  success;
   > 0  number of bytes allocated when run out of space.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.96 SRC/scomplex.c File Reference

Common arithmetic for complex type.

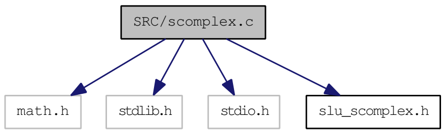
`#include <math.h>`

`#include <stdlib.h>`

`#include <stdio.h>`

`#include "slu_scomplex.h"`

Include dependency graph for scomplex.c:



Functions

- void c_div (complex ∗c, complex ∗a, complex ∗b)

  Complex Division c = a/b.

- double c_abs (complex ∗z)

  Returns sqrt(z.r^2 + z.i^2).

- double c_abs1 (complex ∗z)

  Approximates the abs. Returns abs(z.r) + abs(z.i).

- void c_exp (complex ∗r, complex ∗z)

  Return the exponentiation.

- void r_cnjg (complex ∗r, complex ∗z)

  Return the complex conjugate.

- double r_imag (complex ∗z)

  Return the imaginary part.

4.96.1 Detailed Description

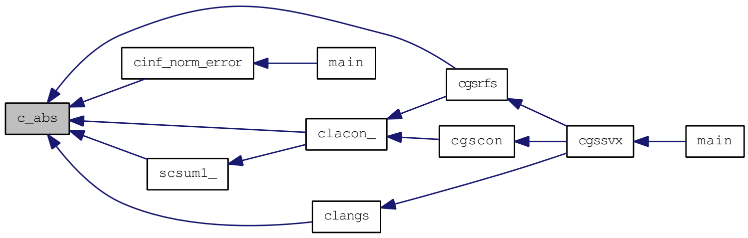
```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997


This file defines common arithmetic operations for complex type.
```

4.96.2 Function Documentation

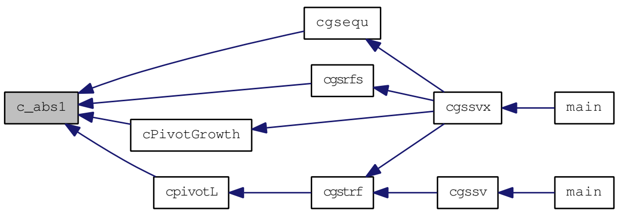
4.96.2.1 double c_abs (complex ∗ z)

Here is the caller graph for this function:



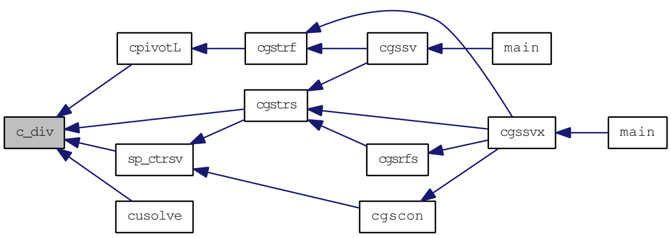
4.96.2.2 double c_abs1 (complex ∗ z)

Here is the caller graph for this function:



4.96.2.3 void c_div (complex ∗ c, complex ∗ a, complex ∗ b)

Here is the caller graph for this function:



4.96.2.4 void c_exp (complex ∗ r, complex ∗ z)

4.96.2.5 void r_cnjg (complex ∗ r, complex ∗ z)

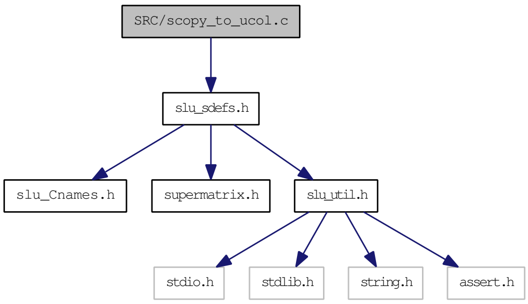
4.96.2.6 double r_imag (complex ∗ z)

4.97 SRC/scopy_to_ucol.c File Reference

Copy a computed column of U to the compressed data structure.

`#include "slu_sdefs.h"`

Include dependency graph for scopy_to_ucol.c:



Functions

- int scopy_to_ucol (int jcol, int nseg, int ∗segrep, int ∗repfnz, int ∗perm_r, float ∗dense, GlobalLU_t ∗Glu)

4.97.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997
Copyright (c) 1994 by Xerox Corporation.  All rights reserved.



THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

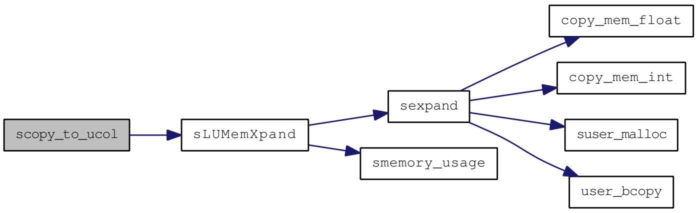


Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

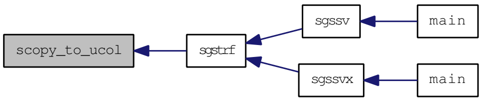
4.97.2 Function Documentation

4.97.2.1 int scopy_to_ucol (int *jcol*, int *nseg*, int ∗ *segrep*, int ∗ *repfnz*, int ∗ *perm_r*, float ∗ *dense*, GlobalLU_t ∗ *Glu*)

Here is the call graph for this function:



Here is the caller graph for this function:



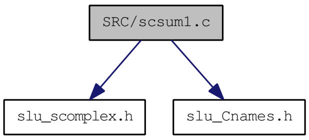
4.98 SRC/scsum1.c File Reference

Takes sum of the absolute values of a complex vector and returns a single precision result.

#include "slu_scomplex.h"

#include "slu_Cnames.h"

Include dependency graph for scsum1.c:



Defines

- #define CX(I) cx[(I)-1]

Functions

- double scsum1_ (int *n, complex *cx, int *incx)

4.98.1 Detailed Description

```
-- LAPACK auxiliary routine (version 2.0) --
Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
Courant Institute, Argonne National Lab, and Rice University
October 31, 1992
```

4.98.2 Define Documentation

4.98.2.1 #define CX(I) cx[(I)-1]

4.98.3 Function Documentation

4.98.3.1 double scsum1_ (int * *n*, complex * *cx*, int * *incx*)

```
Purpose
=======

SCSUM1 takes the sum of the absolute values of a complex
vector and returns a single precision result.

Based on SCASUM from the Level 1 BLAS.
The change is to use the 'genuine' absolute value.

Contributed by Nick Higham for use with CLACON.
```

```
Arguments
=========

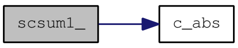
N       (input) INT
        The number of elements in the vector CX.

CX      (input) COMPLEX array, dimension (N)
        The vector whose elements will be summed.

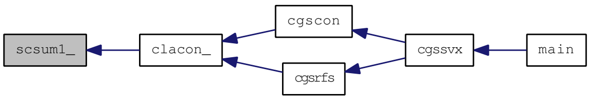
INCX    (input) INT
        The spacing between successive values of CX.  INCX > 0.

=====================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:



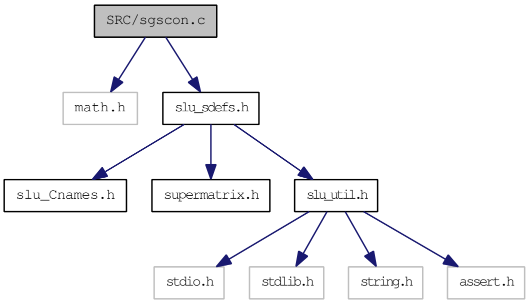
4.99 SRC/sgscon.c File Reference

Estimates reciprocal of the condition number of a general matrix.

`#include <math.h>`

`#include "slu_sdefs.h"`

Include dependency graph for sgscon.c:



Functions

- void sgscon (char ∗norm, SuperMatrix ∗L, SuperMatrix ∗U, float anorm, float ∗rcond, SuperLUStat_t ∗stat, int ∗info)

4.99.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003


Modified from lapack routines SGECON.
```

4.99.2 Function Documentation

4.99.2.1 void sgscon (char ∗ *norm*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, float *anorm*, float ∗ *rcond*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======


SGSCON estimates the reciprocal of the condition number of a general
real matrix A, in either the 1-norm or the infinity-norm, using
the LU factorization computed by SGETRF.   *


An estimate is obtained for norm(inv(A)), and the reciprocal of the
condition number is computed as
   RCOND = 1 / ( norm(A) * norm(inv(A)) ).
```

```
See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

 NORM    (input) char*
         Specifies whether the 1-norm condition number or the
         infinity-norm condition number is required:
         = '1' or 'O':  1-norm;
         = 'I':         Infinity-norm.

 L       (input) SuperMatrix*
         The factor L from the factorization Pr*A*Pc=L*U as computed by
         sgstrf(). Use compressed row subscripts storage for supernodes,
         i.e., L has types: Stype = SLU_SC, Dtype = SLU_S, Mtype = SLU_TRLU.

 U       (input) SuperMatrix*
         The factor U from the factorization Pr*A*Pc=L*U as computed by
         sgstrf(). Use column-wise storage scheme, i.e., U has types:
         Stype = SLU_NC, Dtype = SLU_S, Mtype = SLU_TRU.

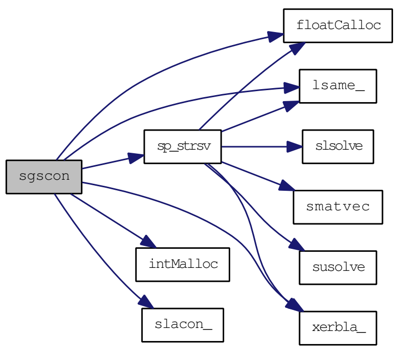
 ANORM   (input) float
         If NORM = '1' or 'O', the 1-norm of the original matrix A.
         If NORM = 'I', the infinity-norm of the original matrix A.

 RCOND   (output) float*
        The reciprocal of the condition number of the matrix A,
        computed as RCOND = 1/(norm(A) * norm(inv(A))).

 INFO    (output) int*
        = 0:  successful exit
        < 0:  if INFO = -i, the i-th argument had an illegal value

 ======================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:

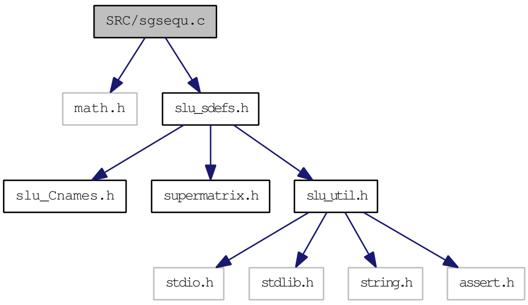
4.100 SRC/sgsequ.c File Reference

Computes row and column scalings.

`#include <math.h>`

`#include "slu_sdefs.h"`

Include dependency graph for sgsequ.c:



Functions

- void sgsequ (SuperMatrix ∗A, float ∗r, float ∗c, float ∗rowcnd, float ∗colcnd, float ∗amax, int ∗info)

  Driver related.

4.100.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997


Modified from LAPACK routine SGEEQU
```

4.100.2 Function Documentation

4.100.2.1 void sgsequ (SuperMatrix ∗ *A*, float ∗ *r*, float ∗ *c*, float ∗ *rowcnd*, float ∗ *colcnd*, float ∗ *amax*, int ∗ *info*)

```
Purpose
  =======


  SGSEQU computes row and column scalings intended to equilibrate an
  M-by-N sparse matrix A and reduce its condition number. R returns the row
  scale factors and C the column scale factors, chosen to try to make
  the largest element in each row and column of the matrix B with
  elements B(i,j)=R(i)*A(i,j)*C(j) have absolute value 1.
```

```
R(i) and C(j) are restricted to be between SMLNUM = smallest safe
number and BIGNUM = largest safe number.  Use of these scaling
factors is not guaranteed to reduce the condition number of A but
works well in practice.


See supermatrix.h for the definition of 'SuperMatrix' structure.


Arguments
=========


A       (input) SuperMatrix*
        The matrix of dimension (A->nrow, A->ncol) whose equilibration
        factors are to be computed. The type of A can be:
        Stype = SLU_NC; Dtype = SLU_S; Mtype = SLU_GE.


R       (output) float*, size A->nrow
        If INFO = 0 or INFO > M, R contains the row scale factors
        for A.


C       (output) float*, size A->ncol
        If INFO = 0,  C contains the column scale factors for A.


ROWCND  (output) float*
        If INFO = 0 or INFO > M, ROWCND contains the ratio of the
        smallest R(i) to the largest R(i).  If ROWCND >= 0.1 and
        AMAX is neither too large nor too small, it is not worth
        scaling by R.

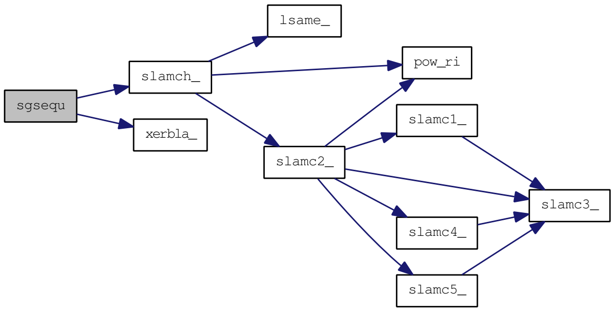

COLCND  (output) float*
        If INFO = 0, COLCND contains the ratio of the smallest
        C(i) to the largest C(i).  If COLCND >= 0.1, it is not
        worth scaling by C.


AMAX    (output) float*
        Absolute value of largest matrix element.  If AMAX is very
        close to overflow or very close to underflow, the matrix
        should be scaled.


INFO    (output) int*
        = 0:  successful exit
        < 0:  if INFO = -i, the i-th argument had an illegal value
        > 0:  if INFO = i,  and i is
              <= A->nrow:  the i-th row of A is exactly zero
              >  A->ncol:  the (i-M)-th column of A is exactly zero


=====================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:

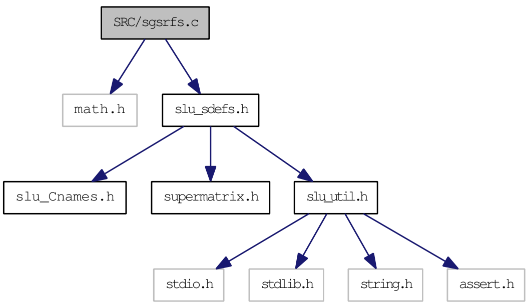
4.101 SRC/sgsrfs.c File Reference

Improves computed solution to a system of inear equations.

`#include <math.h>`

`#include "slu_sdefs.h"`

Include dependency graph for sgsrfs.c:



Defines

- #define ITMAX 5

Functions

- void sgsrfs (trans_t trans, SuperMatrix ∗A, SuperMatrix ∗L, SuperMatrix ∗U, int ∗perm_c, int ∗perm_r, char ∗equed, float ∗R, float ∗C, SuperMatrix ∗B, SuperMatrix ∗X, float ∗ferr, float ∗berr, SuperLUStat_t ∗stat, int ∗info)

4.101.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003
```

```
Modified from lapack routine SGERFS
```

4.101.2 Define Documentation

4.101.2.1 #define ITMAX 5

4.101.3 Function Documentation

4.101.3.1 void sgsrfs (trans_t *trans*, SuperMatrix ∗ *A*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, int ∗ *perm_c*, int ∗ *perm_r*, char ∗ *equed*, float ∗ *R*, float ∗ *C*, SuperMatrix ∗ *B*, SuperMatrix ∗ *X*, float ∗ *ferr*, float ∗ *berr*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
  Purpose
  =======

  SGSRFS improves the computed solution to a system of linear
  equations and provides error bounds and backward error estimates for
  the solution.

  If equilibration was performed, the system becomes:
          (diag(R)*A_original*diag(C)) * X = diag(R)*B_original.

  See supermatrix.h for the definition of 'SuperMatrix' structure.

  Arguments
  =========

 trans   (input) trans_t
          Specifies the form of the system of equations:
          = NOTRANS: A * X = B  (No transpose)
          = TRANS:   A'* X = B  (Transpose)
          = CONJ:    A**H * X = B  (Conjugate transpose)

  A       (input) SuperMatrix*
          The original matrix A in the system, or the scaled A if
          equilibration was done. The type of A can be:
          Stype = SLU_NC, Dtype = SLU_S, Mtype = SLU_GE.

  L       (input) SuperMatrix*
    The factor L from the factorization Pr*A*Pc=L*U. Use
          compressed row subscripts storage for supernodes,
          i.e., L has types: Stype = SLU_SC, Dtype = SLU_S, Mtype = SLU_TRLU.

  U       (input) SuperMatrix*
          The factor U from the factorization Pr*A*Pc=L*U as computed by
          sgstrf(). Use column-wise storage scheme,
          i.e., U has types: Stype = SLU_NC, Dtype = SLU_S, Mtype = SLU_TRU.

  perm_c  (input) int*, dimension (A->ncol)
    Column permutation vector, which defines the
          permutation matrix Pc; perm_c[i] = j means column i of A is
          in position j in A*Pc.

  perm_r  (input) int*, dimension (A->nrow)
          Row permutation vector, which defines the permutation matrix Pr;
          perm_r[i] = j means row i of A is in position j in Pr*A.
```

```
equed   (input) Specifies the form of equilibration that was done.
        = 'N': No equilibration.
        = 'R': Row equilibration, i.e., A was premultiplied by diag(R).
        = 'C': Column equilibration, i.e., A was postmultiplied by
               diag(C).
        = 'B': Both row and column equilibration, i.e., A was replaced
               by diag(R)*A*diag(C).

R       (input) float*, dimension (A->nrow)
        The row scale factors for A.
        If equed = 'R' or 'B', A is premultiplied by diag(R).
        If equed = 'N' or 'C', R is not accessed.

C       (input) float*, dimension (A->ncol)
        The column scale factors for A.
        If equed = 'C' or 'B', A is postmultiplied by diag(C).
        If equed = 'N' or 'R', C is not accessed.

B       (input) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_S, Mtype = SLU_GE.
        The right hand side matrix B.
        if equed = 'R' or 'B', B is premultiplied by diag(R).

X       (input/output) SuperMatrix*
        X has types: Stype = SLU_DN, Dtype = SLU_S, Mtype = SLU_GE.
        On entry, the solution matrix X, as computed by sgstrs().
        On exit, the improved solution matrix X.
        if *equed = 'C' or 'B', X should be premultiplied by diag(C)
            in order to obtain the solution to the original system.

FERR    (output) float*, dimension (B->ncol)
        The estimated forward error bound for each solution vector
        X(j) (the j-th column of the solution matrix X).
        If XTRUE is the true solution corresponding to X(j), FERR(j)
        is an estimated upper bound for the magnitude of the largest
        element in (X(j) - XTRUE) divided by the magnitude of the
        largest element in X(j).  The estimate is as reliable as
        the estimate for RCOND, and is almost always a slight
        overestimate of the true error.

BERR    (output) float*, dimension (B->ncol)
        The componentwise relative backward error of each solution
        vector X(j) (i.e., the smallest relative change in
        any element of A or B that makes X(j) an exact solution).

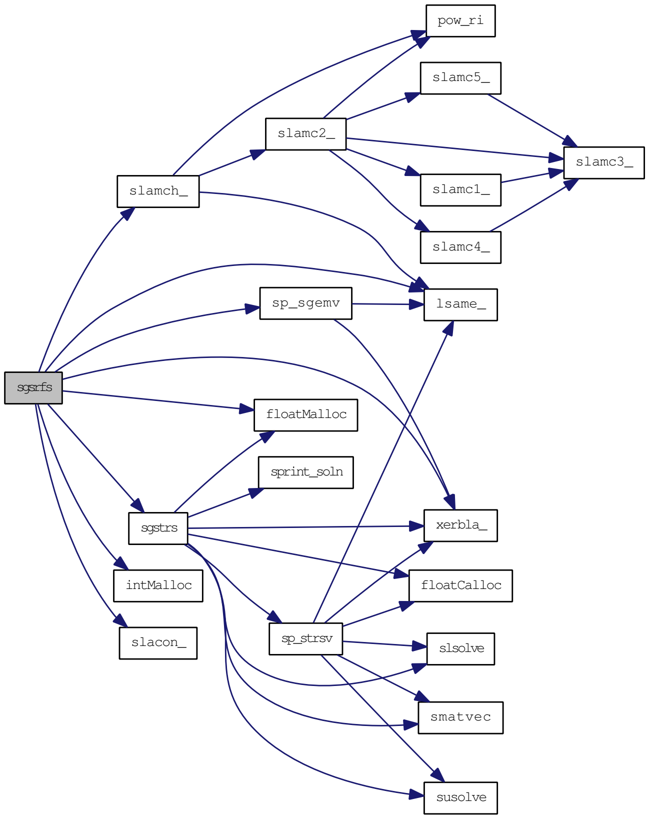
stat     (output) SuperLUStat_t*
         Record the statistics on runtime and floating-point operation count.
         See util.h for the definition of 'SuperLUStat_t'.

info    (output) int*
        = 0:  successful exit
         < 0:  if INFO = -i, the i-th argument had an illegal value

 Internal Parameters
 ===================
```

```
ITMAX is the maximum number of steps of iterative refinement.
```

Here is the call graph for this function:



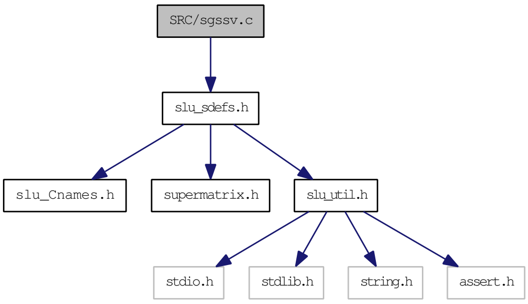
Here is the caller graph for this function:

4.102 SRC/sgssv.c File Reference

Solves the system of linear equations A∗X=B.

`#include "slu_sdefs.h"`

Include dependency graph for sgssv.c:



Functions

- void sgssv (superlu_options_t ∗options, SuperMatrix ∗A, int ∗perm_c, int ∗perm_r, SuperMatrix ∗L, SuperMatrix ∗U, SuperMatrix ∗B, SuperLUStat_t ∗stat, int ∗info)

    Driver routines.

4.102.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003
```

4.102.2 Function Documentation

4.102.2.1 void sgssv (superlu_options_t ∗ *options*, SuperMatrix ∗ *A*, int ∗ *perm_c*, int ∗ *perm_r*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, SuperMatrix ∗ *B*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======

SGSSV solves the system of linear equations A*X=B, using the
LU factorization from SGSTRF. It performs the following steps:

  1. If A is stored column-wise (A->Stype = SLU_NC):

     1.1. Permute the columns of A, forming A*Pc, where Pc
          is a permutation matrix. For more details of this step,
          see sp_preorder.c.
```

```
      1.2. Factor A as Pr*A*Pc=L*U with the permutation Pr determined
           by Gaussian elimination with partial pivoting.
           L is unit lower triangular with offdiagonal entries
           bounded by 1 in magnitude, and U is upper triangular.

      1.3. Solve the system of equations A*X=B using the factored
           form of A.

   2. If A is stored row-wise (A->Stype = SLU_NR), apply the
      above algorithm to the transpose of A:

      2.1. Permute columns of transpose(A) (rows of A),
           forming transpose(A)*Pc, where Pc is a permutation matrix.
           For more details of this step, see sp_preorder.c.

      2.2. Factor A as Pr*transpose(A)*Pc=L*U with the permutation Pr
           determined by Gaussian elimination with partial pivoting.
           L is unit lower triangular with offdiagonal entries
           bounded by 1 in magnitude, and U is upper triangular.

      2.3. Solve the system of equations A*X=B using the factored
           form of A.

   See supermatrix.h for the definition of 'SuperMatrix' structure.

 Arguments
 =========

 options (input) superlu_options_t*
         The structure defines the input parameters to control
         how the LU decomposition will be performed and how the
         system will be solved.

 A       (input) SuperMatrix*
         Matrix A in A*X=B, of dimension (A->nrow, A->ncol). The number
         of linear equations is A->nrow. Currently, the type of A can be:
         Stype = SLU_NC or SLU_NR; Dtype = SLU_S; Mtype = SLU_GE.
         In the future, more general A may be handled.

 perm_c  (input/output) int*
         If A->Stype = SLU_NC, column permutation vector of size A->ncol
         which defines the permutation matrix Pc; perm_c[i] = j means
         column i of A is in position j in A*Pc.
         If A->Stype = SLU_NR, column permutation vector of size A->nrow
         which describes permutation of columns of transpose(A)
         (rows of A) as described above.

         If options->ColPerm = MY_PERMC or options->Fact = SamePattern or
            options->Fact = SamePattern_SameRowPerm, it is an input argument.
            On exit, perm_c may be overwritten by the product of the input
            perm_c and a permutation that postorders the elimination tree
            of Pc'*A'*A*Pc; perm_c is not changed if the elimination tree
            is already in postorder.
         Otherwise, it is an output argument.
```

```
perm_r  (input/output) int*
        If A->Stype = SLU_NC, row permutation vector of size A->nrow,
        which defines the permutation matrix Pr, and is determined
        by partial pivoting.  perm_r[i] = j means row i of A is in
        position j in Pr*A.
        If A->Stype = SLU_NR, permutation vector of size A->ncol, which
        determines permutation of rows of transpose(A)
        (columns of A) as described above.



        If options->RowPerm = MY_PERMR or
           options->Fact = SamePattern_SameRowPerm, perm_r is an
           input argument.
        otherwise it is an output argument.



L       (output) SuperMatrix*
        The factor L from the factorization
            Pr*A*Pc=L*U              (if A->Stype = SLU_NC) or
            Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
        Uses compressed row subscripts storage for supernodes, i.e.,
        L has types: Stype = SLU_SC, Dtype = SLU_S, Mtype = SLU_TRLU.



U       (output) SuperMatrix*
  The factor U from the factorization
            Pr*A*Pc=L*U              (if A->Stype = SLU_NC) or
            Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
        Uses column-wise storage scheme, i.e., U has types:
        Stype = SLU_NC, Dtype = SLU_S, Mtype = SLU_TRU.



B       (input/output) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_S, Mtype = SLU_GE.
        On entry, the right hand side matrix.
        On exit, the solution matrix if info = 0;

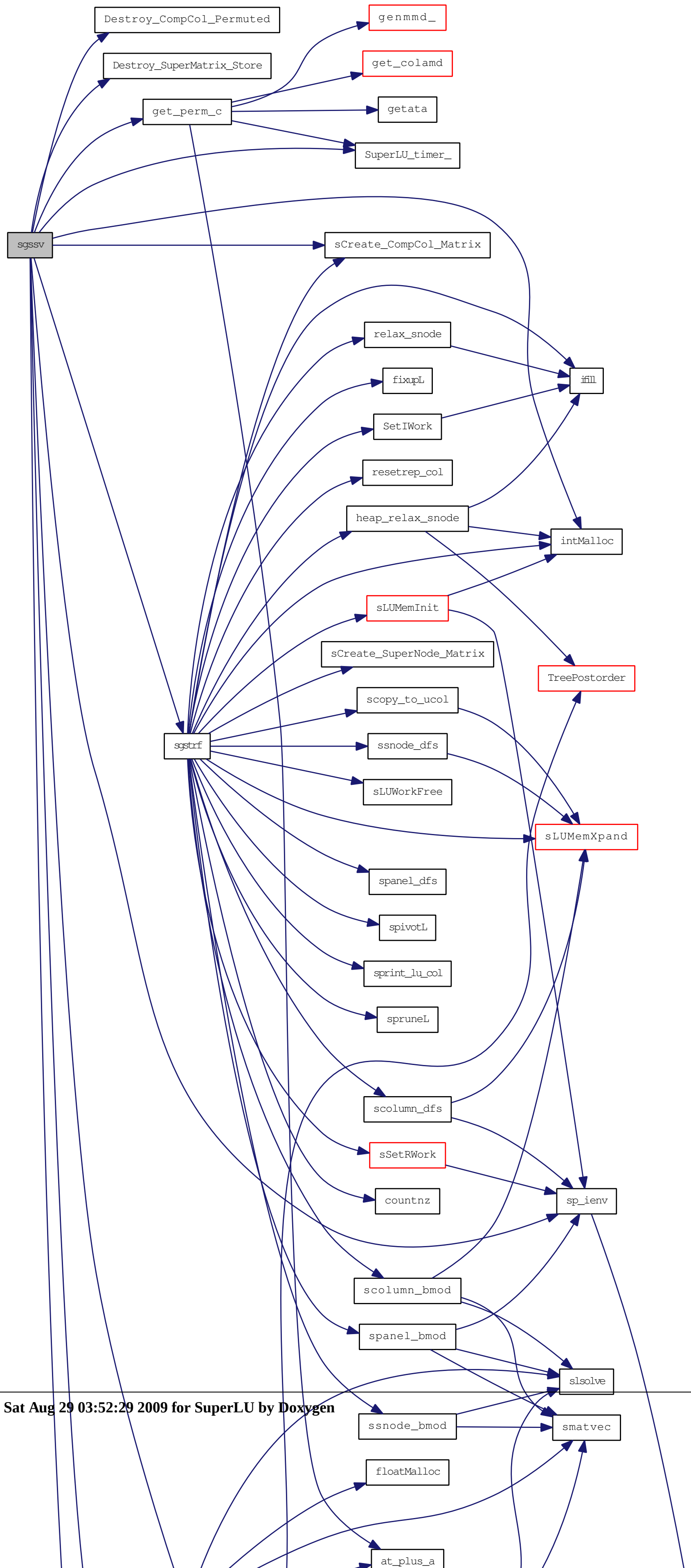


stat   (output) SuperLUStat_t*
       Record the statistics on runtime and floating-point operation count.
       See util.h for the definition of 'SuperLUStat_t'.

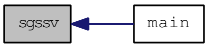


info    (output) int*
  = 0: successful exit
        > 0: if info = i, and i is
            <= A->ncol: U(i,i) is exactly zero. The factorization has
               been completed, but the factor U is exactly singular,
               so the solution could not be computed.
            > A->ncol: number of bytes allocated when memory allocation
               failure occurred, plus A->ncol.
```

Here is the call graph for this function:



Here is the caller graph for this function:

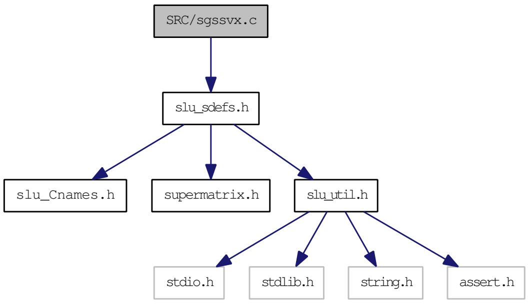


4.103 SRC/sgssvx.c File Reference

Solves the system of linear equations A∗X=B or A'∗X=B.

`#include "slu_sdefs.h"`

Include dependency graph for sgssvx.c:



Functions

- void sgssvx (superlu_options_t ∗options, SuperMatrix ∗A, int ∗perm_c, int ∗perm_r, int ∗etree, char ∗equed, float ∗R, float ∗C, SuperMatrix ∗L, SuperMatrix ∗U, void ∗work, int lwork, SuperMatrix ∗B, SuperMatrix ∗X, float ∗recip_pivot_growth, float ∗rcond, float ∗ferr, float ∗berr, mem_usage_t ∗mem_usage, SuperLUStat_t ∗stat, int ∗info)

4.103.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003
```

4.103.2 Function Documentation

4.103.2.1 void sgssvx (superlu_options_t ∗ *options*, SuperMatrix ∗ *A*, int ∗ *perm_c*, int ∗ *perm_r*, int ∗ *etree*, char ∗ *equed*, float ∗ *R*, float ∗ *C*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, void ∗ *work*, int *lwork*, SuperMatrix ∗ *B*, SuperMatrix ∗ *X*, float ∗ *recip_pivot_growth*, float ∗ *rcond*, float ∗ *ferr*, float ∗ *berr*, mem_usage_t ∗ *mem_usage*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======


SGSSVX solves the system of linear equations A*X=B or A'*X=B, using
the LU factorization from sgstrf(). Error bounds on the solution and
a condition estimate are also provided. It performs the following steps:


  1. If A is stored column-wise (A->Stype = SLU_NC):
```

```
   1.1. If options->Equil = YES, scaling factors are computed to
        equilibrate the system:
        options->Trans = NOTRANS:
            diag(R)*A*diag(C) *inv(diag(C))*X = diag(R)*B
        options->Trans = TRANS:
            (diag(R)*A*diag(C))**T *inv(diag(R))*X = diag(C)*B
        options->Trans = CONJ:
            (diag(R)*A*diag(C))**H *inv(diag(R))*X = diag(C)*B
        Whether or not the system will be equilibrated depends on the
        scaling of the matrix A, but if equilibration is used, A is
        overwritten by diag(R)*A*diag(C) and B by diag(R)*B
        (if options->Trans=NOTRANS) or diag(C)*B (if options->Trans
        = TRANS or CONJ).

   1.2. Permute columns of A, forming A*Pc, where Pc is a permutation
        matrix that usually preserves sparsity.
        For more details of this step, see sp_preorder.c.

   1.3. If options->Fact != FACTORED, the LU decomposition is used to
        factor the matrix A (after equilibration if options->Equil = YES)
        as Pr*A*Pc = L*U, with Pr determined by partial pivoting.

   1.4. Compute the reciprocal pivot growth factor.

   1.5. If some U(i,i) = 0, so that U is exactly singular, then the
        routine returns with info = i. Otherwise, the factored form of
        A is used to estimate the condition number of the matrix A. If
        the reciprocal of the condition number is less than machine
        precision, info = A->ncol+1 is returned as a warning, but the
        routine still goes on to solve for X and computes error bounds
        as described below.

   1.6. The system of equations is solved for X using the factored form
        of A.

   1.7. If options->IterRefine != NOREFINE, iterative refinement is
        applied to improve the computed solution matrix and calculate
        error bounds and backward error estimates for it.

   1.8. If equilibration was used, the matrix X is premultiplied by
        diag(C) (if options->Trans = NOTRANS) or diag(R)
        (if options->Trans = TRANS or CONJ) so that it solves the
        original system before equilibration.

2. If A is stored row-wise (A->Stype = SLU_NR), apply the above algorithm
   to the transpose of A:

   2.1. If options->Equil = YES, scaling factors are computed to
        equilibrate the system:
        options->Trans = NOTRANS:
            diag(R)*A*diag(C) *inv(diag(C))*X = diag(R)*B
        options->Trans = TRANS:
            (diag(R)*A*diag(C))**T *inv(diag(R))*X = diag(C)*B
        options->Trans = CONJ:
            (diag(R)*A*diag(C))**H *inv(diag(R))*X = diag(C)*B
```

```
          Whether or not the system will be equilibrated depends on the
          scaling of the matrix A, but if equilibration is used, A' is
          overwritten by diag(R)*A'*diag(C) and B by diag(R)*B
          (if trans='N') or diag(C)*B (if trans = 'T' or 'C').

     2.2. Permute columns of transpose(A) (rows of A),
          forming transpose(A)*Pc, where Pc is a permutation matrix that
          usually preserves sparsity.
          For more details of this step, see sp_preorder.c.

     2.3. If options->Fact != FACTORED, the LU decomposition is used to
          factor the transpose(A) (after equilibration if
          options->Fact = YES) as Pr*transpose(A)*Pc = L*U with the
          permutation Pr determined by partial pivoting.

     2.4. Compute the reciprocal pivot growth factor.

     2.5. If some U(i,i) = 0, so that U is exactly singular, then the
          routine returns with info = i. Otherwise, the factored form
          of transpose(A) is used to estimate the condition number of the
          matrix A. If the reciprocal of the condition number
          is less than machine precision, info = A->nrow+1 is returned as
          a warning, but the routine still goes on to solve for X and
          computes error bounds as described below.

     2.6. The system of equations is solved for X using the factored form
          of transpose(A).

     2.7. If options->IterRefine != NOREFINE, iterative refinement is
          applied to improve the computed solution matrix and calculate
          error bounds and backward error estimates for it.

     2.8. If equilibration was used, the matrix X is premultiplied by
          diag(C) (if options->Trans = NOTRANS) or diag(R)
          (if options->Trans = TRANS or CONJ) so that it solves the
          original system before equilibration.

   See supermatrix.h for the definition of 'SuperMatrix' structure.

 Arguments
 =========

 options (input) superlu_options_t*
         The structure defines the input parameters to control
         how the LU decomposition will be performed and how the
         system will be solved.

 A       (input/output) SuperMatrix*
         Matrix A in A*X=B, of dimension (A->nrow, A->ncol). The number
         of the linear equations is A->nrow. Currently, the type of A can be:
         Stype = SLU_NC or SLU_NR, Dtype = SLU_D, Mtype = SLU_GE.
         In the future, more general A may be handled.
```

```
        On entry, If options->Fact = FACTORED and equed is not 'N',
        then A must have been equilibrated by the scaling factors in
        R and/or C.
        On exit, A is not modified if options->Equil = NO, or if
        options->Equil = YES but equed = 'N' on exit.
        Otherwise, if options->Equil = YES and equed is not 'N',
        A is scaled as follows:
        If A->Stype = SLU_NC:
          equed = 'R':  A := diag(R) * A
          equed = 'C':  A := A * diag(C)
          equed = 'B':  A := diag(R) * A * diag(C).
        If A->Stype = SLU_NR:
          equed = 'R':  transpose(A) := diag(R) * transpose(A)
          equed = 'C':  transpose(A) := transpose(A) * diag(C)
          equed = 'B':  transpose(A) := diag(R) * transpose(A) * diag(C).

perm_c  (input/output) int*
  If A->Stype = SLU_NC, Column permutation vector of size A->ncol,
        which defines the permutation matrix Pc; perm_c[i] = j means
        column i of A is in position j in A*Pc.
        On exit, perm_c may be overwritten by the product of the input
        perm_c and a permutation that postorders the elimination tree
        of Pc'*A'*A*Pc; perm_c is not changed if the elimination tree
        is already in postorder.

        If A->Stype = SLU_NR, column permutation vector of size A->nrow,
        which describes permutation of columns of transpose(A)
        (rows of A) as described above.

perm_r  (input/output) int*
        If A->Stype = SLU_NC, row permutation vector of size A->nrow,
        which defines the permutation matrix Pr, and is determined
        by partial pivoting.  perm_r[i] = j means row i of A is in
        position j in Pr*A.

        If A->Stype = SLU_NR, permutation vector of size A->ncol, which
        determines permutation of rows of transpose(A)
        (columns of A) as described above.

        If options->Fact = SamePattern_SameRowPerm, the pivoting routine
        will try to use the input perm_r, unless a certain threshold
        criterion is violated. In that case, perm_r is overwritten by a
        new permutation determined by partial pivoting or diagonal
        threshold pivoting.
        Otherwise, perm_r is output argument.

etree   (input/output) int*,  dimension (A->ncol)
        Elimination tree of Pc'*A'*A*Pc.
        If options->Fact != FACTORED and options->Fact != DOFACT,
        etree is an input argument, otherwise it is an output argument.
        Note: etree is a vector of parent pointers for a forest whose
        vertices are the integers 0 to A->ncol-1; etree[root]==A->ncol.

equed   (input/output) char*
        Specifies the form of equilibration that was done.
        = 'N': No equilibration.
```

```
        = 'R': Row equilibration, i.e., A was premultiplied by diag(R).
        = 'C': Column equilibration, i.e., A was postmultiplied by diag(C).
        = 'B': Both row and column equilibration, i.e., A was replaced
               by diag(R)*A*diag(C).
        If options->Fact = FACTORED, equed is an input argument,
        otherwise it is an output argument.

R       (input/output) float*, dimension (A->nrow)
        The row scale factors for A or transpose(A).
        If equed = 'R' or 'B', A (if A->Stype = SLU_NC) or transpose(A)
            (if A->Stype = SLU_NR) is multiplied on the left by diag(R).
        If equed = 'N' or 'C', R is not accessed.
        If options->Fact = FACTORED, R is an input argument,
            otherwise, R is output.
        If options->zFact = FACTORED and equed = 'R' or 'B', each element
            of R must be positive.

C       (input/output) float*, dimension (A->ncol)
        The column scale factors for A or transpose(A).
        If equed = 'C' or 'B', A (if A->Stype = SLU_NC) or transpose(A)
            (if A->Stype = SLU_NR) is multiplied on the right by diag(C).
        If equed = 'N' or 'R', C is not accessed.
        If options->Fact = FACTORED, C is an input argument,
            otherwise, C is output.
        If options->Fact = FACTORED and equed = 'C' or 'B', each element
            of C must be positive.

L       (output) SuperMatrix*
  The factor L from the factorization
            Pr*A*Pc=L*U              (if A->Stype SLU_= NC) or
            Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
        Uses compressed row subscripts storage for supernodes, i.e.,
        L has types: Stype = SLU_SC, Dtype = SLU_S, Mtype = SLU_TRLU.

U       (output) SuperMatrix*
  The factor U from the factorization
            Pr*A*Pc=L*U              (if A->Stype = SLU_NC) or
            Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
        Uses column-wise storage scheme, i.e., U has types:
        Stype = SLU_NC, Dtype = SLU_S, Mtype = SLU_TRU.

work    (workspace/output) void*, size (lwork) (in bytes)
        User supplied workspace, should be large enough
        to hold data structures for factors L and U.
        On exit, if fact is not 'F', L and U point to this array.

lwork   (input) int
        Specifies the size of work array in bytes.
        = 0:  allocate space internally by system malloc;
        > 0:  use user-supplied work array of length lwork in bytes,
              returns error if space runs out.
        = -1: the routine guesses the amount of space needed without
              performing the factorization, and returns it in
              mem_usage->total_needed; no other side effects.

        See argument 'mem_usage' for memory usage statistics.
```

```
B       (input/output) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_S, Mtype = SLU_GE.
        On entry, the right hand side matrix.
        If B->ncol = 0, only LU decomposition is performed, the triangular
                        solve is skipped.
        On exit,
           if equed = 'N', B is not modified; otherwise
           if A->Stype = SLU_NC:
              if options->Trans = NOTRANS and equed = 'R' or 'B',
                 B is overwritten by diag(R)*B;
              if options->Trans = TRANS or CONJ and equed = 'C' of 'B',
                 B is overwritten by diag(C)*B;
           if A->Stype = SLU_NR:
              if options->Trans = NOTRANS and equed = 'C' or 'B',
                 B is overwritten by diag(C)*B;
              if options->Trans = TRANS or CONJ and equed = 'R' of 'B',
                 B is overwritten by diag(R)*B.

X       (output) SuperMatrix*
        X has types: Stype = SLU_DN, Dtype = SLU_S, Mtype = SLU_GE.
        If info = 0 or info = A->ncol+1, X contains the solution matrix
        to the original system of equations. Note that A and B are modified
        on exit if equed is not 'N', and the solution to the equilibrated
        system is inv(diag(C))*X if options->Trans = NOTRANS and
        equed = 'C' or 'B', or inv(diag(R))*X if options->Trans = 'T' or 'C'
        and equed = 'R' or 'B'.

recip_pivot_growth (output) float*
        The reciprocal pivot growth factor max_j( norm(A_j)/norm(U_j) ).
        The infinity norm is used. If recip_pivot_growth is much less
        than 1, the stability of the LU factorization could be poor.

rcond   (output) float*
        The estimate of the reciprocal condition number of the matrix A
        after equilibration (if done). If rcond is less than the machine
        precision (in particular, if rcond = 0), the matrix is singular
        to working precision. This condition is indicated by a return
        code of info > 0.

FERR    (output) float*, dimension (B->ncol)
        The estimated forward error bound for each solution vector
        X(j) (the j-th column of the solution matrix X).
        If XTRUE is the true solution corresponding to X(j), FERR(j)
        is an estimated upper bound for the magnitude of the largest
        element in (X(j) - XTRUE) divided by the magnitude of the
        largest element in X(j).  The estimate is as reliable as
        the estimate for RCOND, and is almost always a slight
        overestimate of the true error.
        If options->IterRefine = NOREFINE, ferr = 1.0.

BERR    (output) float*, dimension (B->ncol)
        The componentwise relative backward error of each solution
        vector X(j) (i.e., the smallest relative change in
        any element of A or B that makes X(j) an exact solution).
        If options->IterRefine = NOREFINE, berr = 1.0.

mem_usage (output) mem_usage_t*
```

```
Record the memory usage statistics, consisting of following fields:
```

- for_lu (float)

```
The amount of space used in bytes for L data structures.
```

- total_needed (float)

```
The amount of space needed in bytes to perform factorization.
```

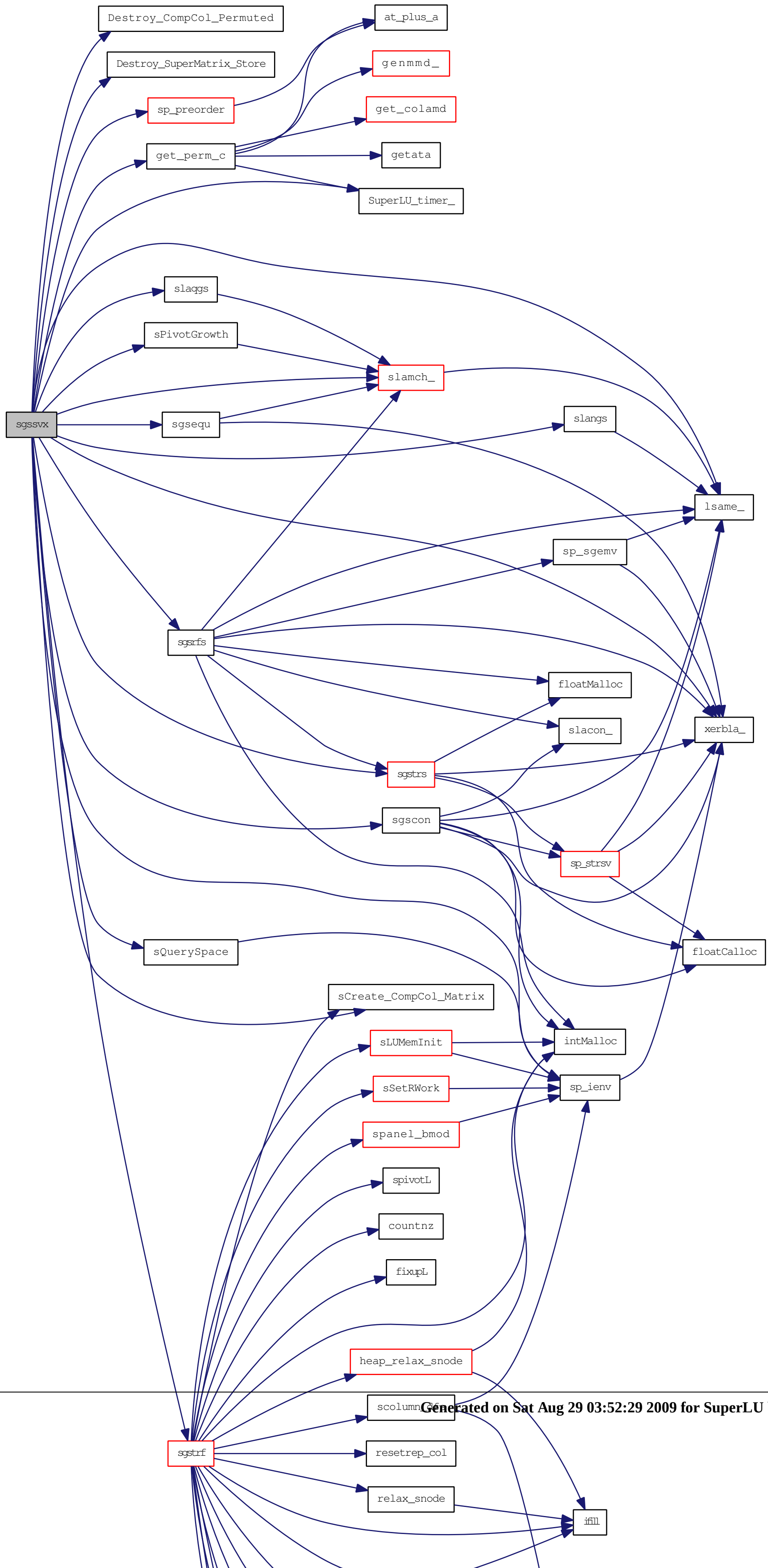
- expansions (int)

```
The number of memory expansions during the LU factorization.
```

```
stat    (output) SuperLUStat_t*
        Record the statistics on runtime and floating-point operation count.
        See util.h for the definition of 'SuperLUStat_t'.
```

```
info    (output) int*
        = 0: successful exit
        < 0: if info = -i, the i-th argument had an illegal value
        > 0: if info = i, and i is
             <= A->ncol: U(i,i) is exactly zero. The factorization has
                   been completed, but the factor U is exactly
                   singular, so the solution and error bounds
                   could not be computed.
             = A->ncol+1: U is nonsingular, but RCOND is less than machine
                   precision, meaning that the matrix is singular to
                   working precision. Nevertheless, the solution and
                   error bounds are computed because there are a number
                   of situations where the computed solution can be more
                   accurate than the value of RCOND would suggest.
             > A->ncol+1: number of bytes allocated when memory allocation
                   failure occurred, plus A->ncol.
```

Here is the call graph for this function:



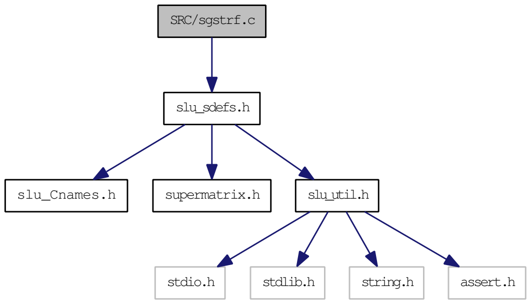
Here is the caller graph for this function:

4.104 SRC/sgstrf.c File Reference

Computes an LU factorization of a general sparse matrix.

```
#include "slu_sdefs.h"
```

Include dependency graph for sgstrf.c:



Functions

- void sgstrf (superlu_options_t *options, SuperMatrix *A, float drop_tol, int relax, int panel_size, int *etree, void *work, int lwork, int *perm_c, int *perm_r, SuperMatrix *L, SuperMatrix *U, SuperLUStat_t *stat, int *info)

4.104.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.104.2 Function Documentation

4.104.2.1 void sgstrf (superlu_options_t * *options*, SuperMatrix * *A*, float *drop_tol*, int *relax*, int *panel_size*, int * *etree*, void * *work*, int *lwork*, int * *perm_c*, int * *perm_r*, SuperMatrix * *L*, SuperMatrix * *U*, SuperLUStat_t * *stat*, int * *info*)

```
Purpose
=======
```

```
SGSTRF computes an LU factorization of a general sparse m-by-n
matrix A using partial pivoting with row interchanges.
The factorization has the form
    Pr * A = L * U
where Pr is a row permutation matrix, L is lower triangular with unit
diagonal elements (lower trapezoidal if A->nrow > A->ncol), and U is upper
triangular (upper trapezoidal if A->nrow < A->ncol).

See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

options (input) superlu_options_t*
        The structure defines the input parameters to control
        how the LU decomposition will be performed.

A        (input) SuperMatrix*
   Original matrix A, permuted by columns, of dimension
         (A->nrow, A->ncol). The type of A can be:
         Stype = SLU_NCP; Dtype = SLU_S; Mtype = SLU_GE.

drop_tol (input) float (NOT IMPLEMENTED)
   Drop tolerance parameter. At step j of the Gaussian elimination,
         if abs(A_ij)/(max_i abs(A_ij)) < drop_tol, drop entry A_ij.
         0 <= drop_tol <= 1. The default value of drop_tol is 0.

relax    (input) int
         To control degree of relaxing supernodes. If the number
         of nodes (columns) in a subtree of the elimination tree is less
         than relax, this subtree is considered as one supernode,
         regardless of the row structures of those columns.

panel_size (input) int
         A panel consists of at most panel_size consecutive columns.

etree    (input) int*, dimension (A->ncol)
         Elimination tree of A'*A.
         Note: etree is a vector of parent pointers for a forest whose
         vertices are the integers 0 to A->ncol-1; etree[root]==A->ncol.
         On input, the columns of A should be permuted so that the
         etree is in a certain postorder.

work     (input/output) void*, size (lwork) (in bytes)
         User-supplied work space and space for the output data structures.
         Not referenced if lwork = 0;

lwork   (input) int
        Specifies the size of work array in bytes.
        = 0:  allocate space internally by system malloc;
        > 0:  use user-supplied work array of length lwork in bytes,
              returns error if space runs out.
        = -1: the routine guesses the amount of space needed without
              performing the factorization, and returns it in
              *info; no other side effects.
```

```
perm_c   (input) int*, dimension (A->ncol)
   Column permutation vector, which defines the
         permutation matrix Pc; perm_c[i] = j means column i of A is
         in position j in A*Pc.
         When searching for diagonal, perm_c[*] is applied to the
         row subscripts of A, so that diagonal threshold pivoting
         can find the diagonal of A, rather than that of A*Pc.


perm_r   (input/output) int*, dimension (A->nrow)
         Row permutation vector which defines the permutation matrix Pr,
         perm_r[i] = j means row i of A is in position j in Pr*A.
         If options->Fact = SamePattern_SameRowPerm, the pivoting routine
            will try to use the input perm_r, unless a certain threshold
            criterion is violated. In that case, perm_r is overwritten by
            a new permutation determined by partial pivoting or diagonal
            threshold pivoting.
         Otherwise, perm_r is output argument;


L        (output) SuperMatrix*
         The factor L from the factorization Pr*A=L*U; use compressed row
         subscripts storage for supernodes, i.e., L has type:
         Stype = SLU_SC, Dtype = SLU_S, Mtype = SLU_TRLU.


U        (output) SuperMatrix*
   The factor U from the factorization Pr*A*Pc=L*U. Use column-wise
         storage scheme, i.e., U has types: Stype = SLU_NC,
         Dtype = SLU_S, Mtype = SLU_TRU.


stat     (output) SuperLUStat_t*
         Record the statistics on runtime and floating-point operation count.
         See util.h for the definition of 'SuperLUStat_t'.


info     (output) int*
         = 0: successful exit
         < 0: if info = -i, the i-th argument had an illegal value
         > 0: if info = i, and i is
            <= A->ncol: U(i,i) is exactly zero. The factorization has
               been completed, but the factor U is exactly singular,
               and division by zero will occur if it is used to solve a
               system of equations.
            > A->ncol: number of bytes allocated when memory allocation
               failure occurred, plus A->ncol. If lwork = -1, it is
               the estimated amount of space needed, plus A->ncol.

======================================================================

Local Working Arrays:
======================
  m = number of rows in the matrix
  n = number of columns in the matrix


  xprune[0:n-1]: xprune[*] points to locations in subscript
vector lsub[*]. For column i, xprune[i] denotes the point where
structural pruning begins. I.e. only xlsub[i],..,xprune[i]-1 need
to be traversed for symbolic factorization.
```

```
   marker[0:3*m-1]: marker[i] = j means that node i has been
reached when working on column j.
Storage: relative to original row subscripts
NOTE: There are 3 of them: marker/marker1 are used for panel dfs,
      see spanel_dfs.c; marker2 is used for inner-factorization,
            see scolumn_dfs.c.
```

```
   parent[0:m-1]: parent vector used during dfs
      Storage: relative to new row subscripts
```

```
   xplore[0:m-1]: xplore[i] gives the location of the next (dfs)
unexplored neighbor of i in lsub[*]
```

```
   segrep[0:nseg-1]: contains the list of supernodal representatives
in topological order of the dfs. A supernode representative is the
last column of a supernode.
      The maximum size of segrep[] is n.
```

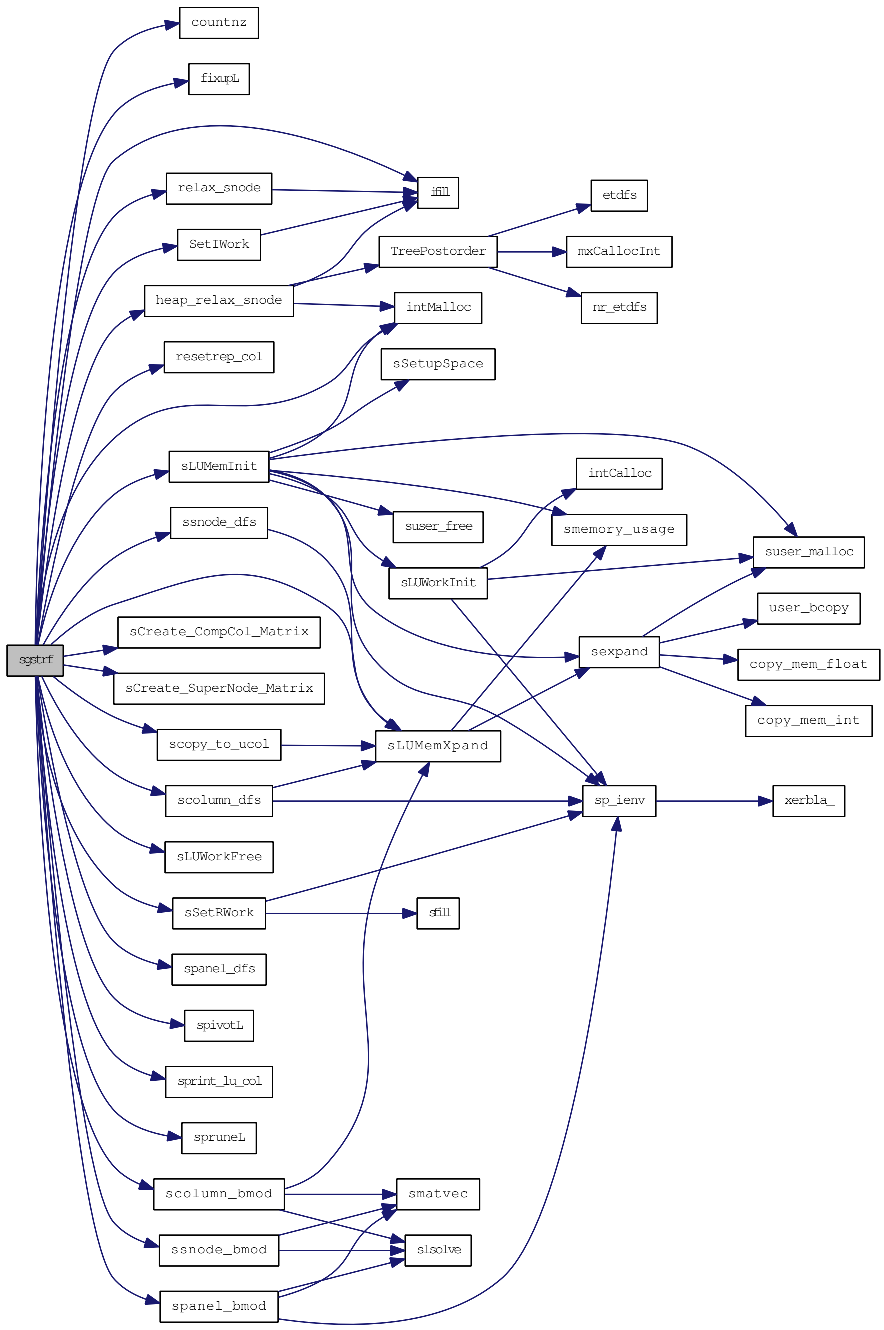
```
   repfnz[0:W*m-1]: for a nonzero segment U[*,j] that ends at a
supernodal representative r, repfnz[r] is the location of the first
nonzero in this segment.  It is also used during the dfs: repfnz[r]>0
indicates the supernode r has been explored.
NOTE: There are W of them, each used for one column of a panel.
```

```
   panel_lsub[0:W*m-1]: temporary for the nonzeros row indices below
      the panel diagonal. These are filled in during spanel_dfs(), and are
      used later in the inner LU factorization within the panel.
panel_lsub[]/dense[] pair forms the SPA data structure.
NOTE: There are W of them.
```

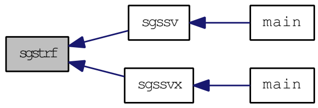
```
   dense[0:W*m-1]: sparse accumulating (SPA) vector for intermediate values;
        NOTE: there are W of them.
```

```
   tempv[0:*]: real temporary used for dense numeric kernels;
The size of this array is defined by NUM_TEMPV() in slu_sdefs.h.
```

Here is the call graph for this function:



Here is the caller graph for this function:

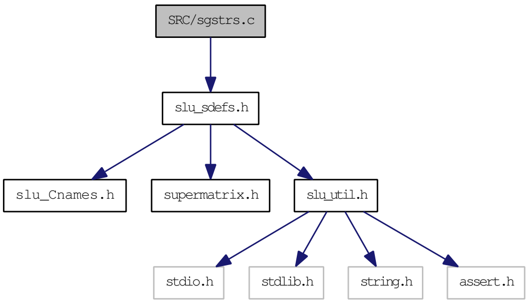


4.105 SRC/sgstrs.c File Reference

Solves a system using LU factorization.

`#include "slu_sdefs.h"`

Include dependency graph for sgstrs.c:



Functions

- void susolve (int, int, float ∗, float ∗)

  Solves a dense upper triangular system.

- void slsolve (int, int, float ∗, float ∗)

  Solves a dense UNIT lower triangular system.

- void smatvec (int, int, int, float ∗, float ∗, float ∗)

  Performs a dense matrix-vector multiply: Mxvec = Mxvec + M ∗ vec.

- void sgstrs (trans_t trans, SuperMatrix ∗L, SuperMatrix ∗U, int ∗perm_c, int ∗perm_r, SuperMatrix ∗B, SuperLUStat_t ∗stat, int ∗info)
- void sprint_soln (int n, int nrhs, float ∗soln)

4.105.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.105.2 Function Documentation

4.105.2.1 void sgstrs (trans_t *trans*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, int ∗ *perm_c*, int ∗ *perm_r*, SuperMatrix ∗ *B*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======

SGSTRS solves a system of linear equations A*X=B or A'*X=B
with A sparse and B dense, using the LU factorization computed by
SGSTRF.

See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

trans   (input) trans_t
         Specifies the form of the system of equations:
         = NOTRANS: A * X = B  (No transpose)
         = TRANS:   A'* X = B  (Transpose)
         = CONJ:    A**H * X = B  (Conjugate transpose)

L       (input) SuperMatrix*
        The factor L from the factorization Pr*A*Pc=L*U as computed by
        sgstrf(). Use compressed row subscripts storage for supernodes,
        i.e., L has types: Stype = SLU_SC, Dtype = SLU_S, Mtype = SLU_TRLU.

U       (input) SuperMatrix*
        The factor U from the factorization Pr*A*Pc=L*U as computed by
        sgstrf(). Use column-wise storage scheme, i.e., U has types:
        Stype = SLU_NC, Dtype = SLU_S, Mtype = SLU_TRU.

perm_c  (input) int*, dimension (L->ncol)
  Column permutation vector, which defines the
        permutation matrix Pc; perm_c[i] = j means column i of A is
        in position j in A*Pc.

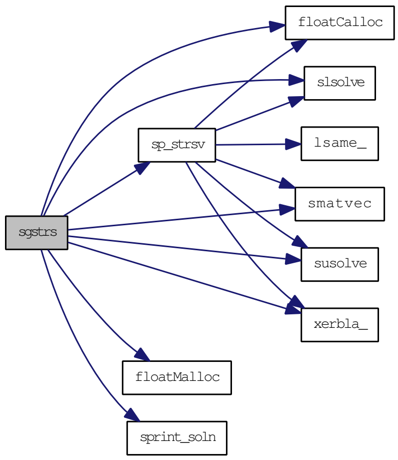
perm_r  (input) int*, dimension (L->nrow)
        Row permutation vector, which defines the permutation matrix Pr;
        perm_r[i] = j means row i of A is in position j in Pr*A.

B       (input/output) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_S, Mtype = SLU_GE.
        On entry, the right hand side matrix.
        On exit, the solution matrix if info = 0;

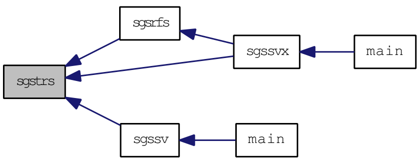
stat     (output) SuperLUStat_t*
         Record the statistics on runtime and floating-point operation count.
         See util.h for the definition of 'SuperLUStat_t'.

info    (output) int*
    = 0: successful exit
  < 0: if info = -i, the i-th argument had an illegal value
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.105.2.2 void slsolve (int *ldm*, int *ncol*, float ∗ *M*, float ∗ *rhs*)

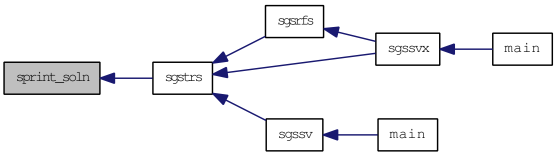
The unit lower triangular matrix is stored in a 2D array M(1:nrow,1:ncol). The solution will be returned in the rhs vector.

4.105.2.3 void smatvec (int *ldm*, int *nrow*, int *ncol*, float ∗ *M*, float ∗ *vec*, float ∗ *Mxvec*)

The input matrix is M(1:nrow,1:ncol); The product is returned in Mxvec[].

4.105.2.4 void sprint_soln (int *n*, int *nrhs*, float ∗ *soln*)

Here is the caller graph for this function:



4.105.2.5 void susolve (int *ldm*, int *ncol*, float ∗ *M*, float ∗ *rhs*)

The upper triangular matrix is stored in a 2-dim array M(1:ldm,1:ncol). The solution will be returned in the rhs vector.

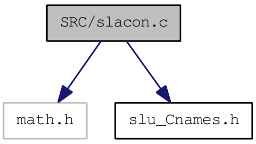
4.106 SRC/slacon.c File Reference

Estimates the 1-norm.

`#include <math.h>`

`#include "slu_Cnames.h"`

Include dependency graph for slacon.c:



Defines

- #define d_sign(a, b) (b >= 0 ? fabs(a) : -fabs(a))
- #define i_dnnt(a) (a>=0 ? floor(a+.5) : -floor(.5-a))

Functions

- int slacon_ (int ∗n, float ∗v, float ∗x, int ∗isgn, float ∗est, int ∗kase)

4.106.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997
```

4.106.2 Define Documentation

4.106.2.1 #define d_sign(a, b) (b >= 0 ? fabs(a) : -fabs(a))

4.106.2.2 #define i_dnnt(a) (a>=0 ? floor(a+.5) : -floor(.5-a))

4.106.3 Function Documentation

4.106.3.1 int slacon_ (int ∗ *n*, float ∗ *v*, float ∗ *x*, int ∗ *isgn*, float ∗ *est*, int ∗ *kase*)

```
Purpose
=======

SLACON estimates the 1-norm of a square matrix A.
Reverse communication is used for evaluating matrix-vector products.

Arguments
=========
```

```
N       (input) INT
        The order of the matrix.  N >= 1.

V       (workspace) FLOAT PRECISION array, dimension (N)
        On the final return, V = A*W,  where  EST = norm(V)/norm(W)
        (W is not returned).

X       (input/output) FLOAT PRECISION array, dimension (N)
        On an intermediate return, X should be overwritten by
              A * X,   if KASE=1,
              A' * X,  if KASE=2,
       and SLACON must be re-called with all the other parameters
        unchanged.

ISGN    (workspace) INT array, dimension (N)

EST     (output) FLOAT PRECISION
        An estimate (a lower bound) for norm(A).

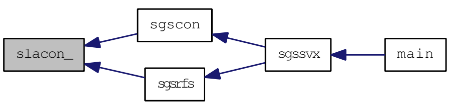
KASE    (input/output) INT
        On the initial call to SLACON, KASE should be 0.
        On an intermediate return, KASE will be 1 or 2, indicating
        whether X should be overwritten by A * X  or A' * X.
        On the final return from SLACON, KASE will again be 0.

Further Details
======= =======

Contributed by Nick Higham, University of Manchester.
Originally named CONEST, dated March 16, 1988.

Reference: N.J. Higham, "FORTRAN codes for estimating the one-norm of
a real or complex matrix, with applications to condition estimation",
ACM Trans. Math. Soft., vol. 14, no. 4, pp. 381-396, December 1988.
=====================================================================
```

Here is the caller graph for this function:



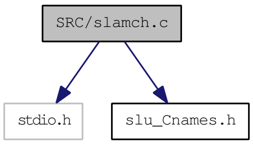
4.107 SRC/slamch.c File Reference

Determines single precision machine parameters and other service routines.

`#include <stdio.h>`

`#include "slu_Cnames.h"`

Include dependency graph for slamch.c:



Defines

- #define TRUE_ (1)
- #define FALSE_ (0)
- #define min(a, b) ((a) <= (b) ? (a) : (b))
- #define max(a, b) ((a) >= (b) ? (a) : (b))
- #define abs(x) ((x) >= 0 ? (x) : -(x))
- #define dabs(x) (double)abs(x)

Functions

- double slamch_ (char ∗cmach)
- int slamc1_ (int ∗beta, int ∗t, int ∗rnd, int ∗ieee1)
- int slamc2_ (int ∗beta, int ∗t, int ∗rnd, float ∗eps, int ∗emin, float ∗rmin, int ∗emax, float ∗rmax)
- double slamc3_ (float ∗a, float ∗b)
- int slamc4_ (int ∗emin, float ∗start, int ∗base)
- int slamc5_ (int ∗beta, int ∗p, int ∗emin, int ∗ieee, int ∗emax, float ∗rmax)
- double pow_ri (float ∗ap, int ∗bp)

4.107.1 Detailed Description

```
-- LAPACK auxiliary routine (version 2.0) --
   Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
   Courant Institute, Argonne National Lab, and Rice University
   October 31, 1992
```

4.107.2 Define Documentation

4.107.2.1 #define abs(x) ((x) >= 0 ? (x) : -(x))

4.107.2.2 #define dabs(x) (double)abs(x)

4.107.2.3 #define FALSE_ (0)

4.107.2.4 #define max(a, b) ((a) >= (b) ? (a) : (b))

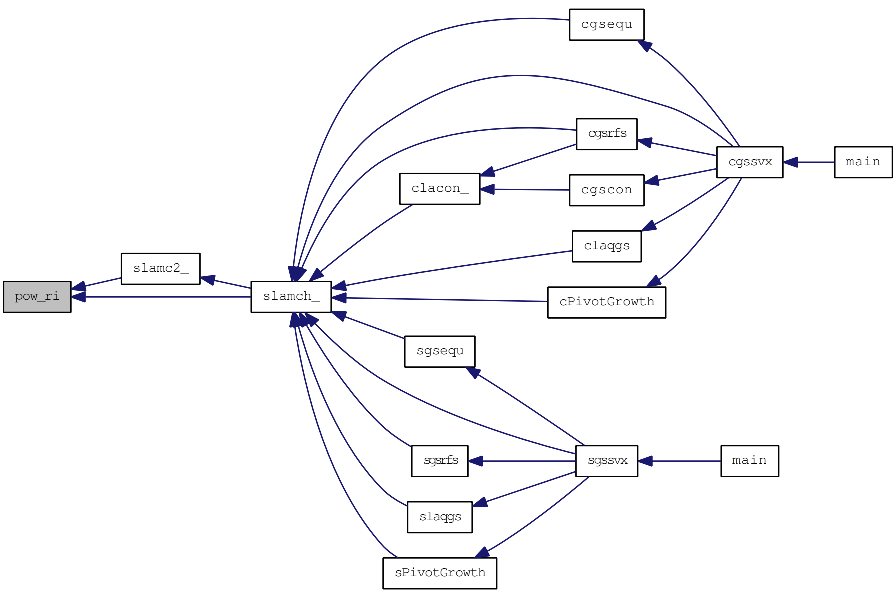
4.107.2.5 #define min(a, b) ((a) <= (b) ? (a) : (b))

4.107.2.6 #define TRUE_ (1)

4.107.3 Function Documentation

4.107.3.1 double pow_ri (float * *ap*, int * *bp*)

Here is the caller graph for this function:



4.107.3.2 int slamc1_ (int * *beta*, int * *t*, int * *rnd*, int * *ieee1*)

```
Purpose
   =======

   SLAMC1 determines the machine parameters given by BETA, T, RND, and
   IEEE1.

   Arguments
   =========
```

```
BETA    (output) INT
        The base of the machine.


T       (output) INT
        The number of ( BETA ) digits in the mantissa.


RND     (output) INT
        Specifies whether proper rounding  ( RND = .TRUE. )  or
        chopping  ( RND = .FALSE. )  occurs in addition. This may not


        be a reliable guide to the way in which the machine performs


        its arithmetic.


IEEE1   (output) INT
        Specifies whether rounding appears to be done in the IEEE
        'round to nearest' style.


Further Details
===============

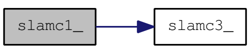

The routine is based on the routine  ENVRON  by Malcolm and
incorporates suggestions by Gentleman and Marovich. See


   Malcolm M. A. (1972) Algorithms to reveal properties of
      floating-point arithmetic. Comms. of the ACM, 15, 949-951.

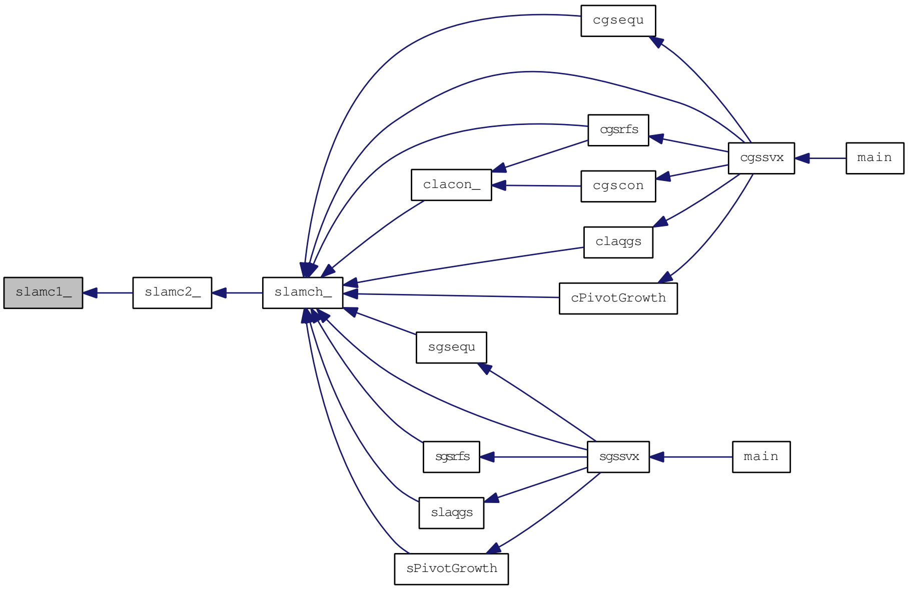

   Gentleman W. M. and Marovich S. B. (1974) More on algorithms
      that reveal properties of floating point arithmetic units.
      Comms. of the ACM, 17, 276-277.


=====================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.107.3.3 int slamc2_ (int * *beta*, int * *t*, int * *rnd*, float * *eps*, int * *emin*, float * *rmin*, int * *emax*, float * *rmax*)

```
Purpose
=======

SLAMC2 determines the machine parameters specified in its argument
list.

Arguments
=========

BETA     (output) INT
         The base of the machine.

T        (output) INT
         The number of ( BETA ) digits in the mantissa.

RND      (output) INT
         Specifies whether proper rounding  ( RND = .TRUE. )  or
         chopping  ( RND = .FALSE. )  occurs in addition. This may not

         be a reliable guide to the way in which the machine performs

         its arithmetic.

EPS      (output) FLOAT
         The smallest positive number such that
```

```
           fl( 1.0 - EPS ) .LT. 1.0,


        where fl denotes the computed value.


EMIN    (output) INT
        The minimum exponent before (gradual) underflow occurs.


RMIN    (output) FLOAT
        The smallest normalized number for the machine, given by
        BASE**( EMIN - 1 ), where  BASE  is the floating point value


        of BETA.


EMAX    (output) INT
        The maximum exponent before overflow occurs.


RMAX    (output) FLOAT
        The largest positive number for the machine, given by
        BASE**EMAX * ( 1 - EPS ), where  BASE  is the floating point

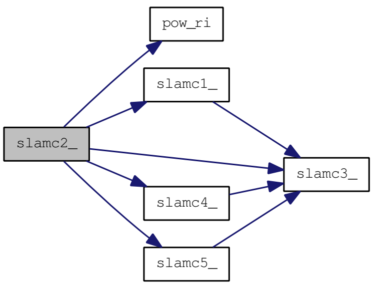

        value of BETA.


Further Details
===============

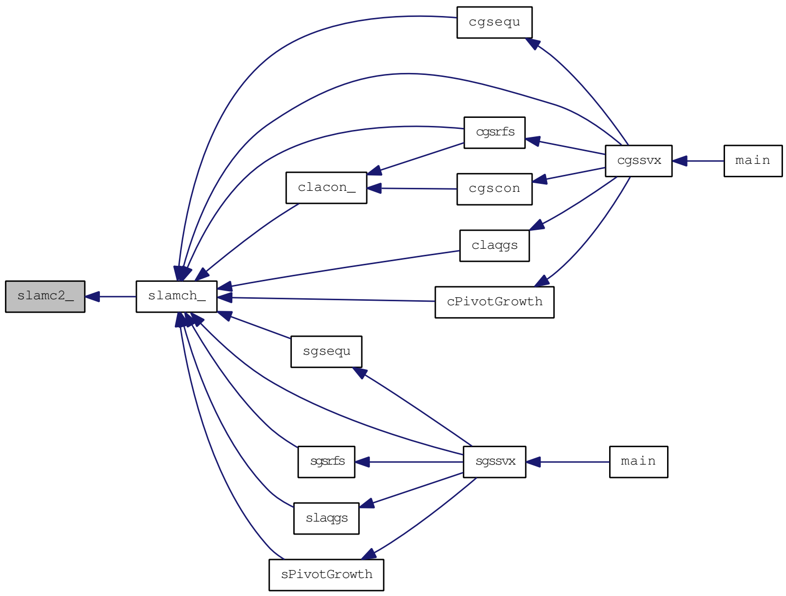

The computation of  EPS  is based on a routine PARANOIA by
W. Kahan of the University of California at Berkeley.


=====================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.107.3.4 double slamc3_ (float ∗ *a*, float ∗ *b*)

```
Purpose
=======


SLAMC3  is intended to force  A  and  B  to be stored prior to doing


the addition of  A  and  B ,  for use in situations where optimizers

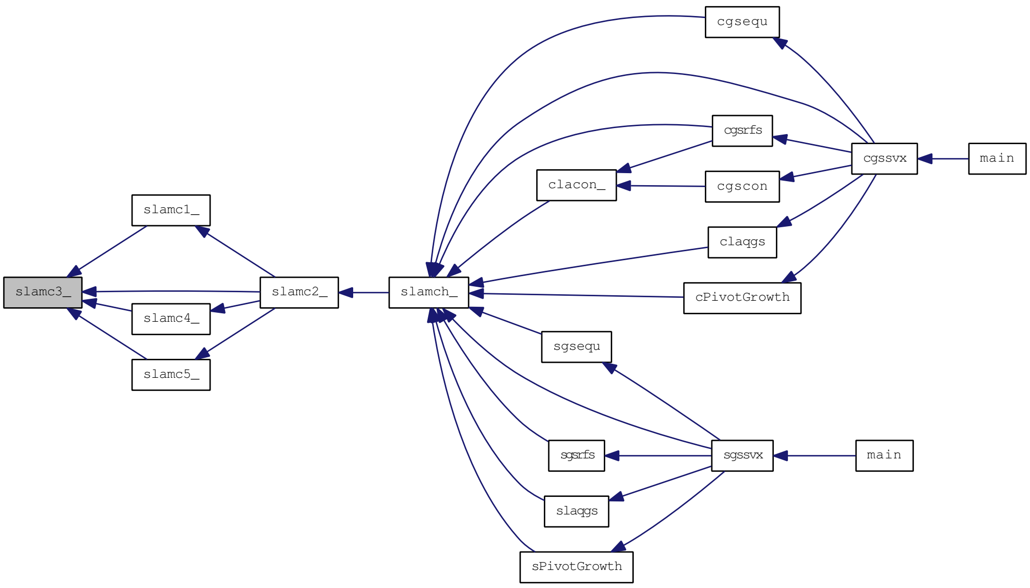

might hold one of these in a register.


Arguments
=========


A, B    (input) FLOAT
        The values A and B.


 =====================================================================
```

Here is the caller graph for this function:



4.107.3.5 int slamc4_ (int ∗ *emin*, float ∗ *start*, int ∗ *base*)

```
Purpose
=======

SLAMC4 is a service routine for SLAMC2.

Arguments
=========

EMIN    (output) EMIN
        The minimum exponent before (gradual) underflow, computed by

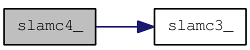
        setting A = START and dividing by BASE until the previous A
        can not be recovered.

START   (input) FLOAT
        The starting point for determining EMIN.

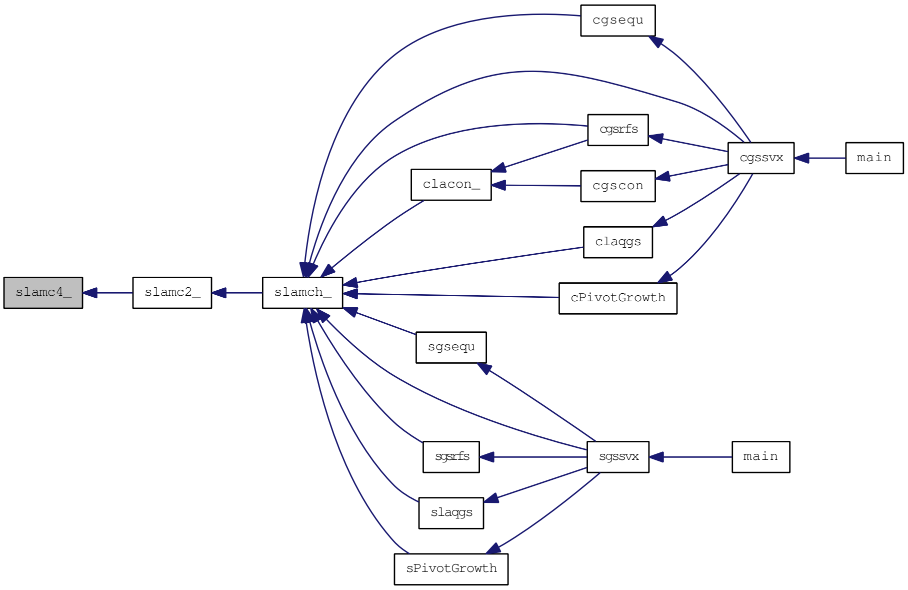
BASE    (input) INT
        The base of the machine.

 =====================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.107.3.6 int slamc5_ (int ∗ *beta*, int ∗ *p*, int ∗ *emin*, int ∗ *ieee*, int ∗ *emax*, float ∗ *rmax*)

```
Purpose
=======

SLAMC5 attempts to compute RMAX, the largest machine floating-point
number, without overflow.  It assumes that EMAX + abs(EMIN) sum
approximately to a power of 2.  It will fail on machines where this
assumption does not hold, for example, the Cyber 205 (EMIN = -28625,

EMAX = 28718).  It will also fail if the value supplied for EMIN is
too large (i.e. too close to zero), probably with overflow.

Arguments
=========

BETA    (input) INT
        The base of floating-point arithmetic.

P       (input) INT
        The number of base BETA digits in the mantissa of a
        floating-point value.

EMIN    (input) INT
        The minimum exponent before (gradual) underflow.

IEEE    (input) INT
        A logical flag specifying whether or not the arithmetic
        system is thought to comply with the IEEE standard.
```

```
    EMAX    (output) INT
            The largest exponent before overflow

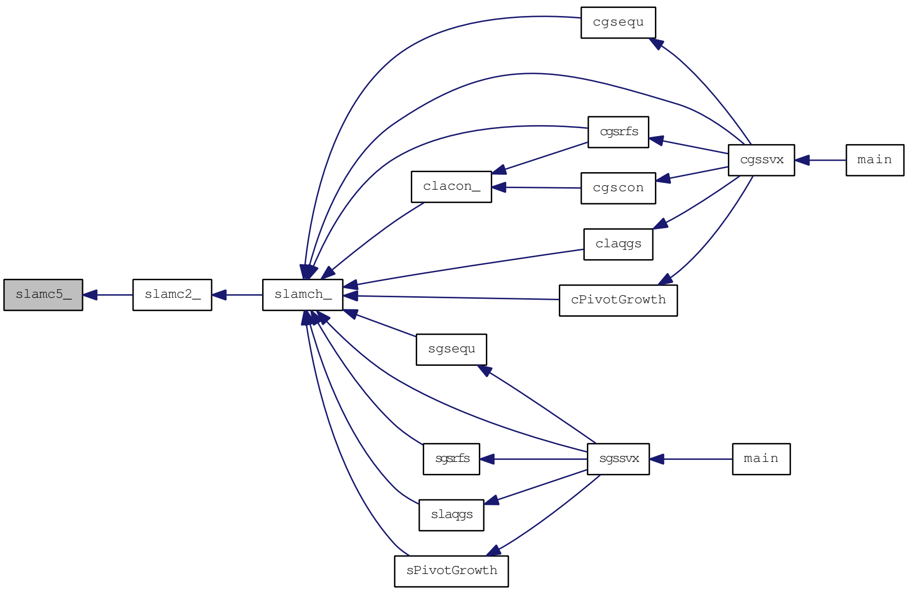
    RMAX    (output) FLOAT
            The largest machine floating-point number.

   =====================================================================

      First compute LEXP and UEXP, two powers of 2 that bound
      abs(EMIN). We then assume that EMAX + abs(EMIN) will sum
      approximately to the bound that is closest to abs(EMIN).
      (EMAX is the exponent of the required number RMAX).
```

Here is the call graph for this function:

Here is the caller graph for this function:



4.107.3.7 double slamch_ (char ∗ *cmach*)

```
 Purpose
    =======

    SLAMCH determines single precision machine parameters.

    Arguments
    =========
```

```
CMACH   (input) CHARACTER*1
        Specifies the value to be returned by SLAMCH:
        = 'E' or 'e',   SLAMCH := eps
        = 'S' or 's ,   SLAMCH := sfmin
        = 'B' or 'b',   SLAMCH := base
        = 'P' or 'p',   SLAMCH := eps*base
        = 'N' or 'n',   SLAMCH := t
        = 'R' or 'r',   SLAMCH := rnd
        = 'M' or 'm',   SLAMCH := emin
        = 'U' or 'u',   SLAMCH := rmin
        = 'L' or 'l',   SLAMCH := emax
        = 'O' or 'o',   SLAMCH := rmax



        where

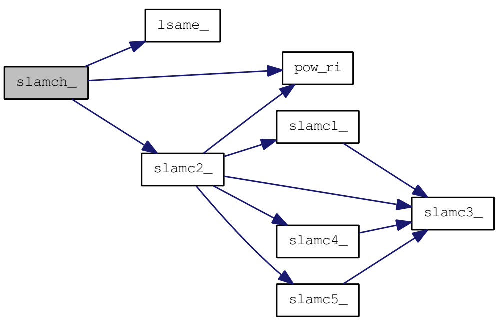


        eps   = relative machine precision
        sfmin = safe minimum, such that 1/sfmin does not overflow
        base  = base of the machine
        prec  = eps*base
        t     = number of (base) digits in the mantissa
        rnd   = 1.0 when rounding occurs in addition, 0.0 otherwise
        emin  = minimum exponent before (gradual) underflow
        rmin  = underflow threshold - base**(emin-1)
        emax  = largest exponent before overflow
        rmax  = overflow threshold  - (base**emax)*(1-eps)

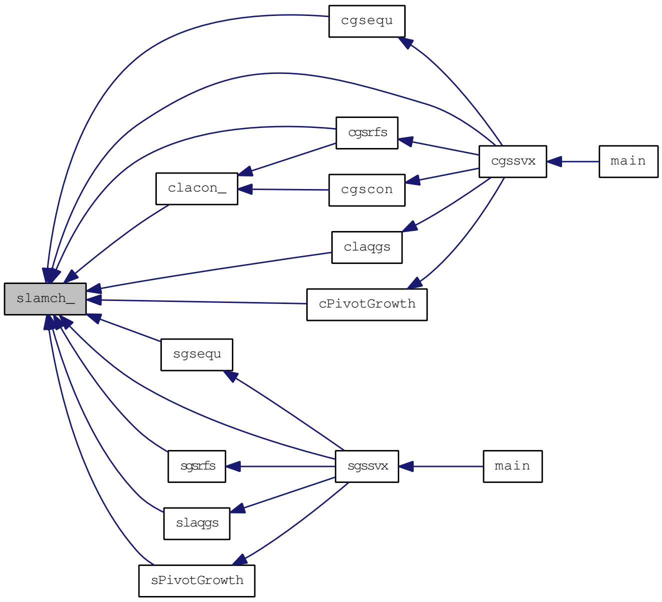


 =====================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:



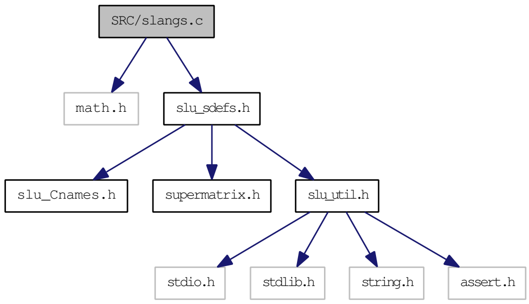
4.108 SRC/slangs.c File Reference

Returns the value of the one norm.

`#include <math.h>`

`#include "slu_sdefs.h"`

Include dependency graph for slangs.c:



Functions

- float slangs (char ∗norm, SuperMatrix ∗A)

4.108.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997


Modified from lapack routine SLANGE
```

4.108.2 Function Documentation

4.108.2.1 float slangs (char ∗ *norm*, SuperMatrix ∗ *A*)

```
Purpose
  =======


  SLANGS returns the value of the one norm, or the Frobenius norm, or
  the infinity norm, or the element of largest absolute value of a
  real matrix A.


  Description
  ===========


  SLANGE returns the value
```

```
   SLANGE = ( max(abs(A(i,j))), NORM = 'M' or 'm'
            (
            ( norm1(A),         NORM = '1', 'O' or 'o'
            (
            ( normI(A),         NORM = 'I' or 'i'
            (
            ( normF(A),         NORM = 'F', 'f', 'E' or 'e'

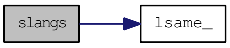
where  norm1  denotes the  one norm of a matrix (maximum column sum),
normI  denotes the  infinity norm  of a matrix  (maximum row sum) and
normF  denotes the  Frobenius norm of a matrix (square root of sum of
squares).  Note that  max(abs(A(i,j)))  is not a  matrix norm.

Arguments
=========

NORM    (input) CHARACTER*1
        Specifies the value to be returned in SLANGE as described above.
A       (input) SuperMatrix*
        The M by N sparse matrix A.

=====================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:

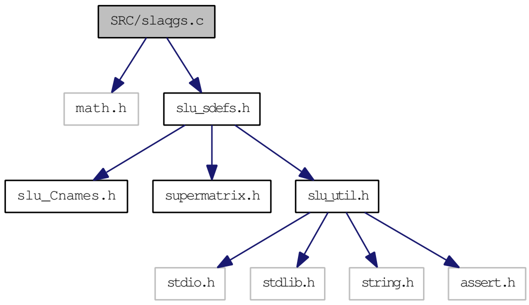
4.109 SRC/slaqgs.c File Reference

Equlibrates a general sprase matrix.

`#include <math.h>`

`#include "slu_sdefs.h"`

Include dependency graph for slaqgs.c:



Defines

- #define THRESH (0.1)

Functions

- void slaqgs (SuperMatrix ∗A, float ∗r, float ∗c, float rowcnd, float colcnd, float amax, char ∗equed)

4.109.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997

Modified from LAPACK routine SLAQGE
```

4.109.2 Define Documentation

4.109.2.1 #define THRESH (0.1)

4.109.3 Function Documentation

4.109.3.1 void slaqgs (SuperMatrix ∗ *A*, float ∗ *r*, float ∗ *c*, float *rowcnd*, float *colcnd*, float *amax*, char ∗ *equed*)

```
Purpose
=======
```

```
SLAQGS equilibrates a general sparse M by N matrix A using the row and
scaling factors in the vectors R and C.

See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

A       (input/output) SuperMatrix*
        On exit, the equilibrated matrix.  See EQUED for the form of
        the equilibrated matrix. The type of A can be:
 Stype = NC; Dtype = SLU_S; Mtype = GE.

R       (input) float*, dimension (A->nrow)
        The row scale factors for A.

C       (input) float*, dimension (A->ncol)
        The column scale factors for A.

ROWCND  (input) float
        Ratio of the smallest R(i) to the largest R(i).

COLCND  (input) float
        Ratio of the smallest C(i) to the largest C(i).

AMAX    (input) float
        Absolute value of largest matrix entry.

EQUED   (output) char*
        Specifies the form of equilibration that was done.
        = 'N':  No equilibration
        = 'R':  Row equilibration, i.e., A has been premultiplied by
                diag(R).
        = 'C':  Column equilibration, i.e., A has been postmultiplied
                by diag(C).
        = 'B':  Both row and column equilibration, i.e., A has been
                replaced by diag(R) * A * diag(C).

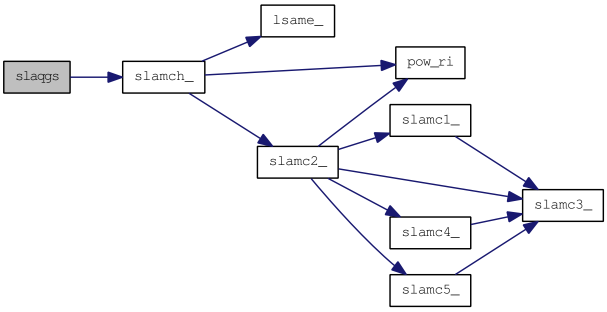
Internal Parameters
===================

THRESH is a threshold value used to decide if row or column scaling
should be done based on the ratio of the row or column scaling
factors.  If ROWCND < THRESH, row scaling is done, and if
COLCND < THRESH, column scaling is done.

LARGE and SMALL are threshold values used to decide if row scaling
should be done based on the absolute size of the largest matrix
element.  If AMAX > LARGE or AMAX < SMALL, row scaling is done.

=====================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.110 SRC/slu_cdefs.h File Reference

Header file for real operations.

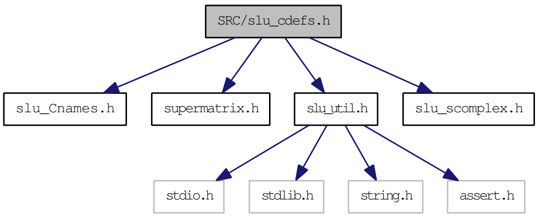
`#include "slu_Cnames.h"`

`#include "supermatrix.h"`

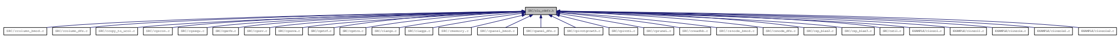
`#include "slu_util.h"`

`#include "slu_scomplex.h"`

Include dependency graph for slu_cdefs.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct GlobalLU_t

Typedefs

- typedef int int_t

Functions

- void cgssv (superlu_options_t ∗, SuperMatrix ∗, int ∗, int ∗, SuperMatrix ∗, SuperMatrix ∗, SuperMatrix ∗, SuperLUStat_t ∗, int ∗)

  Driver routines.

- void cgssvx (superlu_options_t ∗, SuperMatrix ∗, int ∗, int ∗, int ∗, char ∗, float ∗, float ∗, SuperMatrix ∗, SuperMatrix ∗, void ∗, int, SuperMatrix ∗, SuperMatrix ∗, float ∗, float ∗, float ∗, float ∗, mem_usage_t ∗, SuperLUStat_t ∗, int ∗)
- void cCreate_CompCol_Matrix (SuperMatrix ∗, int, int, int, complex ∗, int ∗, int ∗, Stype_t, Dtype_t, Mtype_t)

  Supernodal LU factor related.

- void cCreate_CompRow_Matrix (SuperMatrix ∗, int, int, int, complex ∗, int ∗, int ∗, Stype_t, Dtype_t, Mtype_t)
- void cCopy_CompCol_Matrix (SuperMatrix ∗, SuperMatrix ∗)

Copy matrix A into matrix B.

- void cCreate_Dense_Matrix (SuperMatrix ∗, int, int, complex ∗, int, Stype_t, Dtype_t, Mtype_t)
- void cCreate_SuperNode_Matrix (SuperMatrix ∗, int, int, int, complex ∗, int ∗, int ∗, int ∗, int ∗, int ∗, Stype_t, Dtype_t, Mtype_t)
- void cCopy_Dense_Matrix (int, int, complex ∗, int, complex ∗, int)
- void countnz (const int, int ∗, int ∗, int ∗, GlobalLU_t ∗)

  Count the total number of nonzeros in factors L and U, and in the symmetrically reduced L.

- void fixupL (const int, const int ∗, GlobalLU_t ∗)

  Fix up the data storage lsub for L-subscripts. It removes the subscript sets for structural pruning, and applies permuation to the remaining subscripts.

- void callocateA (int, int, complex ∗∗, int ∗∗, int ∗∗)

  Allocate storage for original matrix A.

- void cgstrf (superlu_options_t ∗, SuperMatrix ∗, float, int, int, int ∗, void ∗, int, int ∗, int ∗, SuperMatrix ∗, SuperMatrix ∗, SuperLUStat_t ∗, int ∗)
- int csnode_dfs (const int, const int, const int ∗, const int ∗, const int ∗, int ∗, int ∗, GlobalLU_t ∗)
- int csnode_bmod (const int, const int, const int, complex ∗, complex ∗, GlobalLU_t ∗, SuperLUStat_t ∗)

  Performs numeric block updates within the relaxed snode.

- void cpanel_dfs (const int, const int, const int, SuperMatrix ∗, int ∗, int ∗, complex ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, GlobalLU_t ∗)
- void cpanel_bmod (const int, const int, const int, const int, complex ∗, complex ∗, int ∗, int ∗, GlobalLU_t ∗, SuperLUStat_t ∗)
- int ccolumn_dfs (const int, const int, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, GlobalLU_t ∗)
- int ccolumn_bmod (const int, const int, complex ∗, complex ∗, int ∗, int ∗, int, GlobalLU_t ∗, SuperLUStat_t ∗)
- int ccopy_to_ucol (int, int, int ∗, int ∗, int ∗, complex ∗, GlobalLU_t ∗)
- int cpivotL (const int, const float, int ∗, int ∗, int ∗, int ∗, int ∗, GlobalLU_t ∗, SuperLUStat_t ∗)
- void cpruneL (const int, const int ∗, const int, const int, const int ∗, const int ∗, int ∗, GlobalLU_t ∗)
- void creadmt (int ∗, int ∗, int ∗, complex ∗∗, int ∗∗, int ∗∗)
- void cGenXtrue (int, int, complex ∗, int)
- void cFillRHS (trans_t, int, complex ∗, int, SuperMatrix ∗, SuperMatrix ∗)

  Let rhs[i] = sum of i-th row of A, so the solution vector is all 1's.

- void cgstrs (trans_t, SuperMatrix ∗, SuperMatrix ∗, int ∗, int ∗, SuperMatrix ∗, SuperLUStat_t ∗, int ∗)
- void cgsequ (SuperMatrix ∗, float ∗, float ∗, float ∗, float ∗, float ∗, int ∗)

  Driver related.

- void claqgs (SuperMatrix ∗, float ∗, float ∗, float, float, float, char ∗)
- void cgscon (char ∗, SuperMatrix ∗, SuperMatrix ∗, float, float ∗, SuperLUStat_t ∗, int ∗)
- float cPivotGrowth (int, SuperMatrix ∗, int ∗, SuperMatrix ∗, SuperMatrix ∗)
- void cgsrfs (trans_t, SuperMatrix ∗, SuperMatrix ∗, SuperMatrix ∗, int ∗, int ∗, char ∗, float ∗, float ∗, SuperMatrix ∗, SuperMatrix ∗, float ∗, float ∗, SuperLUStat_t ∗, int ∗)
- int sp_ctrsv (char ∗, char ∗, char ∗, SuperMatrix ∗, SuperMatrix ∗, complex ∗, SuperLUStat_t ∗, int ∗)

Solves one of the systems of equations A∗x = b, or A'∗x = b.

- int sp_cgemv (char ∗, complex, SuperMatrix ∗, complex ∗, int, complex, complex ∗, int)

  Performs one of the matrix-vector operations y := alpha∗A∗x + beta∗y, or y := alpha∗A'∗x + beta∗y.

- int sp_cgemm (char ∗, char ∗, int, int, int, complex, SuperMatrix ∗, complex ∗, int, complex, complex ∗, int)
- int cLUMemInit (fact_t, void ∗, int, int, int, int, int, SuperMatrix ∗, SuperMatrix ∗, GlobalLU_t ∗, int ∗∗, complex ∗∗)

  Memory-related.

- void cSetRWork (int, int, complex ∗, complex ∗∗, complex ∗∗)

  Set up pointers for real working arrays.

- void cLUWorkFree (int ∗, complex ∗, GlobalLU_t ∗)

  Free the working storage used by factor routines.

- int cLUMemXpand (int, int, MemType, int ∗, GlobalLU_t ∗)

  Expand the data structures for L and U during the factorization.

- complex ∗ complexMalloc (int)
- complex ∗ complexCalloc (int)
- float ∗ floatMalloc (int)
- float ∗ floatCalloc (int)
- int cmemory_usage (const int, const int, const int, const int)
- int cQuerySpace (SuperMatrix ∗, SuperMatrix ∗, mem_usage_t ∗)
- void creadhb (int ∗, int ∗, int ∗, complex ∗∗, int ∗∗, int ∗∗)

  Auxiliary routines.

- void cCompRow_to_CompCol (int, int, int, complex ∗, int ∗, int ∗, complex ∗∗, int ∗∗, int ∗∗)

  Convert a row compressed storage into a column compressed storage.

- void cfill (complex ∗, int, complex)

  Fills a complex precision array with a given value.

- void cinf_norm_error (int, SuperMatrix ∗, complex ∗)

  Check the inf-norm of the error vector.

- void PrintPerf (SuperMatrix ∗, SuperMatrix ∗, mem_usage_t ∗, complex, complex, complex ∗, complex ∗, char ∗)
- void cPrint_CompCol_Matrix (char ∗, SuperMatrix ∗)

  Routines for debugging.

- void cPrint_SuperNode_Matrix (char ∗, SuperMatrix ∗)
- void cPrint_Dense_Matrix (char ∗, SuperMatrix ∗)
- void print_lu_col (char ∗, int, int, int ∗, GlobalLU_t ∗)
- void check_tempv (int, complex ∗)

4.110.1 Detailed Description

```
 -- SuperLU routine (version 3.0) --
 Univ. of California Berkeley, Xerox Palo Alto Research Center,
 and Lawrence Berkeley National Lab.
 October 15, 2003

 Global data structures used in LU factorization -

   nsuper: supernodes = nsuper + 1, numbered [0, nsuper].
   (xsup,supno): supno[i] is the supernode no to which i belongs;
xsup(s) points to the beginning of the s-th supernode.
e.g.   supno 0 1 2 2 3 3 3 4 4 4 4 4   (n=12)
        xsup 0 1 2 4 7 12
Note: dfs will be performed on supernode rep. relative to the new
      row pivoting ordering

   (xlsub,lsub): lsub[*] contains the compressed subscript of
rectangular supernodes; xlsub[j] points to the starting
location of the j-th column in lsub[*]. Note that xlsub
is indexed by column.
Storage: original row subscripts

      During the course of sparse LU factorization, we also use
(xlsub,lsub) for the purpose of symmetric pruning. For each
supernode {s,s+1,...,t=s+r} with first column s and last
column t, the subscript set
lsub[j], j=xlsub[s], .., xlsub[s+1]-1
is the structure of column s (i.e. structure of this supernode).
It is used for the storage of numerical values.
Furthermore,
lsub[j], j=xlsub[t], .., xlsub[t+1]-1
is the structure of the last column t of this supernode.
It is for the purpose of symmetric pruning. Therefore, the
structural subscripts can be rearranged without making physical
interchanges among the numerical values.

However, if the supernode has only one column, then we
only keep one set of subscripts. For any subscript interchange
performed, similar interchange must be done on the numerical
values.

The last column structures (for pruning) will be removed
after the numercial LU factorization phase.

   (xlusup,lusup): lusup[*] contains the numerical values of the
rectangular supernodes; xlusup[j] points to the starting
location of the j-th column in storage vector lusup[*]
Note: xlusup is indexed by column.
Each rectangular supernode is stored by column-major
scheme, consistent with Fortran 2-dim array storage.

   (xusub,ucol,usub): ucol[*] stores the numerical values of
U-columns outside the rectangular supernodes. The row
subscript of nonzero ucol[k] is stored in usub[k].
xusub[i] points to the starting location of column i in ucol.
Storage: new row subscripts; that is subscripts of PA.
```

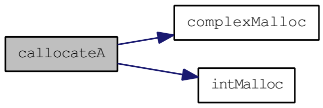
4.110.2 Typedef Documentation

4.110.2.1 typedef int int_t

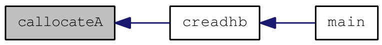
4.110.3 Function Documentation

4.110.3.1 void callocateA (int, int, complex ∗∗, int ∗∗, int ∗∗)

Here is the call graph for this function:



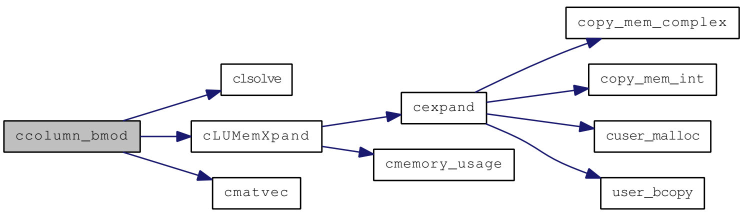
Here is the caller graph for this function:



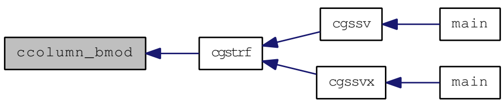
4.110.3.2 int ccolumn_bmod (const int *jcol*, const int *nseg*, complex ∗ *dense*, complex ∗ *tempv*, int ∗ *segrep*, int ∗ *repfnz*, int *fpanelc*, GlobalLU_t ∗ *Glu*, SuperLUStat_t ∗ *stat*)

```
Purpose:
========
Performs numeric block updates (sup-col) in topological order.
It features: col-col, 2cols-col, 3cols-col, and sup-col updates.
Special processing on the supernodal portion of L[*,j]
Return value:   0 - successful return
              > 0 - number of bytes allocated when run out of space
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.110.3.3 int ccolumn_dfs (const int *m*, const int *jcol*, int ∗ *perm_r*, int ∗ *nseg*, int ∗ *lsub_col*, int ∗ *segrep*, int ∗ *repfnz*, int ∗ *xprune*, int ∗ *marker*, int ∗ *parent*, int ∗ *xplore*, GlobalLU_t ∗ *Glu*)

```
 Purpose
 =======
   "column_dfs" performs a symbolic factorization on column jcol, and
   decide the supernode boundary.

   This routine does not use numeric values, but only use the RHS
   row indices to start the dfs.

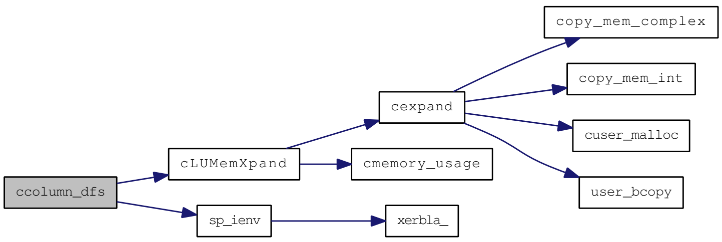
   A supernode representative is the last column of a supernode.
   The nonzeros in U[*,j] are segments that end at supernodal
   representatives. The routine returns a list of such supernodal
   representatives in topological order of the dfs that generates them.
   The location of the first nonzero in each such supernodal segment
   (supernodal entry location) is also returned.

 Local parameters
 ================
   nseg: no of segments in current U[*,j]
   jsuper: jsuper=EMPTY if column j does not belong to the same
supernode as j-1. Otherwise, jsuper=nsuper.

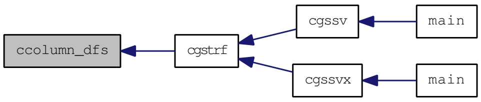
   marker2: A-row --> A-row/col (0/1)
   repfnz: SuperA-col --> PA-row
   parent: SuperA-col --> SuperA-col
   xplore: SuperA-col --> index to L-structure

 Return value
 ============
     0  success;
   > 0  number of bytes allocated when run out of space.
```

Here is the call graph for this function:

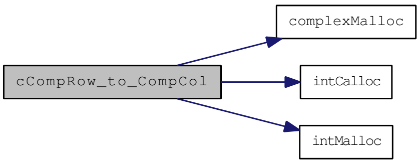


Here is the caller graph for this function:



4.110.3.4 void cCompRow_to_CompCol (int, int, int, complex ∗, int ∗, int ∗, complex ∗∗, int ∗∗, int ∗∗)

Here is the call graph for this function:



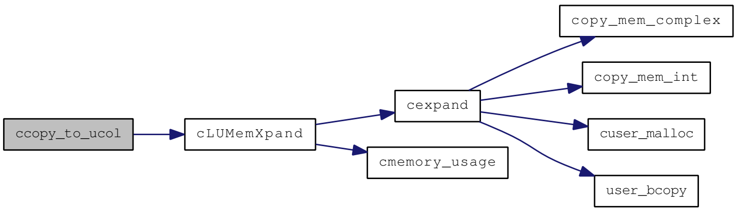
4.110.3.5 void cCopy_CompCol_Matrix (SuperMatrix ∗, SuperMatrix ∗)

4.110.3.6 void cCopy_Dense_Matrix (int, int, complex ∗, int, complex ∗, int)

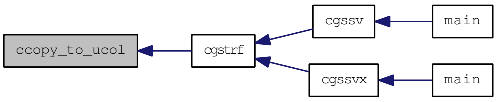
Copies a two-dimensional matrix X to another matrix Y.

4.110.3.7 int ccopy_to_ucol (int, int, int ∗, int ∗, int ∗, complex ∗, GlobalLU_t ∗)

Here is the call graph for this function:

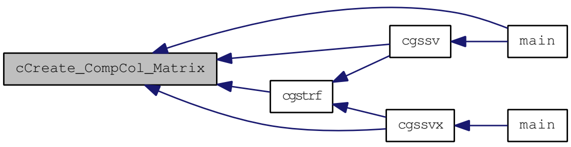


Here is the caller graph for this function:



4.110.3.8 void cCreate_CompCol_Matrix (SuperMatrix ∗, int, int, int, complex ∗, int ∗, int ∗, Stype_t, Dtype_t, Mtype_t)

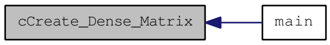
Here is the caller graph for this function:



4.110.3.9 void cCreate_CompRow_Matrix (SuperMatrix ∗, int, int, int, complex ∗, int ∗, int ∗, Stype_t, Dtype_t, Mtype_t)

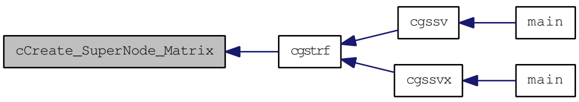
4.110.3.10 void cCreate_Dense_Matrix (SuperMatrix ∗, int, int, complex ∗, int, Stype_t, Dtype_t, Mtype_t)

Here is the caller graph for this function:



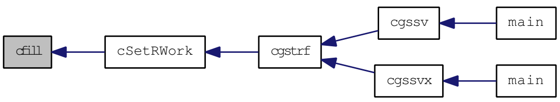
4.110.3.11 void cCreate_SuperNode_Matrix (SuperMatrix ∗, int, int, int, complex ∗, int ∗, int ∗, int ∗, int ∗, int ∗, Stype_t, Dtype_t, Mtype_t)

Here is the caller graph for this function:



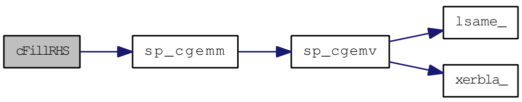
4.110.3.12 void cfill (complex ∗, int, complex)

Here is the caller graph for this function:

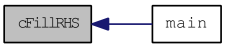


4.110.3.13 void cFillRHS (trans_t, int, complex ∗, int, SuperMatrix ∗, SuperMatrix ∗)

Here is the call graph for this function:

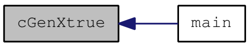


Here is the caller graph for this function:



4.110.3.14 void cGenXtrue (int, int, complex ∗, int)

Here is the caller graph for this function:



4.110.3.15 void cgscon (char ∗ *norm*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, float *anorm*, float ∗ *rcond*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======

CGSCON estimates the reciprocal of the condition number of a general
real matrix A, in either the 1-norm or the infinity-norm, using
the LU factorization computed by CGETRF.   *

An estimate is obtained for norm(inv(A)), and the reciprocal of the
condition number is computed as
   RCOND = 1 / ( norm(A) * norm(inv(A)) ).

See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

 NORM    (input) char*
         Specifies whether the 1-norm condition number or the
         infinity-norm condition number is required:
         = '1' or 'O':  1-norm;
         = 'I':         Infinity-norm.

 L       (input) SuperMatrix*
         The factor L from the factorization Pr*A*Pc=L*U as computed by
         cgstrf(). Use compressed row subscripts storage for supernodes,
         i.e., L has types: Stype = SLU_SC, Dtype = SLU_C, Mtype = SLU_TRLU.

 U       (input) SuperMatrix*
         The factor U from the factorization Pr*A*Pc=L*U as computed by
         cgstrf(). Use column-wise storage scheme, i.e., U has types:
         Stype = SLU_NC, Dtype = SLU_C, Mtype = SLU_TRU.

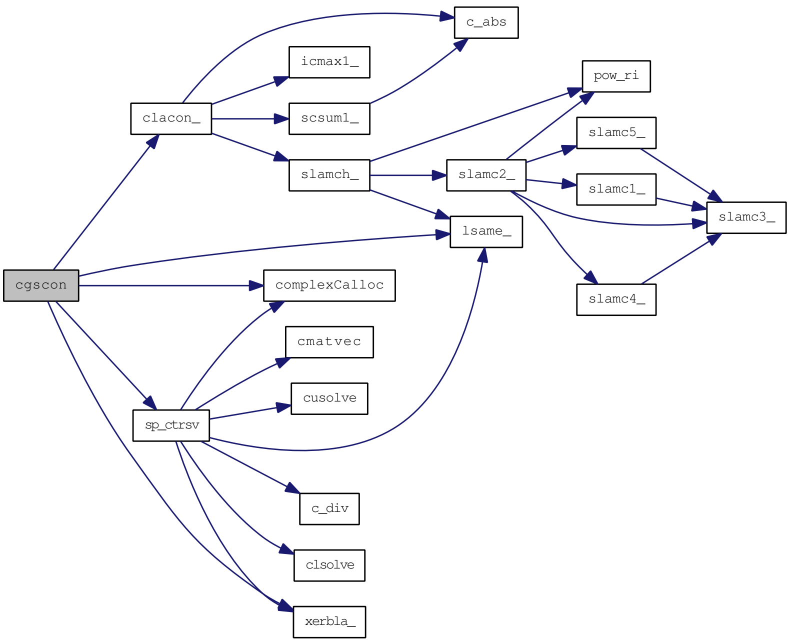
 ANORM   (input) float
         If NORM = '1' or 'O', the 1-norm of the original matrix A.
         If NORM = 'I', the infinity-norm of the original matrix A.

 RCOND   (output) float*
        The reciprocal of the condition number of the matrix A,
        computed as RCOND = 1/(norm(A) * norm(inv(A))).

 INFO    (output) int*
        = 0:  successful exit
        < 0:  if INFO = -i, the i-th argument had an illegal value
```

```
    ======================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.110.3.16 void cgsequ (SuperMatrix ∗ *A*, float ∗ *r*, float ∗ *c*, float ∗ *rowcnd*, float ∗ *colcnd*, float ∗ *amax*, int ∗ *info*)

```
 Purpose
   =======

   CGSEQU computes row and column scalings intended to equilibrate an
   M-by-N sparse matrix A and reduce its condition number. R returns the row
   scale factors and C the column scale factors, chosen to try to make
   the largest element in each row and column of the matrix B with
   elements B(i,j)=R(i)*A(i,j)*C(j) have absolute value 1.

   R(i) and C(j) are restricted to be between SMLNUM = smallest safe
   number and BIGNUM = largest safe number.  Use of these scaling
   factors is not guaranteed to reduce the condition number of A but
   works well in practice.

   See supermatrix.h for the definition of 'SuperMatrix' structure.
```

```
Arguments
=========

A       (input) SuperMatrix*
        The matrix of dimension (A->nrow, A->ncol) whose equilibration
        factors are to be computed. The type of A can be:
        Stype = SLU_NC; Dtype = SLU_C; Mtype = SLU_GE.

R       (output) float*, size A->nrow
        If INFO = 0 or INFO > M, R contains the row scale factors
        for A.

C       (output) float*, size A->ncol
        If INFO = 0,  C contains the column scale factors for A.

ROWCND  (output) float*
        If INFO = 0 or INFO > M, ROWCND contains the ratio of the
        smallest R(i) to the largest R(i).  If ROWCND >= 0.1 and
        AMAX is neither too large nor too small, it is not worth
        scaling by R.

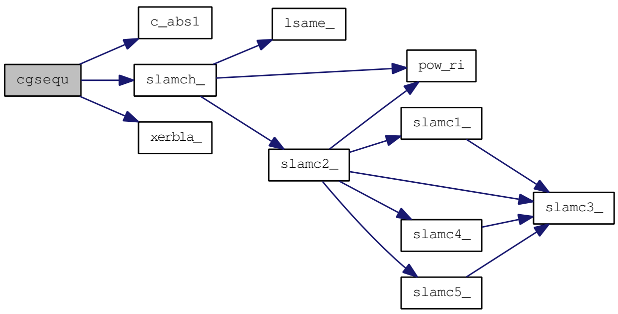
COLCND  (output) float*
        If INFO = 0, COLCND contains the ratio of the smallest
        C(i) to the largest C(i).  If COLCND >= 0.1, it is not
        worth scaling by C.

AMAX    (output) float*
        Absolute value of largest matrix element.  If AMAX is very
        close to overflow or very close to underflow, the matrix
        should be scaled.

INFO    (output) int*
        = 0:  successful exit
        < 0:  if INFO = -i, the i-th argument had an illegal value
        > 0:  if INFO = i,  and i is
              <= A->nrow:  the i-th row of A is exactly zero
              >  A->ncol:  the (i-M)-th column of A is exactly zero

=====================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.110.3.17 void cgsrfs (trans_t *trans*, SuperMatrix ∗ *A*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, int ∗ *perm_c*, int ∗ *perm_r*, char ∗ *equed*, float ∗ *R*, float ∗ *C*, SuperMatrix ∗ *B*, SuperMatrix ∗ *X*, float ∗ *ferr*, float ∗ *berr*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
  Purpose
  =======

  CGSRFS improves the computed solution to a system of linear
  equations and provides error bounds and backward error estimates for
  the solution.

  If equilibration was performed, the system becomes:
          (diag(R)*A_original*diag(C)) * X = diag(R)*B_original.

  See supermatrix.h for the definition of 'SuperMatrix' structure.

  Arguments
  =========

 trans   (input) trans_t
         Specifies the form of the system of equations:
         = NOTRANS: A * X = B  (No transpose)
         = TRANS:   A'* X = B  (Transpose)
         = CONJ:    A**H * X = B  (Conjugate transpose)

  A       (input) SuperMatrix*
          The original matrix A in the system, or the scaled A if
          equilibration was done. The type of A can be:
          Stype = SLU_NC, Dtype = SLU_C, Mtype = SLU_GE.

  L       (input) SuperMatrix*
    The factor L from the factorization Pr*A*Pc=L*U. Use
          compressed row subscripts storage for supernodes,
          i.e., L has types: Stype = SLU_SC, Dtype = SLU_C, Mtype = SLU_TRLU.

  U       (input) SuperMatrix*
          The factor U from the factorization Pr*A*Pc=L*U as computed by
          cgstrf(). Use column-wise storage scheme,
          i.e., U has types: Stype = SLU_NC, Dtype = SLU_C, Mtype = SLU_TRU.

  perm_c  (input) int*, dimension (A->ncol)
    Column permutation vector, which defines the
          permutation matrix Pc; perm_c[i] = j means column i of A is
          in position j in A*Pc.

  perm_r  (input) int*, dimension (A->nrow)
          Row permutation vector, which defines the permutation matrix Pr;
          perm_r[i] = j means row i of A is in position j in Pr*A.
```

```
equed   (input) Specifies the form of equilibration that was done.
        = 'N': No equilibration.
        = 'R': Row equilibration, i.e., A was premultiplied by diag(R).
        = 'C': Column equilibration, i.e., A was postmultiplied by
               diag(C).
        = 'B': Both row and column equilibration, i.e., A was replaced
               by diag(R)*A*diag(C).

R       (input) float*, dimension (A->nrow)
        The row scale factors for A.
        If equed = 'R' or 'B', A is premultiplied by diag(R).
        If equed = 'N' or 'C', R is not accessed.

C       (input) float*, dimension (A->ncol)
        The column scale factors for A.
        If equed = 'C' or 'B', A is postmultiplied by diag(C).
        If equed = 'N' or 'R', C is not accessed.

B       (input) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_C, Mtype = SLU_GE.
        The right hand side matrix B.
        if equed = 'R' or 'B', B is premultiplied by diag(R).

X       (input/output) SuperMatrix*
        X has types: Stype = SLU_DN, Dtype = SLU_C, Mtype = SLU_GE.
        On entry, the solution matrix X, as computed by cgstrs().
        On exit, the improved solution matrix X.
        if *equed = 'C' or 'B', X should be premultiplied by diag(C)
            in order to obtain the solution to the original system.

FERR    (output) float*, dimension (B->ncol)
        The estimated forward error bound for each solution vector
        X(j) (the j-th column of the solution matrix X).
        If XTRUE is the true solution corresponding to X(j), FERR(j)
        is an estimated upper bound for the magnitude of the largest
        element in (X(j) - XTRUE) divided by the magnitude of the
        largest element in X(j).  The estimate is as reliable as
        the estimate for RCOND, and is almost always a slight
        overestimate of the true error.

BERR    (output) float*, dimension (B->ncol)
        The componentwise relative backward error of each solution
        vector X(j) (i.e., the smallest relative change in
        any element of A or B that makes X(j) an exact solution).

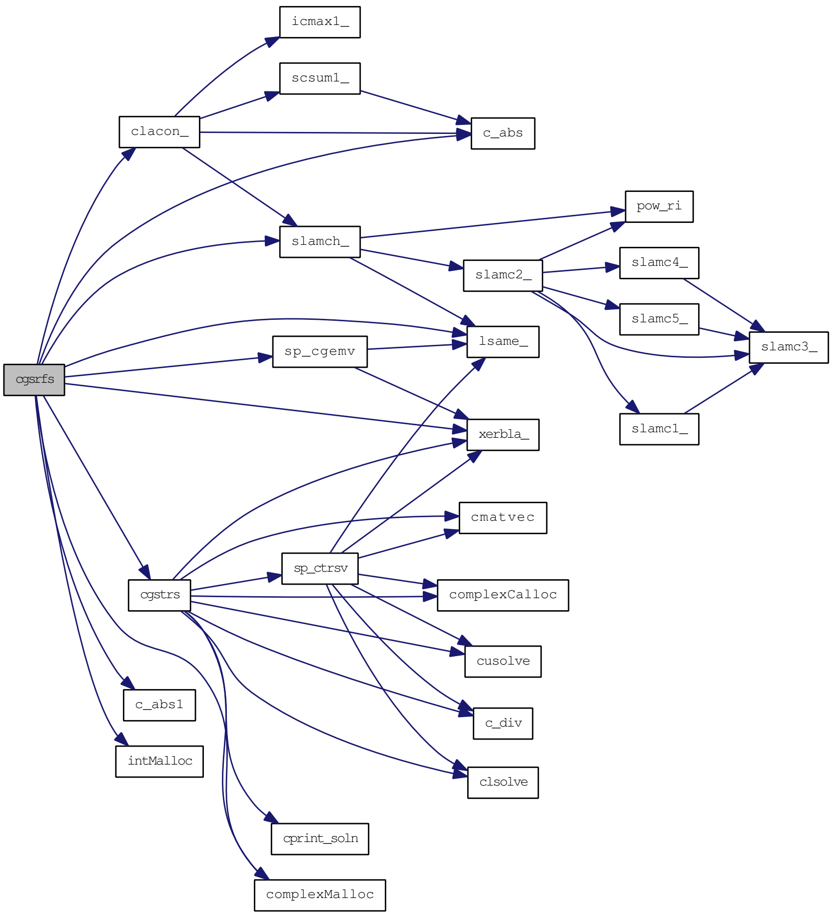
stat     (output) SuperLUStat_t*
         Record the statistics on runtime and floating-point operation count.
         See util.h for the definition of 'SuperLUStat_t'.

info    (output) int*
        = 0:  successful exit
         < 0:  if INFO = -i, the i-th argument had an illegal value

 Internal Parameters
 ===================
```

```
ITMAX is the maximum number of steps of iterative refinement.
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.110.3.18 void cgssv (superlu_options_t ∗ *options*, SuperMatrix ∗ *A*, int ∗ *perm_c*, int ∗ *perm_r*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, SuperMatrix ∗ *B*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======

CGSSV solves the system of linear equations A*X=B, using the
LU factorization from CGSTRF. It performs the following steps:
```

```
  1. If A is stored column-wise (A->Stype = SLU_NC):

     1.1. Permute the columns of A, forming A*Pc, where Pc
          is a permutation matrix. For more details of this step,
          see sp_preorder.c.

     1.2. Factor A as Pr*A*Pc=L*U with the permutation Pr determined
          by Gaussian elimination with partial pivoting.
          L is unit lower triangular with offdiagonal entries
          bounded by 1 in magnitude, and U is upper triangular.

     1.3. Solve the system of equations A*X=B using the factored
          form of A.

  2. If A is stored row-wise (A->Stype = SLU_NR), apply the
     above algorithm to the transpose of A:

     2.1. Permute columns of transpose(A) (rows of A),
          forming transpose(A)*Pc, where Pc is a permutation matrix.
          For more details of this step, see sp_preorder.c.

     2.2. Factor A as Pr*transpose(A)*Pc=L*U with the permutation Pr
          determined by Gaussian elimination with partial pivoting.
          L is unit lower triangular with offdiagonal entries
          bounded by 1 in magnitude, and U is upper triangular.

     2.3. Solve the system of equations A*X=B using the factored
          form of A.

  See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

options (input) superlu_options_t*
        The structure defines the input parameters to control
        how the LU decomposition will be performed and how the
        system will be solved.

A       (input) SuperMatrix*
        Matrix A in A*X=B, of dimension (A->nrow, A->ncol). The number
        of linear equations is A->nrow. Currently, the type of A can be:
        Stype = SLU_NC or SLU_NR; Dtype = SLU_C; Mtype = SLU_GE.
        In the future, more general A may be handled.

perm_c  (input/output) int*
        If A->Stype = SLU_NC, column permutation vector of size A->ncol
        which defines the permutation matrix Pc; perm_c[i] = j means
        column i of A is in position j in A*Pc.
        If A->Stype = SLU_NR, column permutation vector of size A->nrow
        which describes permutation of columns of transpose(A)
        (rows of A) as described above.
```

```
        If options->ColPerm = MY_PERMC or options->Fact = SamePattern or
           options->Fact = SamePattern_SameRowPerm, it is an input argument.
           On exit, perm_c may be overwritten by the product of the input
           perm_c and a permutation that postorders the elimination tree
           of Pc'*A'*A*Pc; perm_c is not changed if the elimination tree
           is already in postorder.
        Otherwise, it is an output argument.


perm_r  (input/output) int*
        If A->Stype = SLU_NC, row permutation vector of size A->nrow,
        which defines the permutation matrix Pr, and is determined
        by partial pivoting.  perm_r[i] = j means row i of A is in
        position j in Pr*A.
        If A->Stype = SLU_NR, permutation vector of size A->ncol, which
        determines permutation of rows of transpose(A)
        (columns of A) as described above.


        If options->RowPerm = MY_PERMR or
           options->Fact = SamePattern_SameRowPerm, perm_r is an
           input argument.
        otherwise it is an output argument.


L       (output) SuperMatrix*
        The factor L from the factorization
            Pr*A*Pc=L*U              (if A->Stype = SLU_NC) or
            Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
        Uses compressed row subscripts storage for supernodes, i.e.,
        L has types: Stype = SLU_SC, Dtype = SLU_C, Mtype = SLU_TRLU.

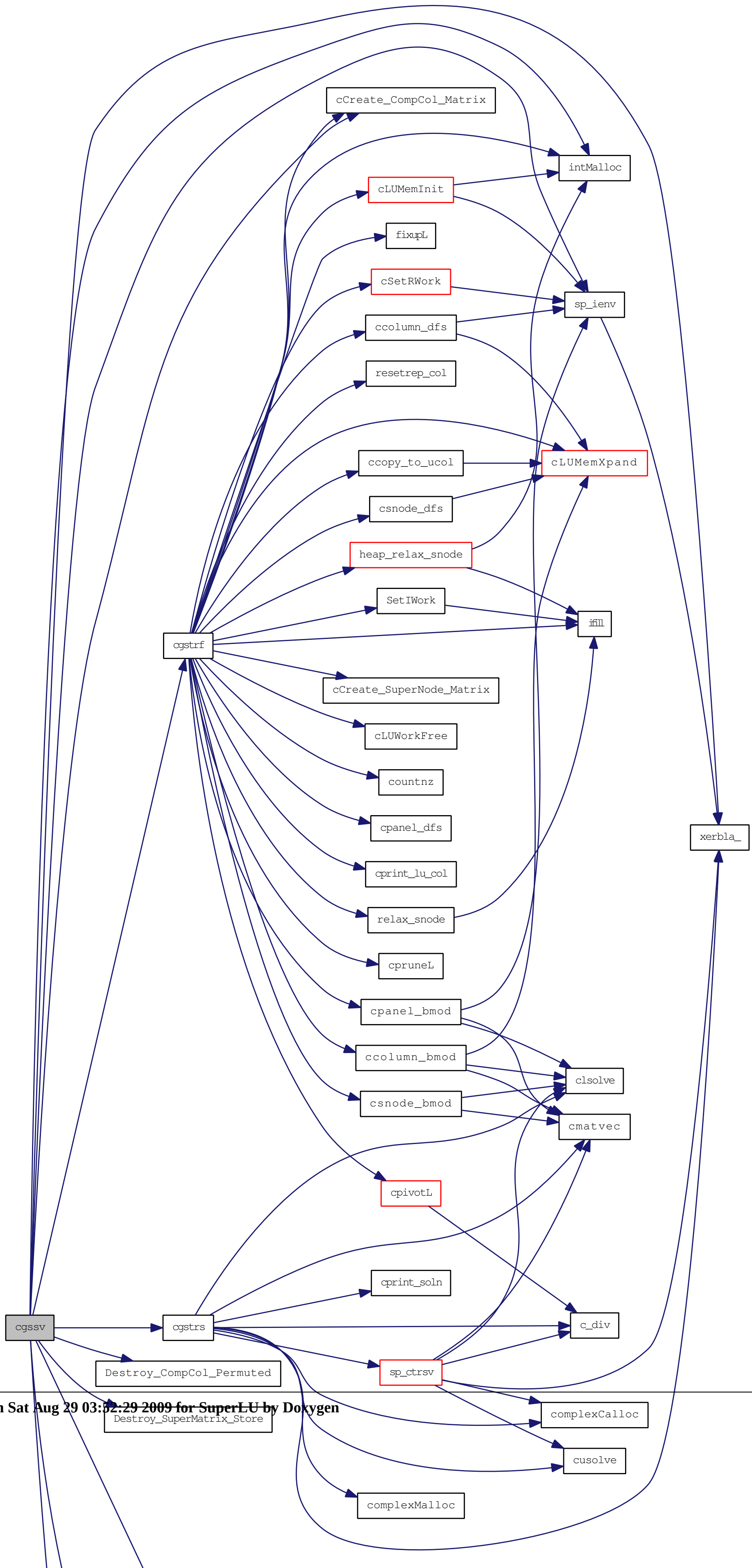

U       (output) SuperMatrix*
  The factor U from the factorization
            Pr*A*Pc=L*U              (if A->Stype = SLU_NC) or
            Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
        Uses column-wise storage scheme, i.e., U has types:
        Stype = SLU_NC, Dtype = SLU_C, Mtype = SLU_TRU.


B       (input/output) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_C, Mtype = SLU_GE.
        On entry, the right hand side matrix.
        On exit, the solution matrix if info = 0;


stat   (output) SuperLUStat_t*
       Record the statistics on runtime and floating-point operation count.
       See util.h for the definition of 'SuperLUStat_t'.


info    (output) int*
  = 0: successful exit
        > 0: if info = i, and i is
            <= A->ncol: U(i,i) is exactly zero. The factorization has
               been completed, but the factor U is exactly singular,
               so the solution could not be computed.
            > A->ncol: number of bytes allocated when memory allocation
               failure occurred, plus A->ncol.
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.110.3.19 void cgssvx (superlu_options_t ∗ *options*, SuperMatrix ∗ *A*, int ∗ *perm_c*, int ∗ *perm_r*, int ∗ *etree*, char ∗ *equed*, float ∗ *R*, float ∗ *C*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, void ∗ *work*, int *lwork*, SuperMatrix ∗ *B*, SuperMatrix ∗ *X*, float ∗ *recip_pivot_growth*, float ∗ *rcond*, float ∗ *ferr*, float ∗ *berr*, mem_usage_t ∗ *mem_usage*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======

CGSSVX solves the system of linear equations A*X=B or A'*X=B, using
the LU factorization from cgstrf(). Error bounds on the solution and
a condition estimate are also provided. It performs the following steps:

  1. If A is stored column-wise (A->Stype = SLU_NC):

     1.1. If options->Equil = YES, scaling factors are computed to
          equilibrate the system:
          options->Trans = NOTRANS:
              diag(R)*A*diag(C) *inv(diag(C))*X = diag(R)*B
          options->Trans = TRANS:
              (diag(R)*A*diag(C))**T *inv(diag(R))*X = diag(C)*B
          options->Trans = CONJ:
              (diag(R)*A*diag(C))**H *inv(diag(R))*X = diag(C)*B
          Whether or not the system will be equilibrated depends on the
          scaling of the matrix A, but if equilibration is used, A is
          overwritten by diag(R)*A*diag(C) and B by diag(R)*B
          (if options->Trans=NOTRANS) or diag(C)*B (if options->Trans
          = TRANS or CONJ).

     1.2. Permute columns of A, forming A*Pc, where Pc is a permutation
          matrix that usually preserves sparsity.
          For more details of this step, see sp_preorder.c.

     1.3. If options->Fact != FACTORED, the LU decomposition is used to
          factor the matrix A (after equilibration if options->Equil = YES)
          as Pr*A*Pc = L*U, with Pr determined by partial pivoting.

     1.4. Compute the reciprocal pivot growth factor.

     1.5. If some U(i,i) = 0, so that U is exactly singular, then the
          routine returns with info = i. Otherwise, the factored form of
          A is used to estimate the condition number of the matrix A. If
          the reciprocal of the condition number is less than machine
          precision, info = A->ncol+1 is returned as a warning, but the
          routine still goes on to solve for X and computes error bounds
          as described below.
```

```
    1.6. The system of equations is solved for X using the factored form
         of A.

    1.7. If options->IterRefine != NOREFINE, iterative refinement is
         applied to improve the computed solution matrix and calculate
         error bounds and backward error estimates for it.

    1.8. If equilibration was used, the matrix X is premultiplied by
         diag(C) (if options->Trans = NOTRANS) or diag(R)
         (if options->Trans = TRANS or CONJ) so that it solves the
         original system before equilibration.

 2. If A is stored row-wise (A->Stype = SLU_NR), apply the above algorithm
    to the transpose of A:

    2.1. If options->Equil = YES, scaling factors are computed to
         equilibrate the system:
         options->Trans = NOTRANS:
             diag(R)*A*diag(C) *inv(diag(C))*X = diag(R)*B
         options->Trans = TRANS:
             (diag(R)*A*diag(C))**T *inv(diag(R))*X = diag(C)*B
         options->Trans = CONJ:
             (diag(R)*A*diag(C))**H *inv(diag(R))*X = diag(C)*B
         Whether or not the system will be equilibrated depends on the
         scaling of the matrix A, but if equilibration is used, A' is
         overwritten by diag(R)*A'*diag(C) and B by diag(R)*B
         (if trans='N') or diag(C)*B (if trans = 'T' or 'C').

    2.2. Permute columns of transpose(A) (rows of A),
         forming transpose(A)*Pc, where Pc is a permutation matrix that
         usually preserves sparsity.
         For more details of this step, see sp_preorder.c.

    2.3. If options->Fact != FACTORED, the LU decomposition is used to
         factor the transpose(A) (after equilibration if
         options->Fact = YES) as Pr*transpose(A)*Pc = L*U with the
         permutation Pr determined by partial pivoting.

    2.4. Compute the reciprocal pivot growth factor.

    2.5. If some U(i,i) = 0, so that U is exactly singular, then the
         routine returns with info = i. Otherwise, the factored form
         of transpose(A) is used to estimate the condition number of the
         matrix A. If the reciprocal of the condition number
         is less than machine precision, info = A->nrow+1 is returned as
         a warning, but the routine still goes on to solve for X and
         computes error bounds as described below.

    2.6. The system of equations is solved for X using the factored form
         of transpose(A).

    2.7. If options->IterRefine != NOREFINE, iterative refinement is
         applied to improve the computed solution matrix and calculate
         error bounds and backward error estimates for it.
```

```
    2.8. If equilibration was used, the matrix X is premultiplied by
         diag(C) (if options->Trans = NOTRANS) or diag(R)
         (if options->Trans = TRANS or CONJ) so that it solves the
         original system before equilibration.

  See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

options (input) superlu_options_t*
        The structure defines the input parameters to control
        how the LU decomposition will be performed and how the
        system will be solved.

A       (input/output) SuperMatrix*
        Matrix A in A*X=B, of dimension (A->nrow, A->ncol). The number
        of the linear equations is A->nrow. Currently, the type of A can be:
        Stype = SLU_NC or SLU_NR, Dtype = SLU_D, Mtype = SLU_GE.
        In the future, more general A may be handled.

        On entry, If options->Fact = FACTORED and equed is not 'N',
        then A must have been equilibrated by the scaling factors in
        R and/or C.
        On exit, A is not modified if options->Equil = NO, or if
        options->Equil = YES but equed = 'N' on exit.
        Otherwise, if options->Equil = YES and equed is not 'N',
        A is scaled as follows:
        If A->Stype = SLU_NC:
          equed = 'R':  A := diag(R) * A
          equed = 'C':  A := A * diag(C)
          equed = 'B':  A := diag(R) * A * diag(C).
        If A->Stype = SLU_NR:
          equed = 'R':  transpose(A) := diag(R) * transpose(A)
          equed = 'C':  transpose(A) := transpose(A) * diag(C)
          equed = 'B':  transpose(A) := diag(R) * transpose(A) * diag(C).

perm_c  (input/output) int*
  If A->Stype = SLU_NC, Column permutation vector of size A->ncol,
        which defines the permutation matrix Pc; perm_c[i] = j means
        column i of A is in position j in A*Pc.
        On exit, perm_c may be overwritten by the product of the input
        perm_c and a permutation that postorders the elimination tree
        of Pc'*A'*A*Pc; perm_c is not changed if the elimination tree
        is already in postorder.

        If A->Stype = SLU_NR, column permutation vector of size A->nrow,
        which describes permutation of columns of transpose(A)
        (rows of A) as described above.

perm_r  (input/output) int*
        If A->Stype = SLU_NC, row permutation vector of size A->nrow,
        which defines the permutation matrix Pr, and is determined
        by partial pivoting.  perm_r[i] = j means row i of A is in
        position j in Pr*A.
```

```
        If A->Stype = SLU_NR, permutation vector of size A->ncol, which
        determines permutation of rows of transpose(A)
        (columns of A) as described above.

        If options->Fact = SamePattern_SameRowPerm, the pivoting routine
        will try to use the input perm_r, unless a certain threshold
        criterion is violated. In that case, perm_r is overwritten by a
        new permutation determined by partial pivoting or diagonal
        threshold pivoting.
        Otherwise, perm_r is output argument.

etree   (input/output) int*,  dimension (A->ncol)
        Elimination tree of Pc'*A'*A*Pc.
        If options->Fact != FACTORED and options->Fact != DOFACT,
        etree is an input argument, otherwise it is an output argument.
        Note: etree is a vector of parent pointers for a forest whose
        vertices are the integers 0 to A->ncol-1; etree[root]==A->ncol.

equed   (input/output) char*
        Specifies the form of equilibration that was done.
        = 'N': No equilibration.
        = 'R': Row equilibration, i.e., A was premultiplied by diag(R).
        = 'C': Column equilibration, i.e., A was postmultiplied by diag(C).
        = 'B': Both row and column equilibration, i.e., A was replaced
               by diag(R)*A*diag(C).
        If options->Fact = FACTORED, equed is an input argument,
        otherwise it is an output argument.

R       (input/output) float*, dimension (A->nrow)
        The row scale factors for A or transpose(A).
        If equed = 'R' or 'B', A (if A->Stype = SLU_NC) or transpose(A)
            (if A->Stype = SLU_NR) is multiplied on the left by diag(R).
        If equed = 'N' or 'C', R is not accessed.
        If options->Fact = FACTORED, R is an input argument,
            otherwise, R is output.
        If options->zFact = FACTORED and equed = 'R' or 'B', each element
            of R must be positive.

C       (input/output) float*, dimension (A->ncol)
        The column scale factors for A or transpose(A).
        If equed = 'C' or 'B', A (if A->Stype = SLU_NC) or transpose(A)
            (if A->Stype = SLU_NR) is multiplied on the right by diag(C).
        If equed = 'N' or 'R', C is not accessed.
        If options->Fact = FACTORED, C is an input argument,
            otherwise, C is output.
        If options->Fact = FACTORED and equed = 'C' or 'B', each element
            of C must be positive.

L       (output) SuperMatrix*
  The factor L from the factorization
            Pr*A*Pc=L*U              (if A->Stype SLU_= NC) or
            Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
        Uses compressed row subscripts storage for supernodes, i.e.,
        L has types: Stype = SLU_SC, Dtype = SLU_C, Mtype = SLU_TRLU.

U       (output) SuperMatrix*
```

```
  The factor U from the factorization
            Pr*A*Pc=L*U              (if A->Stype = SLU_NC) or
            Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
        Uses column-wise storage scheme, i.e., U has types:
        Stype = SLU_NC, Dtype = SLU_C, Mtype = SLU_TRU.

work    (workspace/output) void*, size (lwork) (in bytes)
        User supplied workspace, should be large enough
        to hold data structures for factors L and U.
        On exit, if fact is not 'F', L and U point to this array.

lwork   (input) int
        Specifies the size of work array in bytes.
        = 0:  allocate space internally by system malloc;
        > 0:  use user-supplied work array of length lwork in bytes,
              returns error if space runs out.
        = -1: the routine guesses the amount of space needed without
              performing the factorization, and returns it in
              mem_usage->total_needed; no other side effects.

        See argument 'mem_usage' for memory usage statistics.

B       (input/output) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_C, Mtype = SLU_GE.
        On entry, the right hand side matrix.
        If B->ncol = 0, only LU decomposition is performed, the triangular
                        solve is skipped.
        On exit,
           if equed = 'N', B is not modified; otherwise
           if A->Stype = SLU_NC:
              if options->Trans = NOTRANS and equed = 'R' or 'B',
                 B is overwritten by diag(R)*B;
              if options->Trans = TRANS or CONJ and equed = 'C' of 'B',
                 B is overwritten by diag(C)*B;
           if A->Stype = SLU_NR:
              if options->Trans = NOTRANS and equed = 'C' or 'B',
                 B is overwritten by diag(C)*B;
              if options->Trans = TRANS or CONJ and equed = 'R' of 'B',
                 B is overwritten by diag(R)*B.

X       (output) SuperMatrix*
        X has types: Stype = SLU_DN, Dtype = SLU_C, Mtype = SLU_GE.
        If info = 0 or info = A->ncol+1, X contains the solution matrix
        to the original system of equations. Note that A and B are modified
        on exit if equed is not 'N', and the solution to the equilibrated
        system is inv(diag(C))*X if options->Trans = NOTRANS and
        equed = 'C' or 'B', or inv(diag(R))*X if options->Trans = 'T' or 'C'
        and equed = 'R' or 'B'.

recip_pivot_growth (output) float*
        The reciprocal pivot growth factor max_j( norm(A_j)/norm(U_j) ).
        The infinity norm is used. If recip_pivot_growth is much less
        than 1, the stability of the LU factorization could be poor.

rcond   (output) float*
        The estimate of the reciprocal condition number of the matrix A
```

```
        after equilibration (if done). If rcond is less than the machine
        precision (in particular, if rcond = 0), the matrix is singular
        to working precision. This condition is indicated by a return
        code of info > 0.

FERR    (output) float*, dimension (B->ncol)
        The estimated forward error bound for each solution vector
        X(j) (the j-th column of the solution matrix X).
        If XTRUE is the true solution corresponding to X(j), FERR(j)
        is an estimated upper bound for the magnitude of the largest
        element in (X(j) - XTRUE) divided by the magnitude of the
        largest element in X(j).  The estimate is as reliable as
        the estimate for RCOND, and is almost always a slight
        overestimate of the true error.
        If options->IterRefine = NOREFINE, ferr = 1.0.

BERR    (output) float*, dimension (B->ncol)
        The componentwise relative backward error of each solution
        vector X(j) (i.e., the smallest relative change in
        any element of A or B that makes X(j) an exact solution).
        If options->IterRefine = NOREFINE, berr = 1.0.

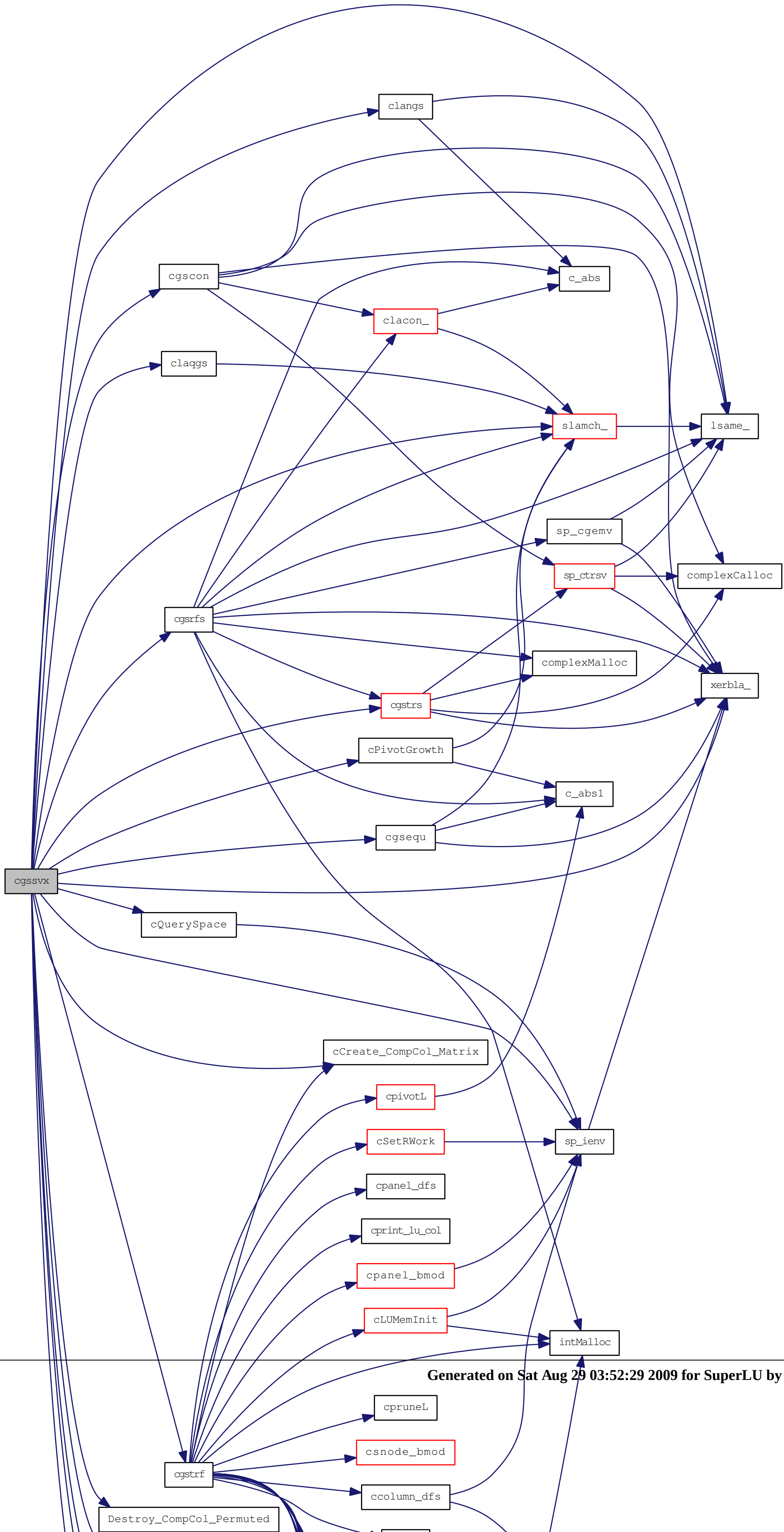
mem_usage (output) mem_usage_t*
        Record the memory usage statistics, consisting of following fields:
```

- for_lu (float)
  The amount of space used in bytes for L data structures.
- total_needed (float)
  The amount of space needed in bytes to perform factorization.
- expansions (int)
  The number of memory expansions during the LU factorization.

```
stat   (output) SuperLUStat_t*
       Record the statistics on runtime and floating-point operation count.
       See util.h for the definition of 'SuperLUStat_t'.

info    (output) int*
        = 0: successful exit
        < 0: if info = -i, the i-th argument had an illegal value
        > 0: if info = i, and i is
             <= A->ncol: U(i,i) is exactly zero. The factorization has
                   been completed, but the factor U is exactly
                   singular, so the solution and error bounds
                   could not be computed.
             = A->ncol+1: U is nonsingular, but RCOND is less than machine
                   precision, meaning that the matrix is singular to
                   working precision. Nevertheless, the solution and
                   error bounds are computed because there are a number
                   of situations where the computed solution can be more
                   accurate than the value of RCOND would suggest.
             > A->ncol+1: number of bytes allocated when memory allocation
                   failure occurred, plus A->ncol.
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.110.3.20 void cgstrf (superlu_options_t ∗ *options*, SuperMatrix ∗ *A*, float *drop_tol*, int *relax*, int *panel_size*, int ∗ *etree*, void ∗ *work*, int *lwork*, int ∗ *perm_c*, int ∗ *perm_r*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======

CGSTRF computes an LU factorization of a general sparse m-by-n
matrix A using partial pivoting with row interchanges.
The factorization has the form
    Pr * A = L * U
where Pr is a row permutation matrix, L is lower triangular with unit
diagonal elements (lower trapezoidal if A->nrow > A->ncol), and U is upper
triangular (upper trapezoidal if A->nrow < A->ncol).

See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

options (input) superlu_options_t*
        The structure defines the input parameters to control
        how the LU decomposition will be performed.

A        (input) SuperMatrix*
   Original matrix A, permuted by columns, of dimension
         (A->nrow, A->ncol). The type of A can be:
         Stype = SLU_NCP; Dtype = SLU_C; Mtype = SLU_GE.

drop_tol (input) float (NOT IMPLEMENTED)
   Drop tolerance parameter. At step j of the Gaussian elimination,
         if abs(A_ij)/(max_i abs(A_ij)) < drop_tol, drop entry A_ij.
         0 <= drop_tol <= 1. The default value of drop_tol is 0.

relax    (input) int
         To control degree of relaxing supernodes. If the number
         of nodes (columns) in a subtree of the elimination tree is less
         than relax, this subtree is considered as one supernode,
         regardless of the row structures of those columns.

panel_size (input) int
         A panel consists of at most panel_size consecutive columns.

etree    (input) int*, dimension (A->ncol)
         Elimination tree of A'*A.
         Note: etree is a vector of parent pointers for a forest whose
         vertices are the integers 0 to A->ncol-1; etree[root]==A->ncol.
         On input, the columns of A should be permuted so that the
         etree is in a certain postorder.
```

```
work     (input/output) void*, size (lwork) (in bytes)
         User-supplied work space and space for the output data structures.
         Not referenced if lwork = 0;

lwork   (input) int
        Specifies the size of work array in bytes.
        = 0:  allocate space internally by system malloc;
        > 0:  use user-supplied work array of length lwork in bytes,
              returns error if space runs out.
        = -1: the routine guesses the amount of space needed without
              performing the factorization, and returns it in
              *info; no other side effects.

perm_c   (input) int*, dimension (A->ncol)
   Column permutation vector, which defines the
         permutation matrix Pc; perm_c[i] = j means column i of A is
         in position j in A*Pc.
         When searching for diagonal, perm_c[*] is applied to the
         row subscripts of A, so that diagonal threshold pivoting
         can find the diagonal of A, rather than that of A*Pc.

perm_r   (input/output) int*, dimension (A->nrow)
         Row permutation vector which defines the permutation matrix Pr,
         perm_r[i] = j means row i of A is in position j in Pr*A.
         If options->Fact = SamePattern_SameRowPerm, the pivoting routine
            will try to use the input perm_r, unless a certain threshold
            criterion is violated. In that case, perm_r is overwritten by
            a new permutation determined by partial pivoting or diagonal
            threshold pivoting.
         Otherwise, perm_r is output argument;

L        (output) SuperMatrix*
         The factor L from the factorization Pr*A=L*U; use compressed row
         subscripts storage for supernodes, i.e., L has type:
         Stype = SLU_SC, Dtype = SLU_C, Mtype = SLU_TRLU.

U        (output) SuperMatrix*
   The factor U from the factorization Pr*A*Pc=L*U. Use column-wise
         storage scheme, i.e., U has types: Stype = SLU_NC,
         Dtype = SLU_C, Mtype = SLU_TRU.

stat     (output) SuperLUStat_t*
         Record the statistics on runtime and floating-point operation count.
         See util.h for the definition of 'SuperLUStat_t'.

info     (output) int*
         = 0: successful exit
         < 0: if info = -i, the i-th argument had an illegal value
         > 0: if info = i, and i is
            <= A->ncol: U(i,i) is exactly zero. The factorization has
               been completed, but the factor U is exactly singular,
               and division by zero will occur if it is used to solve a
               system of equations.
            > A->ncol: number of bytes allocated when memory allocation
               failure occurred, plus A->ncol. If lwork = -1, it is
               the estimated amount of space needed, plus A->ncol.
```

```
 ======================================================================

 Local Working Arrays:
 ======================
   m = number of rows in the matrix
   n = number of columns in the matrix

   xprune[0:n-1]: xprune[*] points to locations in subscript
vector lsub[*]. For column i, xprune[i] denotes the point where
structural pruning begins. I.e. only xlsub[i],..,xprune[i]-1 need
to be traversed for symbolic factorization.

   marker[0:3*m-1]: marker[i] = j means that node i has been
reached when working on column j.
Storage: relative to original row subscripts
NOTE: There are 3 of them: marker/marker1 are used for panel dfs,
      see cpanel_dfs.c; marker2 is used for inner-factorization,
            see ccolumn_dfs.c.

   parent[0:m-1]: parent vector used during dfs
      Storage: relative to new row subscripts

   xplore[0:m-1]: xplore[i] gives the location of the next (dfs)
unexplored neighbor of i in lsub[*]

   segrep[0:nseg-1]: contains the list of supernodal representatives
in topological order of the dfs. A supernode representative is the
last column of a supernode.
      The maximum size of segrep[] is n.

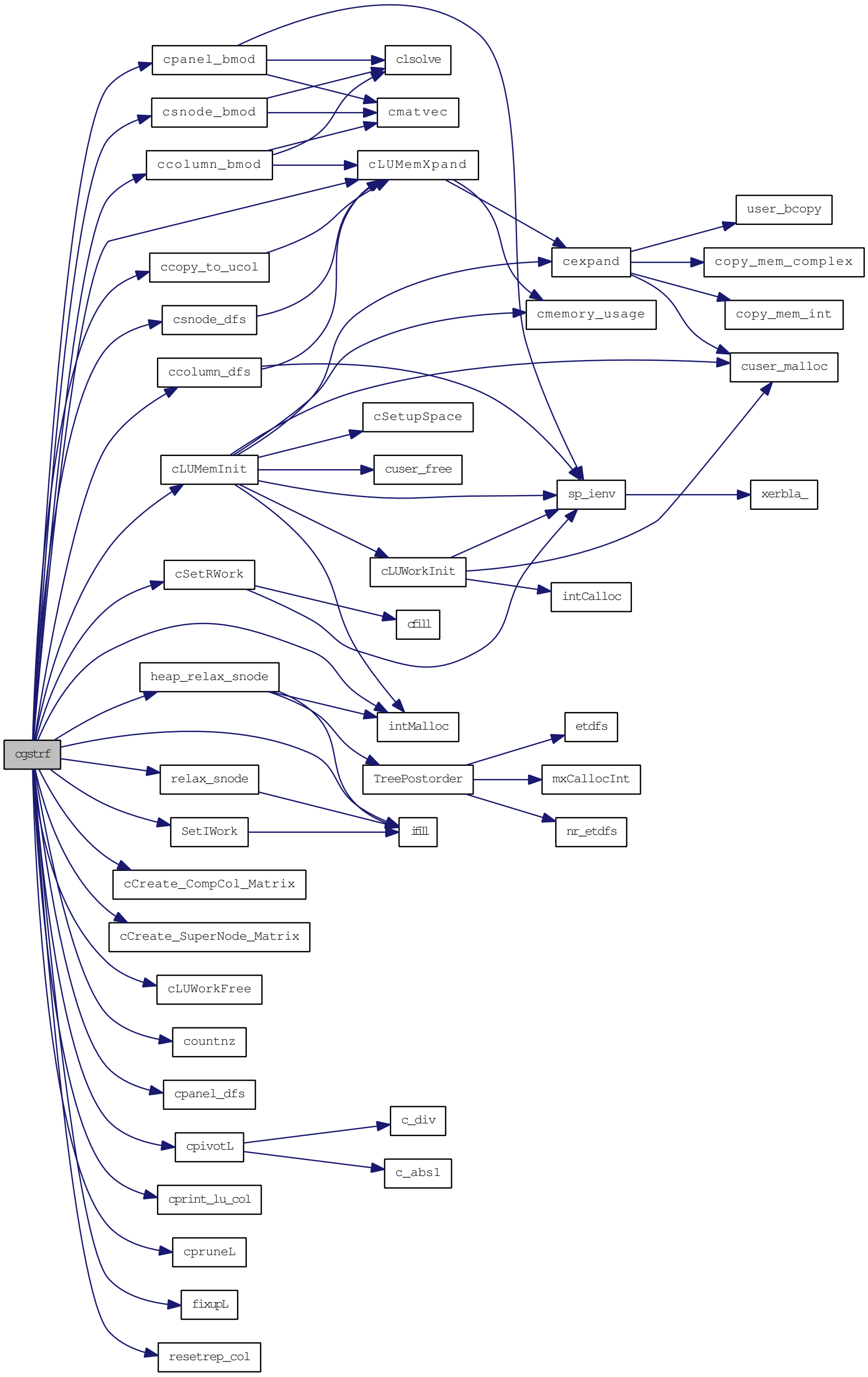
   repfnz[0:W*m-1]: for a nonzero segment U[*,j] that ends at a
supernodal representative r, repfnz[r] is the location of the first
nonzero in this segment.  It is also used during the dfs: repfnz[r]>0
indicates the supernode r has been explored.
NOTE: There are W of them, each used for one column of a panel.

   panel_lsub[0:W*m-1]: temporary for the nonzeros row indices below
      the panel diagonal. These are filled in during cpanel_dfs(), and are
      used later in the inner LU factorization within the panel.
panel_lsub[]/dense[] pair forms the SPA data structure.
NOTE: There are W of them.

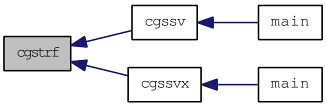
   dense[0:W*m-1]: sparse accumulating (SPA) vector for intermediate values;
        NOTE: there are W of them.

   tempv[0:*]: real temporary used for dense numeric kernels;
The size of this array is defined by NUM_TEMPV() in slu_cdefs.h.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.110.3.21 void cgstrs (trans_t *trans*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, int ∗ *perm_c*, int ∗ *perm_r*, SuperMatrix ∗ *B*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======

CGSTRS solves a system of linear equations A*X=B or A'*X=B
with A sparse and B dense, using the LU factorization computed by
CGSTRF.

See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

trans   (input) trans_t
         Specifies the form of the system of equations:
         = NOTRANS: A * X = B  (No transpose)
         = TRANS:   A'* X = B  (Transpose)
         = CONJ:    A**H * X = B  (Conjugate transpose)

L       (input) SuperMatrix*
        The factor L from the factorization Pr*A*Pc=L*U as computed by
        cgstrf(). Use compressed row subscripts storage for supernodes,
        i.e., L has types: Stype = SLU_SC, Dtype = SLU_C, Mtype = SLU_TRLU.

U       (input) SuperMatrix*
        The factor U from the factorization Pr*A*Pc=L*U as computed by
        cgstrf(). Use column-wise storage scheme, i.e., U has types:
        Stype = SLU_NC, Dtype = SLU_C, Mtype = SLU_TRU.

perm_c  (input) int*, dimension (L->ncol)
  Column permutation vector, which defines the
        permutation matrix Pc; perm_c[i] = j means column i of A is
        in position j in A*Pc.

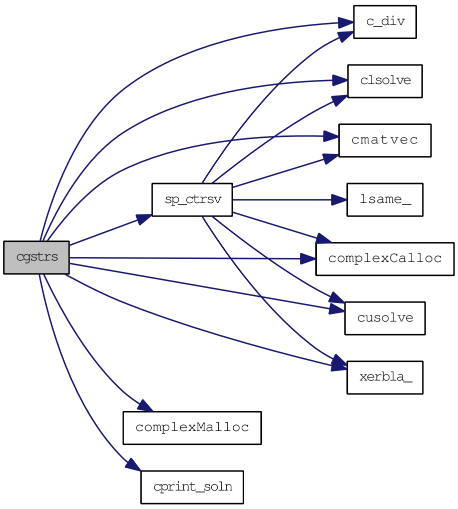
perm_r  (input) int*, dimension (L->nrow)
        Row permutation vector, which defines the permutation matrix Pr;
        perm_r[i] = j means row i of A is in position j in Pr*A.

B       (input/output) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_C, Mtype = SLU_GE.
        On entry, the right hand side matrix.
        On exit, the solution matrix if info = 0;
```

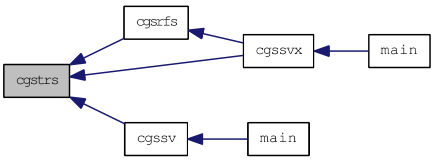
```
stat     (output) SuperLUStat_t*
         Record the statistics on runtime and floating-point operation count.
         See util.h for the definition of 'SuperLUStat_t'.

info    (output) int*
    = 0: successful exit
  < 0: if info = -i, the i-th argument had an illegal value
```

Here is the call graph for this function:



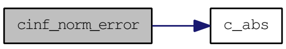
Here is the caller graph for this function:



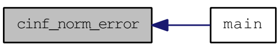
4.110.3.22 void check_tempv (int, complex ∗)

4.110.3.23 void cinf_norm_error (int, SuperMatrix ∗, complex ∗)

Here is the call graph for this function:



Here is the caller graph for this function:



4.110.3.24 void claqgs (SuperMatrix * *A*, float * *r*, float * *c*, float *rowcnd*, float *colcnd*, float *amax*, char * *equed*)

```
Purpose
=======

CLAQGS equilibrates a general sparse M by N matrix A using the row and
scaling factors in the vectors R and C.

See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

A       (input/output) SuperMatrix*
        On exit, the equilibrated matrix.  See EQUED for the form of
        the equilibrated matrix. The type of A can be:
 Stype = NC; Dtype = SLU_C; Mtype = GE.

R       (input) float*, dimension (A->nrow)
        The row scale factors for A.

C       (input) float*, dimension (A->ncol)
        The column scale factors for A.

ROWCND  (input) float
        Ratio of the smallest R(i) to the largest R(i).

COLCND  (input) float
        Ratio of the smallest C(i) to the largest C(i).

AMAX    (input) float
        Absolute value of largest matrix entry.

EQUED   (output) char*
        Specifies the form of equilibration that was done.
        = 'N':  No equilibration
        = 'R':  Row equilibration, i.e., A has been premultiplied by
                diag(R).
        = 'C':  Column equilibration, i.e., A has been postmultiplied
                by diag(C).
        = 'B':  Both row and column equilibration, i.e., A has been
                replaced by diag(R) * A * diag(C).

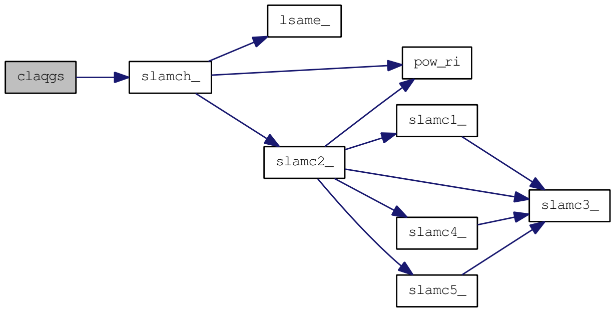
Internal Parameters
===================

THRESH is a threshold value used to decide if row or column scaling
should be done based on the ratio of the row or column scaling
factors.  If ROWCND < THRESH, row scaling is done, and if
COLCND < THRESH, column scaling is done.

LARGE and SMALL are threshold values used to decide if row scaling
should be done based on the absolute size of the largest matrix
element.  If AMAX > LARGE or AMAX < SMALL, row scaling is done.
```

```
=====================================================================
```

Here is the call graph for this function:



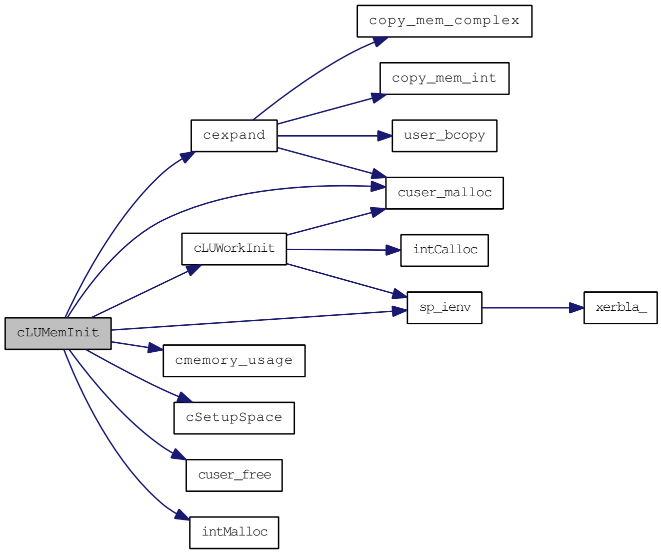
Here is the caller graph for this function:

4.110.3.25 int cLUMemInit (fact_t *fact*, void ∗ *work*, int *lwork*, int *m*, int *n*, int *annz*, int *panel_size*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, GlobalLU_t ∗ *Glu*, int ∗∗ *iwork*, complex ∗∗ *dwork*)

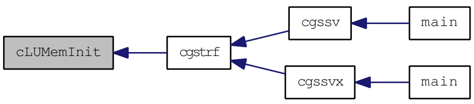
Memory-related.

```
For those unpredictable size, make a guess as FILL * nnz(A).
Return value:
    If lwork = -1, return the estimated amount of space required, plus n;
    otherwise, return the amount of space actually allocated when
    memory allocation failure occurred.
```

Here is the call graph for this function:



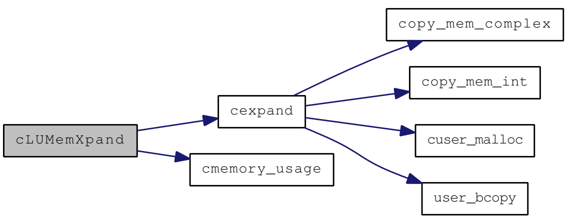
Here is the caller graph for this function:



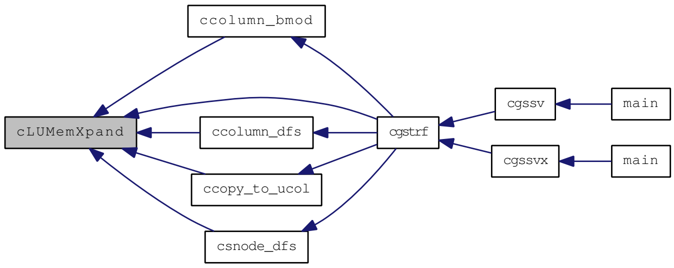
4.110.3.26 int cLUMemXpand (int *jcol*, int *next*, MemType *mem_type*, int ∗ *maxlen*, GlobalLU_t ∗ *Glu*)

```
Return value:   0 - successful return
              > 0 - number of bytes allocated when run out of space
```

Here is the call graph for this function:

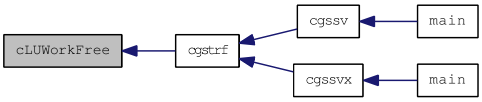


Here is the caller graph for this function:



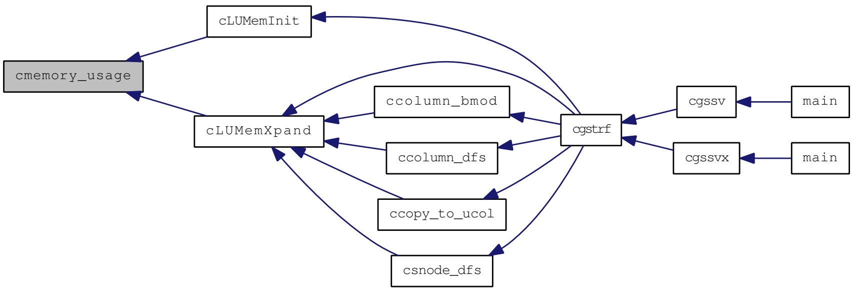
4.110.3.27 void cLUWorkFree (int ∗, complex ∗, GlobalLU_t ∗)

Here is the caller graph for this function:



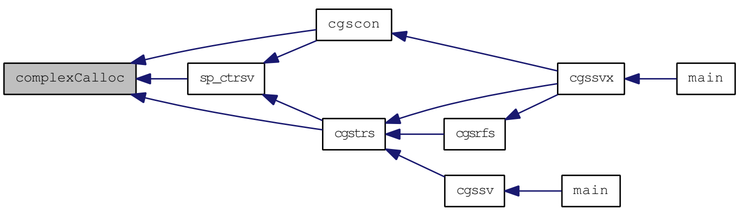
4.110.3.28 int cmemory_usage (const *int*, const *int*, const *int*, const *int*)

Here is the caller graph for this function:



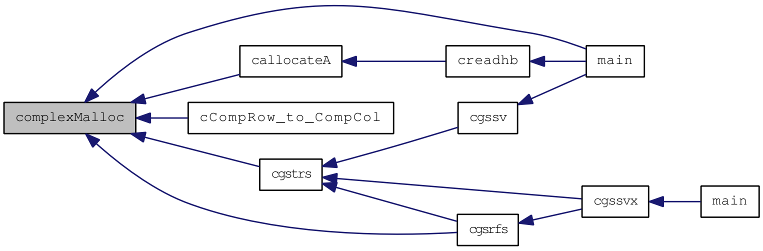
4.110.3.29 complex∗ complexCalloc (int)

Here is the caller graph for this function:



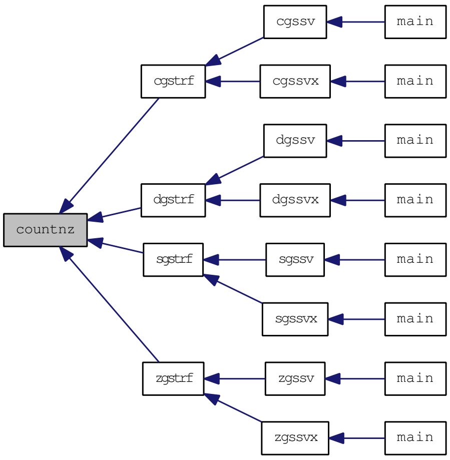
4.110.3.30 complex∗ complexMalloc (int)

Here is the caller graph for this function:



4.110.3.31 void countnz (const *int*, int ∗, int ∗, int ∗, GlobalLU_t ∗)

Here is the caller graph for this function:



4.110.3.32 void cpanel_bmod (const int *m*, const int *w*, const int *jcol*, const int *nseg*, complex ∗ *dense*, complex ∗ *tempv*, int ∗ *segrep*, int ∗ *repfnz*, GlobalLU_t ∗ *Glu*, SuperLUStat_t ∗ *stat*)

```
Purpose
=======

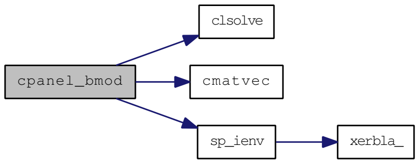
   Performs numeric block updates (sup-panel) in topological order.
   It features: col-col, 2cols-col, 3cols-col, and sup-col updates.
   Special processing on the supernodal portion of L[*,j]


   Before entering this routine, the original nonzeros in the panel
   were already copied into the spa[m,w].

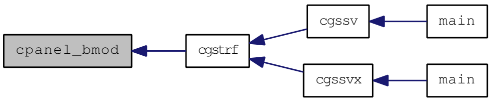

   Updated/Output parameters-
   dense[0:m-1,w]: L[*,j:j+w-1] and U[*,j:j+w-1] are returned
```

```
collectively in the m-by-w vector dense[*].
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.110.3.33 void cpanel_dfs (const int *m*, const int *w*, const int *jcol*, SuperMatrix ∗ *A*, int ∗ *perm_r*, int ∗ *nseg*, complex ∗ *dense*, int ∗ *panel_lsub*, int ∗ *segrep*, int ∗ *repfnz*, int ∗ *xprune*, int ∗ *marker*, int ∗ *parent*, int ∗ *xplore*, GlobalLU_t ∗ *Glu*)

```
Purpose
=======

  Performs a symbolic factorization on a panel of columns [jcol, jcol+w).

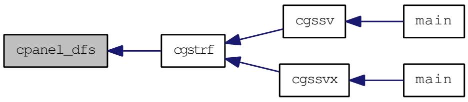
  A supernode representative is the last column of a supernode.
  The nonzeros in U[*,j] are segments that end at supernodal
  representatives.

  The routine returns one list of the supernodal representatives
  in topological order of the dfs that generates them. This list is
  a superset of the topological order of each individual column within
  the panel.
  The location of the first nonzero in each supernodal segment
  (supernodal entry location) is also returned. Each column has a
  separate list for this purpose.

  Two marker arrays are used for dfs:
    marker[i] == jj, if i was visited during dfs of current column jj;
    marker1[i] >= jcol, if i was visited by earlier columns in this panel;

  marker: A-row --> A-row/col (0/1)
  repfnz: SuperA-col --> PA-row
  parent: SuperA-col --> SuperA-col
  xplore: SuperA-col --> index to L-structure
```

Here is the caller graph for this function:



4.110.3.34 float cPivotGrowth (int *ncols*, SuperMatrix ∗ *A*, int ∗ *perm_c*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*)

```
Purpose
=======

Compute the reciprocal pivot growth factor of the leading ncols columns
of the matrix, using the formula:
    min_j ( max_i(abs(A_ij)) / max_i(abs(U_ij)) )

Arguments
=========

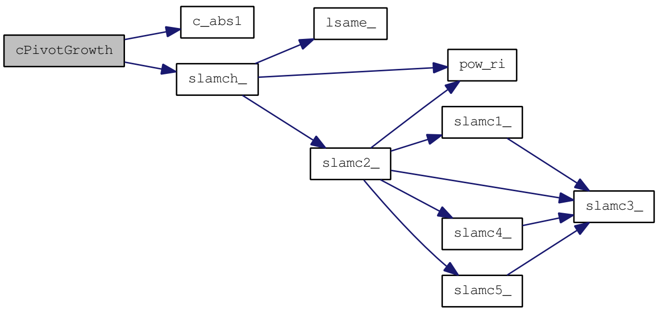
ncols    (input) int
         The number of columns of matrices A, L and U.

A        (input) SuperMatrix*
   Original matrix A, permuted by columns, of dimension
         (A->nrow, A->ncol). The type of A can be:
         Stype = NC; Dtype = SLU_C; Mtype = GE.

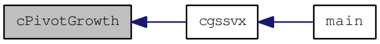
L        (output) SuperMatrix*
         The factor L from the factorization Pr*A=L*U; use compressed row
         subscripts storage for supernodes, i.e., L has type:
         Stype = SC; Dtype = SLU_C; Mtype = TRLU.

U        (output) SuperMatrix*
   The factor U from the factorization Pr*A*Pc=L*U. Use column-wise
         storage scheme, i.e., U has types: Stype = NC;
         Dtype = SLU_C; Mtype = TRU.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.110.3.35 int cpivotL (const int *jcol*, const float *u*, int ∗ *usepr*, int ∗ *perm_r*, int ∗ *iperm_r*, int ∗ *iperm_c*, int ∗ *pivrow*, GlobalLU_t ∗ *Glu*, SuperLUStat_t ∗ *stat*)

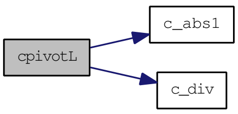
```
Purpose
=======
  Performs the numerical pivoting on the current column of L,
  and the CDIV operation.


  Pivot policy:
  (1) Compute thresh = u * max_(i>=j) abs(A_ij);
  (2) IF user specifies pivot row k and abs(A_kj) >= thresh THEN
          pivot row = k;
      ELSE IF abs(A_jj) >= thresh THEN
          pivot row = j;
      ELSE
          pivot row = m;

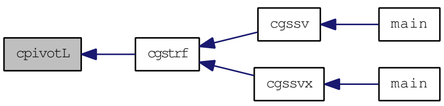

  Note: If you absolutely want to use a given pivot order, then set u=0.0.


  Return value: 0      success;
                i > 0  U(i,i) is exactly zero.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.110.3.36 void cPrint_CompCol_Matrix (char ∗, SuperMatrix ∗)

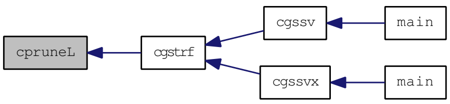
4.110.3.37 void cPrint_Dense_Matrix (char ∗, SuperMatrix ∗)

4.110.3.38 void cPrint_SuperNode_Matrix (char ∗, SuperMatrix ∗)

4.110.3.39 void cpruneL (const int *jcol*, const int ∗ *perm_r*, const int *pivrow*, const int *nseg*, const int ∗ *segrep*, const int ∗ *repfnz*, int ∗ *xprune*, GlobalLU_t ∗ *Glu*)

```
Purpose
=======
  Prunes the L-structure of supernodes whose L-structure
  contains the current pivot row "pivrow"
```

Here is the caller graph for this function:

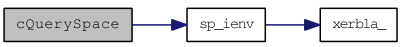


4.110.3.40 int cQuerySpace (SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, mem_usage_t ∗ *mem_usage*)

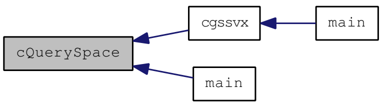
```
mem_usage consists of the following fields:
```

- `for_lu (float)`
  `The amount of space used in bytes for the L data structures.`
- `total_needed (float)`
  `The amount of space needed in bytes to perform factorization.`
- `expansions (int)`
  `Number of memory expansions during the LU factorization.`

Here is the call graph for this function:

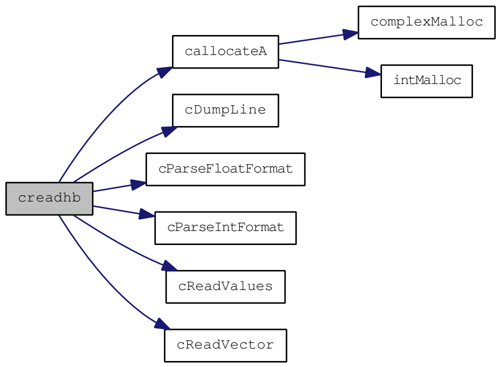


Here is the caller graph for this function:

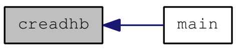


4.110.3.41 void creadhb (int ∗, int ∗, int ∗, complex ∗∗, int ∗∗, int ∗∗)

Here is the call graph for this function:



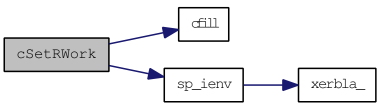
Here is the caller graph for this function:



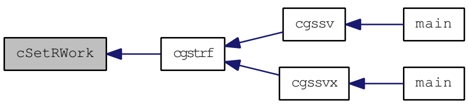
4.110.3.42 void creadmt (int ∗, int ∗, int ∗, complex ∗∗, int ∗∗, int ∗∗)

4.110.3.43 void cSetRWork (int, int, complex ∗, complex ∗∗, complex ∗∗)

Here is the call graph for this function:

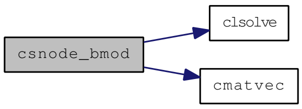


Here is the caller graph for this function:

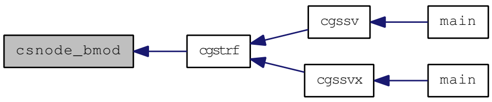


4.110.3.44 int csnode_bmod (const *int*, const *int*, const *int*, complex ∗, complex ∗, GlobalLU_t ∗, SuperLUStat_t ∗)

Here is the call graph for this function:



Here is the caller graph for this function:

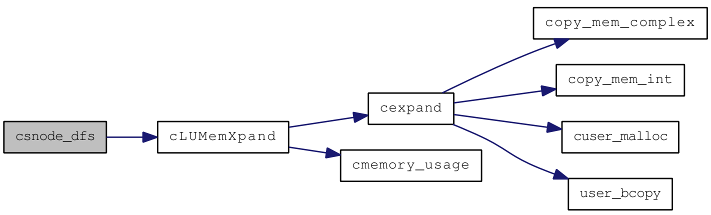


4.110.3.45 int csnode_dfs (const int *jcol*, const int *kcol*, const int ∗ *asub*, const int ∗ *xa_begin*, const int ∗ *xa_end*, int ∗ *xprune*, int ∗ *marker*, GlobalLU_t ∗ *Glu*)

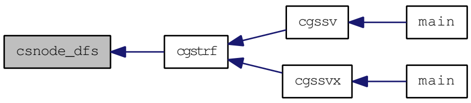
```
Purpose
=======
   csnode_dfs() - Determine the union of the row structures of those
   columns within the relaxed snode.
   Note: The relaxed snodes are leaves of the supernodal etree, therefore,
   the portion outside the rectangular supernode must be zero.

Return value
============
    0   success;
   >0   number of bytes allocated when run out of memory.
```

Here is the call graph for this function:

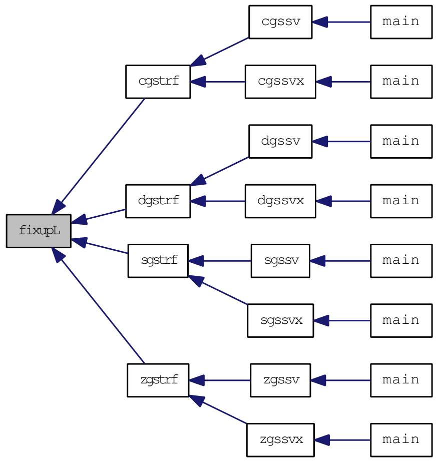


Here is the caller graph for this function:



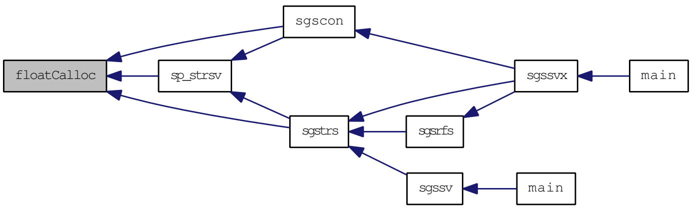
4.110.3.46 void fixupL (const *int*, const int ∗, GlobalLU_t ∗)

Here is the caller graph for this function:



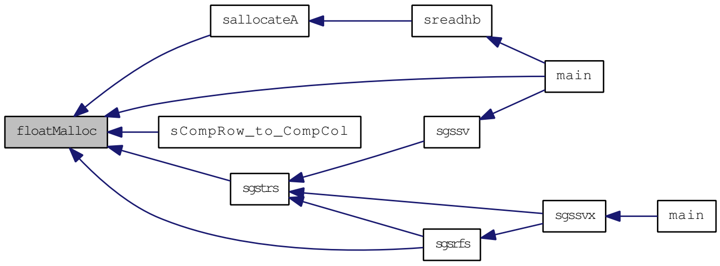
4.110.3.47 float∗ floatCalloc (int)

Here is the caller graph for this function:



4.110.3.48 float∗ floatMalloc (int)

Here is the caller graph for this function:



4.110.3.49 void print_lu_col (char ∗, int, int, int ∗, GlobalLU_t ∗)

4.110.3.50 void PrintPerf (SuperMatrix ∗, SuperMatrix ∗, mem_usage_t ∗, complex, complex, complex ∗, complex ∗, char ∗)

4.110.3.51 int sp_cgemm (char ∗ *transa*, char ∗ *transb*, int *m*, int *n*, int *k*, complex *alpha*, SuperMatrix ∗ *A*, complex ∗ *b*, int *ldb*, complex *beta*, complex ∗ *c*, int *ldc*)

```
Purpose
  =======

  sp_c performs one of the matrix-matrix operations

     C := alpha*op( A )*op( B ) + beta*C,

  where  op( X ) is one of

     op( X ) = X   or   op( X ) = X'   or   op( X ) = conjg( X' ),

  alpha and beta are scalars, and A, B and C are matrices, with op( A )
  an m by k matrix,  op( B )  a  k by n matrix and  C an m by n matrix.

  Parameters
  ==========

  TRANSA - (input) char*
           On entry, TRANSA specifies the form of op( A ) to be used in
           the matrix multiplication as follows:
              TRANSA = 'N' or 'n',  op( A ) = A.
              TRANSA = 'T' or 't',  op( A ) = A'.
              TRANSA = 'C' or 'c',  op( A ) = conjg( A' ).
           Unchanged on exit.

  TRANSB - (input) char*
           On entry, TRANSB specifies the form of op( B ) to be used in
           the matrix multiplication as follows:
              TRANSB = 'N' or 'n',  op( B ) = B.
              TRANSB = 'T' or 't',  op( B ) = B'.
              TRANSB = 'C' or 'c',  op( B ) = conjg( B' ).
           Unchanged on exit.

  M      - (input) int
           On entry,  M  specifies  the number of rows of the matrix
    op( A ) and of the matrix C.  M must be at least zero.
    Unchanged on exit.

  N      - (input) int
           On entry,  N specifies the number of columns of the matrix
    op( B ) and the number of columns of the matrix C. N must be
    at least zero.
    Unchanged on exit.

  K      - (input) int
           On entry, K specifies the number of columns of the matrix
    op( A ) and the number of rows of the matrix op( B ). K must
    be at least  zero.
          Unchanged on exit.
```

```
ALPHA  - (input) complex
         On entry, ALPHA specifies the scalar alpha.

A      - (input) SuperMatrix*
         Matrix A with a sparse format, of dimension (A->nrow, A->ncol).
         Currently, the type of A can be:
             Stype = NC or NCP; Dtype = SLU_C; Mtype = GE.
         In the future, more general A can be handled.

B      - COMPLEX PRECISION array of DIMENSION ( LDB, kb ), where kb is
         n when TRANSB = 'N' or 'n',  and is  k otherwise.
         Before entry with  TRANSB = 'N' or 'n',  the leading k by n
         part of the array B must contain the matrix B, otherwise
         the leading n by k part of the array B must contain the
         matrix B.
         Unchanged on exit.

LDB    - (input) int
         On entry, LDB specifies the first dimension of B as declared
         in the calling (sub) program. LDB must be at least max( 1, n ).
         Unchanged on exit.

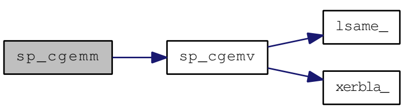
BETA   - (input) complex
         On entry, BETA specifies the scalar beta. When BETA is
         supplied as zero then C need not be set on input.

C      - COMPLEX PRECISION array of DIMENSION ( LDC, n ).
         Before entry, the leading m by n part of the array C must
         contain the matrix C,  except when beta is zero, in which
         case C need not be set on entry.
         On exit, the array C is overwritten by the m by n matrix
  ( alpha*op( A )*B + beta*C ).

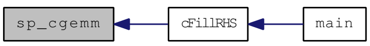
LDC    - (input) int
         On entry, LDC specifies the first dimension of C as declared
         in the calling (sub)program. LDC must be at least max(1,m).
         Unchanged on exit.

==== Sparse Level 3 Blas routine.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.110.3.52 int sp_cgemv (char * *trans*, complex *alpha*, SuperMatrix * *A*, complex * *x*, int *incx*, complex *beta*, complex * *y*, int *incy*)

```
Purpose
=======

sp_cgemv()  performs one of the matrix-vector operations
   y := alpha*A*x + beta*y,   or   y := alpha*A'*x + beta*y,
where alpha and beta are scalars, x and y are vectors and A is a
sparse A->nrow by A->ncol matrix.

Parameters
==========

TRANS  - (input) char*
         On entry, TRANS specifies the operation to be performed as
         follows:
            TRANS = 'N' or 'n'   y := alpha*A*x + beta*y.
            TRANS = 'T' or 't'   y := alpha*A'*x + beta*y.
            TRANS = 'C' or 'c'   y := alpha*A'*x + beta*y.

ALPHA  - (input) complex
         On entry, ALPHA specifies the scalar alpha.

A      - (input) SuperMatrix*
         Before entry, the leading m by n part of the array A must
         contain the matrix of coefficients.

X      - (input) complex*, array of DIMENSION at least
         ( 1 + ( n - 1 )*abs( INCX ) ) when TRANS = 'N' or 'n'
        and at least
         ( 1 + ( m - 1 )*abs( INCX ) ) otherwise.
         Before entry, the incremented array X must contain the
         vector x.

INCX   - (input) int
         On entry, INCX specifies the increment for the elements of
         X. INCX must not be zero.

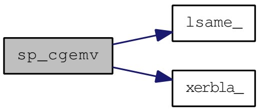
BETA   - (input) complex
         On entry, BETA specifies the scalar beta. When BETA is
         supplied as zero then Y need not be set on input.

Y      - (output) complex*,  array of DIMENSION at least
         ( 1 + ( m - 1 )*abs( INCY ) ) when TRANS = 'N' or 'n'
         and at least
         ( 1 + ( n - 1 )*abs( INCY ) ) otherwise.
         Before entry with BETA non-zero, the incremented array Y
         must contain the vector y. On exit, Y is overwritten by the
         updated vector y.

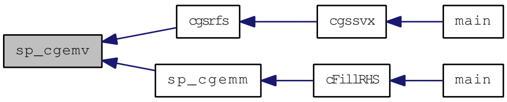
INCY   - (input) int
         On entry, INCY specifies the increment for the elements of
         Y. INCY must not be zero.
```

```
==== Sparse Level 2 Blas routine.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.110.3.53 int sp_ctrsv (char ∗ *uplo*, char ∗ *trans*, char ∗ *diag*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, complex ∗ *x*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======

sp_ctrsv() solves one of the systems of equations
    A*x = b,    or    A'*x = b,
where b and x are n element vectors and A is a sparse unit , or
non-unit, upper or lower triangular matrix.
No test for singularity or near-singularity is included in this
routine. Such tests must be performed before calling this routine.

Parameters
==========

uplo   - (input) char*
         On entry, uplo specifies whether the matrix is an upper or
          lower triangular matrix as follows:
             uplo = 'U' or 'u'   A is an upper triangular matrix.
             uplo = 'L' or 'l'   A is a lower triangular matrix.

trans  - (input) char*
          On entry, trans specifies the equations to be solved as
          follows:
             trans = 'N' or 'n'   A*x = b.
             trans = 'T' or 't'   A'*x = b.
             trans = 'C' or 'c'   A^H*x = b.

diag   - (input) char*
          On entry, diag specifies whether or not A is unit
          triangular as follows:
             diag = 'U' or 'u'   A is assumed to be unit triangular.
             diag = 'N' or 'n'   A is not assumed to be unit
                                 triangular.
```

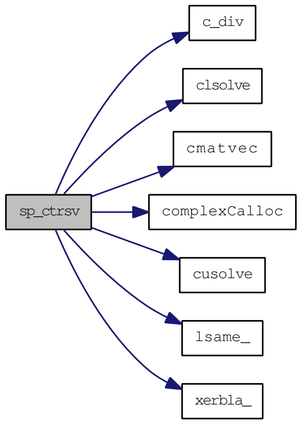
```
L       - (input) SuperMatrix*
    The factor L from the factorization Pr*A*Pc=L*U. Use
          compressed row subscripts storage for supernodes,
          i.e., L has types: Stype = SC, Dtype = SLU_C, Mtype = TRLU.

U       - (input) SuperMatrix*
     The factor U from the factorization Pr*A*Pc=L*U.
     U has types: Stype = NC, Dtype = SLU_C, Mtype = TRU.

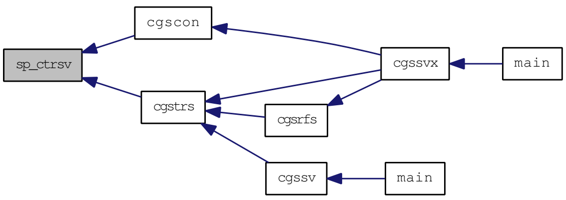
x       - (input/output) complex*
          Before entry, the incremented array X must contain the n
          element right-hand side vector b. On exit, X is overwritten
          with the solution vector x.

info    - (output) int*
          If *info = -i, the i-th argument had an illegal value.
```

Here is the call graph for this function:



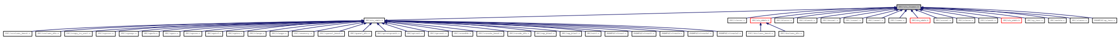
Here is the caller graph for this function:



4.111 SRC/slu_Cnames.h File Reference

Macros defining how C routines will be called.

This graph shows which files directly or indirectly include this file:



Defines

- #define ADD_ 0
- #define ADD__ 1
- #define NOCHANGE 2
- #define UPCASE 3
- #define C_CALL 4
- #define F77_CALL_C ADD_

4.111.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 1, 1997


These macros define how C routines will be called.  ADD_ assumes that
they will be called by fortran, which expects C routines to have an
underscore postfixed to the name (Suns, and the Intel expect this).
NOCHANGE indicates that fortran will be calling, and that it expects
the name called by fortran to be identical to that compiled by the C
(RS6K's do this).  UPCASE says it expects C routines called by fortran
to be in all upcase (CRAY wants this).
```

4.111.2 Define Documentation

4.111.2.1 #define ADD_ 0

4.111.2.2 #define ADD__ 1

4.111.2.3 #define C_CALL 4

4.111.2.4 #define F77_CALL_C ADD_

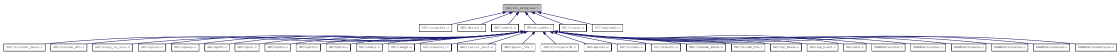
4.111.2.5 #define NOCHANGE 2

4.111.2.6 #define UPCASE 3

4.112 SRC/slu_dcomplex.h File Reference

Header file for complex operations.

This graph shows which files directly or indirectly include this file:



Data Structures

- struct doublecomplex

Defines

- #define z_add(c, a, b)

    Complex Addition c = a + b.

- #define z_sub(c, a, b)

    Complex Subtraction c = a - b.

- #define zd_mult(c, a, b)

    Complex-Double Multiplication.

- #define zz_mult(c, a, b)

    Complex-Complex Multiplication.

- #define zz_conj(a, b)
- #define z_eq(a, b) ((a) → r == (b) → r && (a) → i == (b) → i)

    Complex equality testing.

Functions

- void z_div (doublecomplex ∗, doublecomplex ∗, doublecomplex ∗)

    Complex Division c = a/b.

- double z_abs (doublecomplex ∗)

    Returns sqrt(z.r^2 + z.i^2).

- double z_abs1 (doublecomplex ∗)

    Approximates the abs. Returns abs(z.r) + abs(z.i).

- void z_exp (doublecomplex ∗, doublecomplex ∗)

    Return the exponentiation.

- void d_cnjg (doublecomplex ∗r, doublecomplex ∗z)

    Return the complex conjugate.

- double d_imag (doublecomplex *)

  Return the imaginary part.

4.112.1 Detailed Description

```
 -- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997

Contains definitions for various complex operations.
This header file is to be included in source files z*.c
```

4.112.2 Define Documentation

4.112.2.1 #define z_add(c, a, b)

Value:

```
{ (c)->r = (a)->r + (b)->r; \
       (c)->i = (a)->i + (b)->i; }
```

4.112.2.2 #define z_eq(a, b) ((a) $\rightarrow$ r == (b) $\rightarrow$ r && (a) $\rightarrow$ i == (b) $\rightarrow$ i)

4.112.2.3 #define z_sub(c, a, b)

Value:

```
{ (c)->r = (a)->r - (b)->r; \
       (c)->i = (a)->i - (b)->i; }
```

4.112.2.4 #define zd_mult(c, a, b)

Value:

```
{ (c)->r = (a)->r * (b); \
                           (c)->i = (a)->i * (b); }
```

4.112.2.5 #define zz_conj(a, b)

Value:

```
{ \
        (a)->r = (b)->r; \
        (a)->i = -((b)->i); \
    }
```

4.112.2.6 #define zz_mult(c, a, b)

Value:

```
{ \
  double cr, ci; \
      cr = (a)->r * (b)->r - (a)->i * (b)->i; \
      ci = (a)->i * (b)->r + (a)->r * (b)->i; \
      (c)->r = cr; \
      (c)->i = ci; \
    }
```

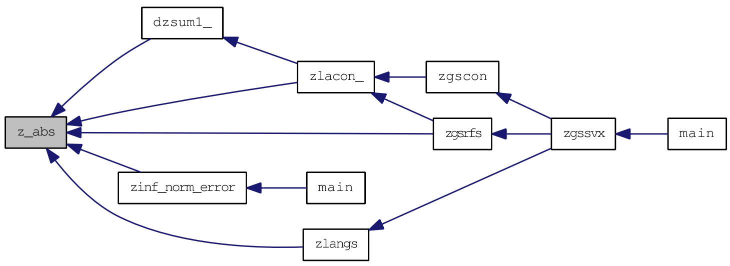
4.112.3 Function Documentation

4.112.3.1 void d_cnjg (doublecomplex ∗ *r*, doublecomplex ∗ *z*)

4.112.3.2 double d_imag (doublecomplex ∗)

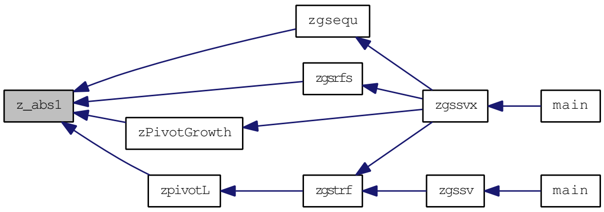
4.112.3.3 double z_abs (doublecomplex ∗)

Here is the caller graph for this function:



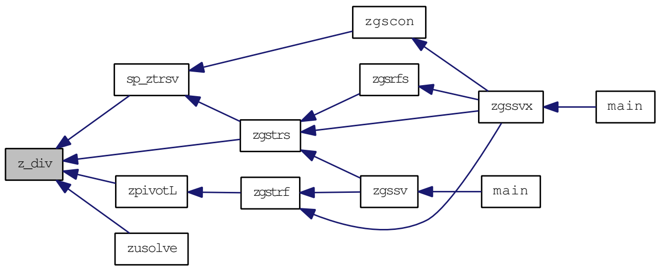
4.112.3.4 double z_abs1 (doublecomplex ∗)

Here is the caller graph for this function:



4.112.3.5 void z_div (doublecomplex ∗, doublecomplex ∗, doublecomplex ∗)

Here is the caller graph for this function:



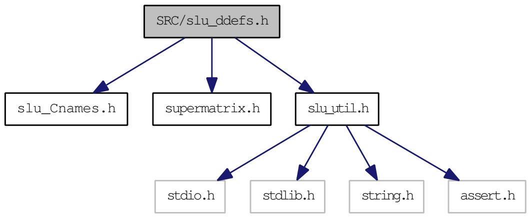
4.112.3.6 void z_exp (doublecomplex ∗, doublecomplex ∗)

4.113 SRC/slu_ddefs.h File Reference

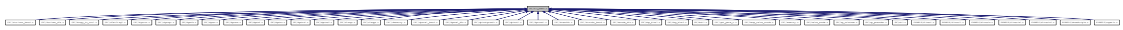
Header file for real operations.

```
#include "slu_Cnames.h"
#include "supermatrix.h"
#include "slu_util.h"
```

Include dependency graph for slu_ddefs.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct GlobalLU_t

Typedefs

- typedef int int_t

Functions

- void dgssv (superlu_options_t ∗, SuperMatrix ∗, int ∗, int ∗, SuperMatrix ∗, SuperMatrix ∗, SuperMatrix ∗, SuperLUStat_t ∗, int ∗)

  Driver routines.

- void dgssvx (superlu_options_t ∗, SuperMatrix ∗, int ∗, int ∗, int ∗, char ∗, double ∗, double ∗, SuperMatrix ∗, SuperMatrix ∗, void ∗, int, SuperMatrix ∗, SuperMatrix ∗, double ∗, double ∗, double ∗, double ∗, mem_usage_t ∗, SuperLUStat_t ∗, int ∗)
- void dCreate_CompCol_Matrix (SuperMatrix ∗, int, int, int, double ∗, int ∗, int ∗, Stype_t, Dtype_t, Mtype_t)

  Supernodal LU factor related.

- void dCreate_CompRow_Matrix (SuperMatrix ∗, int, int, int, double ∗, int ∗, int ∗, Stype_t, Dtype_t, Mtype_t)
- void dCopy_CompCol_Matrix (SuperMatrix ∗, SuperMatrix ∗)

  Copy matrix A into matrix B.

- void dCreate_Dense_Matrix (SuperMatrix ∗, int, int, double ∗, int, Stype_t, Dtype_t, Mtype_t)
- void dCreate_SuperNode_Matrix (SuperMatrix ∗, int, int, int, double ∗, int ∗, int ∗, int ∗, int ∗, int ∗, Stype_t, Dtype_t, Mtype_t)
- void dCopy_Dense_Matrix (int, int, double ∗, int, double ∗, int)
- void countnz (const int, int ∗, int ∗, int ∗, GlobalLU_t ∗)

  Count the total number of nonzeros in factors L and U, and in the symmetrically reduced L.

- void fixupL (const int, const int ∗, GlobalLU_t ∗)

  Fix up the data storage lsub for L-subscripts. It removes the subscript sets for structural pruning, and applies permuation to the remaining subscripts.

- void dallocateA (int, int, double ∗∗, int ∗∗, int ∗∗)

  Allocate storage for original matrix A.

- void dgstrf (superlu_options_t ∗, SuperMatrix ∗, double, int, int, int ∗, void ∗, int, int ∗, int ∗, SuperMatrix ∗, SuperMatrix ∗, SuperLUStat_t ∗, int ∗)
- int dsnode_dfs (const int, const int, const int ∗, const int ∗, const int ∗, int ∗, int ∗, GlobalLU_t ∗)
- int dsnode_bmod (const int, const int, const int, double ∗, double ∗, GlobalLU_t ∗, SuperLUStat_t ∗)

  Performs numeric block updates within the relaxed snode.

- void dpanel_dfs (const int, const int, const int, SuperMatrix ∗, int ∗, int ∗, double ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, GlobalLU_t ∗)
- void dpanel_bmod (const int, const int, const int, const int, double ∗, double ∗, int ∗, int ∗, GlobalLU_t ∗, SuperLUStat_t ∗)
- int dcolumn_dfs (const int, const int, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, GlobalLU_t ∗)
- int dcolumn_bmod (const int, const int, double ∗, double ∗, int ∗, int ∗, int, GlobalLU_t ∗, SuperLUStat_t ∗)
- int dcopy_to_ucol (int, int, int ∗, int ∗, int ∗, double ∗, GlobalLU_t ∗)
- int dpivotL (const int, const double, int ∗, int ∗, int ∗, int ∗, int ∗, GlobalLU_t ∗, SuperLUStat_t ∗)
- void dpruneL (const int, const int ∗, const int, const int, const int ∗, const int ∗, int ∗, GlobalLU_t ∗)
- void dreadmt (int ∗, int ∗, int ∗, double ∗∗, int ∗∗, int ∗∗)
- void dGenXtrue (int, int, double ∗, int)
- void dFillRHS (trans_t, int, double ∗, int, SuperMatrix ∗, SuperMatrix ∗)

  Let rhs[i] = sum of i-th row of A, so the solution vector is all 1's.

- void dgstrs (trans_t, SuperMatrix ∗, SuperMatrix ∗, int ∗, int ∗, SuperMatrix ∗, SuperLUStat_t ∗, int ∗)
- void dgsequ (SuperMatrix ∗, double ∗, double ∗, double ∗, double ∗, double ∗, int ∗)

  Driver related.

- void dlaqgs (SuperMatrix ∗, double ∗, double ∗, double, double, double, char ∗)
- void dgscon (char ∗, SuperMatrix ∗, SuperMatrix ∗, double, double ∗, SuperLUStat_t ∗, int ∗)
- double dPivotGrowth (int, SuperMatrix ∗, int ∗, SuperMatrix ∗, SuperMatrix ∗)
- void dgsrfs (trans_t, SuperMatrix ∗, SuperMatrix ∗, SuperMatrix ∗, int ∗, int ∗, char ∗, double ∗, double ∗, SuperMatrix ∗, SuperMatrix ∗, double ∗, double ∗, SuperLUStat_t ∗, int ∗)
- int sp_dtrsv (char ∗, char ∗, char ∗, SuperMatrix ∗, SuperMatrix ∗, double ∗, SuperLUStat_t ∗, int ∗)

  Solves one of the systems of equations A∗x = b, or A'∗x = b.

- int sp_dgemv (char ∗, double, SuperMatrix ∗, double ∗, int, double, double ∗, int)

  Performs one of the matrix-vector operations y := alpha∗A∗x + beta∗y, or y := alpha∗A'∗x + beta∗y,.

- int sp_dgemm (char ∗, char ∗, int, int, int, double, SuperMatrix ∗, double ∗, int, double, double ∗, int)
- int dLUMemInit (fact_t, void ∗, int, int, int, int, int, SuperMatrix ∗, SuperMatrix ∗, GlobalLU_t ∗, int ∗∗, double ∗∗)

  Memory-related.

- void dSetRWork (int, int, double ∗, double ∗∗, double ∗∗)

  Set up pointers for real working arrays.

- void dLUWorkFree (int ∗, double ∗, GlobalLU_t ∗)

  Free the working storage used by factor routines.

- int dLUMemXpand (int, int, MemType, int ∗, GlobalLU_t ∗)

  Expand the data structures for L and U during the factorization.

- double ∗ doubleMalloc (int)
- double ∗ doubleCalloc (int)
- int dmemory_usage (const int, const int, const int, const int)
- int dQuerySpace (SuperMatrix ∗, SuperMatrix ∗, mem_usage_t ∗)
- void dreadhb (int ∗, int ∗, int ∗, double ∗∗, int ∗∗, int ∗∗)

  Auxiliary routines.

- void dCompRow_to_CompCol (int, int, int, double ∗, int ∗, int ∗, double ∗∗, int ∗∗, int ∗∗)

  Convert a row compressed storage into a column compressed storage.

- void dfill (double ∗, int, double)

  Fills a double precision array with a given value.

- void dinf_norm_error (int, SuperMatrix ∗, double ∗)

  Check the inf-norm of the error vector.

- void PrintPerf (SuperMatrix ∗, SuperMatrix ∗, mem_usage_t ∗, double, double, double ∗, double ∗, char ∗)
- void dPrint_CompCol_Matrix (char ∗, SuperMatrix ∗)

  Routines for debugging.

- void dPrint_SuperNode_Matrix (char ∗, SuperMatrix ∗)
- void dPrint_Dense_Matrix (char ∗, SuperMatrix ∗)
- void print_lu_col (char ∗, int, int, int ∗, GlobalLU_t ∗)
- void check_tempv (int, double ∗)

4.113.1 Detailed Description

```
 -- SuperLU routine (version 3.0) --
 Univ. of California Berkeley, Xerox Palo Alto Research Center,
 and Lawrence Berkeley National Lab.
 October 15, 2003

 Global data structures used in LU factorization -

   nsuper: supernodes = nsuper + 1, numbered [0, nsuper].
   (xsup,supno): supno[i] is the supernode no to which i belongs;
xsup(s) points to the beginning of the s-th supernode.
e.g.   supno 0 1 2 2 3 3 3 4 4 4 4 4   (n=12)
        xsup 0 1 2 4 7 12
Note: dfs will be performed on supernode rep. relative to the new
      row pivoting ordering

   (xlsub,lsub): lsub[*] contains the compressed subscript of
rectangular supernodes; xlsub[j] points to the starting
location of the j-th column in lsub[*]. Note that xlsub
is indexed by column.
Storage: original row subscripts

      During the course of sparse LU factorization, we also use
(xlsub,lsub) for the purpose of symmetric pruning. For each
supernode {s,s+1,...,t=s+r} with first column s and last
column t, the subscript set
lsub[j], j=xlsub[s], .., xlsub[s+1]-1
is the structure of column s (i.e. structure of this supernode).
It is used for the storage of numerical values.
Furthermore,
lsub[j], j=xlsub[t], .., xlsub[t+1]-1
is the structure of the last column t of this supernode.
It is for the purpose of symmetric pruning. Therefore, the
structural subscripts can be rearranged without making physical
interchanges among the numerical values.

However, if the supernode has only one column, then we
only keep one set of subscripts. For any subscript interchange
performed, similar interchange must be done on the numerical
values.

The last column structures (for pruning) will be removed
after the numercial LU factorization phase.

   (xlusup,lusup): lusup[*] contains the numerical values of the
rectangular supernodes; xlusup[j] points to the starting
location of the j-th column in storage vector lusup[*]
Note: xlusup is indexed by column.
Each rectangular supernode is stored by column-major
scheme, consistent with Fortran 2-dim array storage.

   (xusub,ucol,usub): ucol[*] stores the numerical values of
U-columns outside the rectangular supernodes. The row
subscript of nonzero ucol[k] is stored in usub[k].
xusub[i] points to the starting location of column i in ucol.
Storage: new row subscripts; that is subscripts of PA.
```

4.113.2 Typedef Documentation

4.113.2.1 typedef int int_t

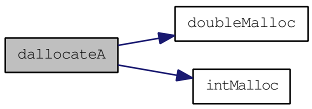
4.113.3 Function Documentation

4.113.3.1 void check_tempv (int, double ∗)

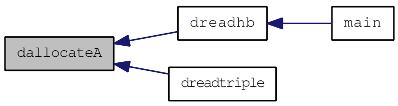
4.113.3.2 void countnz (const *int*, int ∗, int ∗, int ∗, GlobalLU_t ∗)

4.113.3.3 void dallocateA (int, int, double ∗∗, int ∗∗, int ∗∗)

Here is the call graph for this function:



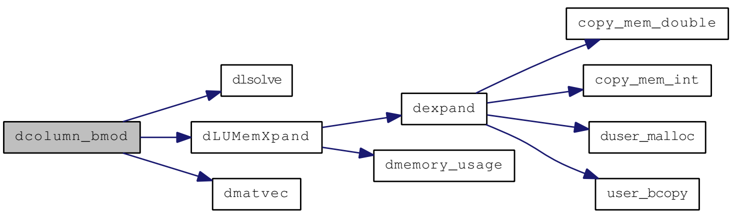
Here is the caller graph for this function:



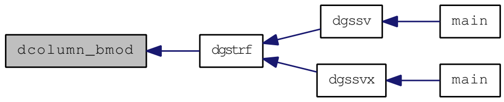
4.113.3.4 int dcolumn_bmod (const int *jcol*, const int *nseg*, double ∗ *dense*, double ∗ *tempv*, int ∗ *segrep*, int ∗ *repfnz*, int *fpanelc*, GlobalLU_t ∗ *Glu*, SuperLUStat_t ∗ *stat*)

```
Purpose:
========
Performs numeric block updates (sup-col) in topological order.
It features: col-col, 2cols-col, 3cols-col, and sup-col updates.
Special processing on the supernodal portion of L[*,j]
Return value:   0 - successful return
              > 0 - number of bytes allocated when run out of space
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.113.3.5 int dcolumn_dfs (const int *m*, const int *jcol*, int * *perm_r*, int * *nseg*, int * *lsub_col*, int * *segrep*, int * *repfnz*, int * *xprune*, int * *marker*, int * *parent*, int * *xplore*, GlobalLU_t * *Glu*)

```
 Purpose
 =======
   "column_dfs" performs a symbolic factorization on column jcol, and
   decide the supernode boundary.

   This routine does not use numeric values, but only use the RHS
   row indices to start the dfs.

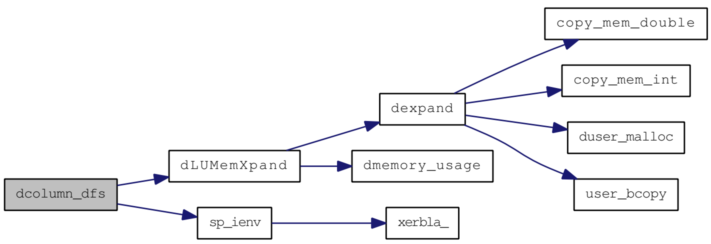
   A supernode representative is the last column of a supernode.
   The nonzeros in U[*,j] are segments that end at supernodal
   representatives. The routine returns a list of such supernodal
   representatives in topological order of the dfs that generates them.
   The location of the first nonzero in each such supernodal segment
   (supernodal entry location) is also returned.

 Local parameters
 ================
   nseg: no of segments in current U[*,j]
   jsuper: jsuper=EMPTY if column j does not belong to the same
supernode as j-1. Otherwise, jsuper=nsuper.

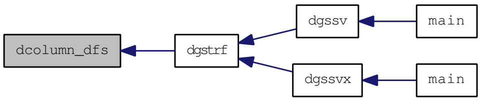
   marker2: A-row --> A-row/col (0/1)
   repfnz: SuperA-col --> PA-row
   parent: SuperA-col --> SuperA-col
   xplore: SuperA-col --> index to L-structure

 Return value
 ============
     0  success;
   > 0  number of bytes allocated when run out of space.
```

Here is the call graph for this function:

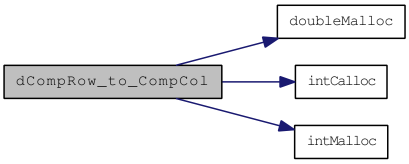


Here is the caller graph for this function:



4.113.3.6 void dCompRow_to_CompCol (int, int, int, double ∗, int ∗, int ∗, double ∗∗, int ∗∗, int ∗∗)

Here is the call graph for this function:



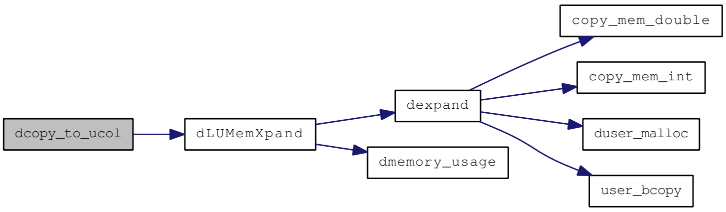
4.113.3.7 void dCopy_CompCol_Matrix (SuperMatrix ∗, SuperMatrix ∗)

4.113.3.8 void dCopy_Dense_Matrix (int, int, double ∗, int, double ∗, int)

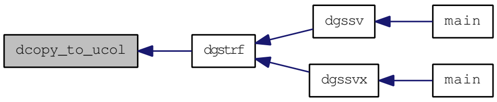
Copies a two-dimensional matrix X to another matrix Y.

4.113.3.9 int dcopy_to_ucol (int, int, int ∗, int ∗, int ∗, double ∗, GlobalLU_t ∗)

Here is the call graph for this function:

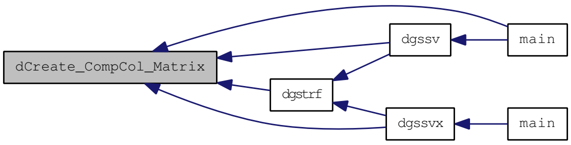


Here is the caller graph for this function:



4.113.3.10 void dCreate_CompCol_Matrix (SuperMatrix ∗, int, int, int, double ∗, int ∗, int ∗, Stype_t, Dtype_t, Mtype_t)

Here is the caller graph for this function:



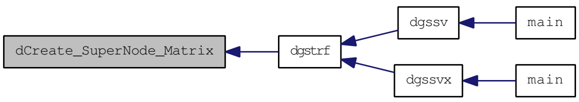
4.113.3.11 void dCreate_CompRow_Matrix (SuperMatrix ∗, int, int, int, double ∗, int ∗, int ∗, Stype_t, Dtype_t, Mtype_t)

4.113.3.12 void dCreate_Dense_Matrix (SuperMatrix ∗, int, int, double ∗, int, Stype_t, Dtype_t, Mtype_t)

Here is the caller graph for this function:

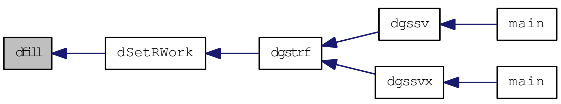
4.113.3.13 void dCreate_SuperNode_Matrix (SuperMatrix ∗, int, int, int, double ∗, int ∗, int ∗, int ∗, int ∗, int ∗, Stype_t, Dtype_t, Mtype_t)

Here is the caller graph for this function:



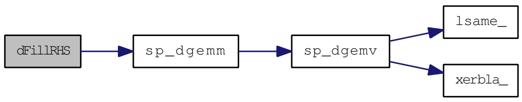
4.113.3.14 void dfill (double ∗, int, double)

Here is the caller graph for this function:

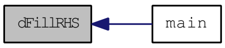


4.113.3.15 void dFillRHS (trans_t, int, double *, int, SuperMatrix *, SuperMatrix *)

Here is the call graph for this function:

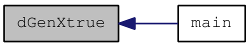


Here is the caller graph for this function:



4.113.3.16 void dGenXtrue (int, int, double *, int)

Here is the caller graph for this function:



4.113.3.17 void dgscon (char * *norm*, SuperMatrix * *L*, SuperMatrix * *U*, double *anorm*, double * *rcond*, SuperLUStat_t * *stat*, int * *info*)

```
Purpose
=======

DGSCON estimates the reciprocal of the condition number of a general
real matrix A, in either the 1-norm or the infinity-norm, using
the LU factorization computed by DGETRF.   *

An estimate is obtained for norm(inv(A)), and the reciprocal of the
condition number is computed as
   RCOND = 1 / ( norm(A) * norm(inv(A)) ).

See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

 NORM    (input) char*
         Specifies whether the 1-norm condition number or the
         infinity-norm condition number is required:
         = '1' or 'O':  1-norm;
         = 'I':         Infinity-norm.

 L       (input) SuperMatrix*
         The factor L from the factorization Pr*A*Pc=L*U as computed by
         dgstrf(). Use compressed row subscripts storage for supernodes,
         i.e., L has types: Stype = SLU_SC, Dtype = SLU_D, Mtype = SLU_TRLU.
```

```
U       (input) SuperMatrix*
        The factor U from the factorization Pr*A*Pc=L*U as computed by
        dgstrf(). Use column-wise storage scheme, i.e., U has types:
        Stype = SLU_NC, Dtype = SLU_D, Mtype = SLU_TRU.

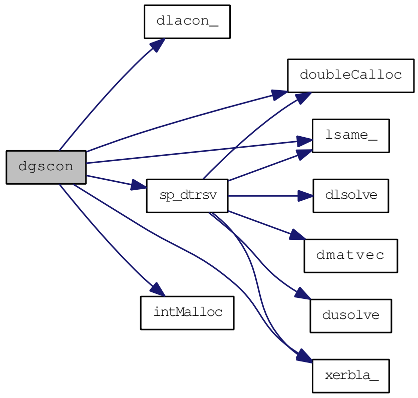
ANORM   (input) double
        If NORM = '1' or 'O', the 1-norm of the original matrix A.
        If NORM = 'I', the infinity-norm of the original matrix A.

RCOND   (output) double*
       The reciprocal of the condition number of the matrix A,
       computed as RCOND = 1/(norm(A) * norm(inv(A))).

INFO    (output) int*
       = 0:  successful exit
       < 0:  if INFO = -i, the i-th argument had an illegal value

=====================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.113.3.18 void dgsequ (SuperMatrix ∗ *A*, double ∗ *r*, double ∗ *c*, double ∗ *rowcnd*, double ∗ *colcnd*, double ∗ *amax*, int ∗ *info*)

```
Purpose
  =======

  DGSEQU computes row and column scalings intended to equilibrate an
  M-by-N sparse matrix A and reduce its condition number. R returns the row
  scale factors and C the column scale factors, chosen to try to make
  the largest element in each row and column of the matrix B with
  elements B(i,j)=R(i)*A(i,j)*C(j) have absolute value 1.
```

```
R(i) and C(j) are restricted to be between SMLNUM = smallest safe
number and BIGNUM = largest safe number.  Use of these scaling
factors is not guaranteed to reduce the condition number of A but
works well in practice.


See supermatrix.h for the definition of 'SuperMatrix' structure.


Arguments
=========


A       (input) SuperMatrix*
        The matrix of dimension (A->nrow, A->ncol) whose equilibration
        factors are to be computed. The type of A can be:
        Stype = SLU_NC; Dtype = SLU_D; Mtype = SLU_GE.


R       (output) double*, size A->nrow
        If INFO = 0 or INFO > M, R contains the row scale factors
        for A.


C       (output) double*, size A->ncol
        If INFO = 0,  C contains the column scale factors for A.


ROWCND  (output) double*
        If INFO = 0 or INFO > M, ROWCND contains the ratio of the
        smallest R(i) to the largest R(i).  If ROWCND >= 0.1 and
        AMAX is neither too large nor too small, it is not worth
        scaling by R.

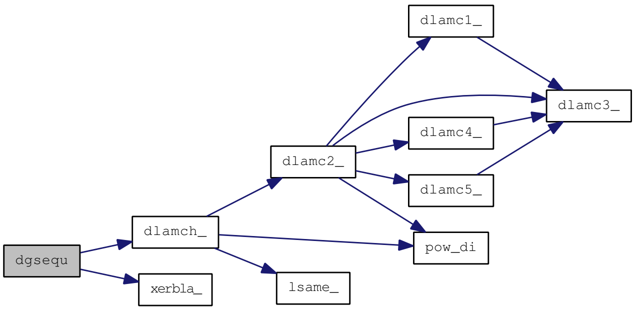

COLCND  (output) double*
        If INFO = 0, COLCND contains the ratio of the smallest
        C(i) to the largest C(i).  If COLCND >= 0.1, it is not
        worth scaling by C.


AMAX    (output) double*
        Absolute value of largest matrix element.  If AMAX is very
        close to overflow or very close to underflow, the matrix
        should be scaled.


INFO    (output) int*
        = 0:  successful exit
        < 0:  if INFO = -i, the i-th argument had an illegal value
        > 0:  if INFO = i,  and i is
              <= A->nrow:  the i-th row of A is exactly zero
              >  A->ncol:  the (i-M)-th column of A is exactly zero


=====================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.113.3.19 void dgsrfs (trans_t *trans*, SuperMatrix ∗ *A*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, int ∗ *perm_c*, int ∗ *perm_r*, char ∗ *equed*, double ∗ *R*, double ∗ *C*, SuperMatrix ∗ *B*, SuperMatrix ∗ *X*, double ∗ *ferr*, double ∗ *berr*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======

DGSRFS improves the computed solution to a system of linear
equations and provides error bounds and backward error estimates for
the solution.

If equilibration was performed, the system becomes:
        (diag(R)*A_original*diag(C)) * X = diag(R)*B_original.

See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

trans   (input) trans_t
        Specifies the form of the system of equations:
        = NOTRANS: A * X = B  (No transpose)
        = TRANS:   A'* X = B  (Transpose)
        = CONJ:    A**H * X = B  (Conjugate transpose)

A        (input) SuperMatrix*
         The original matrix A in the system, or the scaled A if
         equilibration was done. The type of A can be:
         Stype = SLU_NC, Dtype = SLU_D, Mtype = SLU_GE.

L        (input) SuperMatrix*
  The factor L from the factorization Pr*A*Pc=L*U. Use
         compressed row subscripts storage for supernodes,
         i.e., L has types: Stype = SLU_SC, Dtype = SLU_D, Mtype = SLU_TRLU.
```

```
U       (input) SuperMatrix*
        The factor U from the factorization Pr*A*Pc=L*U as computed by
        dgstrf(). Use column-wise storage scheme,
        i.e., U has types: Stype = SLU_NC, Dtype = SLU_D, Mtype = SLU_TRU.

perm_c  (input) int*, dimension (A->ncol)
  Column permutation vector, which defines the
        permutation matrix Pc; perm_c[i] = j means column i of A is
        in position j in A*Pc.

perm_r  (input) int*, dimension (A->nrow)
        Row permutation vector, which defines the permutation matrix Pr;
        perm_r[i] = j means row i of A is in position j in Pr*A.

equed   (input) Specifies the form of equilibration that was done.
        = 'N': No equilibration.
        = 'R': Row equilibration, i.e., A was premultiplied by diag(R).
        = 'C': Column equilibration, i.e., A was postmultiplied by
               diag(C).
        = 'B': Both row and column equilibration, i.e., A was replaced
               by diag(R)*A*diag(C).

R       (input) double*, dimension (A->nrow)
        The row scale factors for A.
        If equed = 'R' or 'B', A is premultiplied by diag(R).
        If equed = 'N' or 'C', R is not accessed.

C       (input) double*, dimension (A->ncol)
        The column scale factors for A.
        If equed = 'C' or 'B', A is postmultiplied by diag(C).
        If equed = 'N' or 'R', C is not accessed.

B       (input) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_D, Mtype = SLU_GE.
        The right hand side matrix B.
        if equed = 'R' or 'B', B is premultiplied by diag(R).

X       (input/output) SuperMatrix*
        X has types: Stype = SLU_DN, Dtype = SLU_D, Mtype = SLU_GE.
        On entry, the solution matrix X, as computed by dgstrs().
        On exit, the improved solution matrix X.
        if *equed = 'C' or 'B', X should be premultiplied by diag(C)
            in order to obtain the solution to the original system.

FERR    (output) double*, dimension (B->ncol)
        The estimated forward error bound for each solution vector
        X(j) (the j-th column of the solution matrix X).
        If XTRUE is the true solution corresponding to X(j), FERR(j)
        is an estimated upper bound for the magnitude of the largest
        element in (X(j) - XTRUE) divided by the magnitude of the
        largest element in X(j).  The estimate is as reliable as
        the estimate for RCOND, and is almost always a slight
        overestimate of the true error.

BERR    (output) double*, dimension (B->ncol)
        The componentwise relative backward error of each solution
        vector X(j) (i.e., the smallest relative change in
        any element of A or B that makes X(j) an exact solution).
```

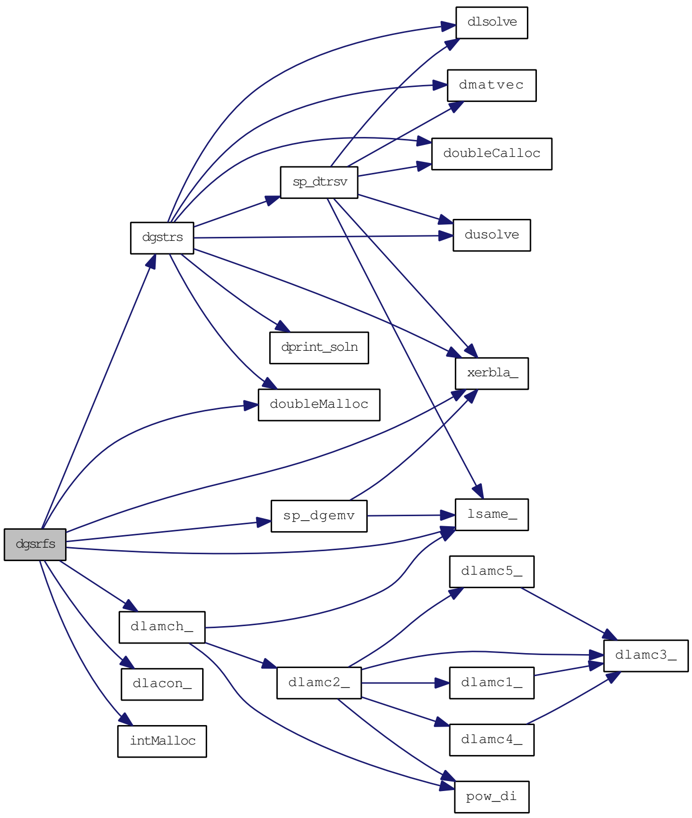
```
  stat     (output) SuperLUStat_t*
           Record the statistics on runtime and floating-point operation count.
           See util.h for the definition of 'SuperLUStat_t'.

  info    (output) int*
          = 0:  successful exit
           < 0:  if INFO = -i, the i-th argument had an illegal value

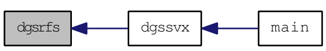
   Internal Parameters
   ===================

   ITMAX is the maximum number of steps of iterative refinement.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.113.3.20 void dgssv (superlu_options_t * *options*, SuperMatrix * *A*, int * *perm_c*, int * *perm_r*, SuperMatrix * *L*, SuperMatrix * *U*, SuperMatrix * *B*, SuperLUStat_t * *stat*, int * *info*)

```
Purpose
=======

DGSSV solves the system of linear equations A*X=B, using the
LU factorization from DGSTRF. It performs the following steps:

  1. If A is stored column-wise (A->Stype = SLU_NC):

     1.1. Permute the columns of A, forming A*Pc, where Pc
          is a permutation matrix. For more details of this step,
          see sp_preorder.c.

     1.2. Factor A as Pr*A*Pc=L*U with the permutation Pr determined
          by Gaussian elimination with partial pivoting.
          L is unit lower triangular with offdiagonal entries
          bounded by 1 in magnitude, and U is upper triangular.

     1.3. Solve the system of equations A*X=B using the factored
          form of A.

  2. If A is stored row-wise (A->Stype = SLU_NR), apply the
     above algorithm to the transpose of A:

     2.1. Permute columns of transpose(A) (rows of A),
          forming transpose(A)*Pc, where Pc is a permutation matrix.
          For more details of this step, see sp_preorder.c.

     2.2. Factor A as Pr*transpose(A)*Pc=L*U with the permutation Pr
          determined by Gaussian elimination with partial pivoting.
          L is unit lower triangular with offdiagonal entries
          bounded by 1 in magnitude, and U is upper triangular.

     2.3. Solve the system of equations A*X=B using the factored
          form of A.

  See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

options (input) superlu_options_t*
        The structure defines the input parameters to control
        how the LU decomposition will be performed and how the
        system will be solved.

A       (input) SuperMatrix*
        Matrix A in A*X=B, of dimension (A->nrow, A->ncol). The number
        of linear equations is A->nrow. Currently, the type of A can be:
        Stype = SLU_NC or SLU_NR; Dtype = SLU_D; Mtype = SLU_GE.
        In the future, more general A may be handled.
```

```
perm_c  (input/output) int*
        If A->Stype = SLU_NC, column permutation vector of size A->ncol
        which defines the permutation matrix Pc; perm_c[i] = j means
        column i of A is in position j in A*Pc.
        If A->Stype = SLU_NR, column permutation vector of size A->nrow
        which describes permutation of columns of transpose(A)
        (rows of A) as described above.

        If options->ColPerm = MY_PERMC or options->Fact = SamePattern or
           options->Fact = SamePattern_SameRowPerm, it is an input argument.
           On exit, perm_c may be overwritten by the product of the input
           perm_c and a permutation that postorders the elimination tree
           of Pc'*A'*A*Pc; perm_c is not changed if the elimination tree
           is already in postorder.
        Otherwise, it is an output argument.

perm_r  (input/output) int*
        If A->Stype = SLU_NC, row permutation vector of size A->nrow,
        which defines the permutation matrix Pr, and is determined
        by partial pivoting.  perm_r[i] = j means row i of A is in
        position j in Pr*A.
        If A->Stype = SLU_NR, permutation vector of size A->ncol, which
        determines permutation of rows of transpose(A)
        (columns of A) as described above.

        If options->RowPerm = MY_PERMR or
           options->Fact = SamePattern_SameRowPerm, perm_r is an
           input argument.
        otherwise it is an output argument.

L       (output) SuperMatrix*
        The factor L from the factorization
            Pr*A*Pc=L*U              (if A->Stype = SLU_NC) or
            Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
        Uses compressed row subscripts storage for supernodes, i.e.,
        L has types: Stype = SLU_SC, Dtype = SLU_D, Mtype = SLU_TRLU.

U       (output) SuperMatrix*
  The factor U from the factorization
            Pr*A*Pc=L*U              (if A->Stype = SLU_NC) or
            Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
        Uses column-wise storage scheme, i.e., U has types:
        Stype = SLU_NC, Dtype = SLU_D, Mtype = SLU_TRU.

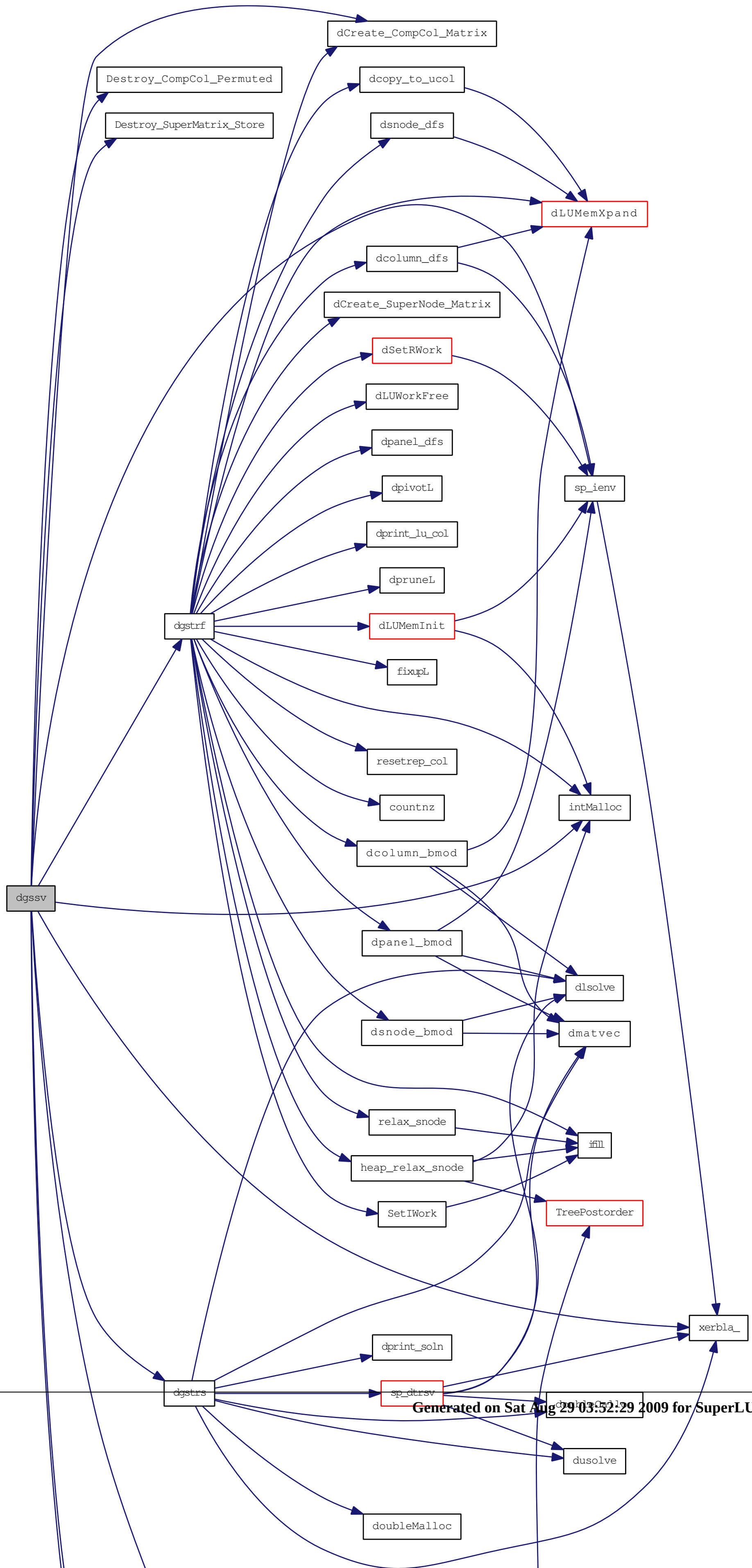
B       (input/output) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_D, Mtype = SLU_GE.
        On entry, the right hand side matrix.
        On exit, the solution matrix if info = 0;

stat   (output) SuperLUStat_t*
       Record the statistics on runtime and floating-point operation count.
       See util.h for the definition of 'SuperLUStat_t'.

info    (output) int*
  = 0: successful exit
        > 0: if info = i, and i is
```

```
<= A->ncol: U(i,i) is exactly zero. The factorization has
   been completed, but the factor U is exactly singular,
   so the solution could not be computed.
> A->ncol: number of bytes allocated when memory allocation
   failure occurred, plus A->ncol.
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.113.3.21 void dgssvx (superlu_options_t ∗ *options*, SuperMatrix ∗ *A*, int ∗ *perm_c*, int ∗ *perm_r*, int ∗ *etree*, char ∗ *equed*, double ∗ *R*, double ∗ *C*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, void ∗ *work*, int *lwork*, SuperMatrix ∗ *B*, SuperMatrix ∗ *X*, double ∗ *recip_pivot_growth*, double ∗ *rcond*, double ∗ *ferr*, double ∗ *berr*, mem_usage_t ∗ *mem_usage*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======

DGSSVX solves the system of linear equations A*X=B or A'*X=B, using
the LU factorization from dgstrf(). Error bounds on the solution and
a condition estimate are also provided. It performs the following steps:

  1. If A is stored column-wise (A->Stype = SLU_NC):

     1.1. If options->Equil = YES, scaling factors are computed to
          equilibrate the system:
          options->Trans = NOTRANS:
              diag(R)*A*diag(C) *inv(diag(C))*X = diag(R)*B
          options->Trans = TRANS:
              (diag(R)*A*diag(C))**T *inv(diag(R))*X = diag(C)*B
          options->Trans = CONJ:
              (diag(R)*A*diag(C))**H *inv(diag(R))*X = diag(C)*B
          Whether or not the system will be equilibrated depends on the
          scaling of the matrix A, but if equilibration is used, A is
          overwritten by diag(R)*A*diag(C) and B by diag(R)*B
          (if options->Trans=NOTRANS) or diag(C)*B (if options->Trans
          = TRANS or CONJ).

     1.2. Permute columns of A, forming A*Pc, where Pc is a permutation
          matrix that usually preserves sparsity.
          For more details of this step, see sp_preorder.c.

     1.3. If options->Fact != FACTORED, the LU decomposition is used to
          factor the matrix A (after equilibration if options->Equil = YES)
          as Pr*A*Pc = L*U, with Pr determined by partial pivoting.

     1.4. Compute the reciprocal pivot growth factor.

     1.5. If some U(i,i) = 0, so that U is exactly singular, then the
          routine returns with info = i. Otherwise, the factored form of
          A is used to estimate the condition number of the matrix A. If
          the reciprocal of the condition number is less than machine
          precision, info = A->ncol+1 is returned as a warning, but the
          routine still goes on to solve for X and computes error bounds
          as described below.
```

```
   1.6. The system of equations is solved for X using the factored form
        of A.

   1.7. If options->IterRefine != NOREFINE, iterative refinement is
        applied to improve the computed solution matrix and calculate
        error bounds and backward error estimates for it.

   1.8. If equilibration was used, the matrix X is premultiplied by
        diag(C) (if options->Trans = NOTRANS) or diag(R)
        (if options->Trans = TRANS or CONJ) so that it solves the
        original system before equilibration.

2. If A is stored row-wise (A->Stype = SLU_NR), apply the above algorithm
   to the transpose of A:

   2.1. If options->Equil = YES, scaling factors are computed to
        equilibrate the system:
        options->Trans = NOTRANS:
            diag(R)*A*diag(C) *inv(diag(C))*X = diag(R)*B
        options->Trans = TRANS:
            (diag(R)*A*diag(C))**T *inv(diag(R))*X = diag(C)*B
        options->Trans = CONJ:
            (diag(R)*A*diag(C))**H *inv(diag(R))*X = diag(C)*B
        Whether or not the system will be equilibrated depends on the
        scaling of the matrix A, but if equilibration is used, A' is
        overwritten by diag(R)*A'*diag(C) and B by diag(R)*B
        (if trans='N') or diag(C)*B (if trans = 'T' or 'C').

   2.2. Permute columns of transpose(A) (rows of A),
        forming transpose(A)*Pc, where Pc is a permutation matrix that
        usually preserves sparsity.
        For more details of this step, see sp_preorder.c.

   2.3. If options->Fact != FACTORED, the LU decomposition is used to
        factor the transpose(A) (after equilibration if
        options->Fact = YES) as Pr*transpose(A)*Pc = L*U with the
        permutation Pr determined by partial pivoting.

   2.4. Compute the reciprocal pivot growth factor.

   2.5. If some U(i,i) = 0, so that U is exactly singular, then the
        routine returns with info = i. Otherwise, the factored form
        of transpose(A) is used to estimate the condition number of the
        matrix A. If the reciprocal of the condition number
        is less than machine precision, info = A->nrow+1 is returned as
        a warning, but the routine still goes on to solve for X and
        computes error bounds as described below.

   2.6. The system of equations is solved for X using the factored form
        of transpose(A).

   2.7. If options->IterRefine != NOREFINE, iterative refinement is
        applied to improve the computed solution matrix and calculate
        error bounds and backward error estimates for it.
```

```
      2.8. If equilibration was used, the matrix X is premultiplied by
           diag(C) (if options->Trans = NOTRANS) or diag(R)
           (if options->Trans = TRANS or CONJ) so that it solves the
           original system before equilibration.

   See supermatrix.h for the definition of 'SuperMatrix' structure.

 Arguments
 =========

 options (input) superlu_options_t*
         The structure defines the input parameters to control
         how the LU decomposition will be performed and how the
         system will be solved.

 A       (input/output) SuperMatrix*
         Matrix A in A*X=B, of dimension (A->nrow, A->ncol). The number
         of the linear equations is A->nrow. Currently, the type of A can be:
         Stype = SLU_NC or SLU_NR, Dtype = SLU_D, Mtype = SLU_GE.
         In the future, more general A may be handled.

         On entry, If options->Fact = FACTORED and equed is not 'N',
         then A must have been equilibrated by the scaling factors in
         R and/or C.
         On exit, A is not modified if options->Equil = NO, or if
         options->Equil = YES but equed = 'N' on exit.
         Otherwise, if options->Equil = YES and equed is not 'N',
         A is scaled as follows:
         If A->Stype = SLU_NC:
           equed = 'R':  A := diag(R) * A
           equed = 'C':  A := A * diag(C)
           equed = 'B':  A := diag(R) * A * diag(C).
         If A->Stype = SLU_NR:
           equed = 'R':  transpose(A) := diag(R) * transpose(A)
           equed = 'C':  transpose(A) := transpose(A) * diag(C)
           equed = 'B':  transpose(A) := diag(R) * transpose(A) * diag(C).

 perm_c  (input/output) int*
   If A->Stype = SLU_NC, Column permutation vector of size A->ncol,
         which defines the permutation matrix Pc; perm_c[i] = j means
         column i of A is in position j in A*Pc.
         On exit, perm_c may be overwritten by the product of the input
         perm_c and a permutation that postorders the elimination tree
         of Pc'*A'*A*Pc; perm_c is not changed if the elimination tree
         is already in postorder.

         If A->Stype = SLU_NR, column permutation vector of size A->nrow,
         which describes permutation of columns of transpose(A)
         (rows of A) as described above.

 perm_r  (input/output) int*
         If A->Stype = SLU_NC, row permutation vector of size A->nrow,
         which defines the permutation matrix Pr, and is determined
         by partial pivoting.  perm_r[i] = j means row i of A is in
         position j in Pr*A.
```

```
        If A->Stype = SLU_NR, permutation vector of size A->ncol, which
        determines permutation of rows of transpose(A)
        (columns of A) as described above.

        If options->Fact = SamePattern_SameRowPerm, the pivoting routine
        will try to use the input perm_r, unless a certain threshold
        criterion is violated. In that case, perm_r is overwritten by a
        new permutation determined by partial pivoting or diagonal
        threshold pivoting.
        Otherwise, perm_r is output argument.

etree   (input/output) int*,  dimension (A->ncol)
        Elimination tree of Pc'*A'*A*Pc.
        If options->Fact != FACTORED and options->Fact != DOFACT,
        etree is an input argument, otherwise it is an output argument.
        Note: etree is a vector of parent pointers for a forest whose
        vertices are the integers 0 to A->ncol-1; etree[root]==A->ncol.

equed   (input/output) char*
        Specifies the form of equilibration that was done.
        = 'N': No equilibration.
        = 'R': Row equilibration, i.e., A was premultiplied by diag(R).
        = 'C': Column equilibration, i.e., A was postmultiplied by diag(C).
        = 'B': Both row and column equilibration, i.e., A was replaced
               by diag(R)*A*diag(C).
        If options->Fact = FACTORED, equed is an input argument,
        otherwise it is an output argument.

R       (input/output) double*, dimension (A->nrow)
        The row scale factors for A or transpose(A).
        If equed = 'R' or 'B', A (if A->Stype = SLU_NC) or transpose(A)
            (if A->Stype = SLU_NR) is multiplied on the left by diag(R).
        If equed = 'N' or 'C', R is not accessed.
        If options->Fact = FACTORED, R is an input argument,
            otherwise, R is output.
        If options->zFact = FACTORED and equed = 'R' or 'B', each element
            of R must be positive.

C       (input/output) double*, dimension (A->ncol)
        The column scale factors for A or transpose(A).
        If equed = 'C' or 'B', A (if A->Stype = SLU_NC) or transpose(A)
            (if A->Stype = SLU_NR) is multiplied on the right by diag(C).
        If equed = 'N' or 'R', C is not accessed.
        If options->Fact = FACTORED, C is an input argument,
            otherwise, C is output.
        If options->Fact = FACTORED and equed = 'C' or 'B', each element
            of C must be positive.

L       (output) SuperMatrix*
  The factor L from the factorization
            Pr*A*Pc=L*U              (if A->Stype SLU_= NC) or
            Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
        Uses compressed row subscripts storage for supernodes, i.e.,
        L has types: Stype = SLU_SC, Dtype = SLU_D, Mtype = SLU_TRLU.

U       (output) SuperMatrix*
```

```
  The factor U from the factorization
          Pr*A*Pc=L*U              (if A->Stype = SLU_NC) or
          Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
      Uses column-wise storage scheme, i.e., U has types:
      Stype = SLU_NC, Dtype = SLU_D, Mtype = SLU_TRU.

work    (workspace/output) void*, size (lwork) (in bytes)
        User supplied workspace, should be large enough
        to hold data structures for factors L and U.
        On exit, if fact is not 'F', L and U point to this array.

lwork   (input) int
        Specifies the size of work array in bytes.
        = 0:  allocate space internally by system malloc;
        > 0:  use user-supplied work array of length lwork in bytes,
              returns error if space runs out.
        = -1: the routine guesses the amount of space needed without
              performing the factorization, and returns it in
              mem_usage->total_needed; no other side effects.

        See argument 'mem_usage' for memory usage statistics.

B       (input/output) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_D, Mtype = SLU_GE.
        On entry, the right hand side matrix.
        If B->ncol = 0, only LU decomposition is performed, the triangular
                        solve is skipped.
        On exit,
           if equed = 'N', B is not modified; otherwise
           if A->Stype = SLU_NC:
              if options->Trans = NOTRANS and equed = 'R' or 'B',
                 B is overwritten by diag(R)*B;
              if options->Trans = TRANS or CONJ and equed = 'C' of 'B',
                 B is overwritten by diag(C)*B;
           if A->Stype = SLU_NR:
              if options->Trans = NOTRANS and equed = 'C' or 'B',
                 B is overwritten by diag(C)*B;
              if options->Trans = TRANS or CONJ and equed = 'R' of 'B',
                 B is overwritten by diag(R)*B.

X       (output) SuperMatrix*
        X has types: Stype = SLU_DN, Dtype = SLU_D, Mtype = SLU_GE.
        If info = 0 or info = A->ncol+1, X contains the solution matrix
        to the original system of equations. Note that A and B are modified
        on exit if equed is not 'N', and the solution to the equilibrated
        system is inv(diag(C))*X if options->Trans = NOTRANS and
        equed = 'C' or 'B', or inv(diag(R))*X if options->Trans = 'T' or 'C'
        and equed = 'R' or 'B'.

recip_pivot_growth (output) double*
        The reciprocal pivot growth factor max_j( norm(A_j)/norm(U_j) ).
        The infinity norm is used. If recip_pivot_growth is much less
        than 1, the stability of the LU factorization could be poor.

rcond   (output) double*
        The estimate of the reciprocal condition number of the matrix A
```

```
        after equilibration (if done). If rcond is less than the machine
        precision (in particular, if rcond = 0), the matrix is singular
        to working precision. This condition is indicated by a return
        code of info > 0.

FERR    (output) double*, dimension (B->ncol)
        The estimated forward error bound for each solution vector
        X(j) (the j-th column of the solution matrix X).
        If XTRUE is the true solution corresponding to X(j), FERR(j)
        is an estimated upper bound for the magnitude of the largest
        element in (X(j) - XTRUE) divided by the magnitude of the
        largest element in X(j).  The estimate is as reliable as
        the estimate for RCOND, and is almost always a slight
        overestimate of the true error.
        If options->IterRefine = NOREFINE, ferr = 1.0.

BERR    (output) double*, dimension (B->ncol)
        The componentwise relative backward error of each solution
        vector X(j) (i.e., the smallest relative change in
        any element of A or B that makes X(j) an exact solution).
        If options->IterRefine = NOREFINE, berr = 1.0.

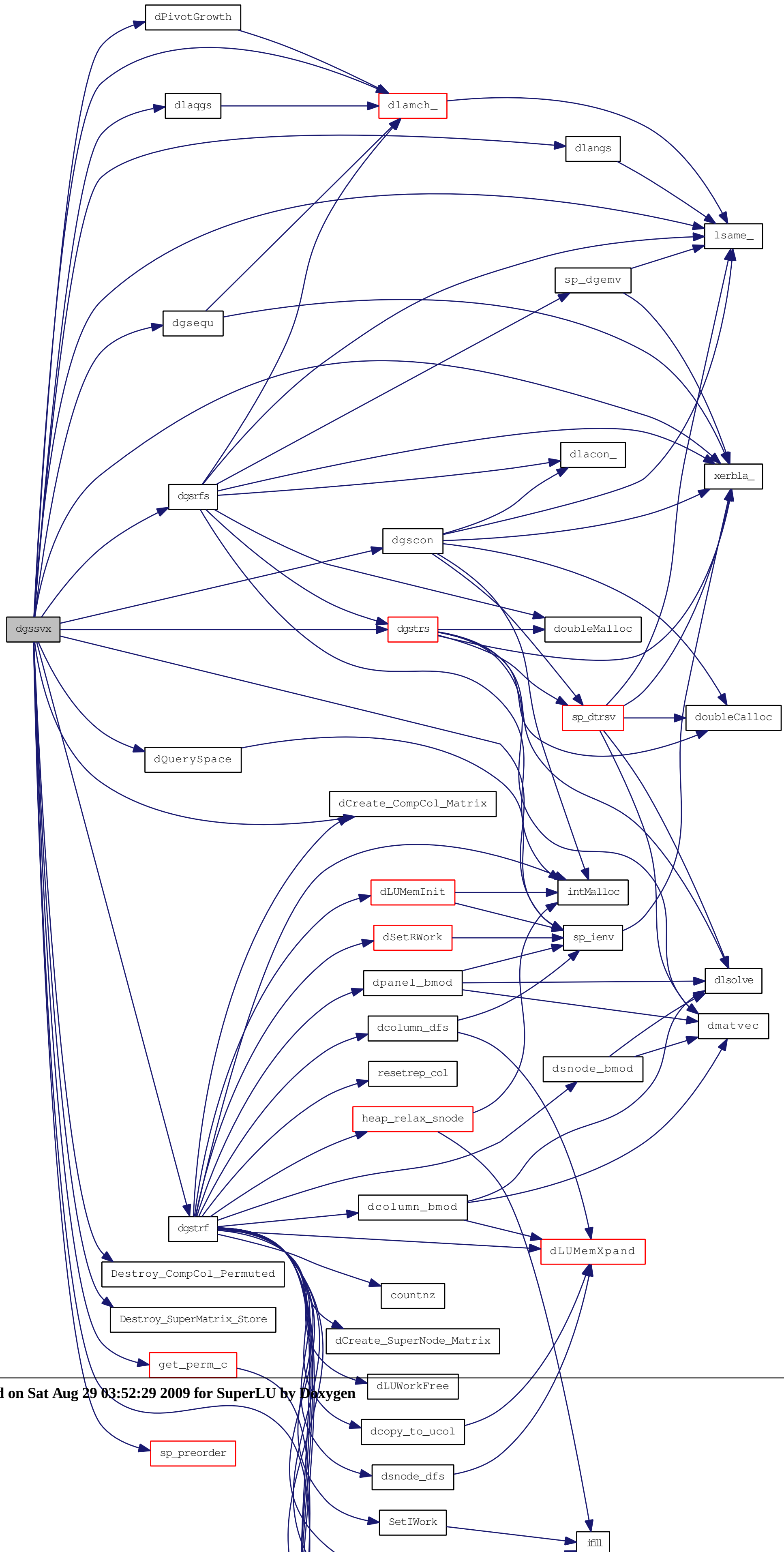
mem_usage (output) mem_usage_t*
        Record the memory usage statistics, consisting of following fields:
```

- for_lu (float)
  ```
    The amount of space used in bytes for L data structures.
  ```
- total_needed (float)
  ```
    The amount of space needed in bytes to perform factorization.
  ```
- expansions (int)
  ```
    The number of memory expansions during the LU factorization.
  ```

```
stat   (output) SuperLUStat_t*
       Record the statistics on runtime and floating-point operation count.
       See util.h for the definition of 'SuperLUStat_t'.

info    (output) int*
        = 0: successful exit
        < 0: if info = -i, the i-th argument had an illegal value
        > 0: if info = i, and i is
             <= A->ncol: U(i,i) is exactly zero. The factorization has
                   been completed, but the factor U is exactly
                   singular, so the solution and error bounds
                   could not be computed.
             = A->ncol+1: U is nonsingular, but RCOND is less than machine
                   precision, meaning that the matrix is singular to
                   working precision. Nevertheless, the solution and
                   error bounds are computed because there are a number
                   of situations where the computed solution can be more
                   accurate than the value of RCOND would suggest.
             > A->ncol+1: number of bytes allocated when memory allocation
                   failure occurred, plus A->ncol.
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.113.3.22 void dgstrf (superlu_options_t * *options*, SuperMatrix * *A*, double *drop_tol*, int *relax*, int *panel_size*, int * *etree*, void * *work*, int *lwork*, int * *perm_c*, int * *perm_r*, SuperMatrix * *L*, SuperMatrix * *U*, SuperLUStat_t * *stat*, int * *info*)

```
Purpose
=======

DGSTRF computes an LU factorization of a general sparse m-by-n
matrix A using partial pivoting with row interchanges.
The factorization has the form
    Pr * A = L * U
where Pr is a row permutation matrix, L is lower triangular with unit
diagonal elements (lower trapezoidal if A->nrow > A->ncol), and U is upper
triangular (upper trapezoidal if A->nrow < A->ncol).

See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

options (input) superlu_options_t*
        The structure defines the input parameters to control
        how the LU decomposition will be performed.

A        (input) SuperMatrix*
   Original matrix A, permuted by columns, of dimension
         (A->nrow, A->ncol). The type of A can be:
         Stype = SLU_NCP; Dtype = SLU_D; Mtype = SLU_GE.

drop_tol (input) double (NOT IMPLEMENTED)
   Drop tolerance parameter. At step j of the Gaussian elimination,
         if abs(A_ij)/(max_i abs(A_ij)) < drop_tol, drop entry A_ij.
         0 <= drop_tol <= 1. The default value of drop_tol is 0.

relax    (input) int
         To control degree of relaxing supernodes. If the number
         of nodes (columns) in a subtree of the elimination tree is less
         than relax, this subtree is considered as one supernode,
         regardless of the row structures of those columns.

panel_size (input) int
         A panel consists of at most panel_size consecutive columns.

etree    (input) int*, dimension (A->ncol)
         Elimination tree of A'*A.
         Note: etree is a vector of parent pointers for a forest whose
         vertices are the integers 0 to A->ncol-1; etree[root]==A->ncol.
         On input, the columns of A should be permuted so that the
         etree is in a certain postorder.
```

```
work     (input/output) void*, size (lwork) (in bytes)
         User-supplied work space and space for the output data structures.
         Not referenced if lwork = 0;

lwork   (input) int
        Specifies the size of work array in bytes.
        = 0:  allocate space internally by system malloc;
        > 0:  use user-supplied work array of length lwork in bytes,
              returns error if space runs out.
        = -1: the routine guesses the amount of space needed without
              performing the factorization, and returns it in
              *info; no other side effects.

perm_c   (input) int*, dimension (A->ncol)
   Column permutation vector, which defines the
         permutation matrix Pc; perm_c[i] = j means column i of A is
         in position j in A*Pc.
         When searching for diagonal, perm_c[*] is applied to the
         row subscripts of A, so that diagonal threshold pivoting
         can find the diagonal of A, rather than that of A*Pc.

perm_r   (input/output) int*, dimension (A->nrow)
         Row permutation vector which defines the permutation matrix Pr,
         perm_r[i] = j means row i of A is in position j in Pr*A.
         If options->Fact = SamePattern_SameRowPerm, the pivoting routine
            will try to use the input perm_r, unless a certain threshold
            criterion is violated. In that case, perm_r is overwritten by
            a new permutation determined by partial pivoting or diagonal
            threshold pivoting.
         Otherwise, perm_r is output argument;

L        (output) SuperMatrix*
         The factor L from the factorization Pr*A=L*U; use compressed row
         subscripts storage for supernodes, i.e., L has type:
         Stype = SLU_SC, Dtype = SLU_D, Mtype = SLU_TRLU.

U        (output) SuperMatrix*
   The factor U from the factorization Pr*A*Pc=L*U. Use column-wise
         storage scheme, i.e., U has types: Stype = SLU_NC,
         Dtype = SLU_D, Mtype = SLU_TRU.

stat     (output) SuperLUStat_t*
         Record the statistics on runtime and floating-point operation count.
         See util.h for the definition of 'SuperLUStat_t'.

info     (output) int*
         = 0: successful exit
         < 0: if info = -i, the i-th argument had an illegal value
         > 0: if info = i, and i is
            <= A->ncol: U(i,i) is exactly zero. The factorization has
               been completed, but the factor U is exactly singular,
               and division by zero will occur if it is used to solve a
               system of equations.
            > A->ncol: number of bytes allocated when memory allocation
               failure occurred, plus A->ncol. If lwork = -1, it is
               the estimated amount of space needed, plus A->ncol.
```

```
 ======================================================================


 Local Working Arrays:
 ======================
   m = number of rows in the matrix
   n = number of columns in the matrix


   xprune[0:n-1]: xprune[*] points to locations in subscript
vector lsub[*]. For column i, xprune[i] denotes the point where
structural pruning begins. I.e. only xlsub[i],..,xprune[i]-1 need
to be traversed for symbolic factorization.


   marker[0:3*m-1]: marker[i] = j means that node i has been
reached when working on column j.
Storage: relative to original row subscripts
NOTE: There are 3 of them: marker/marker1 are used for panel dfs,
      see dpanel_dfs.c; marker2 is used for inner-factorization,
            see dcolumn_dfs.c.


   parent[0:m-1]: parent vector used during dfs
      Storage: relative to new row subscripts


   xplore[0:m-1]: xplore[i] gives the location of the next (dfs)
unexplored neighbor of i in lsub[*]


   segrep[0:nseg-1]: contains the list of supernodal representatives
in topological order of the dfs. A supernode representative is the
last column of a supernode.
      The maximum size of segrep[] is n.

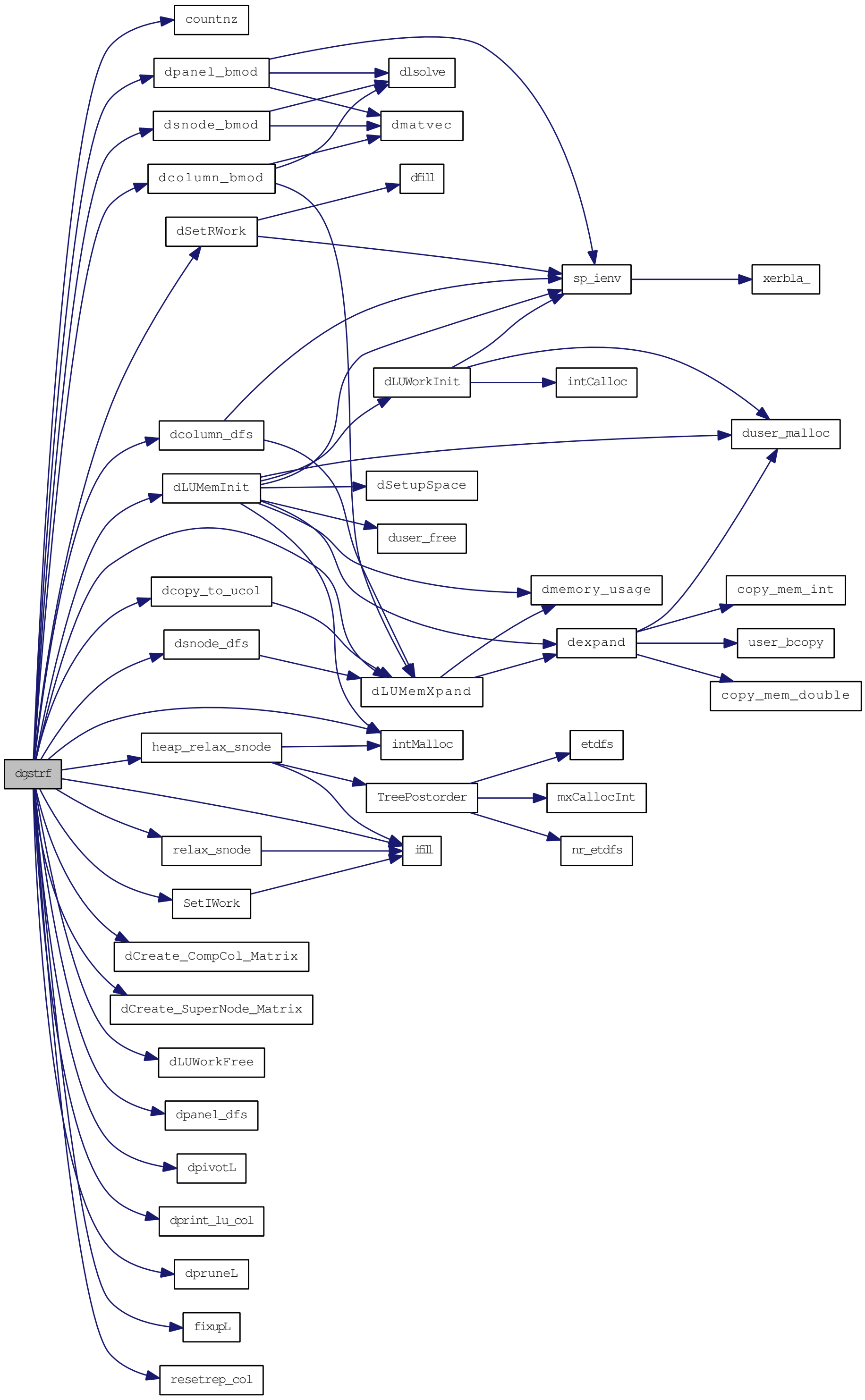

   repfnz[0:W*m-1]: for a nonzero segment U[*,j] that ends at a
supernodal representative r, repfnz[r] is the location of the first
nonzero in this segment.  It is also used during the dfs: repfnz[r]>0
indicates the supernode r has been explored.
NOTE: There are W of them, each used for one column of a panel.


   panel_lsub[0:W*m-1]: temporary for the nonzeros row indices below
      the panel diagonal. These are filled in during dpanel_dfs(), and are
      used later in the inner LU factorization within the panel.
panel_lsub[]/dense[] pair forms the SPA data structure.
NOTE: There are W of them.

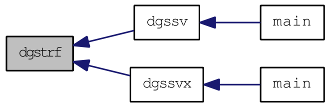

   dense[0:W*m-1]: sparse accumulating (SPA) vector for intermediate values;
        NOTE: there are W of them.


   tempv[0:*]: real temporary used for dense numeric kernels;
The size of this array is defined by NUM_TEMPV() in slu_ddefs.h.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.113.3.23 void dgstrs (trans_t *trans*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, int ∗ *perm_c*, int ∗ *perm_r*, SuperMatrix ∗ *B*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======

DGSTRS solves a system of linear equations A*X=B or A'*X=B
with A sparse and B dense, using the LU factorization computed by
DGSTRF.

See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

trans   (input) trans_t
         Specifies the form of the system of equations:
         = NOTRANS: A * X = B  (No transpose)
         = TRANS:   A'* X = B  (Transpose)
         = CONJ:    A**H * X = B  (Conjugate transpose)

L       (input) SuperMatrix*
        The factor L from the factorization Pr*A*Pc=L*U as computed by
        dgstrf(). Use compressed row subscripts storage for supernodes,
        i.e., L has types: Stype = SLU_SC, Dtype = SLU_D, Mtype = SLU_TRLU.

U       (input) SuperMatrix*
        The factor U from the factorization Pr*A*Pc=L*U as computed by
        dgstrf(). Use column-wise storage scheme, i.e., U has types:
        Stype = SLU_NC, Dtype = SLU_D, Mtype = SLU_TRU.

perm_c  (input) int*, dimension (L->ncol)
  Column permutation vector, which defines the
        permutation matrix Pc; perm_c[i] = j means column i of A is
        in position j in A*Pc.

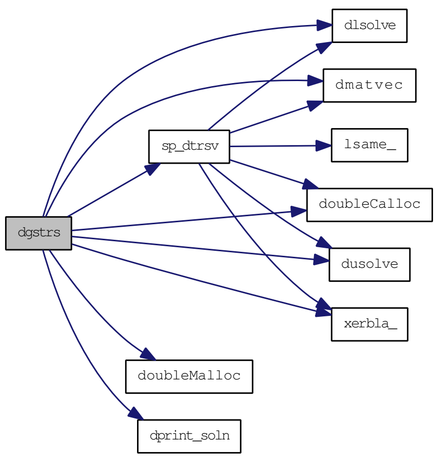
perm_r  (input) int*, dimension (L->nrow)
        Row permutation vector, which defines the permutation matrix Pr;
        perm_r[i] = j means row i of A is in position j in Pr*A.

B       (input/output) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_D, Mtype = SLU_GE.
        On entry, the right hand side matrix.
        On exit, the solution matrix if info = 0;
```

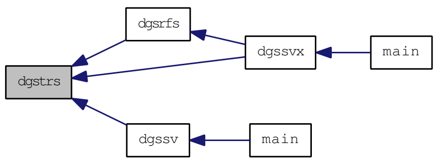
```
stat     (output) SuperLUStat_t*
         Record the statistics on runtime and floating-point operation count.
         See util.h for the definition of 'SuperLUStat_t'.


info    (output) int*
    = 0: successful exit
  < 0: if info = -i, the i-th argument had an illegal value
```

Here is the call graph for this function:

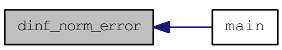


Here is the caller graph for this function:



4.113.3.24 void dinf_norm_error (int, SuperMatrix ∗, double ∗)

Here is the caller graph for this function:



4.113.3.25 void dlaqgs (SuperMatrix ∗ *A*, double ∗ *r*, double ∗ *c*, double *rowcnd*, double *colcnd*, double *amax*, char ∗ *equed*)

```
Purpose
=======
```

```
DLAQGS equilibrates a general sparse M by N matrix A using the row and
scaling factors in the vectors R and C.

See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

A       (input/output) SuperMatrix*
        On exit, the equilibrated matrix.  See EQUED for the form of
        the equilibrated matrix. The type of A can be:
 Stype = NC; Dtype = SLU_D; Mtype = GE.

R       (input) double*, dimension (A->nrow)
        The row scale factors for A.

C       (input) double*, dimension (A->ncol)
        The column scale factors for A.

ROWCND  (input) double
        Ratio of the smallest R(i) to the largest R(i).

COLCND  (input) double
        Ratio of the smallest C(i) to the largest C(i).

AMAX    (input) double
        Absolute value of largest matrix entry.

EQUED   (output) char*
        Specifies the form of equilibration that was done.
        = 'N':  No equilibration
        = 'R':  Row equilibration, i.e., A has been premultiplied by
                diag(R).
        = 'C':  Column equilibration, i.e., A has been postmultiplied
                by diag(C).
        = 'B':  Both row and column equilibration, i.e., A has been
                replaced by diag(R) * A * diag(C).

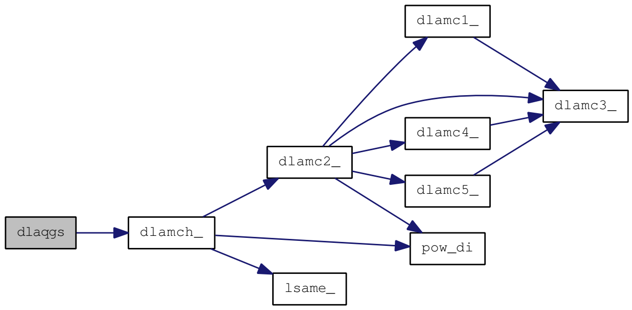
Internal Parameters
===================

THRESH is a threshold value used to decide if row or column scaling
should be done based on the ratio of the row or column scaling
factors.  If ROWCND < THRESH, row scaling is done, and if
COLCND < THRESH, column scaling is done.

LARGE and SMALL are threshold values used to decide if row scaling
should be done based on the absolute size of the largest matrix
element.  If AMAX > LARGE or AMAX < SMALL, row scaling is done.

=====================================================================
```

Here is the call graph for this function:



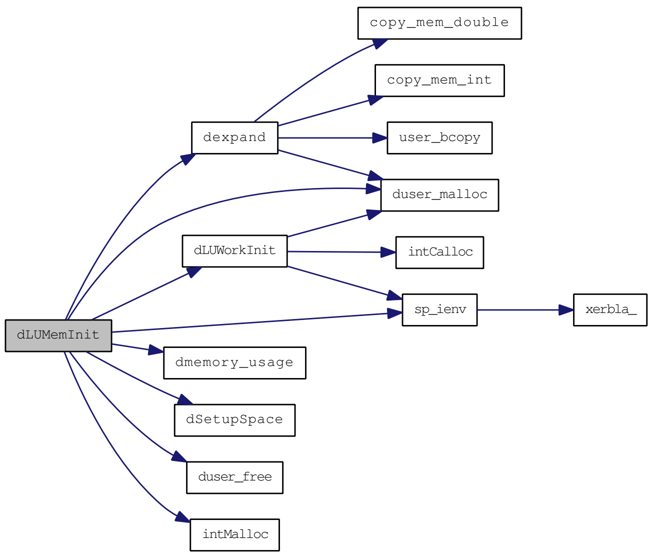
Here is the caller graph for this function:

4.113.3.26 int dLUMemInit (fact_t *fact*, void ∗ *work*, int *lwork*, int *m*, int *n*, int *annz*, int *panel_size*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, GlobalLU_t ∗ *Glu*, int ∗∗ *iwork*, double ∗∗ *dwork*)

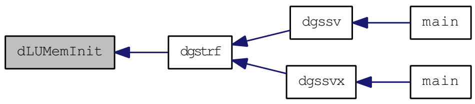
Memory-related.

```
For those unpredictable size, make a guess as FILL * nnz(A).
Return value:
    If lwork = -1, return the estimated amount of space required, plus n;
    otherwise, return the amount of space actually allocated when
    memory allocation failure occurred.
```

Here is the call graph for this function:



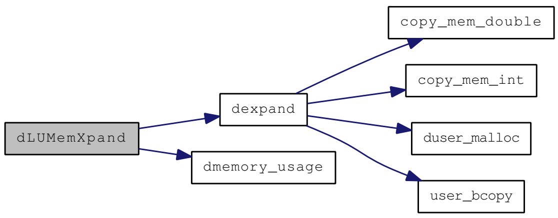
Here is the caller graph for this function:



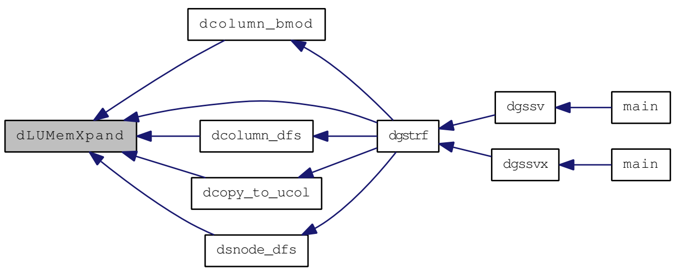
4.113.3.27 int dLUMemXpand (int *jcol*, int *next*, MemType *mem_type*, int ∗ *maxlen*, GlobalLU_t ∗ *Glu*)

```
 Return value:   0 - successful return
               > 0 - number of bytes allocated when run out of space
```

Here is the call graph for this function:

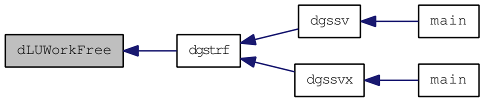


Here is the caller graph for this function:



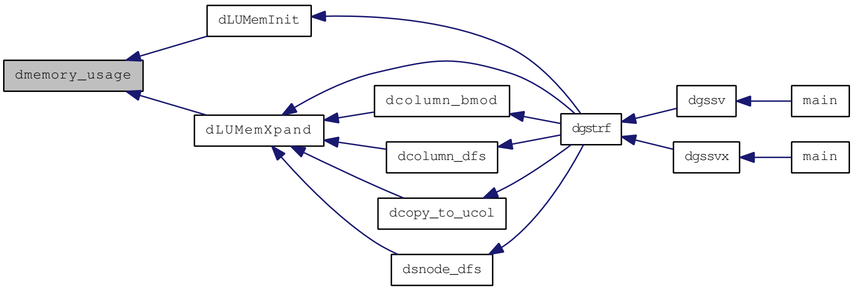
4.113.3.28 void dLUWorkFree (int ∗, double ∗, GlobalLU_t ∗)

Here is the caller graph for this function:



4.113.3.29 int dmemory_usage (const *int*, const *int*, const *int*, const *int*)

Here is the caller graph for this function:



4.113.3.30 double∗ doubleCalloc (int)

4.113.3.31 double∗ doubleMalloc (int)

4.113.3.32 void dpanel_bmod (const int *m*, const int *w*, const int *jcol*, const int *nseg*, double ∗ *dense*, double ∗ *tempv*, int ∗ *segrep*, int ∗ *repfnz*, GlobalLU_t ∗ *Glu*, SuperLUStat_t ∗ *stat*)

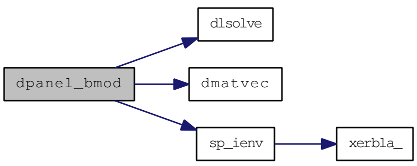
```
Purpose
=======

   Performs numeric block updates (sup-panel) in topological order.
   It features: col-col, 2cols-col, 3cols-col, and sup-col updates.
   Special processing on the supernodal portion of L[*,j]

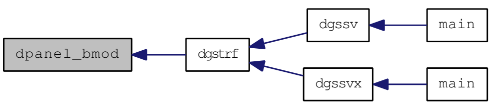
   Before entering this routine, the original nonzeros in the panel
   were already copied into the spa[m,w].

   Updated/Output parameters-
   dense[0:m-1,w]: L[*,j:j+w-1] and U[*,j:j+w-1] are returned
   collectively in the m-by-w vector dense[*].
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.113.3.33 void dpanel_dfs (const int *m*, const int *w*, const int *jcol*, SuperMatrix ∗ *A*, int ∗ *perm_r*, int ∗ *nseg*, double ∗ *dense*, int ∗ *panel_lsub*, int ∗ *segrep*, int ∗ *repfnz*, int ∗ *xprune*, int ∗ *marker*, int ∗ *parent*, int ∗ *xplore*, GlobalLU_t ∗ *Glu*)

```
Purpose
=======

  Performs a symbolic factorization on a panel of columns [jcol, jcol+w).

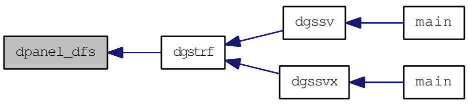
  A supernode representative is the last column of a supernode.
  The nonzeros in U[*,j] are segments that end at supernodal
  representatives.

  The routine returns one list of the supernodal representatives
  in topological order of the dfs that generates them. This list is
  a superset of the topological order of each individual column within
  the panel.
  The location of the first nonzero in each supernodal segment
  (supernodal entry location) is also returned. Each column has a
  separate list for this purpose.

  Two marker arrays are used for dfs:
    marker[i] == jj, if i was visited during dfs of current column jj;
    marker1[i] >= jcol, if i was visited by earlier columns in this panel;

  marker: A-row --> A-row/col (0/1)
  repfnz: SuperA-col --> PA-row
  parent: SuperA-col --> SuperA-col
  xplore: SuperA-col --> index to L-structure
```

Here is the caller graph for this function:



4.113.3.34 double dPivotGrowth (int *ncols*, SuperMatrix ∗ *A*, int ∗ *perm_c*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*)

```
Purpose
=======

Compute the reciprocal pivot growth factor of the leading ncols columns
of the matrix, using the formula:
    min_j ( max_i(abs(A_ij)) / max_i(abs(U_ij)) )

Arguments
=========

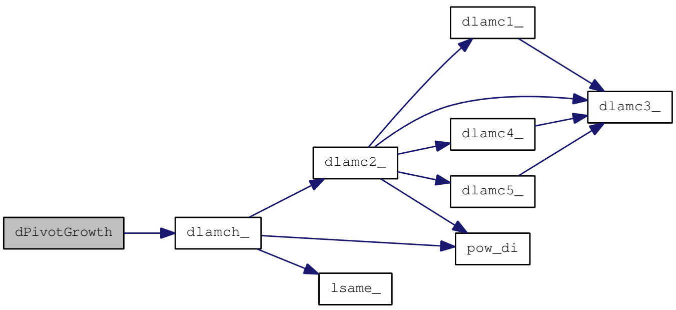
ncols    (input) int
         The number of columns of matrices A, L and U.
```

```
A        (input) SuperMatrix*
   Original matrix A, permuted by columns, of dimension
         (A->nrow, A->ncol). The type of A can be:
         Stype = NC; Dtype = SLU_D; Mtype = GE.

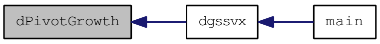
L        (output) SuperMatrix*
         The factor L from the factorization Pr*A=L*U; use compressed row
         subscripts storage for supernodes, i.e., L has type:
         Stype = SC; Dtype = SLU_D; Mtype = TRLU.

U        (output) SuperMatrix*
   The factor U from the factorization Pr*A*Pc=L*U. Use column-wise
         storage scheme, i.e., U has types: Stype = NC;
         Dtype = SLU_D; Mtype = TRU.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.113.3.35 int dpivotL (const int *jcol*, const double *u*, int ∗ *usepr*, int ∗ *perm_r*, int ∗ *iperm_r*, int ∗ *iperm_c*, int ∗ *pivrow*, GlobalLU_t ∗ *Glu*, SuperLUStat_t ∗ *stat*)

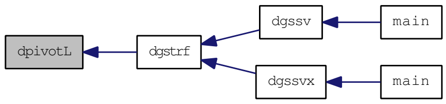
```
Purpose
=======
  Performs the numerical pivoting on the current column of L,
  and the CDIV operation.

  Pivot policy:
  (1) Compute thresh = u * max_(i>=j) abs(A_ij);
  (2) IF user specifies pivot row k and abs(A_kj) >= thresh THEN
          pivot row = k;
      ELSE IF abs(A_jj) >= thresh THEN
          pivot row = j;
      ELSE
          pivot row = m;

  Note: If you absolutely want to use a given pivot order, then set u=0.0.
```

```
Return value: 0      success;
              i > 0  U(i,i) is exactly zero.
```

Here is the caller graph for this function:



4.113.3.36 void dPrint_CompCol_Matrix (char ∗, SuperMatrix ∗)

Here is the caller graph for this function:

4.113.3.37 void dPrint_Dense_Matrix (char ∗, SuperMatrix ∗)

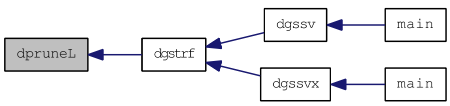
4.113.3.38 void dPrint_SuperNode_Matrix (char ∗, SuperMatrix ∗)

Here is the caller graph for this function:

4.113.3.39 void dpruneL (const int *jcol*, const int ∗ *perm_r*, const int *pivrow*, const int *nseg*, const int ∗ *segrep*, const int ∗ *repfnz*, int ∗ *xprune*, GlobalLU_t ∗ *Glu*)

```
Purpose
=======
  Prunes the L-structure of supernodes whose L-structure
  contains the current pivot row "pivrow"
```

Here is the caller graph for this function:

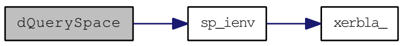


4.113.3.40 int dQuerySpace (SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, mem_usage_t ∗ *mem_usage*)

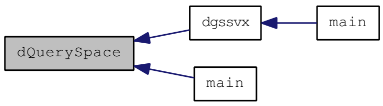
```
mem_usage consists of the following fields:
```

- `for_lu (float)`
  `The amount of space used in bytes for the L data structures.`
- `total_needed (float)`
  `The amount of space needed in bytes to perform factorization.`
- `expansions (int)`
  `Number of memory expansions during the LU factorization.`

Here is the call graph for this function:

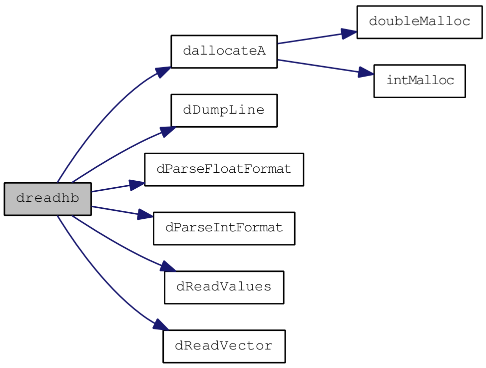


Here is the caller graph for this function:

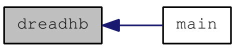


4.113.3.41 void dreadhb (int ∗, int ∗, int ∗, double ∗∗, int ∗∗, int ∗∗)

Here is the call graph for this function:



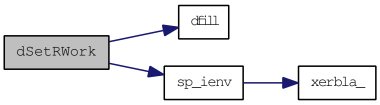
Here is the caller graph for this function:



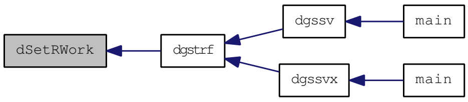
4.113.3.42 void dreadmt (int ∗, int ∗, int ∗, double ∗∗, int ∗∗, int ∗∗)

4.113.3.43 void dSetRWork (int, int, double ∗, double ∗∗, double ∗∗)

Here is the call graph for this function:

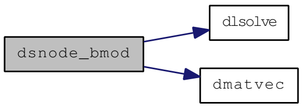


Here is the caller graph for this function:

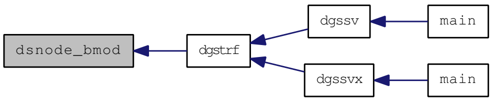


4.113.3.44 int dsnode_bmod (const *int*, const *int*, const *int*, double ∗, double ∗, GlobalLU_t ∗, SuperLUStat_t ∗)

Here is the call graph for this function:



Here is the caller graph for this function:

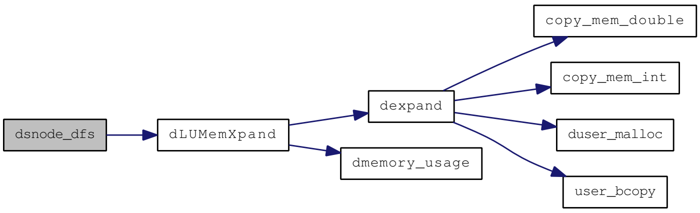


4.113.3.45 int dsnode_dfs (const int *jcol*, const int *kcol*, const int ∗ *asub*, const int ∗ *xa_begin*, const int ∗ *xa_end*, int ∗ *xprune*, int ∗ *marker*, GlobalLU_t ∗ *Glu*)

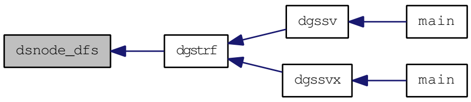
```
Purpose
=======
   dsnode_dfs() - Determine the union of the row structures of those
   columns within the relaxed snode.
   Note: The relaxed snodes are leaves of the supernodal etree, therefore,
   the portion outside the rectangular supernode must be zero.

Return value
============
    0   success;
   >0   number of bytes allocated when run out of memory.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.113.3.46 void fixupL (const *int*, const int ∗, GlobalLU_t ∗)

4.113.3.47 void print_lu_col (char ∗, int, int, int ∗, GlobalLU_t ∗)

4.113.3.48 void PrintPerf (SuperMatrix ∗, SuperMatrix ∗, mem_usage_t ∗, double, double, double ∗, double ∗, char ∗)

4.113.3.49 int sp_dgemm (char ∗ *transa*, char ∗ *transb*, int *m*, int *n*, int *k*, double *alpha*, SuperMatrix ∗ *A*, double ∗ *b*, int *ldb*, double *beta*, double ∗ *c*, int *ldc*)

```
Purpose
  =======

  sp_d performs one of the matrix-matrix operations

     C := alpha*op( A )*op( B ) + beta*C,

  where  op( X ) is one of

     op( X ) = X   or   op( X ) = X'   or   op( X ) = conjg( X' ),

  alpha and beta are scalars, and A, B and C are matrices, with op( A )
  an m by k matrix,  op( B )  a  k by n matrix and  C an m by n matrix.

  Parameters
  ==========

  TRANSA - (input) char*
           On entry, TRANSA specifies the form of op( A ) to be used in
           the matrix multiplication as follows:
              TRANSA = 'N' or 'n',  op( A ) = A.
              TRANSA = 'T' or 't',  op( A ) = A'.
              TRANSA = 'C' or 'c',  op( A ) = conjg( A' ).
           Unchanged on exit.
```

```
TRANSB - (input) char*
         On entry, TRANSB specifies the form of op( B ) to be used in
         the matrix multiplication as follows:
            TRANSB = 'N' or 'n',  op( B ) = B.
            TRANSB = 'T' or 't',  op( B ) = B'.
            TRANSB = 'C' or 'c',  op( B ) = conjg( B' ).
         Unchanged on exit.

M      - (input) int
         On entry,  M  specifies  the number of rows of the matrix
  op( A ) and of the matrix C.  M must be at least zero.
  Unchanged on exit.

N      - (input) int
         On entry,  N specifies the number of columns of the matrix
  op( B ) and the number of columns of the matrix C. N must be
  at least zero.
  Unchanged on exit.

K      - (input) int
         On entry, K specifies the number of columns of the matrix
  op( A ) and the number of rows of the matrix op( B ). K must
  be at least  zero.
        Unchanged on exit.

ALPHA  - (input) double
         On entry, ALPHA specifies the scalar alpha.

A      - (input) SuperMatrix*
         Matrix A with a sparse format, of dimension (A->nrow, A->ncol).
         Currently, the type of A can be:
             Stype = NC or NCP; Dtype = SLU_D; Mtype = GE.
         In the future, more general A can be handled.

B      - DOUBLE PRECISION array of DIMENSION ( LDB, kb ), where kb is
         n when TRANSB = 'N' or 'n',  and is  k otherwise.
         Before entry with  TRANSB = 'N' or 'n',  the leading k by n
         part of the array B must contain the matrix B, otherwise
         the leading n by k part of the array B must contain the
         matrix B.
         Unchanged on exit.

LDB    - (input) int
         On entry, LDB specifies the first dimension of B as declared
         in the calling (sub) program. LDB must be at least max( 1, n ).
         Unchanged on exit.

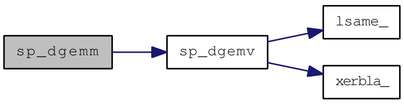
BETA   - (input) double
         On entry, BETA specifies the scalar beta. When BETA is
         supplied as zero then C need not be set on input.

C      - DOUBLE PRECISION array of DIMENSION ( LDC, n ).
         Before entry, the leading m by n part of the array C must
         contain the matrix C,  except when beta is zero, in which
         case C need not be set on entry.
         On exit, the array C is overwritten by the m by n matrix
  ( alpha*op( A )*B + beta*C ).
```

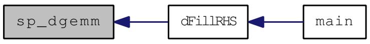
```
LDC     - (input) int
          On entry, LDC specifies the first dimension of C as declared
          in the calling (sub)program. LDC must be at least max(1,m).
          Unchanged on exit.


==== Sparse Level 3 Blas routine.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.113.3.50 int sp_dgemv (char ∗ *trans*, double *alpha*, SuperMatrix ∗ *A*, double ∗ *x*, int *incx*, double *beta*, double ∗ *y*, int *incy*)

```
Purpose
=======


sp_dgemv()  performs one of the matrix-vector operations
   y := alpha*A*x + beta*y,    or   y := alpha*A'*x + beta*y,
where alpha and beta are scalars, x and y are vectors and A is a
sparse A->nrow by A->ncol matrix.


Parameters
==========


TRANS  - (input) char*
         On entry, TRANS specifies the operation to be performed as
         follows:
            TRANS = 'N' or 'n'    y := alpha*A*x + beta*y.
            TRANS = 'T' or 't'    y := alpha*A'*x + beta*y.
            TRANS = 'C' or 'c'    y := alpha*A'*x + beta*y.


ALPHA  - (input) double
         On entry, ALPHA specifies the scalar alpha.


A      - (input) SuperMatrix*
         Matrix A with a sparse format, of dimension (A->nrow, A->ncol).
         Currently, the type of A can be:
             Stype = NC or NCP; Dtype = SLU_D; Mtype = GE.
         In the future, more general A can be handled.
```

```
X       - (input) double*, array of DIMENSION at least
          ( 1 + ( n - 1 )*abs( INCX ) ) when TRANS = 'N' or 'n'
          and at least
          ( 1 + ( m - 1 )*abs( INCX ) ) otherwise.
          Before entry, the incremented array X must contain the
          vector x.


INCX    - (input) int
          On entry, INCX specifies the increment for the elements of
          X. INCX must not be zero.

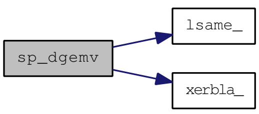

BETA    - (input) double
          On entry, BETA specifies the scalar beta. When BETA is
          supplied as zero then Y need not be set on input.


Y       - (output) double*,  array of DIMENSION at least
          ( 1 + ( m - 1 )*abs( INCY ) ) when TRANS = 'N' or 'n'
          and at least
          ( 1 + ( n - 1 )*abs( INCY ) ) otherwise.
          Before entry with BETA non-zero, the incremented array Y
          must contain the vector y. On exit, Y is overwritten by the
          updated vector y.

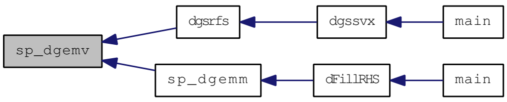

INCY    - (input) int
          On entry, INCY specifies the increment for the elements of
          Y. INCY must not be zero.


==== Sparse Level 2 Blas routine.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.113.3.51 int sp_dtrsv (char * *uplo*, char * *trans*, char * *diag*, SuperMatrix * *L*, SuperMatrix * *U*, double * *x*, SuperLUStat_t * *stat*, int * *info*)

```
Purpose
=======
```

```
sp_dtrsv() solves one of the systems of equations
    A*x = b,   or   A'*x = b,
where b and x are n element vectors and A is a sparse unit , or
non-unit, upper or lower triangular matrix.
No test for singularity or near-singularity is included in this
routine. Such tests must be performed before calling this routine.


Parameters
==========


uplo   - (input) char*
         On entry, uplo specifies whether the matrix is an upper or
          lower triangular matrix as follows:
             uplo = 'U' or 'u'   A is an upper triangular matrix.
             uplo = 'L' or 'l'   A is a lower triangular matrix.


trans  - (input) char*
          On entry, trans specifies the equations to be solved as
          follows:
             trans = 'N' or 'n'   A*x = b.
             trans = 'T' or 't'   A'*x = b.
             trans = 'C' or 'c'   A'*x = b.


diag   - (input) char*
          On entry, diag specifies whether or not A is unit
          triangular as follows:
             diag = 'U' or 'u'   A is assumed to be unit triangular.
             diag = 'N' or 'n'   A is not assumed to be unit
                                 triangular.

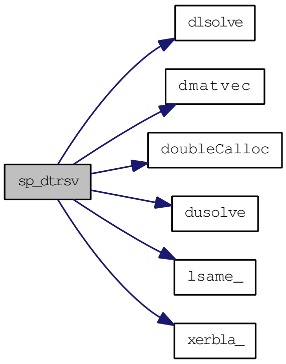

L       - (input) SuperMatrix*
    The factor L from the factorization Pr*A*Pc=L*U. Use
          compressed row subscripts storage for supernodes,
          i.e., L has types: Stype = SC, Dtype = SLU_D, Mtype = TRLU.


U       - (input) SuperMatrix*
     The factor U from the factorization Pr*A*Pc=L*U.
     U has types: Stype = NC, Dtype = SLU_D, Mtype = TRU.

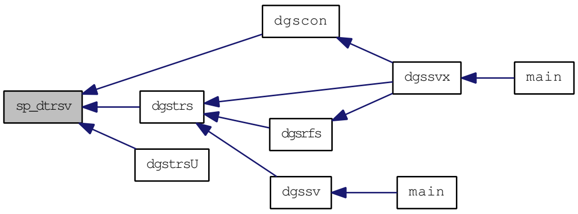

x       - (input/output) double*
          Before entry, the incremented array X must contain the n
          element right-hand side vector b. On exit, X is overwritten
          with the solution vector x.


info    - (output) int*
          If *info = -i, the i-th argument had an illegal value.
```

Here is the call graph for this function:



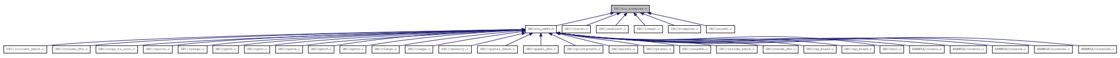
Here is the caller graph for this function:



4.114 SRC/slu_scomplex.h File Reference

Header file for complex operations.

This graph shows which files directly or indirectly include this file:



Data Structures

- struct complex

Defines

- #define c_add(c, a, b)

  Complex Addition c = a + b.

- #define c_sub(c, a, b)

  Complex Subtraction c = a - b.

- #define cs_mult(c, a, b)

  Complex-Double Multiplication.

- #define cc_mult(c, a, b)

  Complex-Complex Multiplication.

- #define cc_conj(a, b)
- #define c_eq(a, b) ((a) → r == (b) → r && (a) → i == (b) → i)

  Complex equality testing.

Functions

- void c_div (complex ∗, complex ∗, complex ∗)

  Complex Division c = a/b.

- double c_abs (complex ∗)

  Returns sqrt(z.r^2 + z.i^2).

- double c_abs1 (complex ∗)

  Approximates the abs. Returns abs(z.r) + abs(z.i).

- void c_exp (complex ∗, complex ∗)

  Return the exponentiation.

- void r_cnjg (complex ∗, complex ∗)

  Return the complex conjugate.

- double r_imag (complex ∗)

  Return the imaginary part.

4.114.1 Detailed Description

```
 -- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997


Contains definitions for various complex operations.
This header file is to be included in source files c*.c
```

4.114.2 Define Documentation

4.114.2.1 #define c_add(c, a, b)

Value:

```
{ (c)->r = (a)->r + (b)->r; \
       (c)->i = (a)->i + (b)->i; }
```

4.114.2.2 #define c_eq(a, b) ((a) → r == (b) → r && (a) → i == (b) → i)

4.114.2.3 #define c_sub(c, a, b)

Value:

```
{ (c)->r = (a)->r - (b)->r; \
       (c)->i = (a)->i - (b)->i; }
```

4.114.2.4 #define cc_conj(a, b)

Value:

```
{ \
        (a)->r = (b)->r; \
        (a)->i = -((b)->i); \
    }
```

4.114.2.5 #define cc_mult(c, a, b)

Value:

```
{ \
  float cr, ci; \
      cr = (a)->r * (b)->r - (a)->i * (b)->i; \
      ci = (a)->i * (b)->r + (a)->r * (b)->i; \
      (c)->r = cr; \
      (c)->i = ci; \
    }
```

4.114.2.6 #define cs_mult(c, a, b)

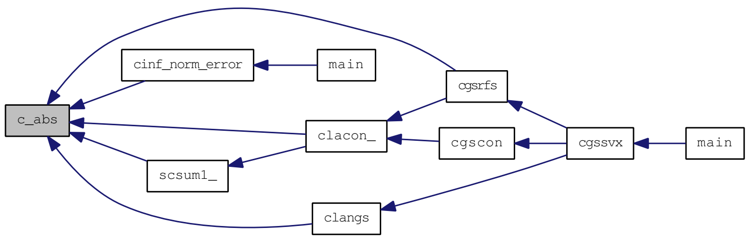
Value:

```
{ (c)->r = (a)->r * (b); \
                           (c)->i = (a)->i * (b); }
```

4.114.3 Function Documentation

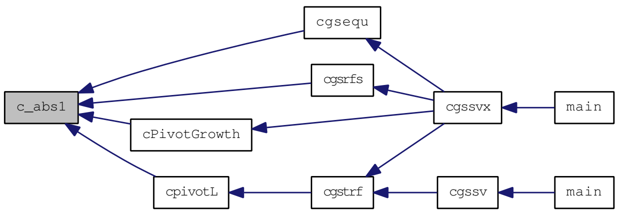
4.114.3.1 double c_abs (complex ∗)

Here is the caller graph for this function:



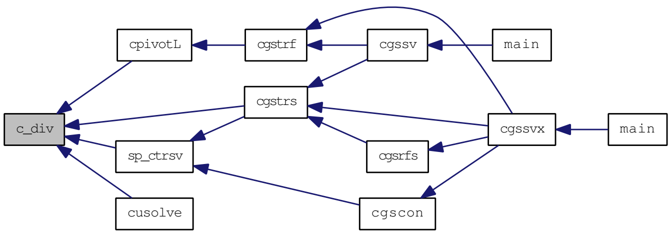
4.114.3.2 double c_abs1 (complex ∗)

Here is the caller graph for this function:



4.114.3.3 void c_div (complex ∗, complex ∗, complex ∗)

Here is the caller graph for this function:



4.114.3.4 void c_exp (complex ∗, complex ∗)

4.114.3.5 void r_cnjg (complex ∗, complex ∗)

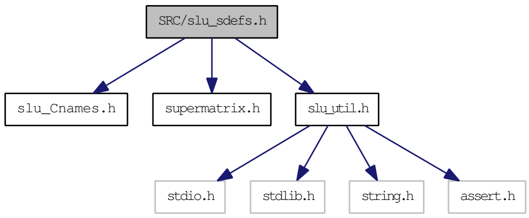
4.114.3.6 double r_imag (complex ∗)

4.115 SRC/slu_sdefs.h File Reference

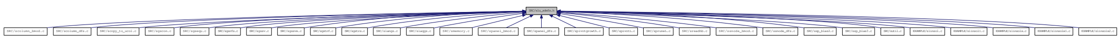
Header file for real operations.

```
#include "slu_Cnames.h"
#include "supermatrix.h"
#include "slu_util.h"
```

Include dependency graph for slu_sdefs.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct GlobalLU_t

Typedefs

- typedef int int_t

Functions

- void sgssv (superlu_options_t ∗, SuperMatrix ∗, int ∗, int ∗, SuperMatrix ∗, SuperMatrix ∗, SuperMatrix ∗, SuperLUStat_t ∗, int ∗)

    Driver routines.

- void sgssvx (superlu_options_t ∗, SuperMatrix ∗, int ∗, int ∗, int ∗, char ∗, float ∗, float ∗, SuperMatrix ∗, SuperMatrix ∗, void ∗, int, SuperMatrix ∗, SuperMatrix ∗, float ∗, float ∗, float ∗, float ∗, mem_usage_t ∗, SuperLUStat_t ∗, int ∗)
- void sCreate_CompCol_Matrix (SuperMatrix ∗, int, int, int, float ∗, int ∗, int ∗, Stype_t, Dtype_t, Mtype_t)

    Supernodal LU factor related.

- void sCreate_CompRow_Matrix (SuperMatrix ∗, int, int, int, float ∗, int ∗, int ∗, Stype_t, Dtype_t, Mtype_t)
- void sCopy_CompCol_Matrix (SuperMatrix ∗, SuperMatrix ∗)

    Copy matrix A into matrix B.

- void sCreate_Dense_Matrix (SuperMatrix ∗, int, int, float ∗, int, Stype_t, Dtype_t, Mtype_t)
- void sCreate_SuperNode_Matrix (SuperMatrix ∗, int, int, int, float ∗, int ∗, int ∗, int ∗, int ∗, int ∗, Stype_t, Dtype_t, Mtype_t)
- void sCopy_Dense_Matrix (int, int, float ∗, int, float ∗, int)
- void countnz (const int, int ∗, int ∗, int ∗, GlobalLU_t ∗)

  Count the total number of nonzeros in factors L and U, and in the symmetrically reduced L.

- void fixupL (const int, const int ∗, GlobalLU_t ∗)

  Fix up the data storage lsub for L-subscripts. It removes the subscript sets for structural pruning, and applies permuation to the remaining subscripts.

- void sallocateA (int, int, float ∗∗, int ∗∗, int ∗∗)

  Allocate storage for original matrix A.

- void sgstrf (superlu_options_t ∗, SuperMatrix ∗, float, int, int, int ∗, void ∗, int, int ∗, int ∗, SuperMatrix ∗, SuperMatrix ∗, SuperLUStat_t ∗, int ∗)
- int ssnode_dfs (const int, const int, const int ∗, const int ∗, const int ∗, int ∗, int ∗, GlobalLU_t ∗)
- int ssnode_bmod (const int, const int, const int, float ∗, float ∗, GlobalLU_t ∗, SuperLUStat_t ∗)

  Performs numeric block updates within the relaxed snode.

- void spanel_dfs (const int, const int, const int, SuperMatrix ∗, int ∗, int ∗, float ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, GlobalLU_t ∗)
- void spanel_bmod (const int, const int, const int, const int, float ∗, float ∗, int ∗, int ∗, GlobalLU_t ∗, SuperLUStat_t ∗)
- int scolumn_dfs (const int, const int, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, GlobalLU_t ∗)
- int scolumn_bmod (const int, const int, float ∗, float ∗, int ∗, int ∗, int, GlobalLU_t ∗, SuperLUStat_t ∗)
- int scopy_to_ucol (int, int, int ∗, int ∗, int ∗, float ∗, GlobalLU_t ∗)
- int spivotL (const int, const float, int ∗, int ∗, int ∗, int ∗, int ∗, GlobalLU_t ∗, SuperLUStat_t ∗)
- void spruneL (const int, const int ∗, const int, const int, const int ∗, const int ∗, int ∗, GlobalLU_t ∗)
- void sreadmt (int ∗, int ∗, int ∗, float ∗∗, int ∗∗, int ∗∗)
- void sGenXtrue (int, int, float ∗, int)
- void sFillRHS (trans_t, int, float ∗, int, SuperMatrix ∗, SuperMatrix ∗)

  Let rhs[i] = sum of i-th row of A, so the solution vector is all 1's.

- void sgstrs (trans_t, SuperMatrix ∗, SuperMatrix ∗, int ∗, int ∗, SuperMatrix ∗, SuperLUStat_t ∗, int ∗)
- void sgsequ (SuperMatrix ∗, float ∗, float ∗, float ∗, float ∗, float ∗, int ∗)

  Driver related.

- void slaqgs (SuperMatrix ∗, float ∗, float ∗, float, float, float, char ∗)
- void sgscon (char ∗, SuperMatrix ∗, SuperMatrix ∗, float, float ∗, SuperLUStat_t ∗, int ∗)
- float sPivotGrowth (int, SuperMatrix ∗, int ∗, SuperMatrix ∗, SuperMatrix ∗)
- void sgsrfs (trans_t, SuperMatrix ∗, SuperMatrix ∗, SuperMatrix ∗, int ∗, int ∗, char ∗, float ∗, float ∗, SuperMatrix ∗, SuperMatrix ∗, float ∗, float ∗, SuperLUStat_t ∗, int ∗)
- int sp_strsv (char ∗, char ∗, char ∗, SuperMatrix ∗, SuperMatrix ∗, float ∗, SuperLUStat_t ∗, int ∗)

  Solves one of the systems of equations A∗x = b, or A'∗x = b.

- int sp_sgemv (char ∗, float, SuperMatrix ∗, float ∗, int, float, float ∗, int)

Performs one of the matrix-vector operations y := alpha∗A∗x + beta∗y, or y := alpha∗A'∗x + beta∗y,.

- int sp_sgemm (char ∗, char ∗, int, int, int, float, SuperMatrix ∗, float ∗, int, float, float ∗, int)
- int sLUMemInit (fact_t, void ∗, int, int, int, int, int, SuperMatrix ∗, SuperMatrix ∗, GlobalLU_t ∗, int ∗∗, float ∗∗)

  Memory-related.

- void sSetRWork (int, int, float ∗, float ∗∗, float ∗∗)

  Set up pointers for real working arrays.

- void sLUWorkFree (int ∗, float ∗, GlobalLU_t ∗)

  Free the working storage used by factor routines.

- int sLUMemXpand (int, int, MemType, int ∗, GlobalLU_t ∗)

  Expand the data structures for L and U during the factorization.

- float ∗ floatMalloc (int)
- float ∗ floatCalloc (int)
- int smemory_usage (const int, const int, const int, const int)
- int sQuerySpace (SuperMatrix ∗, SuperMatrix ∗, mem_usage_t ∗)
- void sreadhb (int ∗, int ∗, int ∗, float ∗∗, int ∗∗, int ∗∗)

  Auxiliary routines.

- void sCompRow_to_CompCol (int, int, int, float ∗, int ∗, int ∗, float ∗∗, int ∗∗, int ∗∗)

  Convert a row compressed storage into a column compressed storage.

- void sfill (float ∗, int, float)

  Fills a float precision array with a given value.

- void sinf_norm_error (int, SuperMatrix ∗, float ∗)

  Check the inf-norm of the error vector.

- void PrintPerf (SuperMatrix ∗, SuperMatrix ∗, mem_usage_t ∗, float, float, float ∗, float ∗, char ∗)
- void sPrint_CompCol_Matrix (char ∗, SuperMatrix ∗)

  Routines for debugging.

- void sPrint_SuperNode_Matrix (char ∗, SuperMatrix ∗)
- void sPrint_Dense_Matrix (char ∗, SuperMatrix ∗)
- void print_lu_col (char ∗, int, int, int ∗, GlobalLU_t ∗)
- void check_tempv (int, float ∗)

4.115.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003


Global data structures used in LU factorization -
```

```
   nsuper: supernodes = nsuper + 1, numbered [0, nsuper].
   (xsup,supno): supno[i] is the supernode no to which i belongs;
xsup(s) points to the beginning of the s-th supernode.
e.g.   supno 0 1 2 2 3 3 3 4 4 4 4 4   (n=12)
        xsup 0 1 2 4 7 12
Note: dfs will be performed on supernode rep. relative to the new
      row pivoting ordering
```

```
   (xlsub,lsub): lsub[*] contains the compressed subscript of
rectangular supernodes; xlsub[j] points to the starting
location of the j-th column in lsub[*]. Note that xlsub
is indexed by column.
Storage: original row subscripts
```

```
      During the course of sparse LU factorization, we also use
(xlsub,lsub) for the purpose of symmetric pruning. For each
supernode {s,s+1,...,t=s+r} with first column s and last
column t, the subscript set
lsub[j], j=xlsub[s], .., xlsub[s+1]-1
is the structure of column s (i.e. structure of this supernode).
It is used for the storage of numerical values.
Furthermore,
lsub[j], j=xlsub[t], .., xlsub[t+1]-1
is the structure of the last column t of this supernode.
It is for the purpose of symmetric pruning. Therefore, the
structural subscripts can be rearranged without making physical
interchanges among the numerical values.
```

```
However, if the supernode has only one column, then we
only keep one set of subscripts. For any subscript interchange
performed, similar interchange must be done on the numerical
values.
```

```
The last column structures (for pruning) will be removed
after the numercial LU factorization phase.
```

```
   (xlusup,lusup): lusup[*] contains the numerical values of the
rectangular supernodes; xlusup[j] points to the starting
location of the j-th column in storage vector lusup[*]
Note: xlusup is indexed by column.
Each rectangular supernode is stored by column-major
scheme, consistent with Fortran 2-dim array storage.
```

```
   (xusub,ucol,usub): ucol[*] stores the numerical values of
U-columns outside the rectangular supernodes. The row
subscript of nonzero ucol[k] is stored in usub[k].
xusub[i] points to the starting location of column i in ucol.
Storage: new row subscripts; that is subscripts of PA.
```

4.115.2 Typedef Documentation

4.115.2.1 typedef int int_t

4.115.3 Function Documentation

4.115.3.1 void check_tempv (int, float ∗)

4.115.3.2 void countnz (const *int*, int ∗, int ∗, int ∗, GlobalLU_t ∗)

4.115.3.3 void fixupL (const *int*, const int ∗, GlobalLU_t ∗)

4.115.3.4 float∗ floatCalloc (int)

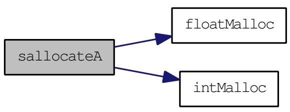
4.115.3.5 float∗ floatMalloc (int)

4.115.3.6 void print_lu_col (char ∗, int, int, int ∗, GlobalLU_t ∗)

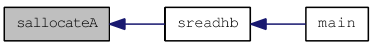
4.115.3.7 void PrintPerf (SuperMatrix ∗, SuperMatrix ∗, mem_usage_t ∗, float, float, float ∗, float ∗, char ∗)

4.115.3.8 void sallocateA (int, int, float ∗∗, int ∗∗, int ∗∗)

Here is the call graph for this function:



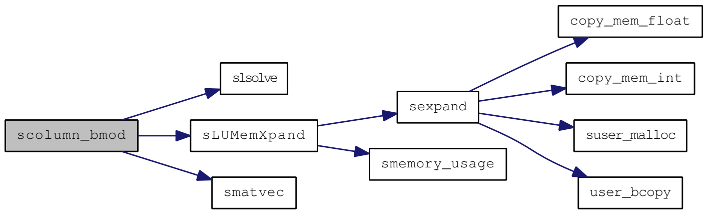
Here is the caller graph for this function:



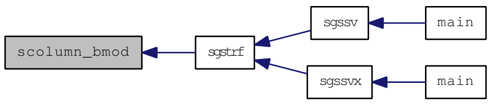
4.115.3.9 int scolumn_bmod (const int *jcol*, const int *nseg*, float ∗ *dense*, float ∗ *tempv*, int ∗ *segrep*, int ∗ *repfnz*, int *fpanelc*, GlobalLU_t ∗ *Glu*, SuperLUStat_t ∗ *stat*)

```
Purpose:
========
Performs numeric block updates (sup-col) in topological order.
It features: col-col, 2cols-col, 3cols-col, and sup-col updates.
Special processing on the supernodal portion of L[*,j]
Return value:   0 - successful return
               > 0 - number of bytes allocated when run out of space
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.115.3.10 int scolumn_dfs (const int *m*, const int *jcol*, int ∗ *perm_r*, int ∗ *nseg*, int ∗ *lsub_col*, int ∗ *segrep*, int ∗ *repfnz*, int ∗ *xprune*, int ∗ *marker*, int ∗ *parent*, int ∗ *xplore*, GlobalLU_t ∗ *Glu*)

```
 Purpose
 =======
   "column_dfs" performs a symbolic factorization on column jcol, and
   decide the supernode boundary.

   This routine does not use numeric values, but only use the RHS
   row indices to start the dfs.

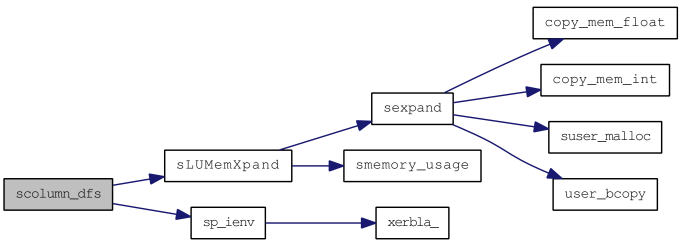
   A supernode representative is the last column of a supernode.
   The nonzeros in U[*,j] are segments that end at supernodal
   representatives. The routine returns a list of such supernodal
   representatives in topological order of the dfs that generates them.
   The location of the first nonzero in each such supernodal segment
   (supernodal entry location) is also returned.

 Local parameters
 ================
   nseg: no of segments in current U[*,j]
   jsuper: jsuper=EMPTY if column j does not belong to the same
supernode as j-1. Otherwise, jsuper=nsuper.

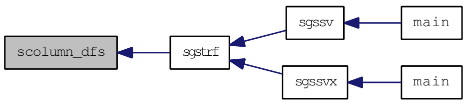
   marker2: A-row --> A-row/col (0/1)
   repfnz: SuperA-col --> PA-row
   parent: SuperA-col --> SuperA-col
   xplore: SuperA-col --> index to L-structure

 Return value
 ============
     0  success;
   > 0  number of bytes allocated when run out of space.
```

Here is the call graph for this function:

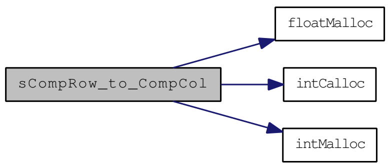


Here is the caller graph for this function:



4.115.3.11 void sCompRow_to_CompCol (int, int, int, float ∗, int ∗, int ∗, float ∗∗, int ∗∗, int ∗∗)

Here is the call graph for this function:



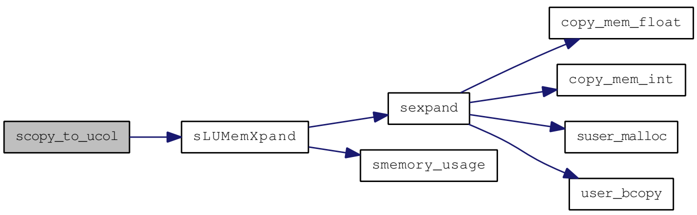
4.115.3.12 void sCopy_CompCol_Matrix (SuperMatrix ∗, SuperMatrix ∗)

4.115.3.13 void sCopy_Dense_Matrix (int, int, float ∗, int, float ∗, int)

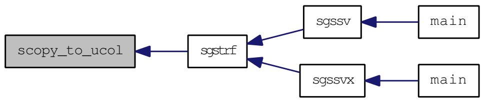
Copies a two-dimensional matrix X to another matrix Y.

4.115.3.14 int scopy_to_ucol (int, int, int ∗, int ∗, int ∗, float ∗, GlobalLU_t ∗)

Here is the call graph for this function:

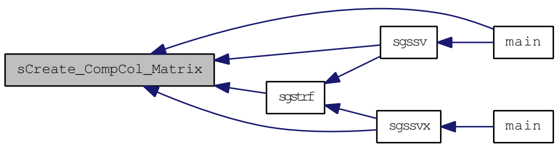


Here is the caller graph for this function:



4.115.3.15 void sCreate_CompCol_Matrix (SuperMatrix ∗, int, int, int, float ∗, int ∗, int ∗, Stype_t, Dtype_t, Mtype_t)

Here is the caller graph for this function:



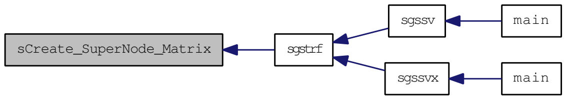
4.115.3.16 void sCreate_CompRow_Matrix (SuperMatrix ∗, int, int, int, float ∗, int ∗, int ∗, Stype_t, Dtype_t, Mtype_t)

4.115.3.17 void sCreate_Dense_Matrix (SuperMatrix ∗, int, int, float ∗, int, Stype_t, Dtype_t, Mtype_t)

Here is the caller graph for this function:

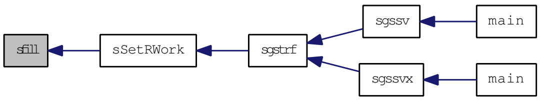
4.115.3.18 void sCreate_SuperNode_Matrix (SuperMatrix ∗, int, int, int, float ∗, int ∗, int ∗, int ∗, int ∗, int ∗, Stype_t, Dtype_t, Mtype_t)

Here is the caller graph for this function:



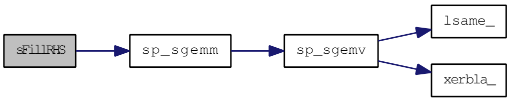
4.115.3.19 void sfill (float ∗, int, float)

Here is the caller graph for this function:

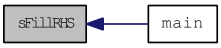


4.115.3.20 void sFillRHS (trans_t, int, float ∗, int, SuperMatrix ∗, SuperMatrix ∗)

Here is the call graph for this function:

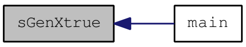


Here is the caller graph for this function:



4.115.3.21 void sGenXtrue (int, int, float ∗, int)

Here is the caller graph for this function:



4.115.3.22 void sgscon (char ∗ *norm*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, float *anorm*, float ∗ *rcond*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======


SGSCON estimates the reciprocal of the condition number of a general
real matrix A, in either the 1-norm or the infinity-norm, using
the LU factorization computed by SGETRF.   *


An estimate is obtained for norm(inv(A)), and the reciprocal of the
condition number is computed as
   RCOND = 1 / ( norm(A) * norm(inv(A)) ).


See supermatrix.h for the definition of 'SuperMatrix' structure.
```

```
Arguments
=========

 NORM    (input) char*
         Specifies whether the 1-norm condition number or the
         infinity-norm condition number is required:
         = '1' or 'O':  1-norm;
         = 'I':         Infinity-norm.

 L       (input) SuperMatrix*
         The factor L from the factorization Pr*A*Pc=L*U as computed by
         sgstrf(). Use compressed row subscripts storage for supernodes,
         i.e., L has types: Stype = SLU_SC, Dtype = SLU_S, Mtype = SLU_TRLU.

 U       (input) SuperMatrix*
         The factor U from the factorization Pr*A*Pc=L*U as computed by
         sgstrf(). Use column-wise storage scheme, i.e., U has types:
         Stype = SLU_NC, Dtype = SLU_S, Mtype = SLU_TRU.

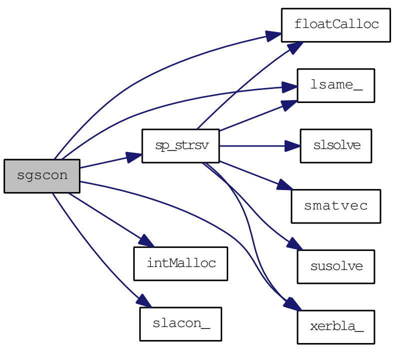
 ANORM   (input) float
         If NORM = '1' or 'O', the 1-norm of the original matrix A.
         If NORM = 'I', the infinity-norm of the original matrix A.

 RCOND   (output) float*
        The reciprocal of the condition number of the matrix A,
        computed as RCOND = 1/(norm(A) * norm(inv(A))).

 INFO    (output) int*
        = 0:  successful exit
        < 0:  if INFO = -i, the i-th argument had an illegal value

 =====================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.115.3.23 void sgsequ (SuperMatrix * *A*, float * *r*, float * *c*, float * *rowcnd*, float * *colcnd*, float * *amax*, int * *info*)

```
Purpose
 =======

 SGSEQU computes row and column scalings intended to equilibrate an
 M-by-N sparse matrix A and reduce its condition number. R returns the row
 scale factors and C the column scale factors, chosen to try to make
 the largest element in each row and column of the matrix B with
 elements B(i,j)=R(i)*A(i,j)*C(j) have absolute value 1.

 R(i) and C(j) are restricted to be between SMLNUM = smallest safe
 number and BIGNUM = largest safe number.  Use of these scaling
 factors is not guaranteed to reduce the condition number of A but
 works well in practice.

 See supermatrix.h for the definition of 'SuperMatrix' structure.

 Arguments
 =========

 A       (input) SuperMatrix*
         The matrix of dimension (A->nrow, A->ncol) whose equilibration
         factors are to be computed. The type of A can be:
         Stype = SLU_NC; Dtype = SLU_S; Mtype = SLU_GE.

 R       (output) float*, size A->nrow
         If INFO = 0 or INFO > M, R contains the row scale factors
         for A.

 C       (output) float*, size A->ncol
         If INFO = 0,  C contains the column scale factors for A.

 ROWCND  (output) float*
         If INFO = 0 or INFO > M, ROWCND contains the ratio of the
         smallest R(i) to the largest R(i).  If ROWCND >= 0.1 and
         AMAX is neither too large nor too small, it is not worth
         scaling by R.

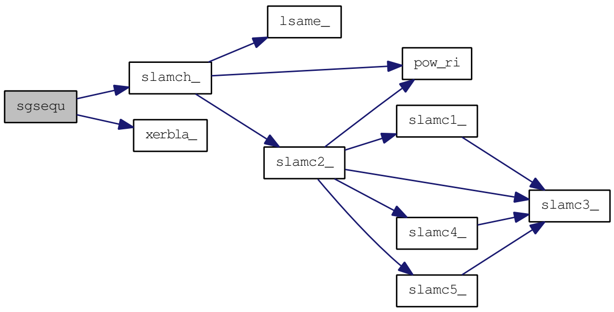
 COLCND  (output) float*
         If INFO = 0, COLCND contains the ratio of the smallest
         C(i) to the largest C(i).  If COLCND >= 0.1, it is not
         worth scaling by C.

 AMAX    (output) float*
         Absolute value of largest matrix element.  If AMAX is very
         close to overflow or very close to underflow, the matrix
         should be scaled.

 INFO    (output) int*
         = 0:  successful exit
         < 0:  if INFO = -i, the i-th argument had an illegal value
         > 0:  if INFO = i,  and i is
               <= A->nrow:  the i-th row of A is exactly zero
               >  A->ncol:  the (i-M)-th column of A is exactly zero
```

```
========================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.115.3.24 void sgsrfs (trans_t *trans*, SuperMatrix ∗ *A*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, int ∗ *perm_c*, int ∗ *perm_r*, char ∗ *equed*, float ∗ *R*, float ∗ *C*, SuperMatrix ∗ *B*, SuperMatrix ∗ *X*, float ∗ *ferr*, float ∗ *berr*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======

SGSRFS improves the computed solution to a system of linear
equations and provides error bounds and backward error estimates for
the solution.

If equilibration was performed, the system becomes:
        (diag(R)*A_original*diag(C)) * X = diag(R)*B_original.

See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

trans   (input) trans_t
         Specifies the form of the system of equations:
         = NOTRANS: A * X = B  (No transpose)
         = TRANS:   A'* X = B  (Transpose)
         = CONJ:    A**H * X = B  (Conjugate transpose)

A       (input) SuperMatrix*
        The original matrix A in the system, or the scaled A if
        equilibration was done. The type of A can be:
        Stype = SLU_NC, Dtype = SLU_S, Mtype = SLU_GE.
```

```
L       (input) SuperMatrix*
  The factor L from the factorization Pr*A*Pc=L*U. Use
        compressed row subscripts storage for supernodes,
        i.e., L has types: Stype = SLU_SC, Dtype = SLU_S, Mtype = SLU_TRLU.

U       (input) SuperMatrix*
        The factor U from the factorization Pr*A*Pc=L*U as computed by
        sgstrf(). Use column-wise storage scheme,
        i.e., U has types: Stype = SLU_NC, Dtype = SLU_S, Mtype = SLU_TRU.

perm_c  (input) int*, dimension (A->ncol)
  Column permutation vector, which defines the
        permutation matrix Pc; perm_c[i] = j means column i of A is
        in position j in A*Pc.

perm_r  (input) int*, dimension (A->nrow)
        Row permutation vector, which defines the permutation matrix Pr;
        perm_r[i] = j means row i of A is in position j in Pr*A.

equed   (input) Specifies the form of equilibration that was done.
        = 'N': No equilibration.
        = 'R': Row equilibration, i.e., A was premultiplied by diag(R).
        = 'C': Column equilibration, i.e., A was postmultiplied by
               diag(C).
        = 'B': Both row and column equilibration, i.e., A was replaced
               by diag(R)*A*diag(C).

R       (input) float*, dimension (A->nrow)
        The row scale factors for A.
        If equed = 'R' or 'B', A is premultiplied by diag(R).
        If equed = 'N' or 'C', R is not accessed.

C       (input) float*, dimension (A->ncol)
        The column scale factors for A.
        If equed = 'C' or 'B', A is postmultiplied by diag(C).
        If equed = 'N' or 'R', C is not accessed.

B       (input) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_S, Mtype = SLU_GE.
        The right hand side matrix B.
        if equed = 'R' or 'B', B is premultiplied by diag(R).

X       (input/output) SuperMatrix*
        X has types: Stype = SLU_DN, Dtype = SLU_S, Mtype = SLU_GE.
        On entry, the solution matrix X, as computed by sgstrs().
        On exit, the improved solution matrix X.
        if *equed = 'C' or 'B', X should be premultiplied by diag(C)
            in order to obtain the solution to the original system.

FERR    (output) float*, dimension (B->ncol)
        The estimated forward error bound for each solution vector
        X(j) (the j-th column of the solution matrix X).
        If XTRUE is the true solution corresponding to X(j), FERR(j)
        is an estimated upper bound for the magnitude of the largest
        element in (X(j) - XTRUE) divided by the magnitude of the
        largest element in X(j).  The estimate is as reliable as
        the estimate for RCOND, and is almost always a slight
        overestimate of the true error.
```

```
BERR    (output) float*, dimension (B->ncol)
        The componentwise relative backward error of each solution
        vector X(j) (i.e., the smallest relative change in
        any element of A or B that makes X(j) an exact solution).

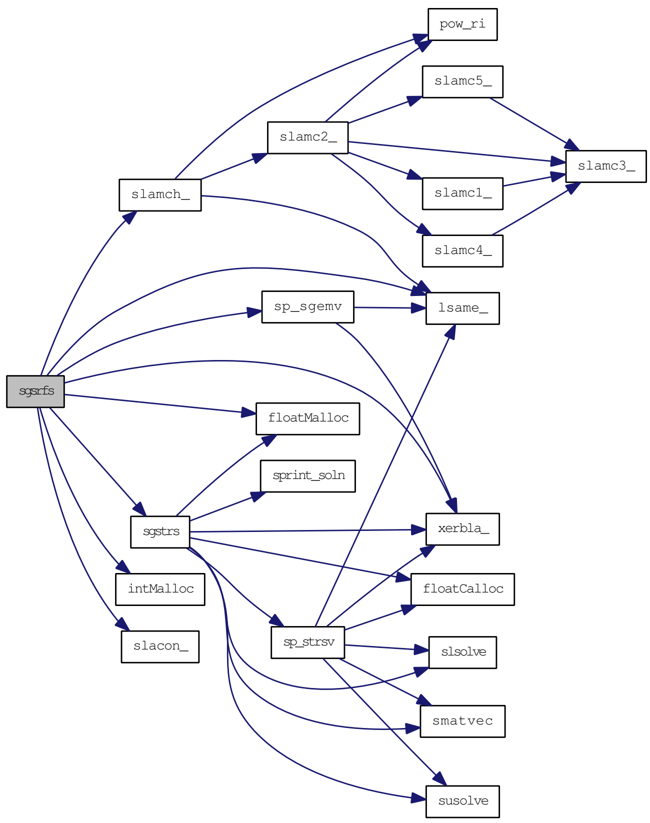
stat     (output) SuperLUStat_t*
         Record the statistics on runtime and floating-point operation count.
         See util.h for the definition of 'SuperLUStat_t'.

info    (output) int*
        = 0:  successful exit
         < 0:  if INFO = -i, the i-th argument had an illegal value

 Internal Parameters
 ===================

 ITMAX is the maximum number of steps of iterative refinement.
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.115.3.25 void sgssv (superlu_options_t * *options*, SuperMatrix * *A*, int * *perm_c*, int * *perm_r*, SuperMatrix * *L*, SuperMatrix * *U*, SuperMatrix * *B*, SuperLUStat_t * *stat*, int * *info*)

```
Purpose
=======

SGSSV solves the system of linear equations A*X=B, using the
LU factorization from SGSTRF. It performs the following steps:

  1. If A is stored column-wise (A->Stype = SLU_NC):

     1.1. Permute the columns of A, forming A*Pc, where Pc
          is a permutation matrix. For more details of this step,
          see sp_preorder.c.

     1.2. Factor A as Pr*A*Pc=L*U with the permutation Pr determined
          by Gaussian elimination with partial pivoting.
          L is unit lower triangular with offdiagonal entries
          bounded by 1 in magnitude, and U is upper triangular.

     1.3. Solve the system of equations A*X=B using the factored
          form of A.

  2. If A is stored row-wise (A->Stype = SLU_NR), apply the
     above algorithm to the transpose of A:

     2.1. Permute columns of transpose(A) (rows of A),
          forming transpose(A)*Pc, where Pc is a permutation matrix.
          For more details of this step, see sp_preorder.c.

     2.2. Factor A as Pr*transpose(A)*Pc=L*U with the permutation Pr
          determined by Gaussian elimination with partial pivoting.
          L is unit lower triangular with offdiagonal entries
          bounded by 1 in magnitude, and U is upper triangular.

     2.3. Solve the system of equations A*X=B using the factored
          form of A.

  See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

options (input) superlu_options_t*
        The structure defines the input parameters to control
        how the LU decomposition will be performed and how the
        system will be solved.

A       (input) SuperMatrix*
        Matrix A in A*X=B, of dimension (A->nrow, A->ncol). The number
        of linear equations is A->nrow. Currently, the type of A can be:
        Stype = SLU_NC or SLU_NR; Dtype = SLU_S; Mtype = SLU_GE.
        In the future, more general A may be handled.
```

```
perm_c  (input/output) int*
        If A->Stype = SLU_NC, column permutation vector of size A->ncol
        which defines the permutation matrix Pc; perm_c[i] = j means
        column i of A is in position j in A*Pc.
        If A->Stype = SLU_NR, column permutation vector of size A->nrow
        which describes permutation of columns of transpose(A)
        (rows of A) as described above.

        If options->ColPerm = MY_PERMC or options->Fact = SamePattern or
           options->Fact = SamePattern_SameRowPerm, it is an input argument.
           On exit, perm_c may be overwritten by the product of the input
           perm_c and a permutation that postorders the elimination tree
           of Pc'*A'*A*Pc; perm_c is not changed if the elimination tree
           is already in postorder.
        Otherwise, it is an output argument.

perm_r  (input/output) int*
        If A->Stype = SLU_NC, row permutation vector of size A->nrow,
        which defines the permutation matrix Pr, and is determined
        by partial pivoting.  perm_r[i] = j means row i of A is in
        position j in Pr*A.
        If A->Stype = SLU_NR, permutation vector of size A->ncol, which
        determines permutation of rows of transpose(A)
        (columns of A) as described above.

        If options->RowPerm = MY_PERMR or
           options->Fact = SamePattern_SameRowPerm, perm_r is an
           input argument.
        otherwise it is an output argument.

L       (output) SuperMatrix*
        The factor L from the factorization
            Pr*A*Pc=L*U              (if A->Stype = SLU_NC) or
            Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
        Uses compressed row subscripts storage for supernodes, i.e.,
        L has types: Stype = SLU_SC, Dtype = SLU_S, Mtype = SLU_TRLU.

U       (output) SuperMatrix*
  The factor U from the factorization
            Pr*A*Pc=L*U              (if A->Stype = SLU_NC) or
            Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
        Uses column-wise storage scheme, i.e., U has types:
        Stype = SLU_NC, Dtype = SLU_S, Mtype = SLU_TRU.

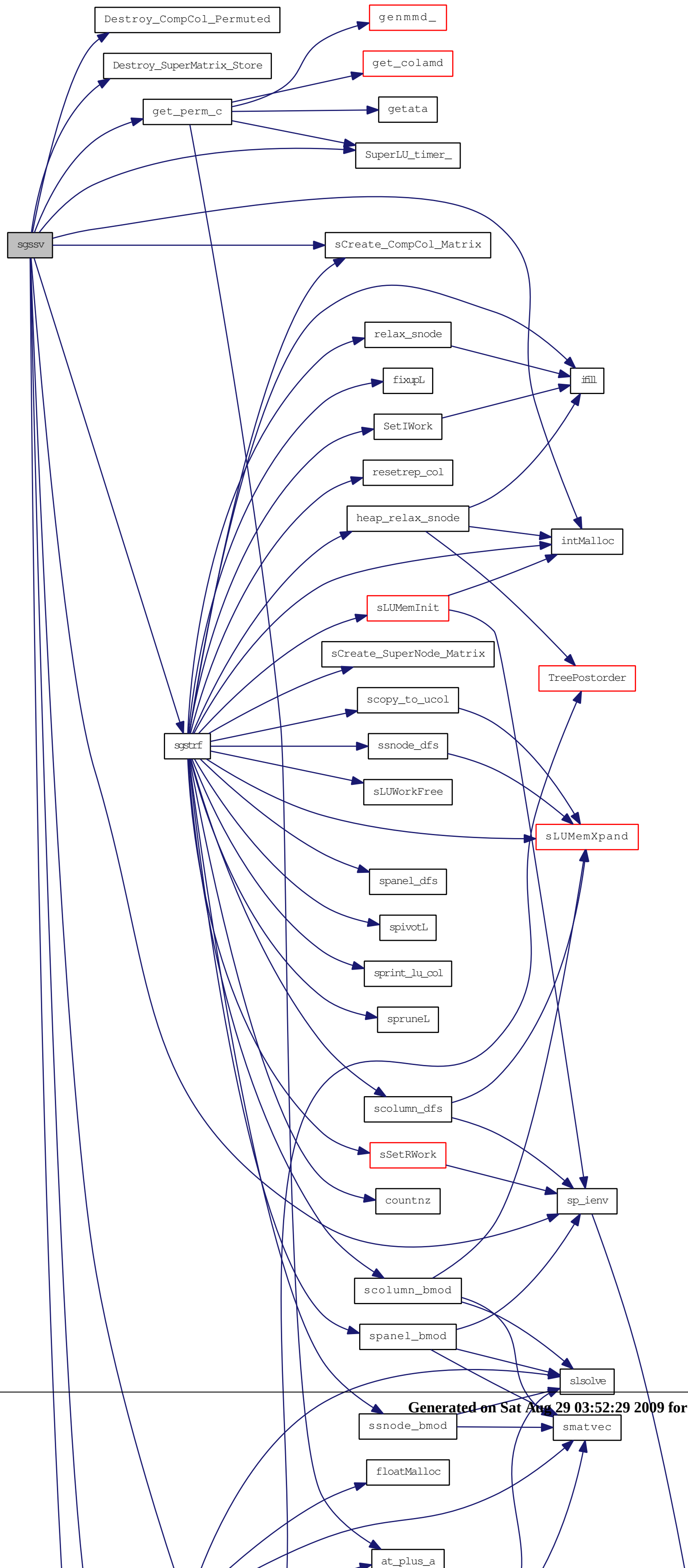
B       (input/output) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_S, Mtype = SLU_GE.
        On entry, the right hand side matrix.
        On exit, the solution matrix if info = 0;

stat   (output) SuperLUStat_t*
       Record the statistics on runtime and floating-point operation count.
       See util.h for the definition of 'SuperLUStat_t'.

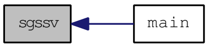
info    (output) int*
  = 0: successful exit
        > 0: if info = i, and i is
```

```
<= A->ncol: U(i,i) is exactly zero. The factorization has
   been completed, but the factor U is exactly singular,
   so the solution could not be computed.
> A->ncol: number of bytes allocated when memory allocation
   failure occurred, plus A->ncol.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.115.3.26 void sgssvx (superlu_options_t ∗ *options*, SuperMatrix ∗ *A*, int ∗ *perm_c*, int ∗ *perm_r*, int ∗ *etree*, char ∗ *equed*, float ∗ *R*, float ∗ *C*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, void ∗ *work*, int *lwork*, SuperMatrix ∗ *B*, SuperMatrix ∗ *X*, float ∗ *recip_pivot_growth*, float ∗ *rcond*, float ∗ *ferr*, float ∗ *berr*, mem_usage_t ∗ *mem_usage*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======

SGSSVX solves the system of linear equations A*X=B or A'*X=B, using
the LU factorization from sgstrf(). Error bounds on the solution and
a condition estimate are also provided. It performs the following steps:

  1. If A is stored column-wise (A->Stype = SLU_NC):

     1.1. If options->Equil = YES, scaling factors are computed to
          equilibrate the system:
          options->Trans = NOTRANS:
              diag(R)*A*diag(C) *inv(diag(C))*X = diag(R)*B
          options->Trans = TRANS:
              (diag(R)*A*diag(C))**T *inv(diag(R))*X = diag(C)*B
          options->Trans = CONJ:
              (diag(R)*A*diag(C))**H *inv(diag(R))*X = diag(C)*B
          Whether or not the system will be equilibrated depends on the
          scaling of the matrix A, but if equilibration is used, A is
          overwritten by diag(R)*A*diag(C) and B by diag(R)*B
          (if options->Trans=NOTRANS) or diag(C)*B (if options->Trans
          = TRANS or CONJ).

     1.2. Permute columns of A, forming A*Pc, where Pc is a permutation
          matrix that usually preserves sparsity.
          For more details of this step, see sp_preorder.c.

     1.3. If options->Fact != FACTORED, the LU decomposition is used to
          factor the matrix A (after equilibration if options->Equil = YES)
          as Pr*A*Pc = L*U, with Pr determined by partial pivoting.

     1.4. Compute the reciprocal pivot growth factor.

     1.5. If some U(i,i) = 0, so that U is exactly singular, then the
          routine returns with info = i. Otherwise, the factored form of
          A is used to estimate the condition number of the matrix A. If
          the reciprocal of the condition number is less than machine
          precision, info = A->ncol+1 is returned as a warning, but the
          routine still goes on to solve for X and computes error bounds
          as described below.
```

```
    1.6. The system of equations is solved for X using the factored form
         of A.

    1.7. If options->IterRefine != NOREFINE, iterative refinement is
         applied to improve the computed solution matrix and calculate
         error bounds and backward error estimates for it.

    1.8. If equilibration was used, the matrix X is premultiplied by
         diag(C) (if options->Trans = NOTRANS) or diag(R)
         (if options->Trans = TRANS or CONJ) so that it solves the
         original system before equilibration.

 2. If A is stored row-wise (A->Stype = SLU_NR), apply the above algorithm
    to the transpose of A:

    2.1. If options->Equil = YES, scaling factors are computed to
         equilibrate the system:
         options->Trans = NOTRANS:
             diag(R)*A*diag(C) *inv(diag(C))*X = diag(R)*B
         options->Trans = TRANS:
             (diag(R)*A*diag(C))**T *inv(diag(R))*X = diag(C)*B
         options->Trans = CONJ:
             (diag(R)*A*diag(C))**H *inv(diag(R))*X = diag(C)*B
         Whether or not the system will be equilibrated depends on the
         scaling of the matrix A, but if equilibration is used, A' is
         overwritten by diag(R)*A'*diag(C) and B by diag(R)*B
         (if trans='N') or diag(C)*B (if trans = 'T' or 'C').

    2.2. Permute columns of transpose(A) (rows of A),
         forming transpose(A)*Pc, where Pc is a permutation matrix that
         usually preserves sparsity.
         For more details of this step, see sp_preorder.c.

    2.3. If options->Fact != FACTORED, the LU decomposition is used to
         factor the transpose(A) (after equilibration if
         options->Fact = YES) as Pr*transpose(A)*Pc = L*U with the
         permutation Pr determined by partial pivoting.

    2.4. Compute the reciprocal pivot growth factor.

    2.5. If some U(i,i) = 0, so that U is exactly singular, then the
         routine returns with info = i. Otherwise, the factored form
         of transpose(A) is used to estimate the condition number of the
         matrix A. If the reciprocal of the condition number
         is less than machine precision, info = A->nrow+1 is returned as
         a warning, but the routine still goes on to solve for X and
         computes error bounds as described below.

    2.6. The system of equations is solved for X using the factored form
         of transpose(A).

    2.7. If options->IterRefine != NOREFINE, iterative refinement is
         applied to improve the computed solution matrix and calculate
         error bounds and backward error estimates for it.
```

```
      2.8. If equilibration was used, the matrix X is premultiplied by
           diag(C) (if options->Trans = NOTRANS) or diag(R)
           (if options->Trans = TRANS or CONJ) so that it solves the
           original system before equilibration.

   See supermatrix.h for the definition of 'SuperMatrix' structure.

 Arguments
 =========

 options (input) superlu_options_t*
         The structure defines the input parameters to control
         how the LU decomposition will be performed and how the
         system will be solved.

 A       (input/output) SuperMatrix*
         Matrix A in A*X=B, of dimension (A->nrow, A->ncol). The number
         of the linear equations is A->nrow. Currently, the type of A can be:
         Stype = SLU_NC or SLU_NR, Dtype = SLU_D, Mtype = SLU_GE.
         In the future, more general A may be handled.

         On entry, If options->Fact = FACTORED and equed is not 'N',
         then A must have been equilibrated by the scaling factors in
         R and/or C.
         On exit, A is not modified if options->Equil = NO, or if
         options->Equil = YES but equed = 'N' on exit.
         Otherwise, if options->Equil = YES and equed is not 'N',
         A is scaled as follows:
         If A->Stype = SLU_NC:
           equed = 'R':  A := diag(R) * A
           equed = 'C':  A := A * diag(C)
           equed = 'B':  A := diag(R) * A * diag(C).
         If A->Stype = SLU_NR:
           equed = 'R':  transpose(A) := diag(R) * transpose(A)
           equed = 'C':  transpose(A) := transpose(A) * diag(C)
           equed = 'B':  transpose(A) := diag(R) * transpose(A) * diag(C).

 perm_c  (input/output) int*
   If A->Stype = SLU_NC, Column permutation vector of size A->ncol,
         which defines the permutation matrix Pc; perm_c[i] = j means
         column i of A is in position j in A*Pc.
         On exit, perm_c may be overwritten by the product of the input
         perm_c and a permutation that postorders the elimination tree
         of Pc'*A'*A*Pc; perm_c is not changed if the elimination tree
         is already in postorder.

         If A->Stype = SLU_NR, column permutation vector of size A->nrow,
         which describes permutation of columns of transpose(A)
         (rows of A) as described above.

 perm_r  (input/output) int*
         If A->Stype = SLU_NC, row permutation vector of size A->nrow,
         which defines the permutation matrix Pr, and is determined
         by partial pivoting.  perm_r[i] = j means row i of A is in
         position j in Pr*A.
```

```
        If A->Stype = SLU_NR, permutation vector of size A->ncol, which
        determines permutation of rows of transpose(A)
        (columns of A) as described above.

        If options->Fact = SamePattern_SameRowPerm, the pivoting routine
        will try to use the input perm_r, unless a certain threshold
        criterion is violated. In that case, perm_r is overwritten by a
        new permutation determined by partial pivoting or diagonal
        threshold pivoting.
        Otherwise, perm_r is output argument.

etree   (input/output) int*,  dimension (A->ncol)
        Elimination tree of Pc'*A'*A*Pc.
        If options->Fact != FACTORED and options->Fact != DOFACT,
        etree is an input argument, otherwise it is an output argument.
        Note: etree is a vector of parent pointers for a forest whose
        vertices are the integers 0 to A->ncol-1; etree[root]==A->ncol.

equed   (input/output) char*
        Specifies the form of equilibration that was done.
        = 'N': No equilibration.
        = 'R': Row equilibration, i.e., A was premultiplied by diag(R).
        = 'C': Column equilibration, i.e., A was postmultiplied by diag(C).
        = 'B': Both row and column equilibration, i.e., A was replaced
               by diag(R)*A*diag(C).
        If options->Fact = FACTORED, equed is an input argument,
        otherwise it is an output argument.

R       (input/output) float*, dimension (A->nrow)
        The row scale factors for A or transpose(A).
        If equed = 'R' or 'B', A (if A->Stype = SLU_NC) or transpose(A)
            (if A->Stype = SLU_NR) is multiplied on the left by diag(R).
        If equed = 'N' or 'C', R is not accessed.
        If options->Fact = FACTORED, R is an input argument,
            otherwise, R is output.
        If options->zFact = FACTORED and equed = 'R' or 'B', each element
            of R must be positive.

C       (input/output) float*, dimension (A->ncol)
        The column scale factors for A or transpose(A).
        If equed = 'C' or 'B', A (if A->Stype = SLU_NC) or transpose(A)
            (if A->Stype = SLU_NR) is multiplied on the right by diag(C).
        If equed = 'N' or 'R', C is not accessed.
        If options->Fact = FACTORED, C is an input argument,
            otherwise, C is output.
        If options->Fact = FACTORED and equed = 'C' or 'B', each element
            of C must be positive.

L       (output) SuperMatrix*
  The factor L from the factorization
            Pr*A*Pc=L*U              (if A->Stype SLU_= NC) or
            Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
        Uses compressed row subscripts storage for supernodes, i.e.,
        L has types: Stype = SLU_SC, Dtype = SLU_S, Mtype = SLU_TRLU.

U       (output) SuperMatrix*
```

```
   The factor U from the factorization
            Pr*A*Pc=L*U              (if A->Stype = SLU_NC) or
            Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
        Uses column-wise storage scheme, i.e., U has types:
        Stype = SLU_NC, Dtype = SLU_S, Mtype = SLU_TRU.

work    (workspace/output) void*, size (lwork) (in bytes)
        User supplied workspace, should be large enough
        to hold data structures for factors L and U.
        On exit, if fact is not 'F', L and U point to this array.

lwork   (input) int
        Specifies the size of work array in bytes.
        = 0:  allocate space internally by system malloc;
        > 0:  use user-supplied work array of length lwork in bytes,
              returns error if space runs out.
        = -1: the routine guesses the amount of space needed without
              performing the factorization, and returns it in
              mem_usage->total_needed; no other side effects.

        See argument 'mem_usage' for memory usage statistics.

B       (input/output) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_S, Mtype = SLU_GE.
        On entry, the right hand side matrix.
        If B->ncol = 0, only LU decomposition is performed, the triangular
                        solve is skipped.
        On exit,
           if equed = 'N', B is not modified; otherwise
           if A->Stype = SLU_NC:
              if options->Trans = NOTRANS and equed = 'R' or 'B',
                 B is overwritten by diag(R)*B;
              if options->Trans = TRANS or CONJ and equed = 'C' of 'B',
                 B is overwritten by diag(C)*B;
           if A->Stype = SLU_NR:
              if options->Trans = NOTRANS and equed = 'C' or 'B',
                 B is overwritten by diag(C)*B;
              if options->Trans = TRANS or CONJ and equed = 'R' of 'B',
                 B is overwritten by diag(R)*B.

X       (output) SuperMatrix*
        X has types: Stype = SLU_DN, Dtype = SLU_S, Mtype = SLU_GE.
        If info = 0 or info = A->ncol+1, X contains the solution matrix
        to the original system of equations. Note that A and B are modified
        on exit if equed is not 'N', and the solution to the equilibrated
        system is inv(diag(C))*X if options->Trans = NOTRANS and
        equed = 'C' or 'B', or inv(diag(R))*X if options->Trans = 'T' or 'C'
        and equed = 'R' or 'B'.

recip_pivot_growth (output) float*
        The reciprocal pivot growth factor max_j( norm(A_j)/norm(U_j) ).
        The infinity norm is used. If recip_pivot_growth is much less
        than 1, the stability of the LU factorization could be poor.

rcond   (output) float*
        The estimate of the reciprocal condition number of the matrix A
```

```
        after equilibration (if done). If rcond is less than the machine
        precision (in particular, if rcond = 0), the matrix is singular
        to working precision. This condition is indicated by a return
        code of info > 0.

FERR    (output) float*, dimension (B->ncol)
        The estimated forward error bound for each solution vector
        X(j) (the j-th column of the solution matrix X).
        If XTRUE is the true solution corresponding to X(j), FERR(j)
        is an estimated upper bound for the magnitude of the largest
        element in (X(j) - XTRUE) divided by the magnitude of the
        largest element in X(j).  The estimate is as reliable as
        the estimate for RCOND, and is almost always a slight
        overestimate of the true error.
        If options->IterRefine = NOREFINE, ferr = 1.0.

BERR    (output) float*, dimension (B->ncol)
        The componentwise relative backward error of each solution
        vector X(j) (i.e., the smallest relative change in
        any element of A or B that makes X(j) an exact solution).
        If options->IterRefine = NOREFINE, berr = 1.0.

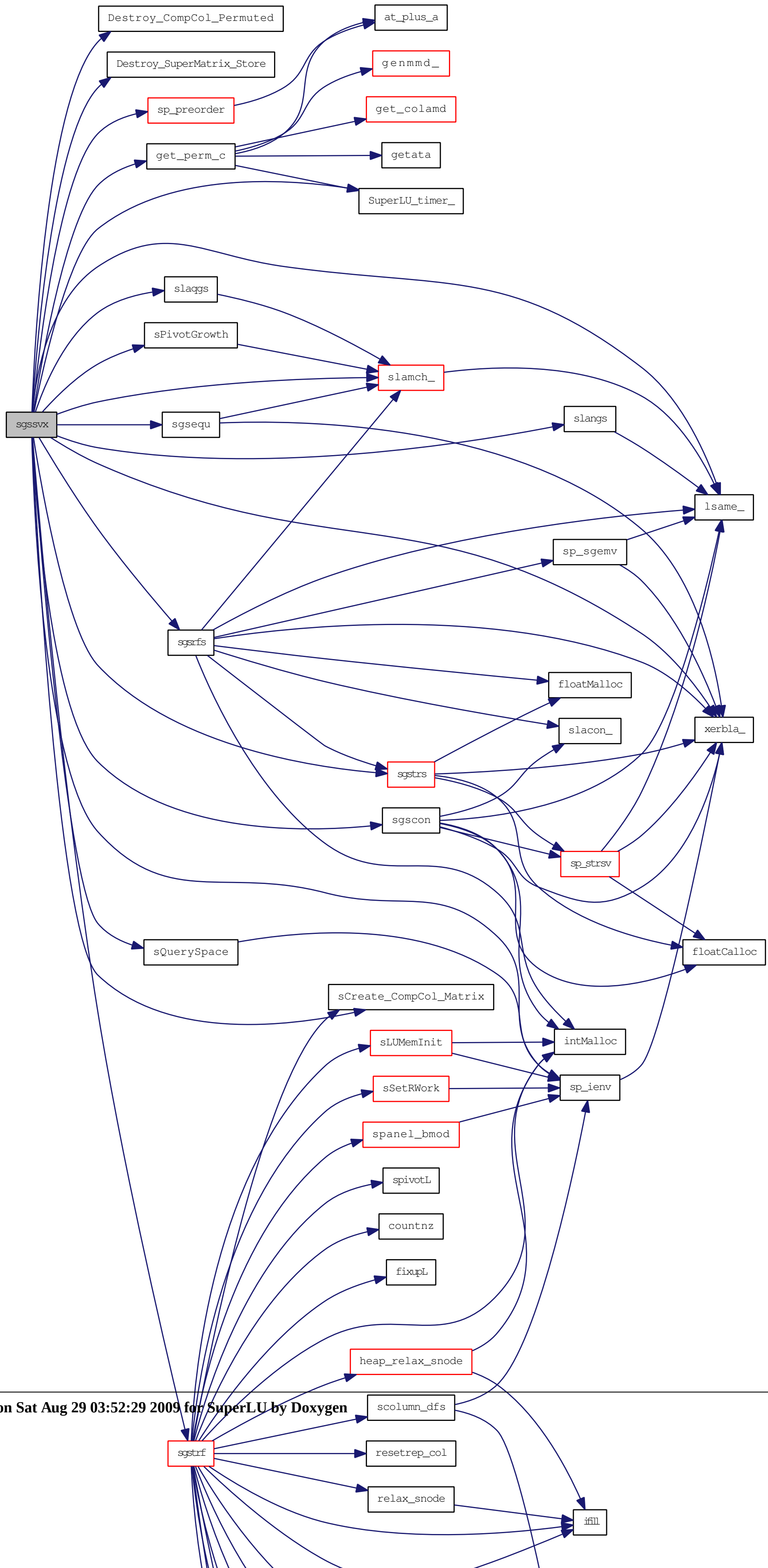
mem_usage (output) mem_usage_t*
        Record the memory usage statistics, consisting of following fields:
```

- for_lu (float)
  ```
  The amount of space used in bytes for L data structures.
  ```
- total_needed (float)
  ```
  The amount of space needed in bytes to perform factorization.
  ```
- expansions (int)
  ```
  The number of memory expansions during the LU factorization.
  ```

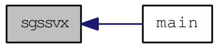
```
stat   (output) SuperLUStat_t*
       Record the statistics on runtime and floating-point operation count.
       See util.h for the definition of 'SuperLUStat_t'.

info    (output) int*
        = 0: successful exit
        < 0: if info = -i, the i-th argument had an illegal value
        > 0: if info = i, and i is
             <= A->ncol: U(i,i) is exactly zero. The factorization has
                   been completed, but the factor U is exactly
                   singular, so the solution and error bounds
                   could not be computed.
             = A->ncol+1: U is nonsingular, but RCOND is less than machine
                   precision, meaning that the matrix is singular to
                   working precision. Nevertheless, the solution and
                   error bounds are computed because there are a number
                   of situations where the computed solution can be more
                   accurate than the value of RCOND would suggest.
             > A->ncol+1: number of bytes allocated when memory allocation
                   failure occurred, plus A->ncol.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.115.3.27 void sgstrf (superlu_options_t ∗ *options*, SuperMatrix ∗ *A*, float *drop_tol*, int *relax*, int *panel_size*, int ∗ *etree*, void ∗ *work*, int *lwork*, int ∗ *perm_c*, int ∗ *perm_r*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======

SGSTRF computes an LU factorization of a general sparse m-by-n
matrix A using partial pivoting with row interchanges.
The factorization has the form
    Pr * A = L * U
where Pr is a row permutation matrix, L is lower triangular with unit
diagonal elements (lower trapezoidal if A->nrow > A->ncol), and U is upper
triangular (upper trapezoidal if A->nrow < A->ncol).

See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

options (input) superlu_options_t*
        The structure defines the input parameters to control
        how the LU decomposition will be performed.

A        (input) SuperMatrix*
   Original matrix A, permuted by columns, of dimension
         (A->nrow, A->ncol). The type of A can be:
         Stype = SLU_NCP; Dtype = SLU_S; Mtype = SLU_GE.

drop_tol (input) float (NOT IMPLEMENTED)
   Drop tolerance parameter. At step j of the Gaussian elimination,
         if abs(A_ij)/(max_i abs(A_ij)) < drop_tol, drop entry A_ij.
         0 <= drop_tol <= 1. The default value of drop_tol is 0.

relax    (input) int
         To control degree of relaxing supernodes. If the number
         of nodes (columns) in a subtree of the elimination tree is less
         than relax, this subtree is considered as one supernode,
         regardless of the row structures of those columns.

panel_size (input) int
         A panel consists of at most panel_size consecutive columns.

etree    (input) int*, dimension (A->ncol)
         Elimination tree of A'*A.
         Note: etree is a vector of parent pointers for a forest whose
         vertices are the integers 0 to A->ncol-1; etree[root]==A->ncol.
         On input, the columns of A should be permuted so that the
         etree is in a certain postorder.
```

```
work     (input/output) void*, size (lwork) (in bytes)
         User-supplied work space and space for the output data structures.
         Not referenced if lwork = 0;

lwork   (input) int
        Specifies the size of work array in bytes.
        = 0:  allocate space internally by system malloc;
        > 0:  use user-supplied work array of length lwork in bytes,
              returns error if space runs out.
        = -1: the routine guesses the amount of space needed without
              performing the factorization, and returns it in
              *info; no other side effects.

perm_c   (input) int*, dimension (A->ncol)
   Column permutation vector, which defines the
         permutation matrix Pc; perm_c[i] = j means column i of A is
         in position j in A*Pc.
         When searching for diagonal, perm_c[*] is applied to the
         row subscripts of A, so that diagonal threshold pivoting
         can find the diagonal of A, rather than that of A*Pc.

perm_r   (input/output) int*, dimension (A->nrow)
         Row permutation vector which defines the permutation matrix Pr,
         perm_r[i] = j means row i of A is in position j in Pr*A.
         If options->Fact = SamePattern_SameRowPerm, the pivoting routine
            will try to use the input perm_r, unless a certain threshold
            criterion is violated. In that case, perm_r is overwritten by
            a new permutation determined by partial pivoting or diagonal
            threshold pivoting.
         Otherwise, perm_r is output argument;

L        (output) SuperMatrix*
         The factor L from the factorization Pr*A=L*U; use compressed row
         subscripts storage for supernodes, i.e., L has type:
         Stype = SLU_SC, Dtype = SLU_S, Mtype = SLU_TRLU.

U        (output) SuperMatrix*
   The factor U from the factorization Pr*A*Pc=L*U. Use column-wise
         storage scheme, i.e., U has types: Stype = SLU_NC,
         Dtype = SLU_S, Mtype = SLU_TRU.

stat     (output) SuperLUStat_t*
         Record the statistics on runtime and floating-point operation count.
         See util.h for the definition of 'SuperLUStat_t'.

info     (output) int*
         = 0: successful exit
         < 0: if info = -i, the i-th argument had an illegal value
         > 0: if info = i, and i is
            <= A->ncol: U(i,i) is exactly zero. The factorization has
               been completed, but the factor U is exactly singular,
               and division by zero will occur if it is used to solve a
               system of equations.
            > A->ncol: number of bytes allocated when memory allocation
               failure occurred, plus A->ncol. If lwork = -1, it is
               the estimated amount of space needed, plus A->ncol.
```

```
 ======================================================================


 Local Working Arrays:
 ======================
   m = number of rows in the matrix
   n = number of columns in the matrix


   xprune[0:n-1]: xprune[*] points to locations in subscript
vector lsub[*]. For column i, xprune[i] denotes the point where
structural pruning begins. I.e. only xlsub[i],..,xprune[i]-1 need
to be traversed for symbolic factorization.


   marker[0:3*m-1]: marker[i] = j means that node i has been
reached when working on column j.
Storage: relative to original row subscripts
NOTE: There are 3 of them: marker/marker1 are used for panel dfs,
      see spanel_dfs.c; marker2 is used for inner-factorization,
            see scolumn_dfs.c.


   parent[0:m-1]: parent vector used during dfs
      Storage: relative to new row subscripts


   xplore[0:m-1]: xplore[i] gives the location of the next (dfs)
unexplored neighbor of i in lsub[*]


   segrep[0:nseg-1]: contains the list of supernodal representatives
in topological order of the dfs. A supernode representative is the
last column of a supernode.
      The maximum size of segrep[] is n.

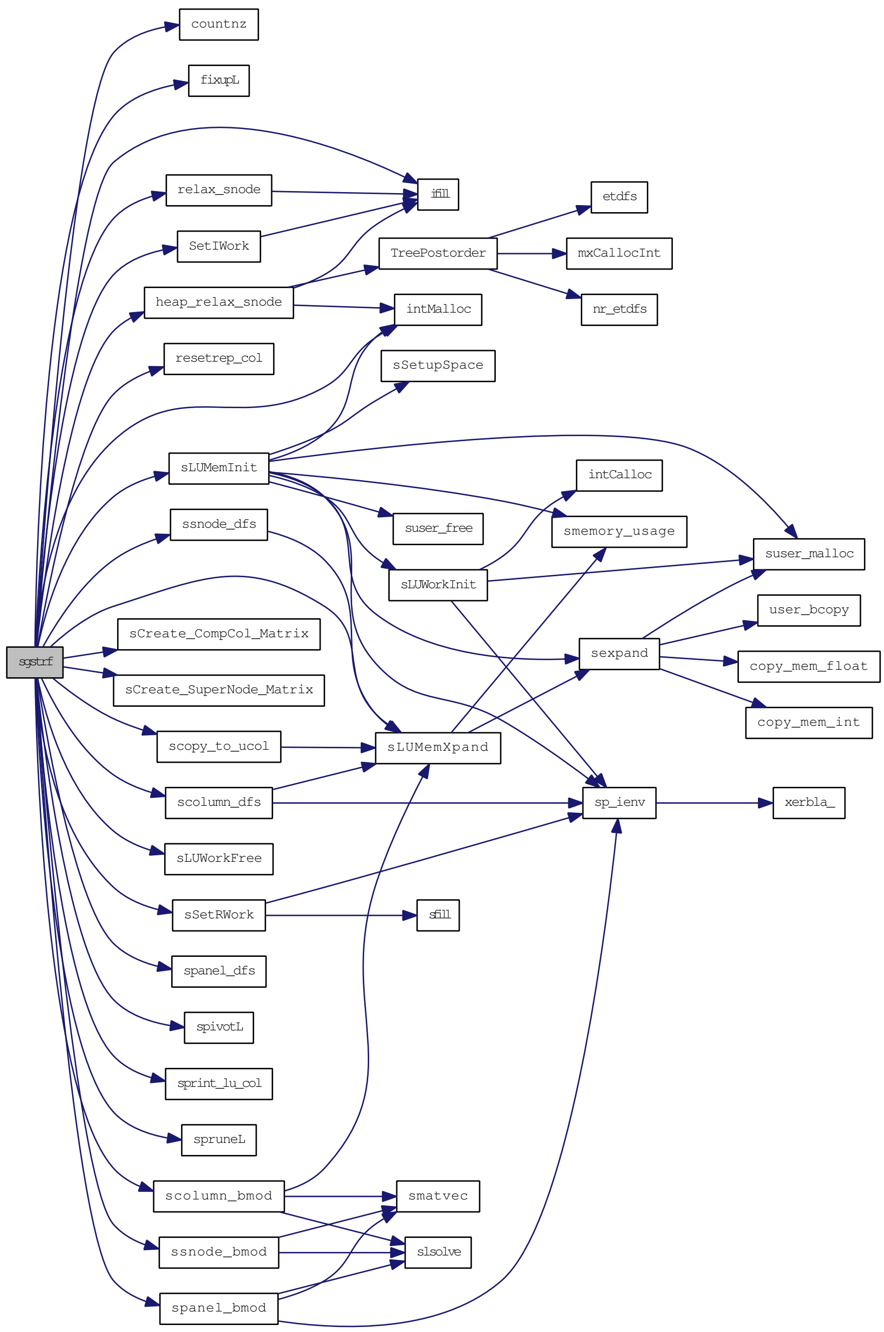

   repfnz[0:W*m-1]: for a nonzero segment U[*,j] that ends at a
supernodal representative r, repfnz[r] is the location of the first
nonzero in this segment.  It is also used during the dfs: repfnz[r]>0
indicates the supernode r has been explored.
NOTE: There are W of them, each used for one column of a panel.


   panel_lsub[0:W*m-1]: temporary for the nonzeros row indices below
      the panel diagonal. These are filled in during spanel_dfs(), and are
      used later in the inner LU factorization within the panel.
panel_lsub[]/dense[] pair forms the SPA data structure.
NOTE: There are W of them.

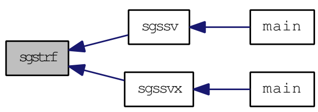

   dense[0:W*m-1]: sparse accumulating (SPA) vector for intermediate values;
        NOTE: there are W of them.


   tempv[0:*]: real temporary used for dense numeric kernels;
The size of this array is defined by NUM_TEMPV() in slu_sdefs.h.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.115.3.28 void sgstrs (trans_t *trans*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, int ∗ *perm_c*, int ∗ *perm_r*, SuperMatrix ∗ *B*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======

SGSTRS solves a system of linear equations A*X=B or A'*X=B
with A sparse and B dense, using the LU factorization computed by
SGSTRF.

See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

trans   (input) trans_t
         Specifies the form of the system of equations:
         = NOTRANS: A * X = B  (No transpose)
         = TRANS:   A'* X = B  (Transpose)
         = CONJ:    A**H * X = B  (Conjugate transpose)

L       (input) SuperMatrix*
        The factor L from the factorization Pr*A*Pc=L*U as computed by
        sgstrf(). Use compressed row subscripts storage for supernodes,
        i.e., L has types: Stype = SLU_SC, Dtype = SLU_S, Mtype = SLU_TRLU.

U       (input) SuperMatrix*
        The factor U from the factorization Pr*A*Pc=L*U as computed by
        sgstrf(). Use column-wise storage scheme, i.e., U has types:
        Stype = SLU_NC, Dtype = SLU_S, Mtype = SLU_TRU.

perm_c  (input) int*, dimension (L->ncol)
  Column permutation vector, which defines the
        permutation matrix Pc; perm_c[i] = j means column i of A is
        in position j in A*Pc.

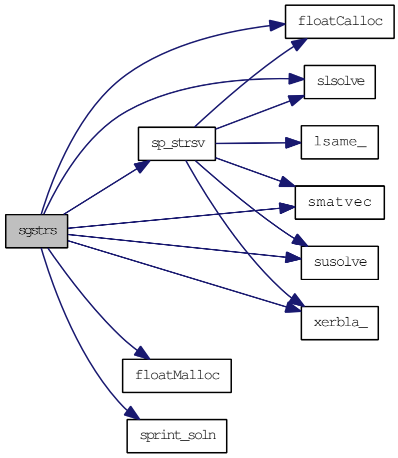
perm_r  (input) int*, dimension (L->nrow)
        Row permutation vector, which defines the permutation matrix Pr;
        perm_r[i] = j means row i of A is in position j in Pr*A.

B       (input/output) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_S, Mtype = SLU_GE.
        On entry, the right hand side matrix.
        On exit, the solution matrix if info = 0;

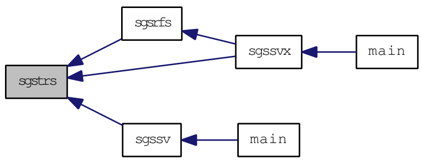
stat     (output) SuperLUStat_t*
         Record the statistics on runtime and floating-point operation count.
         See util.h for the definition of 'SuperLUStat_t'.

info    (output) int*
    = 0: successful exit
  < 0: if info = -i, the i-th argument had an illegal value
```

Here is the call graph for this function:

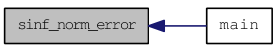


Here is the caller graph for this function:



4.115.3.29 void sinf_norm_error (int, SuperMatrix ∗, float ∗)

Here is the caller graph for this function:



4.115.3.30 void slaqgs (SuperMatrix ∗ *A*, float ∗ *r*, float ∗ *c*, float *rowcnd*, float *colcnd*, float *amax*, char ∗ *equed*)

```
Purpose
=======


SLAQGS equilibrates a general sparse M by N matrix A using the row and
scaling factors in the vectors R and C.


See supermatrix.h for the definition of 'SuperMatrix' structure.


Arguments
=========
```

```
A       (input/output) SuperMatrix*
        On exit, the equilibrated matrix.  See EQUED for the form of
        the equilibrated matrix. The type of A can be:
 Stype = NC; Dtype = SLU_S; Mtype = GE.

R       (input) float*, dimension (A->nrow)
        The row scale factors for A.

C       (input) float*, dimension (A->ncol)
        The column scale factors for A.

ROWCND  (input) float
        Ratio of the smallest R(i) to the largest R(i).

COLCND  (input) float
        Ratio of the smallest C(i) to the largest C(i).

AMAX    (input) float
        Absolute value of largest matrix entry.

EQUED   (output) char*
        Specifies the form of equilibration that was done.
        = 'N':  No equilibration
        = 'R':  Row equilibration, i.e., A has been premultiplied by
                diag(R).
        = 'C':  Column equilibration, i.e., A has been postmultiplied
                by diag(C).
        = 'B':  Both row and column equilibration, i.e., A has been
                replaced by diag(R) * A * diag(C).

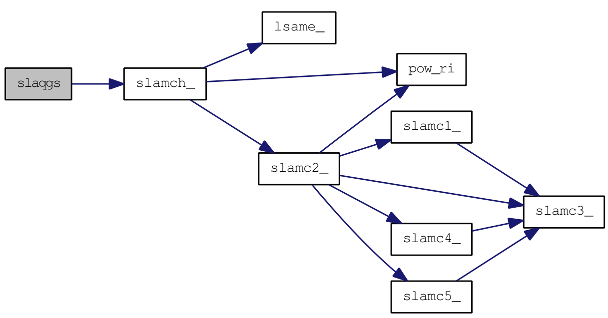
Internal Parameters
===================

THRESH is a threshold value used to decide if row or column scaling
should be done based on the ratio of the row or column scaling
factors.  If ROWCND < THRESH, row scaling is done, and if
COLCND < THRESH, column scaling is done.

LARGE and SMALL are threshold values used to decide if row scaling
should be done based on the absolute size of the largest matrix
element.  If AMAX > LARGE or AMAX < SMALL, row scaling is done.

=====================================================================
```

Here is the call graph for this function:



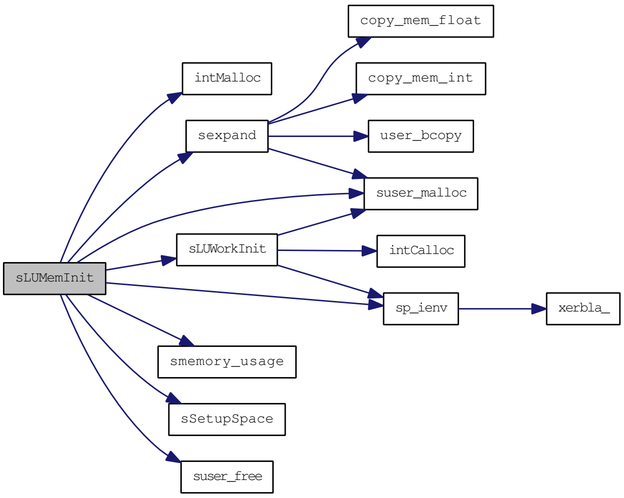
Here is the caller graph for this function:

4.115.3.31 int sLUMemInit (fact_t *fact*, void ∗ *work*, int *lwork*, int *m*, int *n*, int *annz*, int *panel_size*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, GlobalLU_t ∗ *Glu*, int ∗∗ *iwork*, float ∗∗ *dwork*)

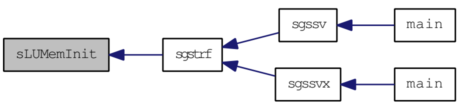
Memory-related.

```
For those unpredictable size, make a guess as FILL * nnz(A).
Return value:
    If lwork = -1, return the estimated amount of space required, plus n;
    otherwise, return the amount of space actually allocated when
    memory allocation failure occurred.
```

Here is the call graph for this function:



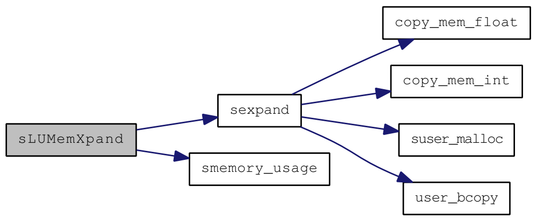
Here is the caller graph for this function:



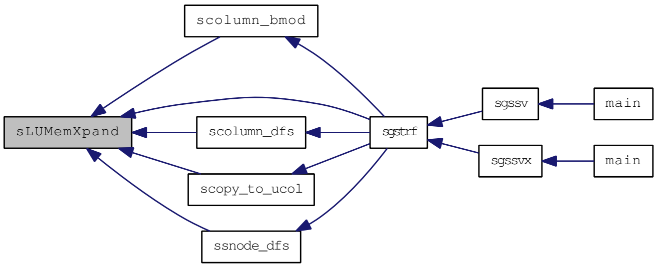
4.115.3.32 int sLUMemXpand (int *jcol*, int *next*, MemType *mem_type*, int ∗ *maxlen*, GlobalLU_t ∗ *Glu*)

```
Return value:   0 - successful return
              > 0 - number of bytes allocated when run out of space
```

Here is the call graph for this function:

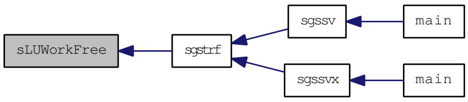


Here is the caller graph for this function:



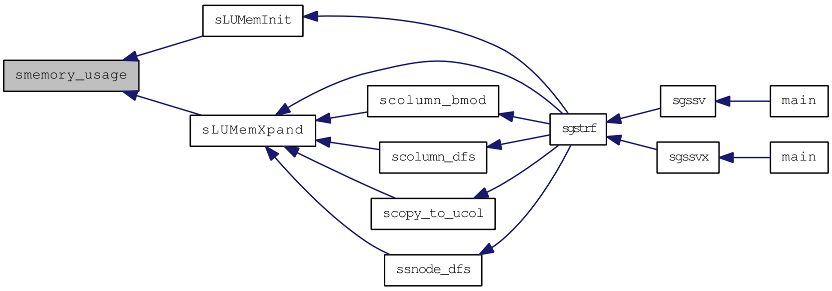
4.115.3.33 void sLUWorkFree (int ∗, float ∗, GlobalLU_t ∗)

Here is the caller graph for this function:



4.115.3.34 int smemory_usage (const *int*, const *int*, const *int*, const *int*)

Here is the caller graph for this function:



4.115.3.35 int sp_sgemm (char * *transa*, char * *transb*, int *m*, int *n*, int *k*, float *alpha*, SuperMatrix * *A*, float * *b*, int *ldb*, float *beta*, float * *c*, int *ldc*)

```
Purpose
  =======

  sp_s performs one of the matrix-matrix operations

     C := alpha*op( A )*op( B ) + beta*C,

  where  op( X ) is one of

     op( X ) = X   or   op( X ) = X'   or   op( X ) = conjg( X' ),

  alpha and beta are scalars, and A, B and C are matrices, with op( A )
  an m by k matrix,  op( B )  a  k by n matrix and  C an m by n matrix.

  Parameters
  ==========

  TRANSA - (input) char*
           On entry, TRANSA specifies the form of op( A ) to be used in
           the matrix multiplication as follows:
              TRANSA = 'N' or 'n',  op( A ) = A.
              TRANSA = 'T' or 't',  op( A ) = A'.
              TRANSA = 'C' or 'c',  op( A ) = conjg( A' ).
           Unchanged on exit.

  TRANSB - (input) char*
           On entry, TRANSB specifies the form of op( B ) to be used in
           the matrix multiplication as follows:
              TRANSB = 'N' or 'n',  op( B ) = B.
              TRANSB = 'T' or 't',  op( B ) = B'.
              TRANSB = 'C' or 'c',  op( B ) = conjg( B' ).
           Unchanged on exit.

  M      - (input) int
           On entry,  M  specifies  the number of rows of the matrix
    op( A ) and of the matrix C.  M must be at least zero.
    Unchanged on exit.

  N      - (input) int
           On entry,  N specifies the number of columns of the matrix
    op( B ) and the number of columns of the matrix C. N must be
    at least zero.
    Unchanged on exit.

  K      - (input) int
           On entry, K specifies the number of columns of the matrix
    op( A ) and the number of rows of the matrix op( B ). K must
    be at least  zero.
          Unchanged on exit.

  ALPHA  - (input) float
           On entry, ALPHA specifies the scalar alpha.
```

```
A       - (input) SuperMatrix*
          Matrix A with a sparse format, of dimension (A->nrow, A->ncol).
          Currently, the type of A can be:
              Stype = NC or NCP; Dtype = SLU_S; Mtype = GE.
          In the future, more general A can be handled.


B       - FLOAT PRECISION array of DIMENSION ( LDB, kb ), where kb is
          n when TRANSB = 'N' or 'n',  and is  k otherwise.
          Before entry with  TRANSB = 'N' or 'n',  the leading k by n
          part of the array B must contain the matrix B, otherwise
          the leading n by k part of the array B must contain the
          matrix B.
          Unchanged on exit.


LDB     - (input) int
          On entry, LDB specifies the first dimension of B as declared
          in the calling (sub) program. LDB must be at least max( 1, n ).
          Unchanged on exit.

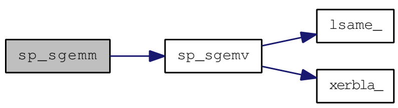

BETA    - (input) float
          On entry, BETA specifies the scalar beta. When BETA is
          supplied as zero then C need not be set on input.


C       - FLOAT PRECISION array of DIMENSION ( LDC, n ).
          Before entry, the leading m by n part of the array C must
          contain the matrix C,  except when beta is zero, in which
          case C need not be set on entry.
          On exit, the array C is overwritten by the m by n matrix
  ( alpha*op( A )*B + beta*C ).

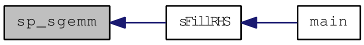

LDC     - (input) int
          On entry, LDC specifies the first dimension of C as declared
          in the calling (sub)program. LDC must be at least max(1,m).
          Unchanged on exit.


==== Sparse Level 3 Blas routine.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.115.3.36 int sp_sgemv (char * *trans*, float *alpha*, SuperMatrix * *A*, float * *x*, int *incx*, float *beta*, float * *y*, int *incy*)

```
Purpose
=======

sp_sgemv()  performs one of the matrix-vector operations
   y := alpha*A*x + beta*y,   or   y := alpha*A'*x + beta*y,
where alpha and beta are scalars, x and y are vectors and A is a
sparse A->nrow by A->ncol matrix.

Parameters
==========

TRANS  - (input) char*
         On entry, TRANS specifies the operation to be performed as
         follows:
            TRANS = 'N' or 'n'   y := alpha*A*x + beta*y.
            TRANS = 'T' or 't'   y := alpha*A'*x + beta*y.
            TRANS = 'C' or 'c'   y := alpha*A'*x + beta*y.

ALPHA  - (input) float
         On entry, ALPHA specifies the scalar alpha.

A      - (input) SuperMatrix*
         Matrix A with a sparse format, of dimension (A->nrow, A->ncol).
         Currently, the type of A can be:
             Stype = NC or NCP; Dtype = SLU_S; Mtype = GE.
         In the future, more general A can be handled.

X      - (input) float*, array of DIMENSION at least
         ( 1 + ( n - 1 )*abs( INCX ) ) when TRANS = 'N' or 'n'
         and at least
         ( 1 + ( m - 1 )*abs( INCX ) ) otherwise.
         Before entry, the incremented array X must contain the
         vector x.

INCX   - (input) int
         On entry, INCX specifies the increment for the elements of
         X. INCX must not be zero.

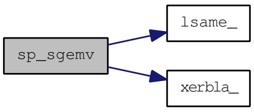
BETA   - (input) float
         On entry, BETA specifies the scalar beta. When BETA is
         supplied as zero then Y need not be set on input.

Y      - (output) float*,  array of DIMENSION at least
         ( 1 + ( m - 1 )*abs( INCY ) ) when TRANS = 'N' or 'n'
         and at least
         ( 1 + ( n - 1 )*abs( INCY ) ) otherwise.
         Before entry with BETA non-zero, the incremented array Y
         must contain the vector y. On exit, Y is overwritten by the
         updated vector y.

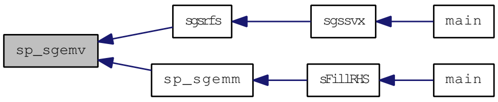
INCY   - (input) int
         On entry, INCY specifies the increment for the elements of
         Y. INCY must not be zero.
```

```
==== Sparse Level 2 Blas routine.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.115.3.37 int sp_strsv (char * *uplo*, char * *trans*, char * *diag*, SuperMatrix * *L*, SuperMatrix * *U*, float * *x*, SuperLUStat_t * *stat*, int * *info*)

```
Purpose
=======

sp_strsv() solves one of the systems of equations
    A*x = b,    or    A'*x = b,
where b and x are n element vectors and A is a sparse unit , or
non-unit, upper or lower triangular matrix.
No test for singularity or near-singularity is included in this
routine. Such tests must be performed before calling this routine.

Parameters
==========

uplo   - (input) char*
         On entry, uplo specifies whether the matrix is an upper or
          lower triangular matrix as follows:
             uplo = 'U' or 'u'   A is an upper triangular matrix.
             uplo = 'L' or 'l'   A is a lower triangular matrix.

trans  - (input) char*
          On entry, trans specifies the equations to be solved as
          follows:
             trans = 'N' or 'n'   A*x = b.
             trans = 'T' or 't'   A'*x = b.
             trans = 'C' or 'c'   A'*x = b.

diag   - (input) char*
          On entry, diag specifies whether or not A is unit
          triangular as follows:
             diag = 'U' or 'u'   A is assumed to be unit triangular.
             diag = 'N' or 'n'   A is not assumed to be unit
                                 triangular.
```

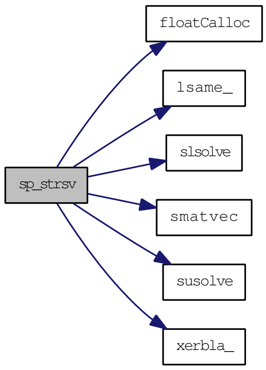
```
L       - (input) SuperMatrix*
    The factor L from the factorization Pr*A*Pc=L*U. Use
          compressed row subscripts storage for supernodes,
          i.e., L has types: Stype = SC, Dtype = SLU_S, Mtype = TRLU.

U       - (input) SuperMatrix*
     The factor U from the factorization Pr*A*Pc=L*U.
     U has types: Stype = NC, Dtype = SLU_S, Mtype = TRU.

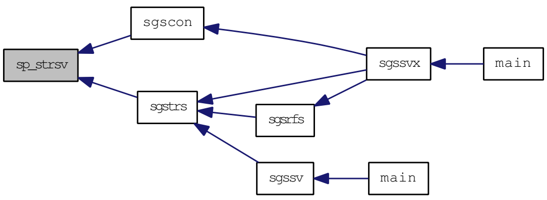
x       - (input/output) float*
          Before entry, the incremented array X must contain the n
          element right-hand side vector b. On exit, X is overwritten
          with the solution vector x.

info    - (output) int*
          If *info = -i, the i-th argument had an illegal value.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.115.3.38 void spanel_bmod (const int *m*, const int *w*, const int *jcol*, const int *nseg*, float ∗ *dense*, float ∗ *tempv*, int ∗ *segrep*, int ∗ *repfnz*, GlobalLU_t ∗ *Glu*, SuperLUStat_t ∗ *stat*)

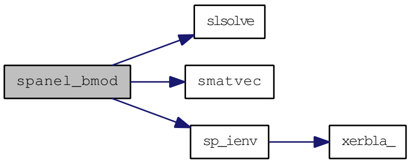
```
Purpose
=======

   Performs numeric block updates (sup-panel) in topological order.
   It features: col-col, 2cols-col, 3cols-col, and sup-col updates.
   Special processing on the supernodal portion of L[*,j]
```

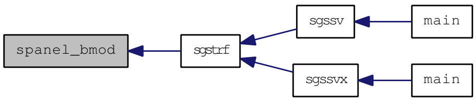
```
Before entering this routine, the original nonzeros in the panel
were already copied into the spa[m,w].

Updated/Output parameters-
dense[0:m-1,w]: L[*,j:j+w-1] and U[*,j:j+w-1] are returned
collectively in the m-by-w vector dense[*].
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.115.3.39 void spanel_dfs (const int *m*, const int *w*, const int *jcol*, SuperMatrix ∗ *A*, int ∗ *perm_r*, int ∗ *nseg*, float ∗ *dense*, int ∗ *panel_lsub*, int ∗ *segrep*, int ∗ *repfnz*, int ∗ *xprune*, int ∗ *marker*, int ∗ *parent*, int ∗ *xplore*, GlobalLU_t ∗ *Glu*)

```
Purpose
=======

  Performs a symbolic factorization on a panel of columns [jcol, jcol+w).

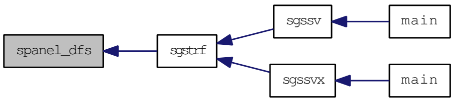
  A supernode representative is the last column of a supernode.
  The nonzeros in U[*,j] are segments that end at supernodal
  representatives.

  The routine returns one list of the supernodal representatives
  in topological order of the dfs that generates them. This list is
  a superset of the topological order of each individual column within
  the panel.
  The location of the first nonzero in each supernodal segment
  (supernodal entry location) is also returned. Each column has a
  separate list for this purpose.

  Two marker arrays are used for dfs:
    marker[i] == jj, if i was visited during dfs of current column jj;
    marker1[i] >= jcol, if i was visited by earlier columns in this panel;

  marker: A-row --> A-row/col (0/1)
  repfnz: SuperA-col --> PA-row
  parent: SuperA-col --> SuperA-col
  xplore: SuperA-col --> index to L-structure
```

Here is the caller graph for this function:



4.115.3.40 float sPivotGrowth (int *ncols*, SuperMatrix ∗ *A*, int ∗ *perm_c*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*)

```
Purpose
=======

Compute the reciprocal pivot growth factor of the leading ncols columns
of the matrix, using the formula:
    min_j ( max_i(abs(A_ij)) / max_i(abs(U_ij)) )

Arguments
=========

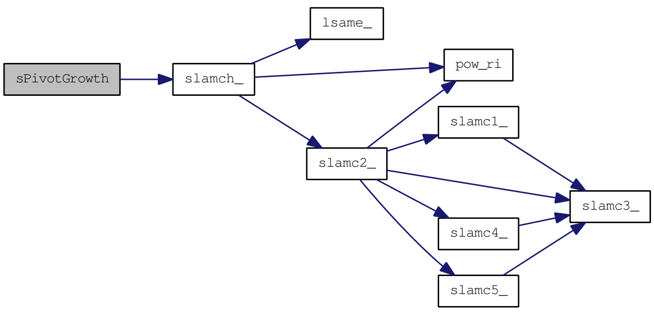
ncols    (input) int
         The number of columns of matrices A, L and U.

A        (input) SuperMatrix*
   Original matrix A, permuted by columns, of dimension
         (A->nrow, A->ncol). The type of A can be:
         Stype = NC; Dtype = SLU_S; Mtype = GE.

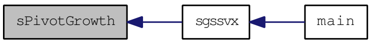
L        (output) SuperMatrix*
         The factor L from the factorization Pr*A=L*U; use compressed row
         subscripts storage for supernodes, i.e., L has type:
         Stype = SC; Dtype = SLU_S; Mtype = TRLU.

U        (output) SuperMatrix*
   The factor U from the factorization Pr*A*Pc=L*U. Use column-wise
         storage scheme, i.e., U has types: Stype = NC;
         Dtype = SLU_S; Mtype = TRU.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.115.3.41 int spivotL (const int *jcol*, const float *u*, int ∗ *usepr*, int ∗ *perm_r*, int ∗ *iperm_r*, int ∗ *iperm_c*, int ∗ *pivrow*, GlobalLU_t ∗ *Glu*, SuperLUStat_t ∗ *stat*)

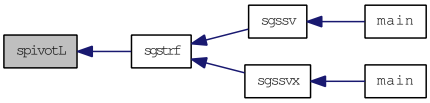
```
Purpose
=======
  Performs the numerical pivoting on the current column of L,
  and the CDIV operation.


  Pivot policy:
  (1) Compute thresh = u * max_(i>=j) abs(A_ij);
  (2) IF user specifies pivot row k and abs(A_kj) >= thresh THEN
          pivot row = k;
      ELSE IF abs(A_jj) >= thresh THEN
          pivot row = j;
      ELSE
          pivot row = m;


  Note: If you absolutely want to use a given pivot order, then set u=0.0.


  Return value: 0      success;
                i > 0  U(i,i) is exactly zero.
```

Here is the caller graph for this function:



4.115.3.42 void sPrint_CompCol_Matrix (char ∗, SuperMatrix ∗)

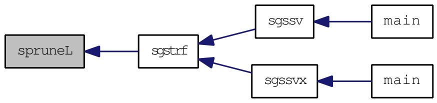
4.115.3.43 void sPrint_Dense_Matrix (char ∗, SuperMatrix ∗)

4.115.3.44 void sPrint_SuperNode_Matrix (char ∗, SuperMatrix ∗)

4.115.3.45 void spruneL (const int *jcol*, const int ∗ *perm_r*, const int *pivrow*, const int *nseg*, const int ∗ *segrep*, const int ∗ *repfnz*, int ∗ *xprune*, GlobalLU_t ∗ *Glu*)

```
Purpose
=======
  Prunes the L-structure of supernodes whose L-structure
  contains the current pivot row "pivrow"
```

Here is the caller graph for this function:

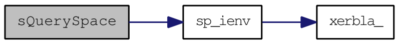


4.115.3.46 int sQuerySpace (SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, mem_usage_t ∗ *mem_usage*)

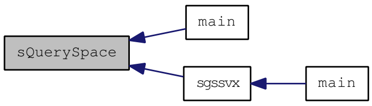
```
mem_usage consists of the following fields:
```

- `for_lu (float)`
  `The amount of space used in bytes for the L data structures.`
- `total_needed (float)`
  `The amount of space needed in bytes to perform factorization.`
- `expansions (int)`
  `Number of memory expansions during the LU factorization.`

Here is the call graph for this function:

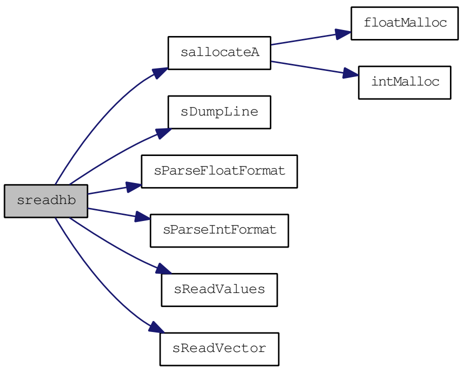


Here is the caller graph for this function:

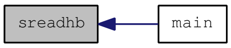


4.115.3.47 void sreadhb (int ∗, int ∗, int ∗, float ∗∗, int ∗∗, int ∗∗)

Here is the call graph for this function:



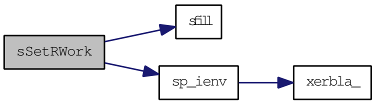
Here is the caller graph for this function:



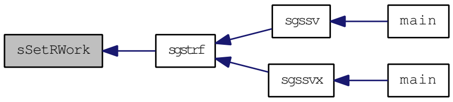
4.115.3.48 void sreadmt (int ∗, int ∗, int ∗, float ∗∗, int ∗∗, int ∗∗)

4.115.3.49 void sSetRWork (int, int, float ∗, float ∗∗, float ∗∗)

Here is the call graph for this function:

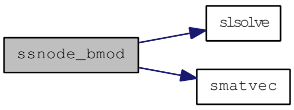


Here is the caller graph for this function:

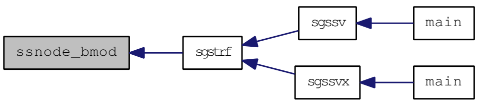


4.115.3.50 int ssnode_bmod (const *int*, const *int*, const *int*, float ∗, float ∗, GlobalLU_t ∗, SuperLUStat_t ∗)

Here is the call graph for this function:



Here is the caller graph for this function:



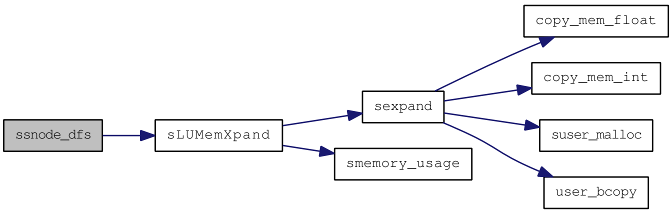
4.115.3.51 int ssnode_dfs (const int *jcol*, const int *kcol*, const int ∗ *asub*, const int ∗ *xa_begin*, const int ∗ *xa_end*, int ∗ *xprune*, int ∗ *marker*, GlobalLU_t ∗ *Glu*)

```
Purpose
=======
   ssnode_dfs() - Determine the union of the row structures of those
```

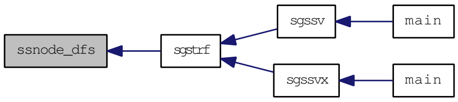
```
    columns within the relaxed snode.
    Note: The relaxed snodes are leaves of the supernodal etree, therefore,
    the portion outside the rectangular supernode must be zero.

 Return value
 ============
     0   success;
    >0   number of bytes allocated when run out of memory.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.116 SRC/slu_util.h File Reference

Utility header file.

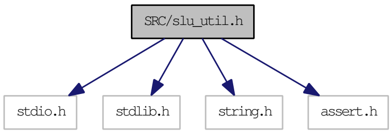
`#include <stdio.h>`

`#include <stdlib.h>`

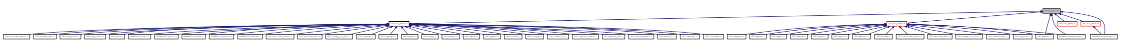
`#include <string.h>`

`#include <assert.h>`

Include dependency graph for slu_util.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct superlu_options_t
- struct SuperLUStat_t
- struct mem_usage_t

Defines

- #define FIRSTCOL_OF_SNODE(i) (xsup[i])
- #define NO_MARKER 3
- #define NUM_TEMPV(m, w, t, b) (SUPERLU_MAX(m, (t + b)*w))
- #define USER_ABORT(msg) superlu_abort_and_exit(msg)
- #define ABORT(err_msg)
- #define USER_MALLOC(size) superlu_malloc(size)
- #define SUPERLU_MALLOC(size) USER_MALLOC(size)
- #define USER_FREE(addr) superlu_free(addr)
- #define SUPERLU_FREE(addr) USER_FREE(addr)
- #define CHECK_MALLOC(where)
- #define SUPERLU_MAX(x, y) ((x) > (y) ? (x) : (y))
- #define SUPERLU_MIN(x, y) ((x) < (y) ? (x) : (y))
- #define L_SUB_START(col) (Lstore → rowind_colptr[col])
- #define L_SUB(ptr) (Lstore → rowind[ptr])
- #define L_NZ_START(col) (Lstore → nzval_colptr[col])
- #define L_FST_SUPC(superno) (Lstore → sup_to_col[superno])
- #define U_NZ_START(col) (Ustore → colptr[col])
- #define U_SUB(ptr) (Ustore → rowind[ptr])
- #define EMPTY (-1)
- #define FALSE 0
- #define TRUE 1

Typedefs

- typedef float flops_t
- typedef unsigned char Logical

Enumerations

- enum yes_no_t { NO, YES }
- enum fact_t { DOFACT, SamePattern, SamePattern_SameRowPerm, FACTORED }
- enum rowperm_t { NOROWPERM, LargeDiag, MY_PERMR }
- enum colperm_t {

  NATURAL, MMD_ATA, MMD_AT_PLUS_A, COLAMD,

  MY_PERMC }
- enum trans_t { NOTRANS, TRANS, CONJ }
- enum DiagScale_t { NOEQUIL, ROW, COL, BOTH }
- enum IterRefine_t { NOREFINE, SINGLE = 1, DOUBLE, EXTRA }
- enum MemType { LUSUP, UCOL, LSUB, USUB }
- enum stack_end_t { HEAD, TAIL }
- enum LU_space_t { SYSTEM, USER }
- enum PhaseType {

  COLPERM, RELAX, ETREE, EQUIL,

  FACT, RCOND, SOLVE, REFINE,

  TRSV, GEMV, FERR, NPHASES }

Functions

- void Destroy_SuperMatrix_Store (SuperMatrix ∗)

  Deallocate the structure pointing to the actual storage of the matrix.

- void Destroy_CompCol_Matrix (SuperMatrix ∗)
- void Destroy_CompRow_Matrix (SuperMatrix ∗)
- void Destroy_SuperNode_Matrix (SuperMatrix ∗)
- void Destroy_CompCol_Permuted (SuperMatrix ∗)

  A is of type Stype==NCP.

- void Destroy_Dense_Matrix (SuperMatrix ∗)

  A is of type Stype==DN.

- void get_perm_c (int, SuperMatrix ∗, int ∗)
- void set_default_options (superlu_options_t ∗options)

  Set the default values for the options argument.

- void sp_preorder (superlu_options_t ∗, SuperMatrix ∗, int ∗, int ∗, SuperMatrix ∗)
- void superlu_abort_and_exit (char ∗)

  Global statistics variale.

- void ∗ superlu_malloc (size_t)
- int ∗ intMalloc (int)

- int ∗ intCalloc (int)
- void superlu_free (void ∗)
- void SetIWork (int, int, int, int ∗, int ∗∗, int ∗∗, int ∗∗, int ∗∗, int ∗∗, int ∗∗, int ∗∗)

    Set up pointers for integer working arrays.

- int sp_coletree (int ∗, int ∗, int ∗, int, int, int ∗)
- void relax_snode (const int, int ∗, const int, int ∗, int ∗)
- void heap_relax_snode (const int, int ∗, const int, int ∗, int ∗)
- void resetrep_col (const int, const int ∗, int ∗)

    Reset repfnz[] for the current column.

- int spcoletree (int ∗, int ∗, int ∗, int, int, int ∗)
- int ∗ TreePostorder (int, int ∗)
- double SuperLU_timer_ ()

    Timer function.

- int sp_ienv (int)
- int lsame_ (char ∗, char ∗)
- int xerbla_ (char ∗, int ∗)
- void ifill (int ∗, int, int)

    Fills an integer array with a given value.

- void snode_profile (int, int ∗)
- void super_stats (int, int ∗)
- void PrintSumm (char ∗, int, int, int)

    Print a summary of the testing results.

- void StatInit (SuperLUStat_t ∗)
- void StatPrint (SuperLUStat_t ∗)
- void StatFree (SuperLUStat_t ∗)
- void print_panel_seg (int, int, int, int, int ∗, int ∗)

    Diagnostic print of segment info after panel_dfs().

- void check_repfnz (int, int, int, int ∗)

    Check whether repfnz[] == EMPTY after reset.

4.116.1 Detailed Description

– SuperLU routine (version 3.1) – Univ. of California Berkeley, Xerox Palo Alto Research Center, and Lawrence Berkeley National Lab. August 1, 2008

4.116.2 Define Documentation

4.116.2.1 #define ABORT(err_msg)

Value:

```
{ char msg[256];\
   sprintf(msg,"%s at line %d in file %s\n",err_msg,__LINE__, __FILE__);\
   USER_ABORT(msg); }
```

4.116.2.2 #define CHECK_MALLOC(where)

Value:

```
{                   \
    extern int superlu_malloc_total;        \
    printf("%s: malloc_total %d Bytes\n",     \
     where, superlu_malloc_total); \
}
```

4.116.2.3 #define EMPTY (-1)

4.116.2.4 #define FALSE 0

4.116.2.5 #define FIRSTCOL_OF_SNODE(i) (xsup[i])

4.116.2.6 #define L_FST_SUPC(superno) (Lstore → sup_to_col[superno])

4.116.2.7 #define L_NZ_START(col) (Lstore → nzval_colptr[col])

4.116.2.8 #define L_SUB(ptr) (Lstore → rowind[ptr])

4.116.2.9 #define L_SUB_START(col) (Lstore → rowind_colptr[col])

4.116.2.10 #define NO_MARKER 3

4.116.2.11 #define NUM_TEMPV(m, w, t, b) (SUPERLU_MAX(m, (t + b)*w))

4.116.2.12 #define SUPERLU_FREE(addr) USER_FREE(addr)

4.116.2.13 #define SUPERLU_MALLOC(size) USER_MALLOC(size)

4.116.2.14 #define SUPERLU_MAX(x, y) ((x) > (y) ? (x) : (y))

4.116.2.15 #define SUPERLU_MIN(x, y) ((x) < (y) ? (x) : (y))

4.116.2.16 #define TRUE 1

4.116.2.17 #define U_NZ_START(col) (Ustore → colptr[col])

4.116.2.18 #define U_SUB(ptr) (Ustore → rowind[ptr])

4.116.2.19 #define USER_ABORT(msg) superlu_abort_and_exit(msg)

4.116.2.20 #define USER_FREE(addr) superlu_free(addr)

4.116.2.21 #define USER_MALLOC(size) superlu_malloc(size)

4.116.3 Typedef Documentation

4.116.3.1 typedef float flops_t

4.116.3.2 typedef unsigned char Logical

4.116.4 Enumeration Type Documentation

4.116.4.1 enum colperm_t

Enumerator:

NATURAL

MMD_ATA

MMD_AT_PLUS_A
COLAMD
MY_PERMC

4.116.4.2 enum DiagScale_t

Enumerator:

NOEQUIL
ROW
COL
BOTH

4.116.4.3 enum fact_t

Enumerator:

DOFACT
SamePattern
SamePattern_SameRowPerm
FACTORED

4.116.4.4 enum IterRefine_t

Enumerator:

NOREFINE
SINGLE
DOUBLE
EXTRA

4.116.4.5 enum LU_space_t

Enumerator:

SYSTEM
USER

4.116.4.6 enum MemType

Enumerator:

LUSUP
UCOL
LSUB
USUB

4.116.4.7 enum PhaseType

Enumerator:

COLPERM
RELAX
ETREE
EQUIL
FACT
RCOND
SOLVE
REFINE
TRSV
GEMV
FERR
NPHASES

4.116.4.8 enum rowperm_t

Enumerator:

NOROWPERM
LargeDiag
MY_PERMR

4.116.4.9 enum stack_end_t

Enumerator:

HEAD
TAIL

4.116.4.10 enum trans_t

Enumerator:

NOTRANS
TRANS
CONJ

4.116.4.11 enum yes_no_t

Enumerator:

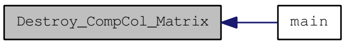
NO
YES

4.116.5 Function Documentation

4.116.5.1 void check_repfnz (int, int, int, int ∗)

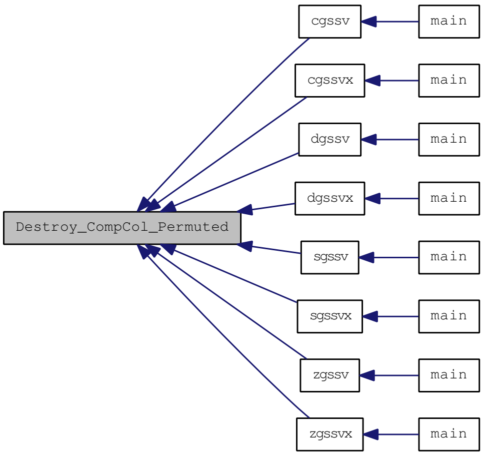
4.116.5.2 void Destroy_CompCol_Matrix (SuperMatrix ∗)

Here is the caller graph for this function:



4.116.5.3 void Destroy_CompCol_Permuted (SuperMatrix ∗)

Here is the caller graph for this function:



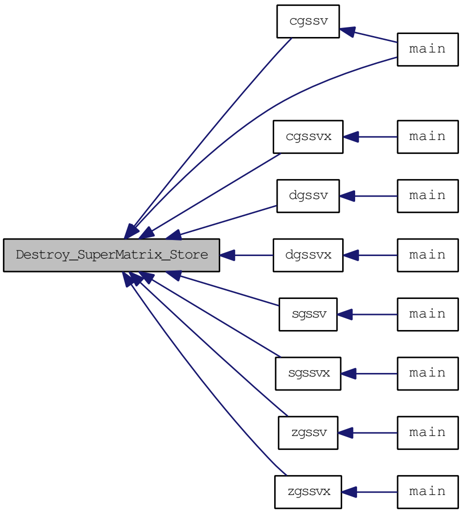
4.116.5.4 void Destroy_CompRow_Matrix (SuperMatrix ∗)

4.116.5.5 void Destroy_Dense_Matrix (SuperMatrix ∗)

Here is the caller graph for this function:

4.116.5.6 void Destroy_SuperMatrix_Store (SuperMatrix ∗)

Here is the caller graph for this function:



4.116.5.7 void Destroy_SuperNode_Matrix (SuperMatrix ∗)

Here is the caller graph for this function:

4.116.5.8 void get_perm_c (int *ispec*, SuperMatrix ∗ *A*, int ∗ *perm_c*)

```
Purpose
=======


GET_PERM_C obtains a permutation matrix Pc, by applying the multiple
minimum degree ordering code by Joseph Liu to matrix A'*A or A+A'.
or using approximate minimum degree column ordering by Davis et. al.
The LU factorization of A*Pc tends to have less fill than the LU
factorization of A.

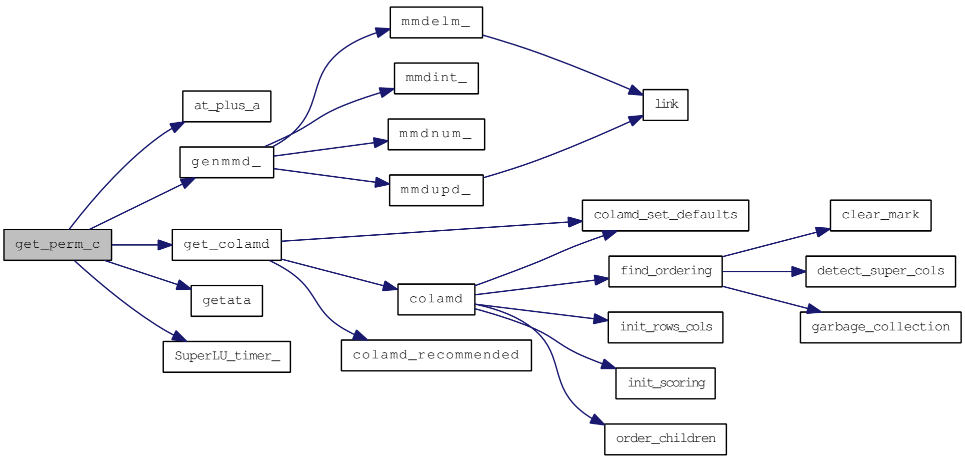

Arguments
=========


ispec   (input) int
        Specifies the type of column ordering to reduce fill:
        = 1: minimum degree on the structure of A^T * A
        = 2: minimum degree on the structure of A^T + A
        = 3: approximate minimum degree for unsymmetric matrices
        If ispec == 0, the natural ordering (i.e., Pc = I) is returned.
```

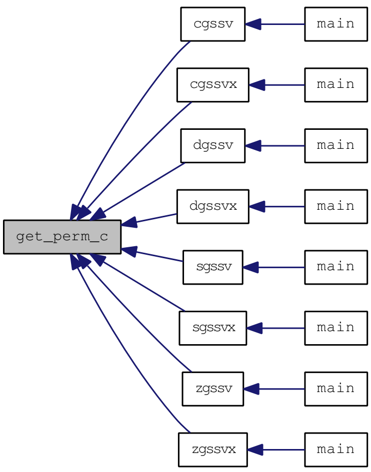
```
A        (input) SuperMatrix*
         Matrix A in A*X=B, of dimension (A->nrow, A->ncol). The number
         of the linear equations is A->nrow. Currently, the type of A
         can be: Stype = NC; Dtype = _D; Mtype = GE. In the future,
         more general A can be handled.

perm_c  (output) int*
  Column permutation vector of size A->ncol, which defines the
         permutation matrix Pc; perm_c[i] = j means column i of A is
         in position j in A*Pc.
```

Here is the call graph for this function:



Here is the caller graph for this function:

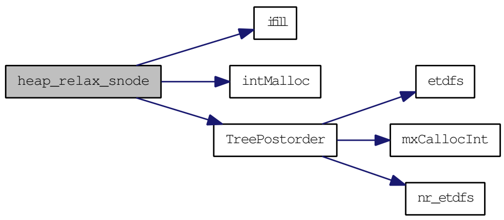


4.116.5.9 void heap_relax_snode (const int *n*, int ∗ *et*, const int *relax_columns*, int ∗ *descendants*, int ∗ *relax_end*)

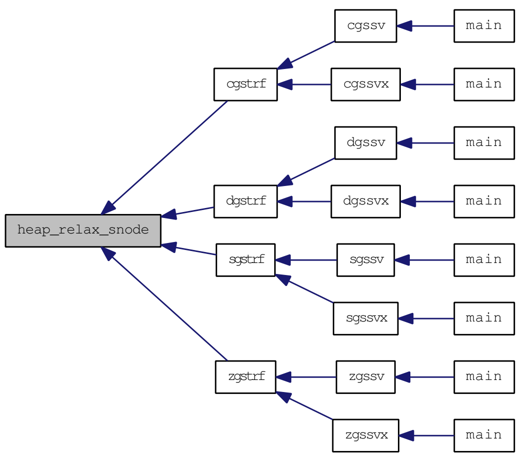
```
Purpose
```

```
=======
   relax_snode() - Identify the initial relaxed supernodes, assuming that
   the matrix has been reordered according to the postorder of the etree.
```

Here is the call graph for this function:

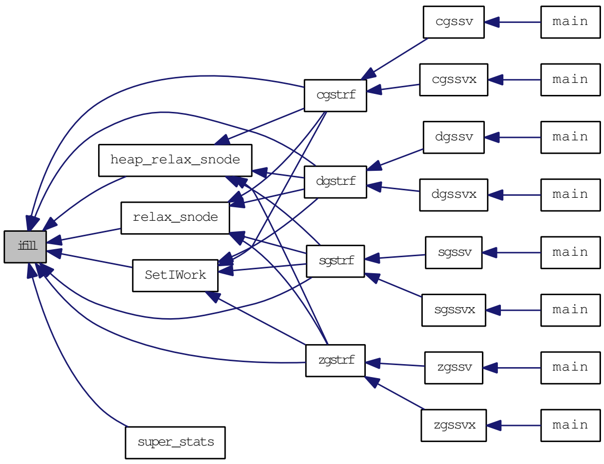


Here is the caller graph for this function:



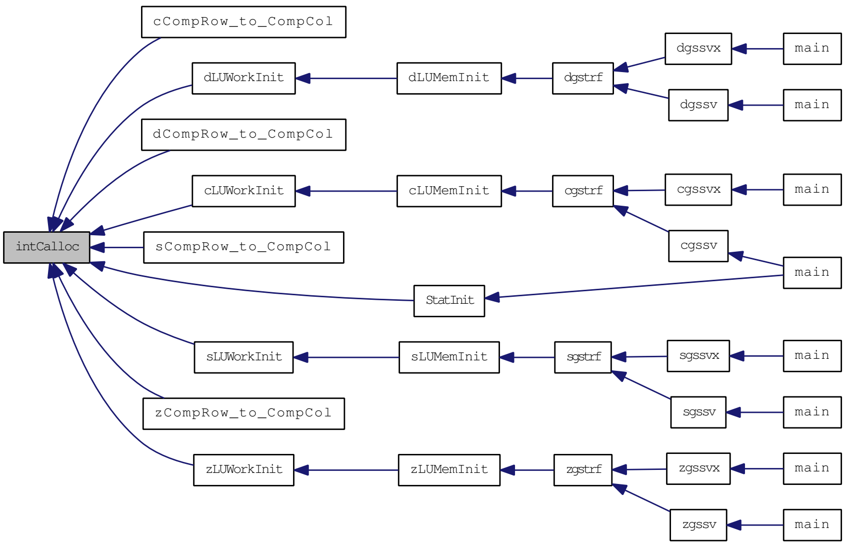
4.116.5.10 void ifill (int *, int, int)

Here is the caller graph for this function:



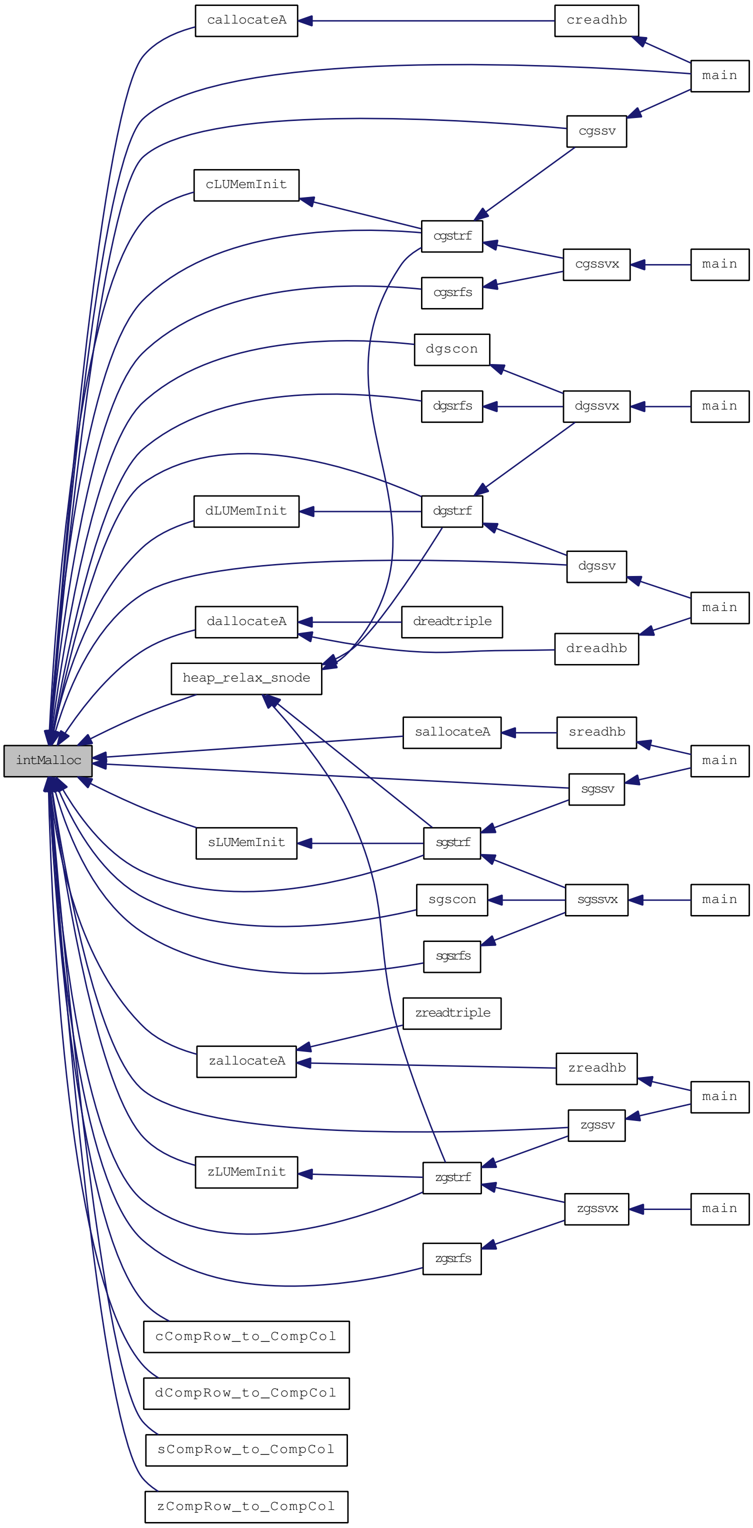
4.116.5.11 int* intCalloc (int)

Here is the caller graph for this function:



4.116.5.12 int∗ intMalloc (int)

Here is the caller graph for this function:



4.116.5.13 int lsame_ (char * *ca*, char * *cb*)

```
Purpose
=======

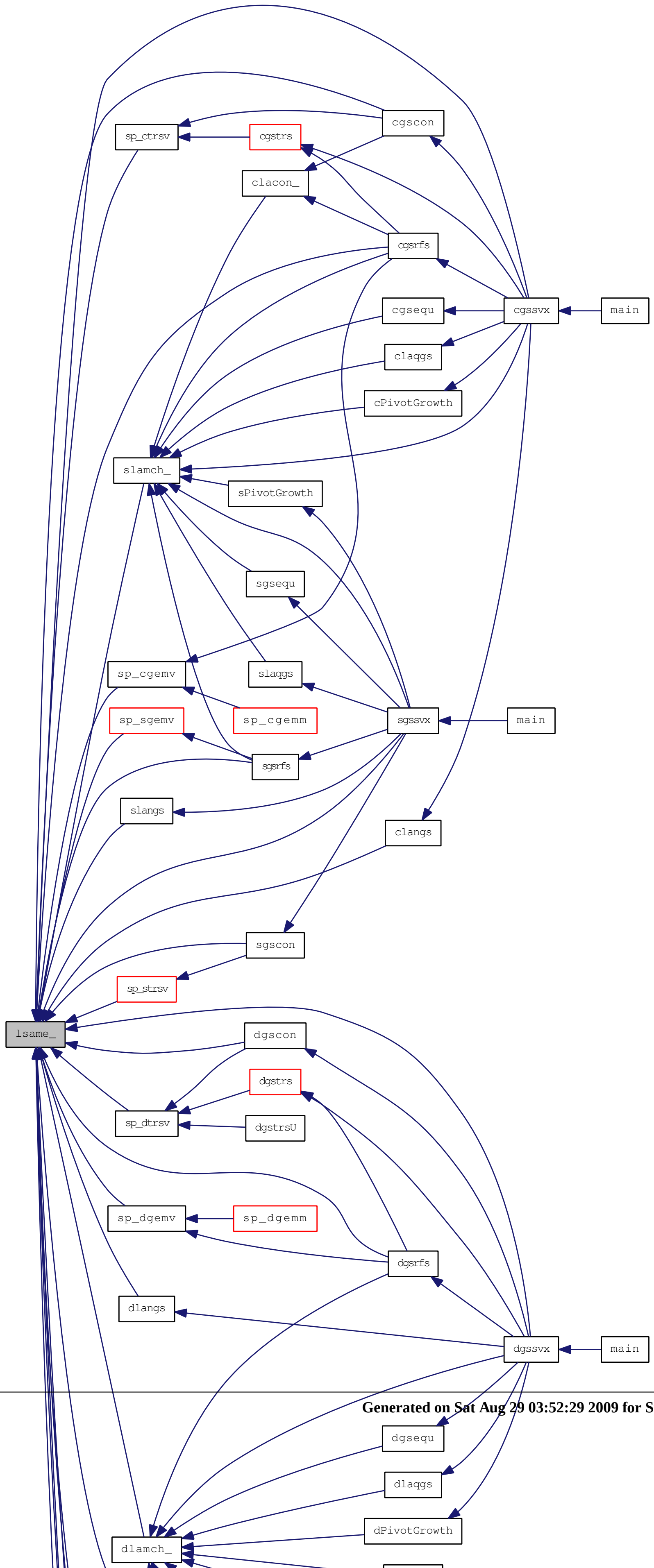
LSAME returns .TRUE. if CA is the same letter as CB regardless of case.

Arguments
=========

CA      (input) CHARACTER*1
CB      (input) CHARACTER*1
        CA and CB specify the single characters to be compared.

=====================================================================
```

Here is the caller graph for this function:



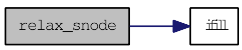
4.116.5.14 void print_panel_seg (int, int, int, int, int ∗, int ∗)

4.116.5.15 void PrintSumm (char ∗, int, int, int)

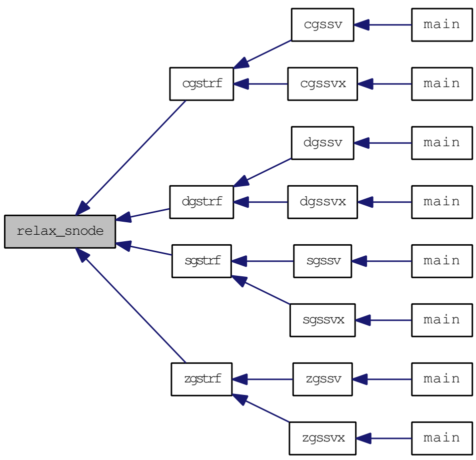
4.116.5.16 void relax_snode (const int *n*, int ∗ *et*, const int *relax_columns*, int ∗ *descendants*, int ∗ *relax_end*)

```
Purpose
=======
   relax_snode() - Identify the initial relaxed supernodes, assuming that
   the matrix has been reordered according to the postorder of the etree.
```

Here is the call graph for this function:

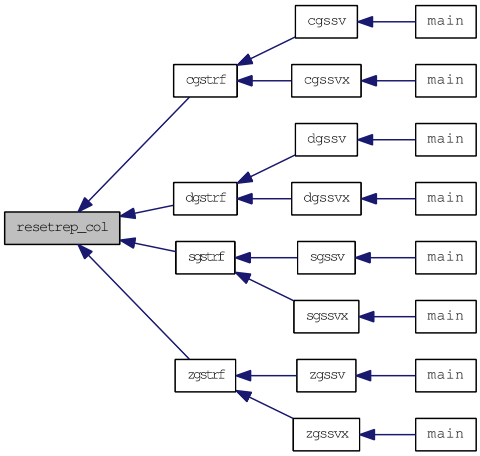


Here is the caller graph for this function:



4.116.5.17 void resetrep_col (const *int*, const int ∗, int ∗)

Here is the caller graph for this function:

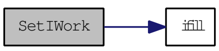


4.116.5.18 void set_default_options (superlu_options_t ∗ *options*)

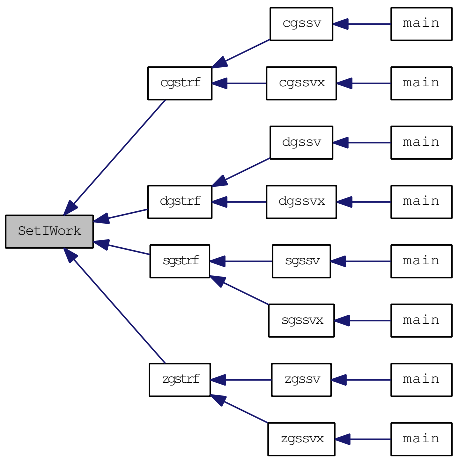
Here is the caller graph for this function:

4.116.5.19 void SetIWork (int, int, int, int ∗, int ∗∗, int ∗∗, int ∗∗, int ∗∗, int ∗∗, int ∗∗, int ∗∗)

Here is the call graph for this function:



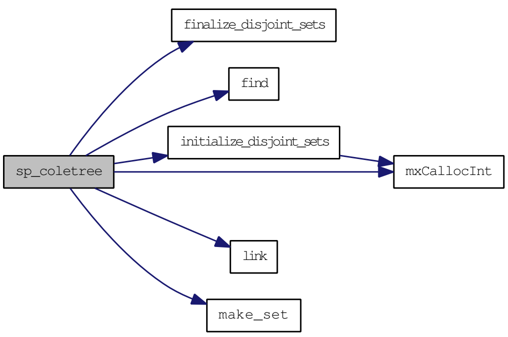
Here is the caller graph for this function:



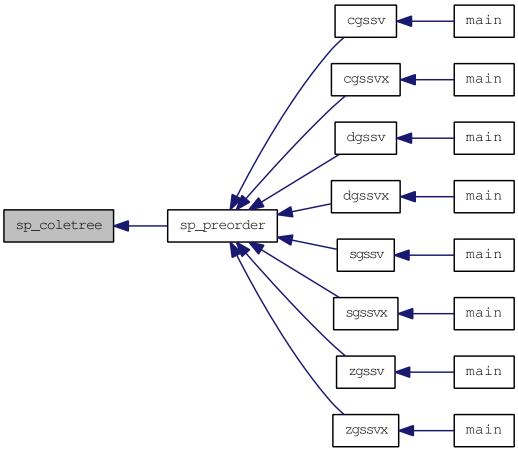
4.116.5.20 void snode_profile (int, int ∗)

4.116.5.21 int sp_coletree (int ∗, int ∗, int ∗, int, int, int ∗)

Here is the call graph for this function:



Here is the caller graph for this function:



4.116.5.22 int sp_ienv (int *ispec*)

```
Purpose
=======

sp_ienv() is inquired to choose machine-dependent parameters for the
local environment. See ISPEC for a description of the parameters.

This version provides a set of parameters which should give good,
but not optimal, performance on many of the currently available
computers.  Users are encouraged to modify this subroutine to set
the tuning parameters for their particular machine using the option
and problem size information in the arguments.

Arguments
=========

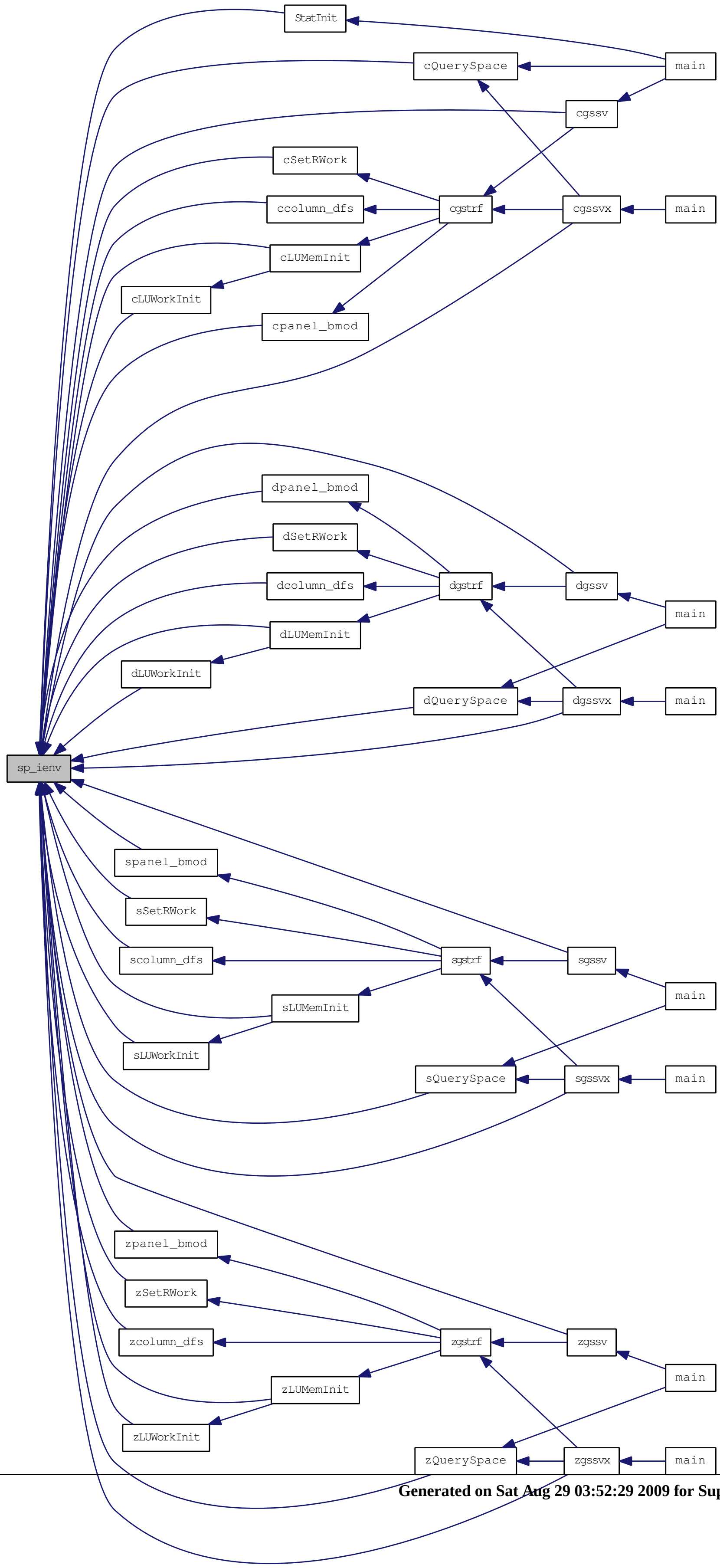
ISPEC   (input) int
        Specifies the parameter to be returned as the value of SP_IENV.
        = 1: the panel size w; a panel consists of w consecutive
     columns of matrix A in the process of Gaussian elimination.
 The best value depends on machine's cache characters.
        = 2: the relaxation parameter relax; if the number of
     nodes (columns) in a subtree of the elimination tree is less
 than relax, this subtree is considered as one supernode,
 regardless of their row structures.
        = 3: the maximum size for a supernode;
    = 4: the minimum row dimension for 2-D blocking to be used;
    = 5: the minimum column dimension for 2-D blocking to be used;
    = 6: the estimated fills factor for L and U, compared with A;

  (SP_IENV) (output) int
        >= 0: the value of the parameter specified by ISPEC
        < 0:  if SP_IENV = -k, the k-th argument had an illegal value.
```

===

Here is the call graph for this function:

Here is the caller graph for this function:



4.116.5.23 void sp_preorder (superlu_options_t * *options*, SuperMatrix * *A*, int * *perm_c*, int * *etree*, SuperMatrix * *AC*)

```
Purpose
=======

sp_preorder() permutes the columns of the original matrix. It performs
the following steps:

   1. Apply column permutation perm_c[] to A's column pointers to form AC;

   2. If options->Fact = DOFACT, then
      (1) Compute column elimination tree etree[] of AC'AC;
      (2) Post order etree[] to get a postordered elimination tree etree[],
          and a postorder permutation post[];
      (3) Apply post[] permutation to columns of AC;
      (4) Overwrite perm_c[] with the product perm_c * post.

Arguments
=========

options (input) superlu_options_t*
        Specifies whether or not the elimination tree will be re-used.
        If options->Fact == DOFACT, this means first time factor A,
        etree is computed, postered, and output.
        Otherwise, re-factor A, etree is input, unchanged on exit.

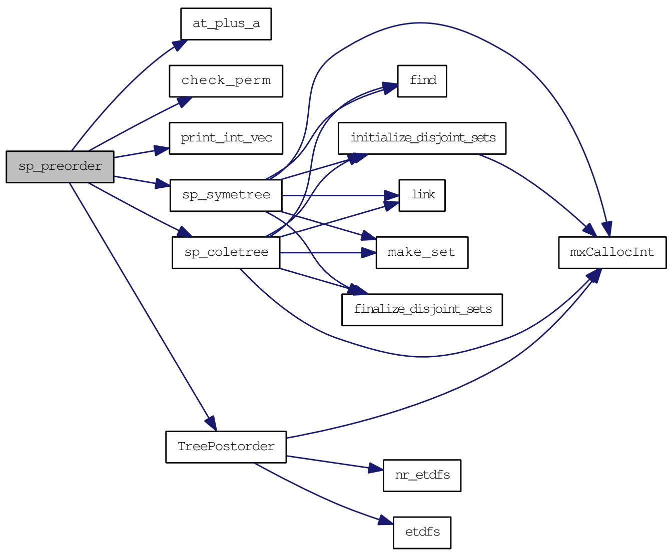
A       (input) SuperMatrix*
        Matrix A in A*X=B, of dimension (A->nrow, A->ncol). The number
        of the linear equations is A->nrow. Currently, the type of A can be:
        Stype = NC or SLU_NCP; Mtype = SLU_GE.
        In the future, more general A may be handled.

perm_c  (input/output) int*
  Column permutation vector of size A->ncol, which defines the
        permutation matrix Pc; perm_c[i] = j means column i of A is
        in position j in A*Pc.
        If options->Fact == DOFACT, perm_c is both input and output.
        On output, it is changed according to a postorder of etree.
        Otherwise, perm_c is input.

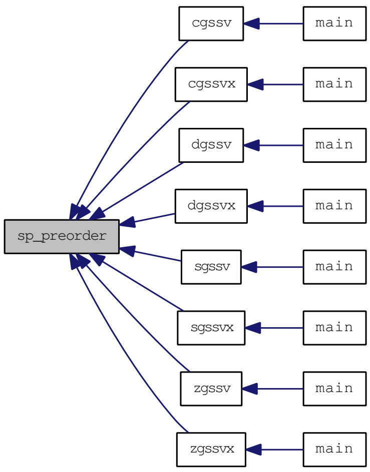
etree   (input/output) int*
        Elimination tree of Pc'*A'*A*Pc, dimension A->ncol.
        If options->Fact == DOFACT, etree is an output argument,
        otherwise it is an input argument.
        Note: etree is a vector of parent pointers for a forest whose
        vertices are the integers 0 to A->ncol-1; etree[root]==A->ncol.

AC      (output) SuperMatrix*
        The resulting matrix after applied the column permutation
        perm_c[] to matrix A. The type of AC can be:
        Stype = SLU_NCP; Dtype = A->Dtype; Mtype = SLU_GE.
```

Here is the call graph for this function:



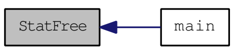
Here is the caller graph for this function:



4.116.5.24 int spcoletree (int ∗, int ∗, int ∗, int, int, int ∗)

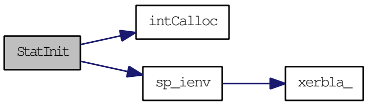
4.116.5.25 void StatFree (SuperLUStat_t ∗)

Here is the caller graph for this function:

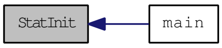


4.116.5.26 void StatInit (SuperLUStat_t ∗)

Here is the call graph for this function:

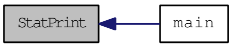


Here is the caller graph for this function:



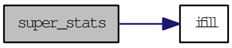
4.116.5.27 void StatPrint (SuperLUStat_t ∗)

Here is the caller graph for this function:



4.116.5.28 void super_stats (int, int ∗)

Here is the call graph for this function:



4.116.5.29 void superlu_abort_and_exit (char ∗)

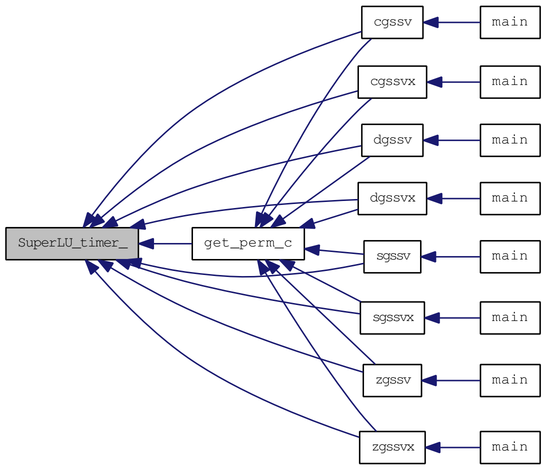
4.116.5.30 void superlu_free (void ∗)

4.116.5.31 void∗ superlu_malloc (size_t *size*)

Precision-independent memory-related routines. (Shared by [sdcz]memory.c)

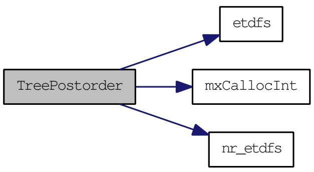
4.116.5.32 double SuperLU_timer_ ()

Here is the caller graph for this function:

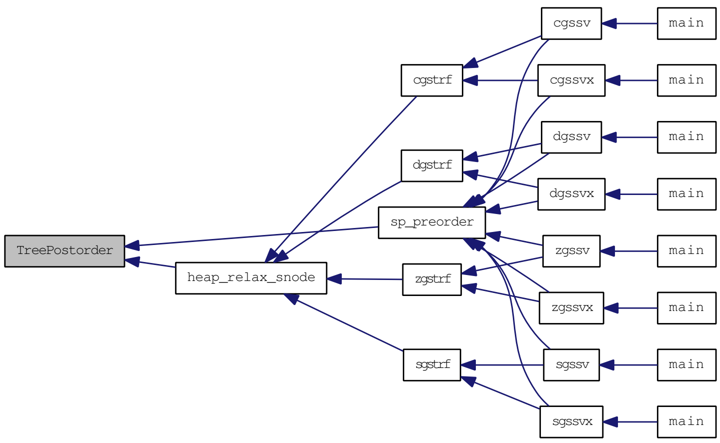


4.116.5.33 int∗ TreePostorder (int, int ∗)

Here is the call graph for this function:

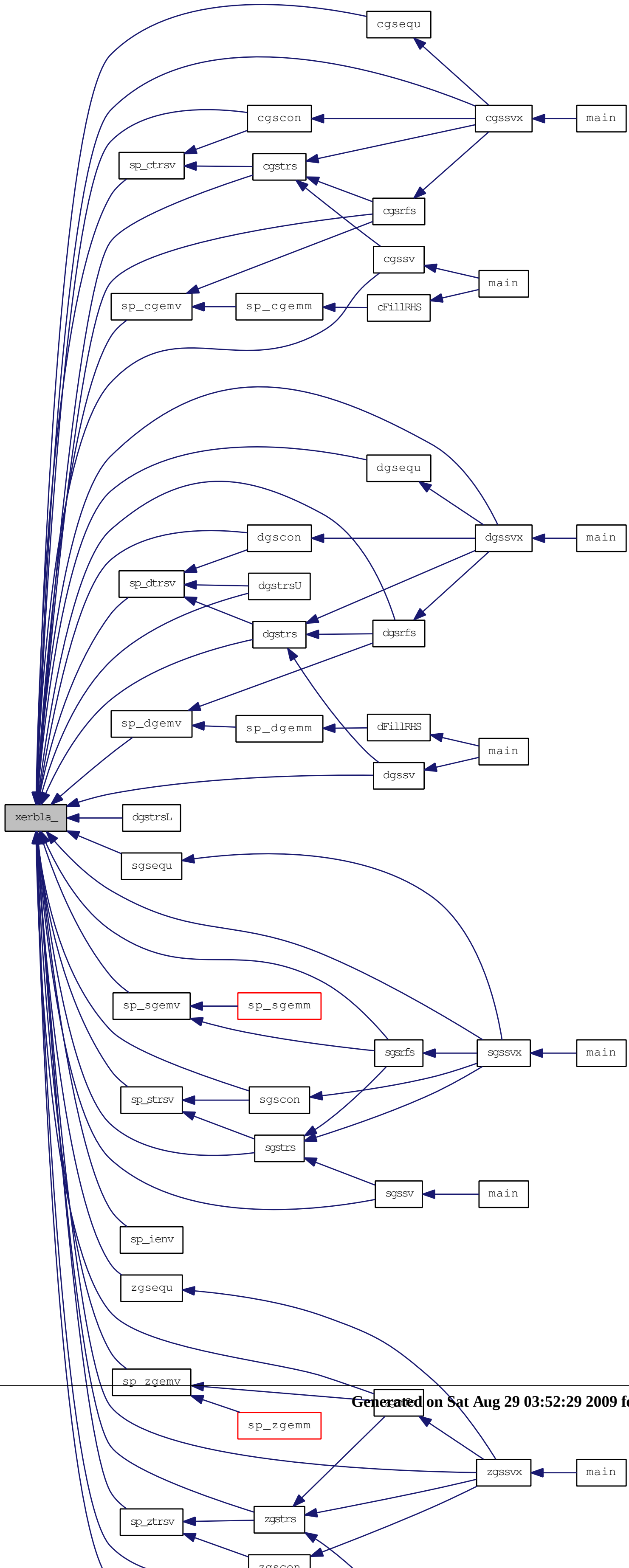


Here is the caller graph for this function:



4.116.5.34 int xerbla_ (char *, int *)

Here is the caller graph for this function:



4.117 SRC/slu_zdefs.h File Reference

Header file for real operations.

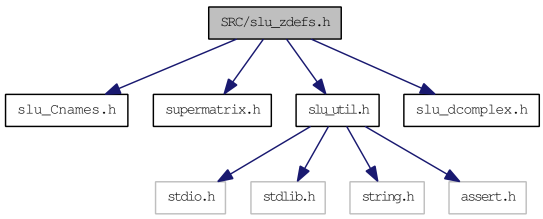
`#include "slu_Cnames.h"`

`#include "supermatrix.h"`

`#include "slu_util.h"`

`#include "slu_dcomplex.h"`

Include dependency graph for slu_zdefs.h:



This graph shows which files directly or indirectly include this file:

Data Structures

- struct GlobalLU_t

Typedefs

- typedef int int_t

Functions

- void zgssv (superlu_options_t ∗, SuperMatrix ∗, int ∗, int ∗, SuperMatrix ∗, SuperMatrix ∗, SuperMatrix ∗, SuperLUStat_t ∗, int ∗)

  Driver routines.

- void zgssvx (superlu_options_t ∗, SuperMatrix ∗, int ∗, int ∗, int ∗, char ∗, double ∗, double ∗, SuperMatrix ∗, SuperMatrix ∗, void ∗, int, SuperMatrix ∗, SuperMatrix ∗, double ∗, double ∗, double ∗, double ∗, mem_usage_t ∗, SuperLUStat_t ∗, int ∗)
- void zCreate_CompCol_Matrix (SuperMatrix ∗, int, int, int, doublecomplex ∗, int ∗, int ∗, Stype_t, Dtype_t, Mtype_t)

  Supernodal LU factor related.

- void zCreate_CompRow_Matrix (SuperMatrix ∗, int, int, int, doublecomplex ∗, int ∗, int ∗, Stype_t, Dtype_t, Mtype_t)
- void zCopy_CompCol_Matrix (SuperMatrix ∗, SuperMatrix ∗)

Copy matrix A into matrix B.

- void zCreate_Dense_Matrix (SuperMatrix ∗, int, int, doublecomplex ∗, int, Stype_t, Dtype_t, Mtype_t)
- void zCreate_SuperNode_Matrix (SuperMatrix ∗, int, int, int, doublecomplex ∗, int ∗, int ∗, int ∗, int ∗, int ∗, Stype_t, Dtype_t, Mtype_t)
- void zCopy_Dense_Matrix (int, int, doublecomplex ∗, int, doublecomplex ∗, int)
- void countnz (const int, int ∗, int ∗, int ∗, GlobalLU_t ∗)

  Count the total number of nonzeros in factors L and U, and in the symmetrically reduced L.

- void fixupL (const int, const int ∗, GlobalLU_t ∗)

  Fix up the data storage lsub for L-subscripts. It removes the subscript sets for structural pruning, and applies permuation to the remaining subscripts.

- void zallocateA (int, int, doublecomplex ∗∗, int ∗∗, int ∗∗)

  Allocate storage for original matrix A.

- void zgstrf (superlu_options_t ∗, SuperMatrix ∗, double, int, int, int ∗, void ∗, int, int ∗, int ∗, SuperMatrix ∗, SuperMatrix ∗, SuperLUStat_t ∗, int ∗)
- int zsnode_dfs (const int, const int, const int ∗, const int ∗, const int ∗, int ∗, int ∗, GlobalLU_t ∗)
- int zsnode_bmod (const int, const int, const int, doublecomplex ∗, doublecomplex ∗, GlobalLU_t ∗, SuperLUStat_t ∗)

  Performs numeric block updates within the relaxed snode.

- void zpanel_dfs (const int, const int, const int, SuperMatrix ∗, int ∗, int ∗, doublecomplex ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, GlobalLU_t ∗)
- void zpanel_bmod (const int, const int, const int, const int, doublecomplex ∗, doublecomplex ∗, int ∗, int ∗, GlobalLU_t ∗, SuperLUStat_t ∗)
- int zcolumn_dfs (const int, const int, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, GlobalLU_t ∗)
- int zcolumn_bmod (const int, const int, doublecomplex ∗, doublecomplex ∗, int ∗, int ∗, int, GlobalLU_t ∗, SuperLUStat_t ∗)
- int zcopy_to_ucol (int, int, int ∗, int ∗, int ∗, doublecomplex ∗, GlobalLU_t ∗)
- int zpivotL (const int, const double, int ∗, int ∗, int ∗, int ∗, int ∗, GlobalLU_t ∗, SuperLUStat_t ∗)
- void zpruneL (const int, const int ∗, const int, const int, const int ∗, const int ∗, int ∗, GlobalLU_t ∗)
- void zreadmt (int ∗, int ∗, int ∗, doublecomplex ∗∗, int ∗∗, int ∗∗)
- void zGenXtrue (int, int, doublecomplex ∗, int)
- void zFillRHS (trans_t, int, doublecomplex ∗, int, SuperMatrix ∗, SuperMatrix ∗)

  Let rhs[i] = sum of i-th row of A, so the solution vector is all 1's.

- void zgstrs (trans_t, SuperMatrix ∗, SuperMatrix ∗, int ∗, int ∗, SuperMatrix ∗, SuperLUStat_t ∗, int ∗)
- void zgsequ (SuperMatrix ∗, double ∗, double ∗, double ∗, double ∗, double ∗, int ∗)

  Driver related.

- void zlaqgs (SuperMatrix ∗, double ∗, double ∗, double, double, double, char ∗)
- void zgscon (char ∗, SuperMatrix ∗, SuperMatrix ∗, double, double ∗, SuperLUStat_t ∗, int ∗)
- double zPivotGrowth (int, SuperMatrix ∗, int ∗, SuperMatrix ∗, SuperMatrix ∗)
- void zgsrfs (trans_t, SuperMatrix ∗, SuperMatrix ∗, SuperMatrix ∗, int ∗, int ∗, char ∗, double ∗, double ∗, SuperMatrix ∗, SuperMatrix ∗, double ∗, double ∗, SuperLUStat_t ∗, int ∗)

- int sp_ztrsv (char ∗, char ∗, char ∗, SuperMatrix ∗, SuperMatrix ∗, doublecomplex ∗, SuperLUStat_t ∗, int ∗)

  Solves one of the systems of equations A∗x = b, or A'∗x = b.

- int sp_zgemv (char ∗, doublecomplex, SuperMatrix ∗, doublecomplex ∗, int, doublecomplex, doublecomplex ∗, int)

  Performs one of the matrix-vector operations y := alpha∗A∗x + beta∗y, or y := alpha∗A'∗x + beta∗y.

- int sp_zgemm (char ∗, char ∗, int, int, int, doublecomplex, SuperMatrix ∗, doublecomplex ∗, int, doublecomplex, doublecomplex ∗, int)
- int zLUMemInit (fact_t, void ∗, int, int, int, int, int, SuperMatrix ∗, SuperMatrix ∗, GlobalLU_t ∗, int ∗∗, doublecomplex ∗∗)

  Memory-related.

- void zSetRWork (int, int, doublecomplex ∗, doublecomplex ∗∗, doublecomplex ∗∗)

  Set up pointers for real working arrays.

- void zLUWorkFree (int ∗, doublecomplex ∗, GlobalLU_t ∗)

  Free the working storage used by factor routines.

- int zLUMemXpand (int, int, MemType, int ∗, GlobalLU_t ∗)

  Expand the data structures for L and U during the factorization.

- doublecomplex ∗ doublecomplexMalloc (int)
- doublecomplex ∗ doublecomplexCalloc (int)
- double ∗ doubleMalloc (int)
- double ∗ doubleCalloc (int)
- int zmemory_usage (const int, const int, const int, const int)
- int zQuerySpace (SuperMatrix ∗, SuperMatrix ∗, mem_usage_t ∗)
- void zreadhb (int ∗, int ∗, int ∗, doublecomplex ∗∗, int ∗∗, int ∗∗)

  Auxiliary routines.

- void zCompRow_to_CompCol (int, int, int, doublecomplex ∗, int ∗, int ∗, doublecomplex ∗∗, int ∗∗, int ∗∗)

  Convert a row compressed storage into a column compressed storage.

- void zfill (doublecomplex ∗, int, doublecomplex)

  Fills a doublecomplex precision array with a given value.

- void zinf_norm_error (int, SuperMatrix ∗, doublecomplex ∗)

  Check the inf-norm of the error vector.

- void PrintPerf (SuperMatrix ∗, SuperMatrix ∗, mem_usage_t ∗, doublecomplex, doublecomplex, doublecomplex ∗, doublecomplex ∗, char ∗)
- void zPrint_CompCol_Matrix (char ∗, SuperMatrix ∗)

  Routines for debugging.

- void zPrint_SuperNode_Matrix (char ∗, SuperMatrix ∗)
- void zPrint_Dense_Matrix (char ∗, SuperMatrix ∗)
- void print_lu_col (char ∗, int, int, int ∗, GlobalLU_t ∗)
- void check_tempv (int, doublecomplex ∗)

4.117.1 Detailed Description

```
 -- SuperLU routine (version 3.0) --
 Univ. of California Berkeley, Xerox Palo Alto Research Center,
 and Lawrence Berkeley National Lab.
 October 15, 2003

 Global data structures used in LU factorization -

   nsuper: supernodes = nsuper + 1, numbered [0, nsuper].
   (xsup,supno): supno[i] is the supernode no to which i belongs;
xsup(s) points to the beginning of the s-th supernode.
e.g.   supno 0 1 2 2 3 3 3 4 4 4 4 4   (n=12)
        xsup 0 1 2 4 7 12
Note: dfs will be performed on supernode rep. relative to the new
      row pivoting ordering

   (xlsub,lsub): lsub[*] contains the compressed subscript of
rectangular supernodes; xlsub[j] points to the starting
location of the j-th column in lsub[*]. Note that xlsub
is indexed by column.
Storage: original row subscripts

      During the course of sparse LU factorization, we also use
(xlsub,lsub) for the purpose of symmetric pruning. For each
supernode {s,s+1,...,t=s+r} with first column s and last
column t, the subscript set
lsub[j], j=xlsub[s], .., xlsub[s+1]-1
is the structure of column s (i.e. structure of this supernode).
It is used for the storage of numerical values.
Furthermore,
lsub[j], j=xlsub[t], .., xlsub[t+1]-1
is the structure of the last column t of this supernode.
It is for the purpose of symmetric pruning. Therefore, the
structural subscripts can be rearranged without making physical
interchanges among the numerical values.

However, if the supernode has only one column, then we
only keep one set of subscripts. For any subscript interchange
performed, similar interchange must be done on the numerical
values.

The last column structures (for pruning) will be removed
after the numercial LU factorization phase.

   (xlusup,lusup): lusup[*] contains the numerical values of the
rectangular supernodes; xlusup[j] points to the starting
location of the j-th column in storage vector lusup[*]
Note: xlusup is indexed by column.
Each rectangular supernode is stored by column-major
scheme, consistent with Fortran 2-dim array storage.

   (xusub,ucol,usub): ucol[*] stores the numerical values of
U-columns outside the rectangular supernodes. The row
subscript of nonzero ucol[k] is stored in usub[k].
xusub[i] points to the starting location of column i in ucol.
Storage: new row subscripts; that is subscripts of PA.
```

4.117.2 Typedef Documentation

4.117.2.1 typedef int int_t

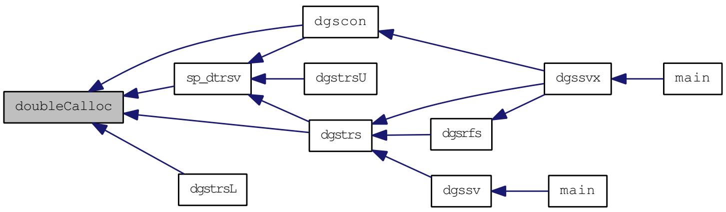
4.117.3 Function Documentation

4.117.3.1 void check_tempv (int, doublecomplex ∗)

4.117.3.2 void countnz (const *int*, int ∗, int ∗, int ∗, GlobalLU_t ∗)

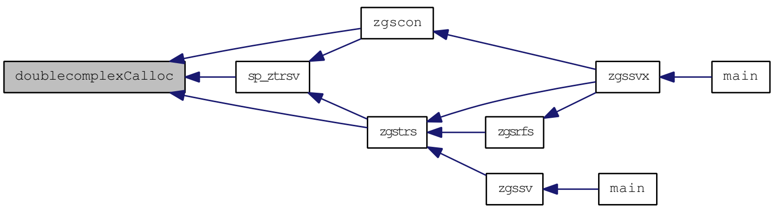
4.117.3.3 double∗ doubleCalloc (int)

Here is the caller graph for this function:



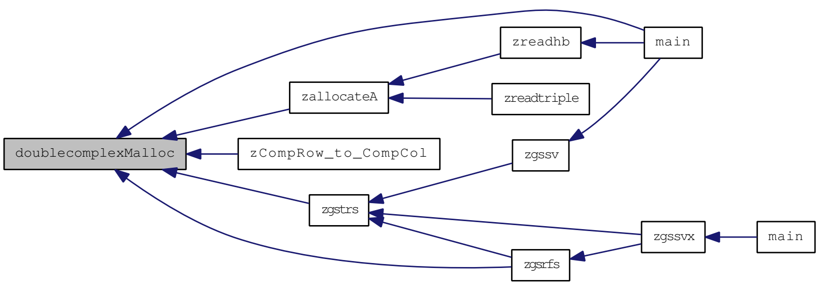
4.117.3.4 doublecomplex∗ doublecomplexCalloc (int)

Here is the caller graph for this function:



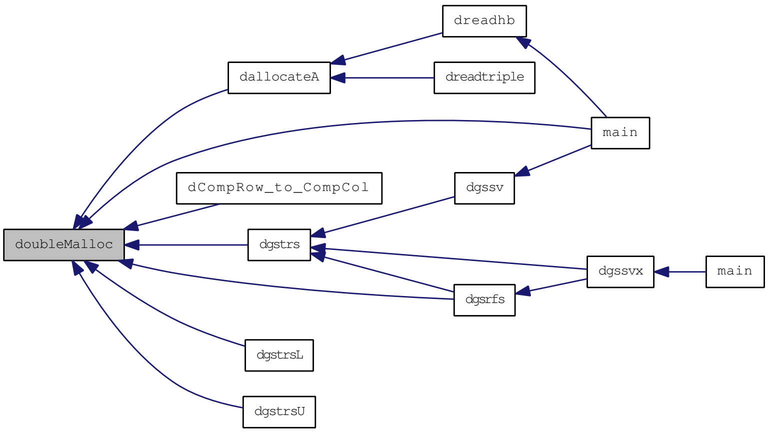
4.117.3.5 doublecomplex∗ doublecomplexMalloc (int)

Here is the caller graph for this function:



4.117.3.6 double* doubleMalloc (int)

Here is the caller graph for this function:



4.117.3.7 void fixupL (const *int*, const int *, GlobalLU_t *)

4.117.3.8 void print_lu_col (char *, int, int, int *, GlobalLU_t *)

4.117.3.9 void PrintPerf (SuperMatrix *, SuperMatrix *, mem_usage_t *, doublecomplex, doublecomplex, doublecomplex *, doublecomplex *, char *)

4.117.3.10 int sp_zgemm (char * *transa*, char * *transb*, int *m*, int *n*, int *k*, doublecomplex *alpha*, SuperMatrix * *A*, doublecomplex * *b*, int *ldb*, doublecomplex *beta*, doublecomplex * *c*, int *ldc*)

```
Purpose
  =======

  sp_z performs one of the matrix-matrix operations

     C := alpha*op( A )*op( B ) + beta*C,

  where  op( X ) is one of

     op( X ) = X   or   op( X ) = X'   or   op( X ) = conjg( X' ),

  alpha and beta are scalars, and A, B and C are matrices, with op( A )
  an m by k matrix,  op( B )  a  k by n matrix and  C an m by n matrix.

  Parameters
  ==========

  TRANSA - (input) char*
           On entry, TRANSA specifies the form of op( A ) to be used in
           the matrix multiplication as follows:
              TRANSA = 'N' or 'n',  op( A ) = A.
```

```
              TRANSA = 'T' or 't',  op( A ) = A'.
              TRANSA = 'C' or 'c',  op( A ) = conjg( A' ).
           Unchanged on exit.


   TRANSB - (input) char*
           On entry, TRANSB specifies the form of op( B ) to be used in
           the matrix multiplication as follows:
              TRANSB = 'N' or 'n',  op( B ) = B.
              TRANSB = 'T' or 't',  op( B ) = B'.
              TRANSB = 'C' or 'c',  op( B ) = conjg( B' ).
           Unchanged on exit.


   M      - (input) int
           On entry,  M  specifies  the number of rows of the matrix
     op( A ) and of the matrix C.  M must be at least zero.
     Unchanged on exit.


   N      - (input) int
           On entry,  N specifies the number of columns of the matrix
     op( B ) and the number of columns of the matrix C. N must be
     at least zero.
     Unchanged on exit.


   K      - (input) int
           On entry, K specifies the number of columns of the matrix
     op( A ) and the number of rows of the matrix op( B ). K must
     be at least  zero.
           Unchanged on exit.


   ALPHA  - (input) doublecomplex
           On entry, ALPHA specifies the scalar alpha.


   A      - (input) SuperMatrix*
           Matrix A with a sparse format, of dimension (A->nrow, A->ncol).
           Currently, the type of A can be:
               Stype = NC or NCP; Dtype = SLU_Z; Mtype = GE.
           In the future, more general A can be handled.


   B      - DOUBLE COMPLEX PRECISION array of DIMENSION ( LDB, kb ), where kb is
           n when TRANSB = 'N' or 'n',  and is  k otherwise.
           Before entry with  TRANSB = 'N' or 'n',  the leading k by n
           part of the array B must contain the matrix B, otherwise
           the leading n by k part of the array B must contain the
           matrix B.
           Unchanged on exit.


   LDB    - (input) int
           On entry, LDB specifies the first dimension of B as declared
           in the calling (sub) program. LDB must be at least max( 1, n ).
           Unchanged on exit.

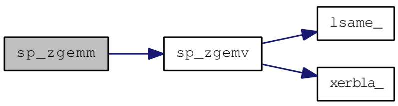

   BETA   - (input) doublecomplex
           On entry, BETA specifies the scalar beta. When BETA is
           supplied as zero then C need not be set on input.
```

```
C       - DOUBLE COMPLEX PRECISION array of DIMENSION ( LDC, n ).
          Before entry, the leading m by n part of the array C must
          contain the matrix C,  except when beta is zero, in which
          case C need not be set on entry.
          On exit, the array C is overwritten by the m by n matrix
  ( alpha*op( A )*B + beta*C ).

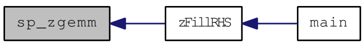
LDC     - (input) int
          On entry, LDC specifies the first dimension of C as declared
          in the calling (sub)program. LDC must be at least max(1,m).
          Unchanged on exit.

==== Sparse Level 3 Blas routine.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.117.3.11 int sp_zgemv (char ∗ *trans*, doublecomplex *alpha*, SuperMatrix ∗ *A*, doublecomplex ∗ *x*, int *incx*, doublecomplex *beta*, doublecomplex ∗ *y*, int *incy*)

```
Purpose
=======

sp_zgemv()  performs one of the matrix-vector operations
   y := alpha*A*x + beta*y,    or    y := alpha*A'*x + beta*y,
where alpha and beta are scalars, x and y are vectors and A is a
sparse A->nrow by A->ncol matrix.

Parameters
==========

TRANS   - (input) char*
          On entry, TRANS specifies the operation to be performed as
          follows:
             TRANS = 'N' or 'n'   y := alpha*A*x + beta*y.
             TRANS = 'T' or 't'   y := alpha*A'*x + beta*y.
             TRANS = 'C' or 'c'   y := alpha*A'*x + beta*y.

ALPHA   - (input) doublecomplex
          On entry, ALPHA specifies the scalar alpha.
```

```
A       - (input) SuperMatrix*
          Before entry, the leading m by n part of the array A must
          contain the matrix of coefficients.

X       - (input) doublecomplex*, array of DIMENSION at least
          ( 1 + ( n - 1 )*abs( INCX ) ) when TRANS = 'N' or 'n'
         and at least
          ( 1 + ( m - 1 )*abs( INCX ) ) otherwise.
          Before entry, the incremented array X must contain the
          vector x.

INCX    - (input) int
          On entry, INCX specifies the increment for the elements of
          X. INCX must not be zero.

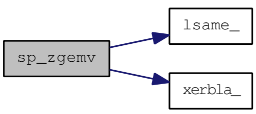
BETA    - (input) doublecomplex
          On entry, BETA specifies the scalar beta. When BETA is
          supplied as zero then Y need not be set on input.

Y       - (output) doublecomplex*,  array of DIMENSION at least
          ( 1 + ( m - 1 )*abs( INCY ) ) when TRANS = 'N' or 'n'
          and at least
          ( 1 + ( n - 1 )*abs( INCY ) ) otherwise.
          Before entry with BETA non-zero, the incremented array Y
          must contain the vector y. On exit, Y is overwritten by the
          updated vector y.

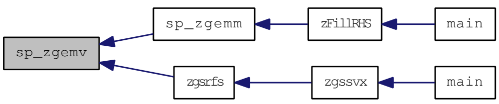
INCY    - (input) int
          On entry, INCY specifies the increment for the elements of
          Y. INCY must not be zero.

 ==== Sparse Level 2 Blas routine.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.117.3.12 int sp_ztrsv (char * *uplo*, char * *trans*, char * *diag*, SuperMatrix * *L*, SuperMatrix * *U*, doublecomplex * *x*, SuperLUStat_t * *stat*, int * *info*)

```
Purpose
=======
```

```
sp_ztrsv() solves one of the systems of equations
    A*x = b,   or   A'*x = b,
where b and x are n element vectors and A is a sparse unit , or
non-unit, upper or lower triangular matrix.
No test for singularity or near-singularity is included in this
routine. Such tests must be performed before calling this routine.


Parameters
==========


uplo   - (input) char*
         On entry, uplo specifies whether the matrix is an upper or
          lower triangular matrix as follows:
             uplo = 'U' or 'u'   A is an upper triangular matrix.
             uplo = 'L' or 'l'   A is a lower triangular matrix.


trans  - (input) char*
          On entry, trans specifies the equations to be solved as
          follows:
             trans = 'N' or 'n'   A*x = b.
             trans = 'T' or 't'   A'*x = b.
             trans = 'C' or 'c'   A^H*x = b.


diag   - (input) char*
          On entry, diag specifies whether or not A is unit
          triangular as follows:
             diag = 'U' or 'u'   A is assumed to be unit triangular.
             diag = 'N' or 'n'   A is not assumed to be unit
                                 triangular.

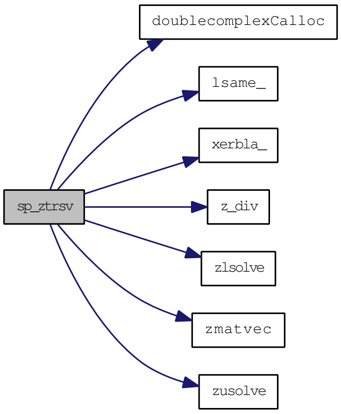

L       - (input) SuperMatrix*
    The factor L from the factorization Pr*A*Pc=L*U. Use
          compressed row subscripts storage for supernodes,
          i.e., L has types: Stype = SC, Dtype = SLU_Z, Mtype = TRLU.


U       - (input) SuperMatrix*
     The factor U from the factorization Pr*A*Pc=L*U.
     U has types: Stype = NC, Dtype = SLU_Z, Mtype = TRU.

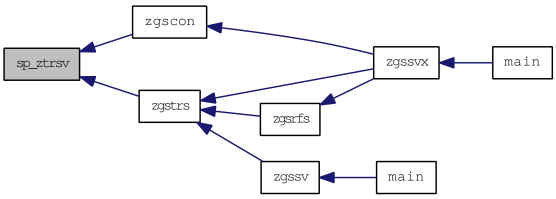

x       - (input/output) doublecomplex*
          Before entry, the incremented array X must contain the n
          element right-hand side vector b. On exit, X is overwritten
          with the solution vector x.


info    - (output) int*
          If *info = -i, the i-th argument had an illegal value.
```

Here is the call graph for this function:

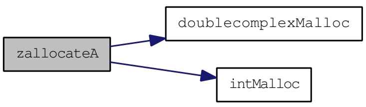


Here is the caller graph for this function:

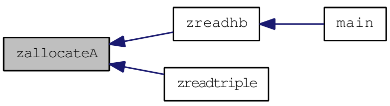


4.117.3.13 void zallocateA (int, int, doublecomplex $**$, int $**$, int $**$)

Here is the call graph for this function:



Here is the caller graph for this function:

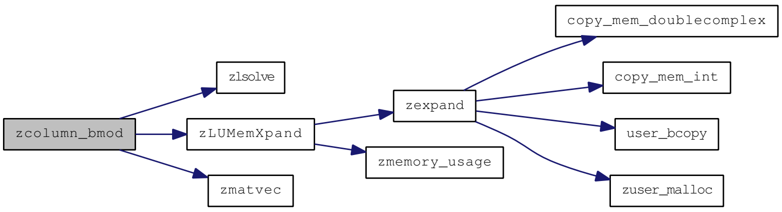


4.117.3.14 int zcolumn_bmod (const int *jcol*, const int *nseg*, doublecomplex $*$ *dense*, doublecomplex $*$ *tempv*, int $*$ *segrep*, int $*$ *repfnz*, int *fpanelc*, GlobalLU_t $*$ *Glu*, SuperLUStat_t $*$ *stat*)

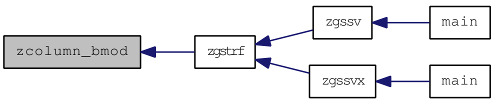
```
Purpose:
```

```
========
Performs numeric block updates (sup-col) in topological order.
It features: col-col, 2cols-col, 3cols-col, and sup-col updates.
Special processing on the supernodal portion of L[*,j]
Return value:   0 - successful return
              > 0 - number of bytes allocated when run out of space
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.117.3.15 int zcolumn_dfs (const int *m*, const int *jcol*, int ∗ *perm_r*, int ∗ *nseg*, int ∗ *lsub_col*, int ∗ *segrep*, int ∗ *repfnz*, int ∗ *xprune*, int ∗ *marker*, int ∗ *parent*, int ∗ *xplore*, GlobalLU_t ∗ *Glu*)

```
Purpose
=======
  "column_dfs" performs a symbolic factorization on column jcol, and
  decide the supernode boundary.

  This routine does not use numeric values, but only use the RHS
  row indices to start the dfs.

  A supernode representative is the last column of a supernode.
  The nonzeros in U[*,j] are segments that end at supernodal
  representatives. The routine returns a list of such supernodal
  representatives in topological order of the dfs that generates them.
  The location of the first nonzero in each such supernodal segment
  (supernodal entry location) is also returned.

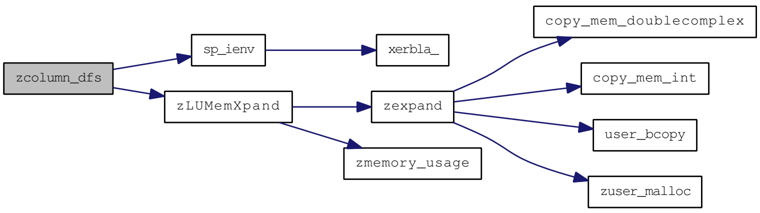
Local parameters
================
  nseg: no of segments in current U[*,j]
  jsuper: jsuper=EMPTY if column j does not belong to the same
supernode as j-1. Otherwise, jsuper=nsuper.

  marker2: A-row --> A-row/col (0/1)
```

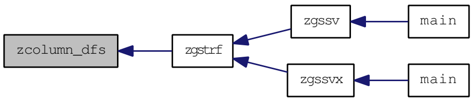
```
   repfnz: SuperA-col --> PA-row
   parent: SuperA-col --> SuperA-col
   xplore: SuperA-col --> index to L-structure


 Return value
 ============
     0  success;
   > 0  number of bytes allocated when run out of space.
```

Here is the call graph for this function:

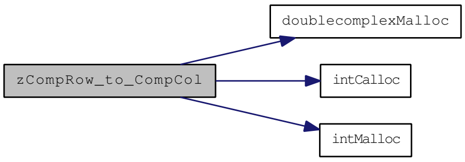


Here is the caller graph for this function:



4.117.3.16 void zCompRow_to_CompCol (int, int, int, doublecomplex ∗, int ∗, int ∗, doublecomplex ∗∗, int ∗∗, int ∗∗)

Here is the call graph for this function:



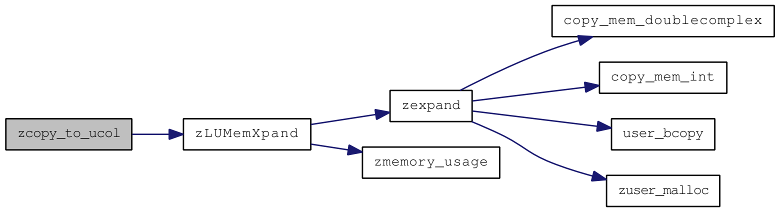
4.117.3.17 void zCopy_CompCol_Matrix (SuperMatrix ∗, SuperMatrix ∗)

4.117.3.18 void zCopy_Dense_Matrix (int, int, doublecomplex ∗, int, doublecomplex ∗, int)

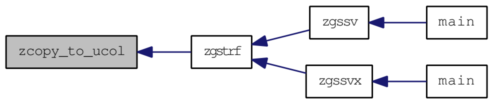
Copies a two-dimensional matrix X to another matrix Y.

4.117.3.19 int zcopy_to_ucol (int, int, int ∗, int ∗, int ∗, doublecomplex ∗, GlobalLU_t ∗)

Here is the call graph for this function:

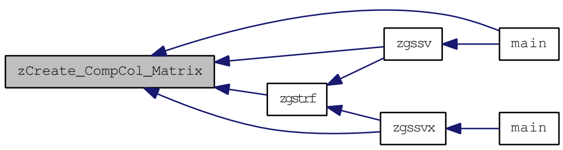


Here is the caller graph for this function:



4.117.3.20 void zCreate_CompCol_Matrix (SuperMatrix ∗, int, int, int, doublecomplex ∗, int ∗, int ∗, Stype_t, Dtype_t, Mtype_t)

Here is the caller graph for this function:



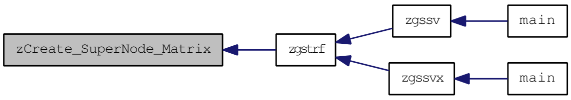
4.117.3.21 void zCreate_CompRow_Matrix (SuperMatrix ∗, int, int, int, doublecomplex ∗, int ∗, int ∗, Stype_t, Dtype_t, Mtype_t)

4.117.3.22 void zCreate_Dense_Matrix (SuperMatrix ∗, int, int, doublecomplex ∗, int, Stype_t, Dtype_t, Mtype_t)

Here is the caller graph for this function:

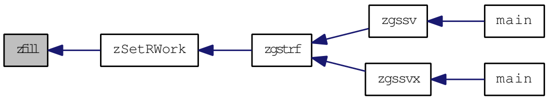
4.117.3.23 void zCreate_SuperNode_Matrix (SuperMatrix ∗, int, int, int, doublecomplex ∗, int ∗, int ∗, int ∗, int ∗, int ∗, Stype_t, Dtype_t, Mtype_t)

Here is the caller graph for this function:



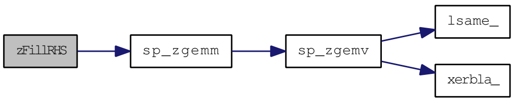
4.117.3.24 void zfill (doublecomplex ∗, int, doublecomplex)

Here is the caller graph for this function:

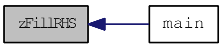


4.117.3.25 void zFillRHS (trans_t, int, doublecomplex ∗, int, SuperMatrix ∗, SuperMatrix ∗)

Here is the call graph for this function:

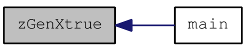


Here is the caller graph for this function:



4.117.3.26 void zGenXtrue (int, int, doublecomplex ∗, int)

Here is the caller graph for this function:



4.117.3.27 void zgscon (char ∗ *norm*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, double *anorm*, double ∗ *rcond*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======
```

```
ZGSCON estimates the reciprocal of the condition number of a general
real matrix A, in either the 1-norm or the infinity-norm, using
the LU factorization computed by ZGETRF.   *


An estimate is obtained for norm(inv(A)), and the reciprocal of the
condition number is computed as
   RCOND = 1 / ( norm(A) * norm(inv(A)) ).


See supermatrix.h for the definition of 'SuperMatrix' structure.


Arguments
=========


 NORM    (input) char*
         Specifies whether the 1-norm condition number or the
         infinity-norm condition number is required:
         = '1' or 'O':  1-norm;
         = 'I':         Infinity-norm.


 L       (input) SuperMatrix*
         The factor L from the factorization Pr*A*Pc=L*U as computed by
         zgstrf(). Use compressed row subscripts storage for supernodes,
         i.e., L has types: Stype = SLU_SC, Dtype = SLU_Z, Mtype = SLU_TRLU.


 U       (input) SuperMatrix*
         The factor U from the factorization Pr*A*Pc=L*U as computed by
         zgstrf(). Use column-wise storage scheme, i.e., U has types:
         Stype = SLU_NC, Dtype = SLU_Z, Mtype = SLU_TRU.

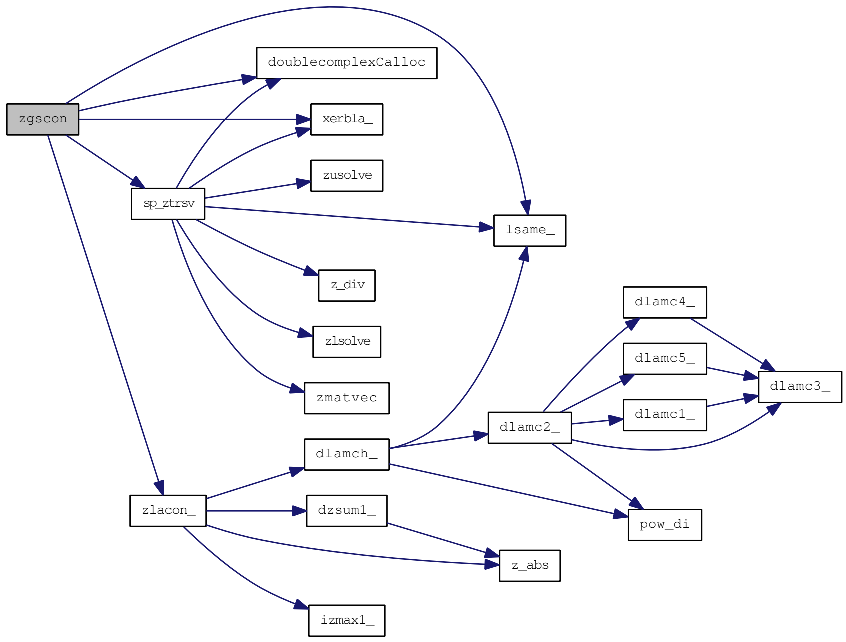

 ANORM   (input) double
         If NORM = '1' or 'O', the 1-norm of the original matrix A.
         If NORM = 'I', the infinity-norm of the original matrix A.


 RCOND   (output) double*
        The reciprocal of the condition number of the matrix A,
        computed as RCOND = 1/(norm(A) * norm(inv(A))).


 INFO    (output) int*
        = 0:  successful exit
        < 0:  if INFO = -i, the i-th argument had an illegal value


 ====================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.117.3.28 void zgsequ (SuperMatrix ∗ *A*, double ∗ *r*, double ∗ *c*, double ∗ *rowcnd*, double ∗ *colcnd*, double ∗ *amax*, int ∗ *info*)

```
Purpose
  =======

  ZGSEQU computes row and column scalings intended to equilibrate an
  M-by-N sparse matrix A and reduce its condition number. R returns the row
  scale factors and C the column scale factors, chosen to try to make
  the largest element in each row and column of the matrix B with
  elements B(i,j)=R(i)*A(i,j)*C(j) have absolute value 1.

  R(i) and C(j) are restricted to be between SMLNUM = smallest safe
  number and BIGNUM = largest safe number.  Use of these scaling
  factors is not guaranteed to reduce the condition number of A but
  works well in practice.

  See supermatrix.h for the definition of 'SuperMatrix' structure.

  Arguments
  =========
```

```
A       (input) SuperMatrix*
        The matrix of dimension (A->nrow, A->ncol) whose equilibration
        factors are to be computed. The type of A can be:
        Stype = SLU_NC; Dtype = SLU_Z; Mtype = SLU_GE.

R       (output) double*, size A->nrow
        If INFO = 0 or INFO > M, R contains the row scale factors
        for A.

C       (output) double*, size A->ncol
        If INFO = 0,  C contains the column scale factors for A.

ROWCND  (output) double*
        If INFO = 0 or INFO > M, ROWCND contains the ratio of the
        smallest R(i) to the largest R(i).  If ROWCND >= 0.1 and
        AMAX is neither too large nor too small, it is not worth
        scaling by R.

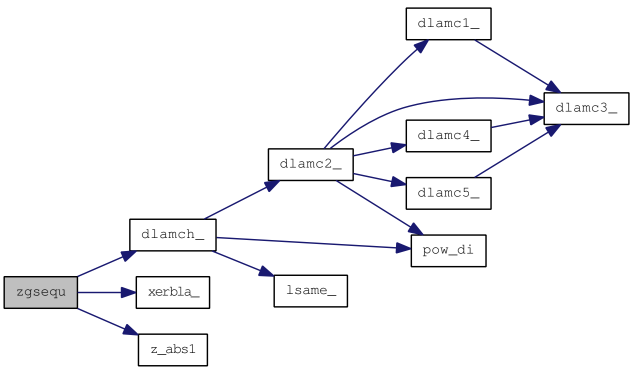
COLCND  (output) double*
        If INFO = 0, COLCND contains the ratio of the smallest
        C(i) to the largest C(i).  If COLCND >= 0.1, it is not
        worth scaling by C.

AMAX    (output) double*
        Absolute value of largest matrix element.  If AMAX is very
        close to overflow or very close to underflow, the matrix
        should be scaled.

INFO    (output) int*
        = 0:  successful exit
        < 0:  if INFO = -i, the i-th argument had an illegal value
        > 0:  if INFO = i,  and i is
              <= A->nrow:  the i-th row of A is exactly zero
              >  A->ncol:  the (i-M)-th column of A is exactly zero

======================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.117.3.29 void zgsrfs (trans_t *trans*, SuperMatrix ∗ *A*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, int ∗ *perm_c*, int ∗ *perm_r*, char ∗ *equed*, double ∗ *R*, double ∗ *C*, SuperMatrix ∗ *B*, SuperMatrix ∗ *X*, double ∗ *ferr*, double ∗ *berr*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
  Purpose
  =======

  ZGSRFS improves the computed solution to a system of linear
  equations and provides error bounds and backward error estimates for
  the solution.

  If equilibration was performed, the system becomes:
          (diag(R)*A_original*diag(C)) * X = diag(R)*B_original.

  See supermatrix.h for the definition of 'SuperMatrix' structure.

  Arguments
  =========

 trans   (input) trans_t
          Specifies the form of the system of equations:
          = NOTRANS: A * X = B  (No transpose)
          = TRANS:   A'* X = B  (Transpose)
          = CONJ:    A**H * X = B  (Conjugate transpose)

  A       (input) SuperMatrix*
          The original matrix A in the system, or the scaled A if
          equilibration was done. The type of A can be:
          Stype = SLU_NC, Dtype = SLU_Z, Mtype = SLU_GE.

  L       (input) SuperMatrix*
    The factor L from the factorization Pr*A*Pc=L*U. Use
          compressed row subscripts storage for supernodes,
          i.e., L has types: Stype = SLU_SC, Dtype = SLU_Z, Mtype = SLU_TRLU.

  U       (input) SuperMatrix*
          The factor U from the factorization Pr*A*Pc=L*U as computed by
          zgstrf(). Use column-wise storage scheme,
          i.e., U has types: Stype = SLU_NC, Dtype = SLU_Z, Mtype = SLU_TRU.

  perm_c  (input) int*, dimension (A->ncol)
    Column permutation vector, which defines the
          permutation matrix Pc; perm_c[i] = j means column i of A is
          in position j in A*Pc.

  perm_r  (input) int*, dimension (A->nrow)
          Row permutation vector, which defines the permutation matrix Pr;
          perm_r[i] = j means row i of A is in position j in Pr*A.
```

```
equed   (input) Specifies the form of equilibration that was done.
        = 'N': No equilibration.
        = 'R': Row equilibration, i.e., A was premultiplied by diag(R).
        = 'C': Column equilibration, i.e., A was postmultiplied by
               diag(C).
        = 'B': Both row and column equilibration, i.e., A was replaced
               by diag(R)*A*diag(C).

R       (input) double*, dimension (A->nrow)
        The row scale factors for A.
        If equed = 'R' or 'B', A is premultiplied by diag(R).
        If equed = 'N' or 'C', R is not accessed.

C       (input) double*, dimension (A->ncol)
        The column scale factors for A.
        If equed = 'C' or 'B', A is postmultiplied by diag(C).
        If equed = 'N' or 'R', C is not accessed.

B       (input) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_Z, Mtype = SLU_GE.
        The right hand side matrix B.
        if equed = 'R' or 'B', B is premultiplied by diag(R).

X       (input/output) SuperMatrix*
        X has types: Stype = SLU_DN, Dtype = SLU_Z, Mtype = SLU_GE.
        On entry, the solution matrix X, as computed by zgstrs().
        On exit, the improved solution matrix X.
        if *equed = 'C' or 'B', X should be premultiplied by diag(C)
            in order to obtain the solution to the original system.

FERR    (output) double*, dimension (B->ncol)
        The estimated forward error bound for each solution vector
        X(j) (the j-th column of the solution matrix X).
        If XTRUE is the true solution corresponding to X(j), FERR(j)
        is an estimated upper bound for the magnitude of the largest
        element in (X(j) - XTRUE) divided by the magnitude of the
        largest element in X(j).  The estimate is as reliable as
        the estimate for RCOND, and is almost always a slight
        overestimate of the true error.

BERR    (output) double*, dimension (B->ncol)
        The componentwise relative backward error of each solution
        vector X(j) (i.e., the smallest relative change in
        any element of A or B that makes X(j) an exact solution).

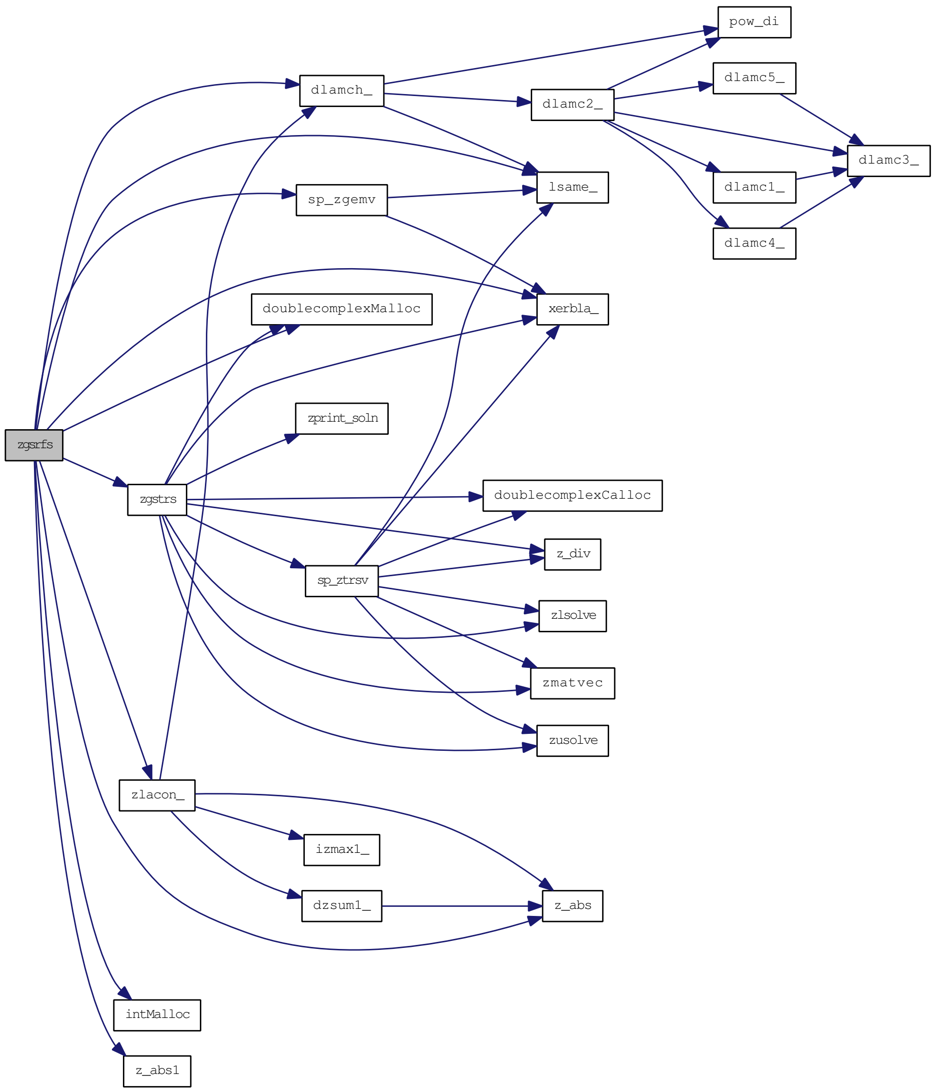
stat     (output) SuperLUStat_t*
         Record the statistics on runtime and floating-point operation count.
         See util.h for the definition of 'SuperLUStat_t'.

info    (output) int*
        = 0:  successful exit
         < 0:  if INFO = -i, the i-th argument had an illegal value

 Internal Parameters
 ===================
```

```
ITMAX is the maximum number of steps of iterative refinement.
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.117.3.30 void zgssv (superlu_options_t ∗ *options*, SuperMatrix ∗ *A*, int ∗ *perm_c*, int ∗ *perm_r*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, SuperMatrix ∗ *B*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
```

```
=======

ZGSSV solves the system of linear equations A*X=B, using the
LU factorization from ZGSTRF. It performs the following steps:

  1. If A is stored column-wise (A->Stype = SLU_NC):

     1.1. Permute the columns of A, forming A*Pc, where Pc
          is a permutation matrix. For more details of this step,
          see sp_preorder.c.

     1.2. Factor A as Pr*A*Pc=L*U with the permutation Pr determined
          by Gaussian elimination with partial pivoting.
          L is unit lower triangular with offdiagonal entries
          bounded by 1 in magnitude, and U is upper triangular.

     1.3. Solve the system of equations A*X=B using the factored
          form of A.

  2. If A is stored row-wise (A->Stype = SLU_NR), apply the
     above algorithm to the transpose of A:

     2.1. Permute columns of transpose(A) (rows of A),
          forming transpose(A)*Pc, where Pc is a permutation matrix.
          For more details of this step, see sp_preorder.c.

     2.2. Factor A as Pr*transpose(A)*Pc=L*U with the permutation Pr
          determined by Gaussian elimination with partial pivoting.
          L is unit lower triangular with offdiagonal entries
          bounded by 1 in magnitude, and U is upper triangular.

     2.3. Solve the system of equations A*X=B using the factored
          form of A.

  See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

options (input) superlu_options_t*
        The structure defines the input parameters to control
        how the LU decomposition will be performed and how the
        system will be solved.

A       (input) SuperMatrix*
        Matrix A in A*X=B, of dimension (A->nrow, A->ncol). The number
        of linear equations is A->nrow. Currently, the type of A can be:
        Stype = SLU_NC or SLU_NR; Dtype = SLU_Z; Mtype = SLU_GE.
        In the future, more general A may be handled.

perm_c  (input/output) int*
        If A->Stype = SLU_NC, column permutation vector of size A->ncol
        which defines the permutation matrix Pc; perm_c[i] = j means
        column i of A is in position j in A*Pc.
        If A->Stype = SLU_NR, column permutation vector of size A->nrow
        which describes permutation of columns of transpose(A)
        (rows of A) as described above.
```

```
        If options->ColPerm = MY_PERMC or options->Fact = SamePattern or
           options->Fact = SamePattern_SameRowPerm, it is an input argument.
           On exit, perm_c may be overwritten by the product of the input
           perm_c and a permutation that postorders the elimination tree
           of Pc'*A'*A*Pc; perm_c is not changed if the elimination tree
           is already in postorder.
        Otherwise, it is an output argument.


perm_r  (input/output) int*
        If A->Stype = SLU_NC, row permutation vector of size A->nrow,
        which defines the permutation matrix Pr, and is determined
        by partial pivoting.  perm_r[i] = j means row i of A is in
        position j in Pr*A.
        If A->Stype = SLU_NR, permutation vector of size A->ncol, which
        determines permutation of rows of transpose(A)
        (columns of A) as described above.


        If options->RowPerm = MY_PERMR or
           options->Fact = SamePattern_SameRowPerm, perm_r is an
           input argument.
        otherwise it is an output argument.


L       (output) SuperMatrix*
        The factor L from the factorization
            Pr*A*Pc=L*U              (if A->Stype = SLU_NC) or
            Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
        Uses compressed row subscripts storage for supernodes, i.e.,
        L has types: Stype = SLU_SC, Dtype = SLU_Z, Mtype = SLU_TRLU.

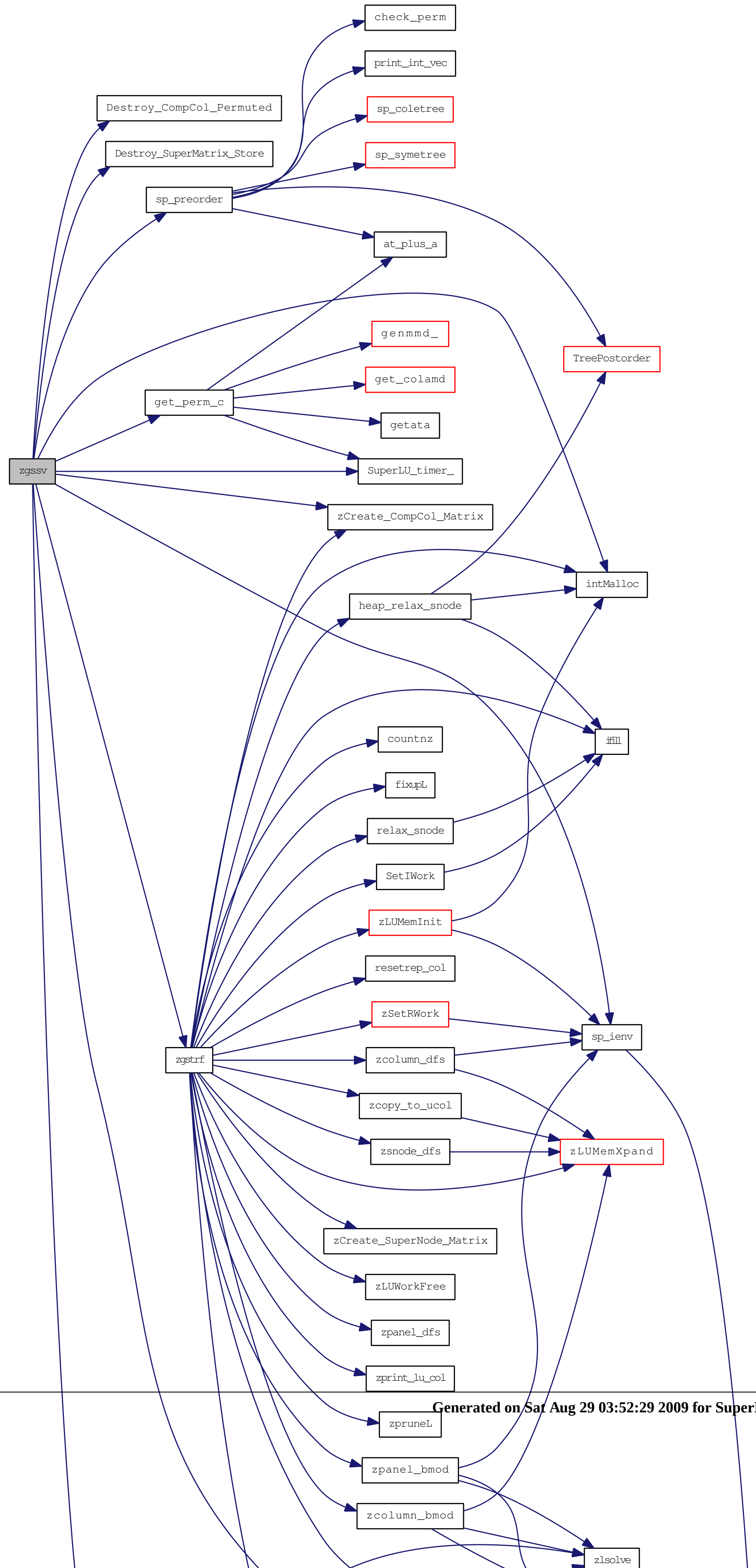

U       (output) SuperMatrix*
  The factor U from the factorization
            Pr*A*Pc=L*U              (if A->Stype = SLU_NC) or
            Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
        Uses column-wise storage scheme, i.e., U has types:
        Stype = SLU_NC, Dtype = SLU_Z, Mtype = SLU_TRU.


B       (input/output) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_Z, Mtype = SLU_GE.
        On entry, the right hand side matrix.
        On exit, the solution matrix if info = 0;


stat   (output) SuperLUStat_t*
       Record the statistics on runtime and floating-point operation count.
       See util.h for the definition of 'SuperLUStat_t'.


info    (output) int*
  = 0: successful exit
        > 0: if info = i, and i is
            <= A->ncol: U(i,i) is exactly zero. The factorization has
               been completed, but the factor U is exactly singular,
               so the solution could not be computed.
            > A->ncol: number of bytes allocated when memory allocation
               failure occurred, plus A->ncol.
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.117.3.31 void zgssvx (superlu_options_t ∗ *options*, SuperMatrix ∗ *A*, int ∗ *perm_c*, int ∗ *perm_r*, int ∗ *etree*, char ∗ *equed*, double ∗ *R*, double ∗ *C*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, void ∗ *work*, int *lwork*, SuperMatrix ∗ *B*, SuperMatrix ∗ *X*, double ∗ *recip_pivot_growth*, double ∗ *rcond*, double ∗ *ferr*, double ∗ *berr*, mem_usage_t ∗ *mem_usage*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======

ZGSSVX solves the system of linear equations A*X=B or A'*X=B, using
the LU factorization from zgstrf(). Error bounds on the solution and
a condition estimate are also provided. It performs the following steps:

  1. If A is stored column-wise (A->Stype = SLU_NC):

     1.1. If options->Equil = YES, scaling factors are computed to
          equilibrate the system:
          options->Trans = NOTRANS:
              diag(R)*A*diag(C) *inv(diag(C))*X = diag(R)*B
          options->Trans = TRANS:
              (diag(R)*A*diag(C))**T *inv(diag(R))*X = diag(C)*B
          options->Trans = CONJ:
              (diag(R)*A*diag(C))**H *inv(diag(R))*X = diag(C)*B
          Whether or not the system will be equilibrated depends on the
          scaling of the matrix A, but if equilibration is used, A is
          overwritten by diag(R)*A*diag(C) and B by diag(R)*B
          (if options->Trans=NOTRANS) or diag(C)*B (if options->Trans
          = TRANS or CONJ).

     1.2. Permute columns of A, forming A*Pc, where Pc is a permutation
          matrix that usually preserves sparsity.
          For more details of this step, see sp_preorder.c.

     1.3. If options->Fact != FACTORED, the LU decomposition is used to
          factor the matrix A (after equilibration if options->Equil = YES)
          as Pr*A*Pc = L*U, with Pr determined by partial pivoting.

     1.4. Compute the reciprocal pivot growth factor.

     1.5. If some U(i,i) = 0, so that U is exactly singular, then the
          routine returns with info = i. Otherwise, the factored form of
          A is used to estimate the condition number of the matrix A. If
          the reciprocal of the condition number is less than machine
          precision, info = A->ncol+1 is returned as a warning, but the
          routine still goes on to solve for X and computes error bounds
          as described below.
```

```
    1.6. The system of equations is solved for X using the factored form
         of A.

    1.7. If options->IterRefine != NOREFINE, iterative refinement is
         applied to improve the computed solution matrix and calculate
         error bounds and backward error estimates for it.

    1.8. If equilibration was used, the matrix X is premultiplied by
         diag(C) (if options->Trans = NOTRANS) or diag(R)
         (if options->Trans = TRANS or CONJ) so that it solves the
         original system before equilibration.

 2. If A is stored row-wise (A->Stype = SLU_NR), apply the above algorithm
    to the transpose of A:

    2.1. If options->Equil = YES, scaling factors are computed to
         equilibrate the system:
         options->Trans = NOTRANS:
             diag(R)*A*diag(C) *inv(diag(C))*X = diag(R)*B
         options->Trans = TRANS:
             (diag(R)*A*diag(C))**T *inv(diag(R))*X = diag(C)*B
         options->Trans = CONJ:
             (diag(R)*A*diag(C))**H *inv(diag(R))*X = diag(C)*B
         Whether or not the system will be equilibrated depends on the
         scaling of the matrix A, but if equilibration is used, A' is
         overwritten by diag(R)*A'*diag(C) and B by diag(R)*B
         (if trans='N') or diag(C)*B (if trans = 'T' or 'C').

    2.2. Permute columns of transpose(A) (rows of A),
         forming transpose(A)*Pc, where Pc is a permutation matrix that
         usually preserves sparsity.
         For more details of this step, see sp_preorder.c.

    2.3. If options->Fact != FACTORED, the LU decomposition is used to
         factor the transpose(A) (after equilibration if
         options->Fact = YES) as Pr*transpose(A)*Pc = L*U with the
         permutation Pr determined by partial pivoting.

    2.4. Compute the reciprocal pivot growth factor.

    2.5. If some U(i,i) = 0, so that U is exactly singular, then the
         routine returns with info = i. Otherwise, the factored form
         of transpose(A) is used to estimate the condition number of the
         matrix A. If the reciprocal of the condition number
         is less than machine precision, info = A->nrow+1 is returned as
         a warning, but the routine still goes on to solve for X and
         computes error bounds as described below.

    2.6. The system of equations is solved for X using the factored form
         of transpose(A).

    2.7. If options->IterRefine != NOREFINE, iterative refinement is
         applied to improve the computed solution matrix and calculate
         error bounds and backward error estimates for it.
```

```
    2.8. If equilibration was used, the matrix X is premultiplied by
         diag(C) (if options->Trans = NOTRANS) or diag(R)
         (if options->Trans = TRANS or CONJ) so that it solves the
         original system before equilibration.

 See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

options (input) superlu_options_t*
        The structure defines the input parameters to control
        how the LU decomposition will be performed and how the
        system will be solved.

A       (input/output) SuperMatrix*
        Matrix A in A*X=B, of dimension (A->nrow, A->ncol). The number
        of the linear equations is A->nrow. Currently, the type of A can be:
        Stype = SLU_NC or SLU_NR, Dtype = SLU_D, Mtype = SLU_GE.
        In the future, more general A may be handled.

        On entry, If options->Fact = FACTORED and equed is not 'N',
        then A must have been equilibrated by the scaling factors in
        R and/or C.
        On exit, A is not modified if options->Equil = NO, or if
        options->Equil = YES but equed = 'N' on exit.
        Otherwise, if options->Equil = YES and equed is not 'N',
        A is scaled as follows:
        If A->Stype = SLU_NC:
          equed = 'R':  A := diag(R) * A
          equed = 'C':  A := A * diag(C)
          equed = 'B':  A := diag(R) * A * diag(C).
        If A->Stype = SLU_NR:
          equed = 'R':  transpose(A) := diag(R) * transpose(A)
          equed = 'C':  transpose(A) := transpose(A) * diag(C)
          equed = 'B':  transpose(A) := diag(R) * transpose(A) * diag(C).

perm_c  (input/output) int*
  If A->Stype = SLU_NC, Column permutation vector of size A->ncol,
        which defines the permutation matrix Pc; perm_c[i] = j means
        column i of A is in position j in A*Pc.
        On exit, perm_c may be overwritten by the product of the input
        perm_c and a permutation that postorders the elimination tree
        of Pc'*A'*A*Pc; perm_c is not changed if the elimination tree
        is already in postorder.

        If A->Stype = SLU_NR, column permutation vector of size A->nrow,
        which describes permutation of columns of transpose(A)
        (rows of A) as described above.

perm_r  (input/output) int*
        If A->Stype = SLU_NC, row permutation vector of size A->nrow,
        which defines the permutation matrix Pr, and is determined
        by partial pivoting.  perm_r[i] = j means row i of A is in
        position j in Pr*A.
```

```
        If A->Stype = SLU_NR, permutation vector of size A->ncol, which
        determines permutation of rows of transpose(A)
        (columns of A) as described above.

        If options->Fact = SamePattern_SameRowPerm, the pivoting routine
        will try to use the input perm_r, unless a certain threshold
        criterion is violated. In that case, perm_r is overwritten by a
        new permutation determined by partial pivoting or diagonal
        threshold pivoting.
        Otherwise, perm_r is output argument.

etree   (input/output) int*,  dimension (A->ncol)
        Elimination tree of Pc'*A'*A*Pc.
        If options->Fact != FACTORED and options->Fact != DOFACT,
        etree is an input argument, otherwise it is an output argument.
        Note: etree is a vector of parent pointers for a forest whose
        vertices are the integers 0 to A->ncol-1; etree[root]==A->ncol.

equed   (input/output) char*
        Specifies the form of equilibration that was done.
        = 'N': No equilibration.
        = 'R': Row equilibration, i.e., A was premultiplied by diag(R).
        = 'C': Column equilibration, i.e., A was postmultiplied by diag(C).
        = 'B': Both row and column equilibration, i.e., A was replaced
               by diag(R)*A*diag(C).
        If options->Fact = FACTORED, equed is an input argument,
        otherwise it is an output argument.

R       (input/output) double*, dimension (A->nrow)
        The row scale factors for A or transpose(A).
        If equed = 'R' or 'B', A (if A->Stype = SLU_NC) or transpose(A)
            (if A->Stype = SLU_NR) is multiplied on the left by diag(R).
        If equed = 'N' or 'C', R is not accessed.
        If options->Fact = FACTORED, R is an input argument,
            otherwise, R is output.
        If options->zFact = FACTORED and equed = 'R' or 'B', each element
            of R must be positive.

C       (input/output) double*, dimension (A->ncol)
        The column scale factors for A or transpose(A).
        If equed = 'C' or 'B', A (if A->Stype = SLU_NC) or transpose(A)
            (if A->Stype = SLU_NR) is multiplied on the right by diag(C).
        If equed = 'N' or 'R', C is not accessed.
        If options->Fact = FACTORED, C is an input argument,
            otherwise, C is output.
        If options->Fact = FACTORED and equed = 'C' or 'B', each element
            of C must be positive.

L       (output) SuperMatrix*
  The factor L from the factorization
            Pr*A*Pc=L*U              (if A->Stype SLU_= NC) or
            Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
        Uses compressed row subscripts storage for supernodes, i.e.,
        L has types: Stype = SLU_SC, Dtype = SLU_Z, Mtype = SLU_TRLU.

U       (output) SuperMatrix*
```

```
  The factor U from the factorization
          Pr*A*Pc=L*U              (if A->Stype = SLU_NC) or
          Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
        Uses column-wise storage scheme, i.e., U has types:
        Stype = SLU_NC, Dtype = SLU_Z, Mtype = SLU_TRU.

work    (workspace/output) void*, size (lwork) (in bytes)
        User supplied workspace, should be large enough
        to hold data structures for factors L and U.
        On exit, if fact is not 'F', L and U point to this array.

lwork   (input) int
        Specifies the size of work array in bytes.
        = 0:  allocate space internally by system malloc;
        > 0:  use user-supplied work array of length lwork in bytes,
              returns error if space runs out.
        = -1: the routine guesses the amount of space needed without
              performing the factorization, and returns it in
              mem_usage->total_needed; no other side effects.

        See argument 'mem_usage' for memory usage statistics.

B       (input/output) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_Z, Mtype = SLU_GE.
        On entry, the right hand side matrix.
        If B->ncol = 0, only LU decomposition is performed, the triangular
                        solve is skipped.
        On exit,
           if equed = 'N', B is not modified; otherwise
           if A->Stype = SLU_NC:
              if options->Trans = NOTRANS and equed = 'R' or 'B',
                 B is overwritten by diag(R)*B;
              if options->Trans = TRANS or CONJ and equed = 'C' of 'B',
                 B is overwritten by diag(C)*B;
           if A->Stype = SLU_NR:
              if options->Trans = NOTRANS and equed = 'C' or 'B',
                 B is overwritten by diag(C)*B;
              if options->Trans = TRANS or CONJ and equed = 'R' of 'B',
                 B is overwritten by diag(R)*B.

X       (output) SuperMatrix*
        X has types: Stype = SLU_DN, Dtype = SLU_Z, Mtype = SLU_GE.
        If info = 0 or info = A->ncol+1, X contains the solution matrix
        to the original system of equations. Note that A and B are modified
        on exit if equed is not 'N', and the solution to the equilibrated
        system is inv(diag(C))*X if options->Trans = NOTRANS and
        equed = 'C' or 'B', or inv(diag(R))*X if options->Trans = 'T' or 'C'
        and equed = 'R' or 'B'.

recip_pivot_growth (output) double*
        The reciprocal pivot growth factor max_j( norm(A_j)/norm(U_j) ).
        The infinity norm is used. If recip_pivot_growth is much less
        than 1, the stability of the LU factorization could be poor.

rcond   (output) double*
        The estimate of the reciprocal condition number of the matrix A
```

```
        after equilibration (if done). If rcond is less than the machine
        precision (in particular, if rcond = 0), the matrix is singular
        to working precision. This condition is indicated by a return
        code of info > 0.

FERR    (output) double*, dimension (B->ncol)
        The estimated forward error bound for each solution vector
        X(j) (the j-th column of the solution matrix X).
        If XTRUE is the true solution corresponding to X(j), FERR(j)
        is an estimated upper bound for the magnitude of the largest
        element in (X(j) - XTRUE) divided by the magnitude of the
        largest element in X(j).  The estimate is as reliable as
        the estimate for RCOND, and is almost always a slight
        overestimate of the true error.
        If options->IterRefine = NOREFINE, ferr = 1.0.

BERR    (output) double*, dimension (B->ncol)
        The componentwise relative backward error of each solution
        vector X(j) (i.e., the smallest relative change in
        any element of A or B that makes X(j) an exact solution).
        If options->IterRefine = NOREFINE, berr = 1.0.

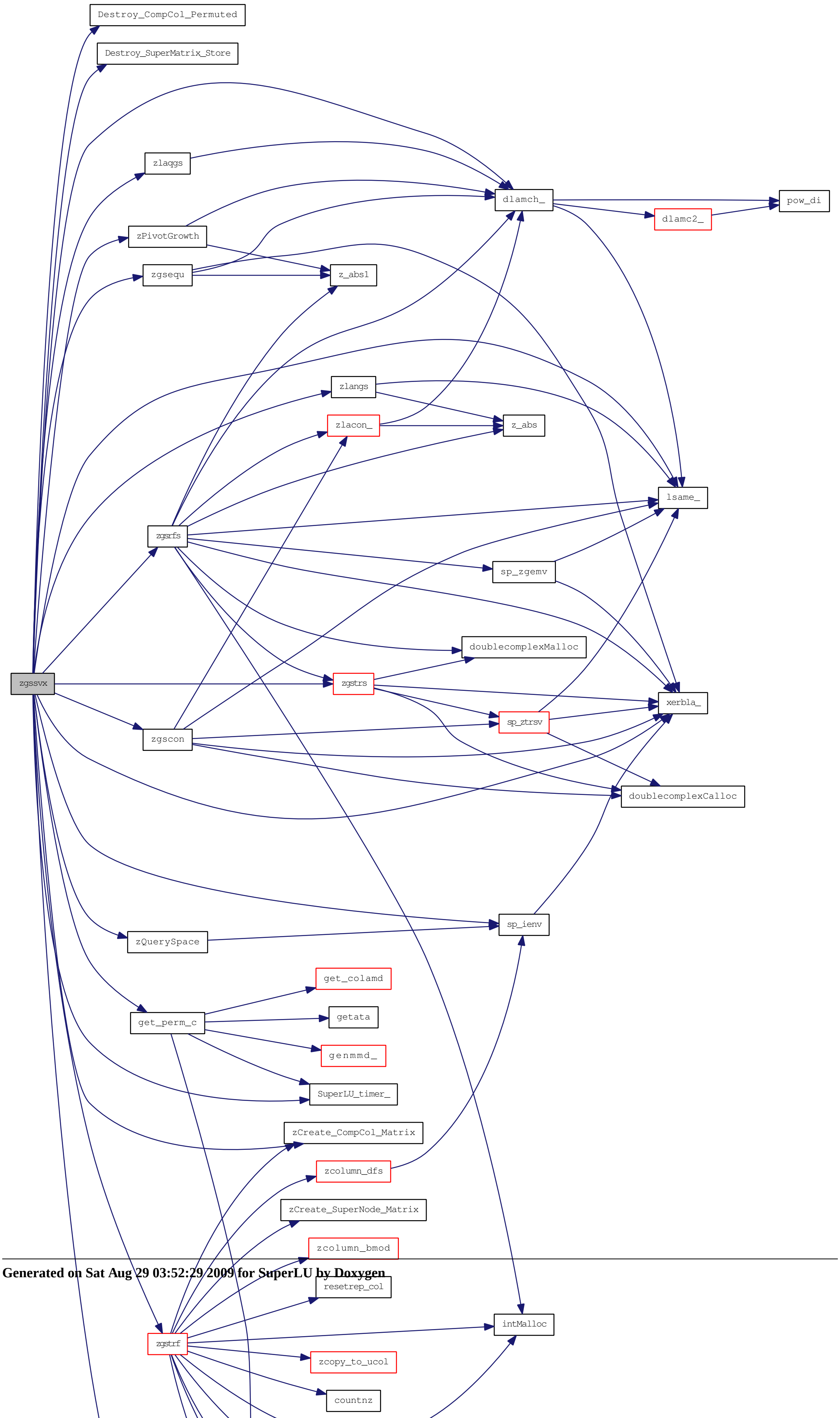
mem_usage (output) mem_usage_t*
        Record the memory usage statistics, consisting of following fields:
```

- for_lu (float)
  ```
  The amount of space used in bytes for L data structures.
  ```
- total_needed (float)
  ```
  The amount of space needed in bytes to perform factorization.
  ```
- expansions (int)
  ```
  The number of memory expansions during the LU factorization.
  ```

```
stat   (output) SuperLUStat_t*
       Record the statistics on runtime and floating-point operation count.
       See util.h for the definition of 'SuperLUStat_t'.

info    (output) int*
        = 0: successful exit
        < 0: if info = -i, the i-th argument had an illegal value
        > 0: if info = i, and i is
             <= A->ncol: U(i,i) is exactly zero. The factorization has
                   been completed, but the factor U is exactly
                   singular, so the solution and error bounds
                   could not be computed.
             = A->ncol+1: U is nonsingular, but RCOND is less than machine
                   precision, meaning that the matrix is singular to
                   working precision. Nevertheless, the solution and
                   error bounds are computed because there are a number
                   of situations where the computed solution can be more
                   accurate than the value of RCOND would suggest.
             > A->ncol+1: number of bytes allocated when memory allocation
                   failure occurred, plus A->ncol.
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.117.3.32 void zgstrf (superlu_options_t * *options*, SuperMatrix * *A*, double *drop_tol*, int *relax*, int *panel_size*, int * *etree*, void * *work*, int *lwork*, int * *perm_c*, int * *perm_r*, SuperMatrix * *L*, SuperMatrix * *U*, SuperLUStat_t * *stat*, int * *info*)

```
Purpose
=======

ZGSTRF computes an LU factorization of a general sparse m-by-n
matrix A using partial pivoting with row interchanges.
The factorization has the form
    Pr * A = L * U
where Pr is a row permutation matrix, L is lower triangular with unit
diagonal elements (lower trapezoidal if A->nrow > A->ncol), and U is upper
triangular (upper trapezoidal if A->nrow < A->ncol).

See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

options (input) superlu_options_t*
        The structure defines the input parameters to control
        how the LU decomposition will be performed.

A        (input) SuperMatrix*
   Original matrix A, permuted by columns, of dimension
         (A->nrow, A->ncol). The type of A can be:
         Stype = SLU_NCP; Dtype = SLU_Z; Mtype = SLU_GE.

drop_tol (input) double (NOT IMPLEMENTED)
   Drop tolerance parameter. At step j of the Gaussian elimination,
         if abs(A_ij)/(max_i abs(A_ij)) < drop_tol, drop entry A_ij.
         0 <= drop_tol <= 1. The default value of drop_tol is 0.

relax    (input) int
         To control degree of relaxing supernodes. If the number
         of nodes (columns) in a subtree of the elimination tree is less
         than relax, this subtree is considered as one supernode,
         regardless of the row structures of those columns.

panel_size (input) int
         A panel consists of at most panel_size consecutive columns.

etree    (input) int*, dimension (A->ncol)
         Elimination tree of A'*A.
         Note: etree is a vector of parent pointers for a forest whose
         vertices are the integers 0 to A->ncol-1; etree[root]==A->ncol.
         On input, the columns of A should be permuted so that the
         etree is in a certain postorder.
```

```
work     (input/output) void*, size (lwork) (in bytes)
         User-supplied work space and space for the output data structures.
         Not referenced if lwork = 0;

lwork   (input) int
        Specifies the size of work array in bytes.
        = 0:  allocate space internally by system malloc;
        > 0:  use user-supplied work array of length lwork in bytes,
              returns error if space runs out.
        = -1: the routine guesses the amount of space needed without
              performing the factorization, and returns it in
              *info; no other side effects.

perm_c   (input) int*, dimension (A->ncol)
   Column permutation vector, which defines the
         permutation matrix Pc; perm_c[i] = j means column i of A is
         in position j in A*Pc.
         When searching for diagonal, perm_c[*] is applied to the
         row subscripts of A, so that diagonal threshold pivoting
         can find the diagonal of A, rather than that of A*Pc.

perm_r   (input/output) int*, dimension (A->nrow)
         Row permutation vector which defines the permutation matrix Pr,
         perm_r[i] = j means row i of A is in position j in Pr*A.
         If options->Fact = SamePattern_SameRowPerm, the pivoting routine
            will try to use the input perm_r, unless a certain threshold
            criterion is violated. In that case, perm_r is overwritten by
            a new permutation determined by partial pivoting or diagonal
            threshold pivoting.
         Otherwise, perm_r is output argument;

L        (output) SuperMatrix*
         The factor L from the factorization Pr*A=L*U; use compressed row
         subscripts storage for supernodes, i.e., L has type:
         Stype = SLU_SC, Dtype = SLU_Z, Mtype = SLU_TRLU.

U        (output) SuperMatrix*
   The factor U from the factorization Pr*A*Pc=L*U. Use column-wise
         storage scheme, i.e., U has types: Stype = SLU_NC,
         Dtype = SLU_Z, Mtype = SLU_TRU.

stat     (output) SuperLUStat_t*
         Record the statistics on runtime and floating-point operation count.
         See util.h for the definition of 'SuperLUStat_t'.

info     (output) int*
         = 0: successful exit
         < 0: if info = -i, the i-th argument had an illegal value
         > 0: if info = i, and i is
            <= A->ncol: U(i,i) is exactly zero. The factorization has
               been completed, but the factor U is exactly singular,
               and division by zero will occur if it is used to solve a
               system of equations.
            > A->ncol: number of bytes allocated when memory allocation
               failure occurred, plus A->ncol. If lwork = -1, it is
               the estimated amount of space needed, plus A->ncol.
```

```
 ======================================================================


 Local Working Arrays:
 ======================
   m = number of rows in the matrix
   n = number of columns in the matrix


   xprune[0:n-1]: xprune[*] points to locations in subscript
vector lsub[*]. For column i, xprune[i] denotes the point where
structural pruning begins. I.e. only xlsub[i],..,xprune[i]-1 need
to be traversed for symbolic factorization.


   marker[0:3*m-1]: marker[i] = j means that node i has been
reached when working on column j.
Storage: relative to original row subscripts
NOTE: There are 3 of them: marker/marker1 are used for panel dfs,
      see zpanel_dfs.c; marker2 is used for inner-factorization,
            see zcolumn_dfs.c.


   parent[0:m-1]: parent vector used during dfs
      Storage: relative to new row subscripts


   xplore[0:m-1]: xplore[i] gives the location of the next (dfs)
unexplored neighbor of i in lsub[*]


   segrep[0:nseg-1]: contains the list of supernodal representatives
in topological order of the dfs. A supernode representative is the
last column of a supernode.
      The maximum size of segrep[] is n.

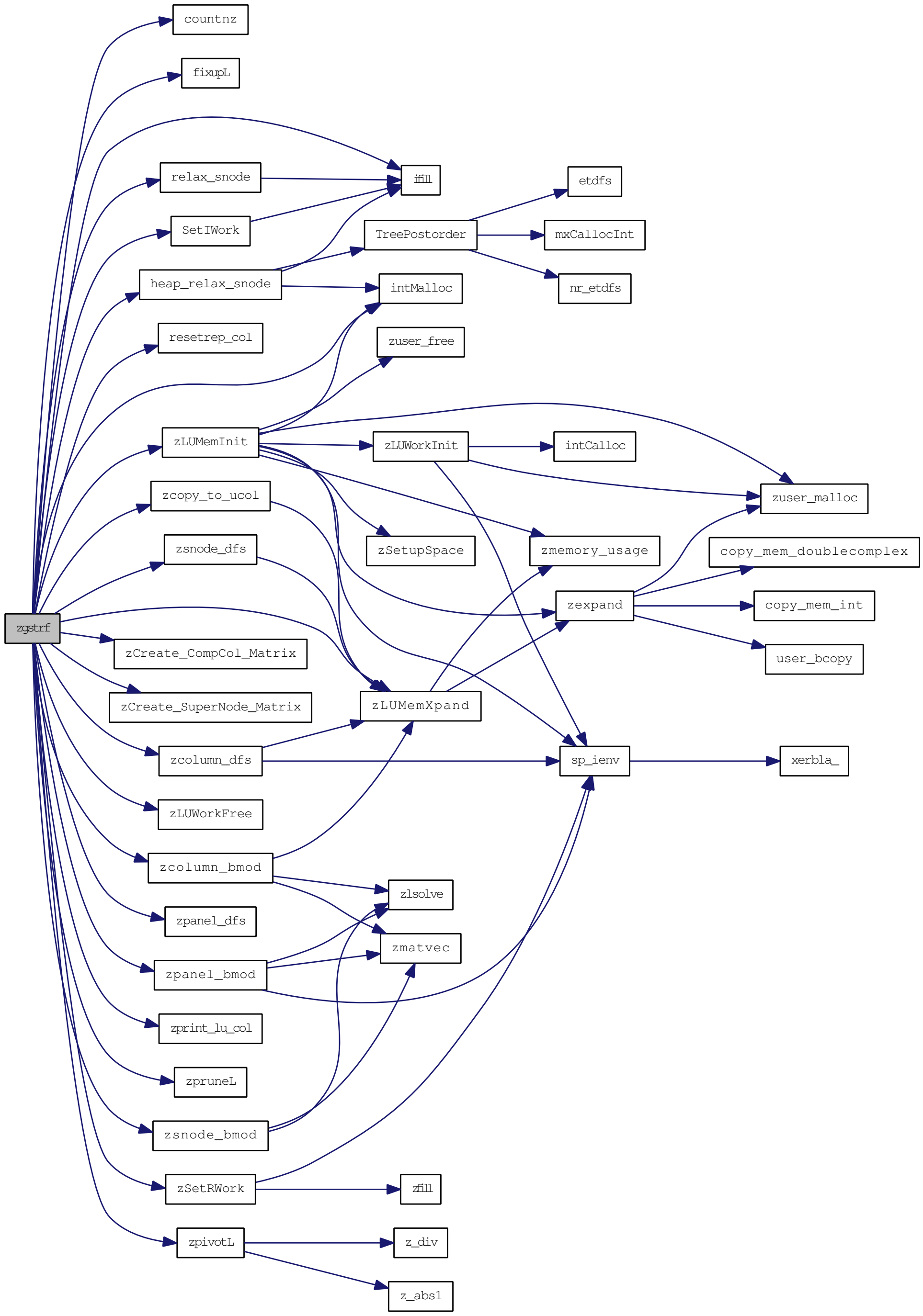

   repfnz[0:W*m-1]: for a nonzero segment U[*,j] that ends at a
supernodal representative r, repfnz[r] is the location of the first
nonzero in this segment.  It is also used during the dfs: repfnz[r]>0
indicates the supernode r has been explored.
NOTE: There are W of them, each used for one column of a panel.


   panel_lsub[0:W*m-1]: temporary for the nonzeros row indices below
      the panel diagonal. These are filled in during zpanel_dfs(), and are
      used later in the inner LU factorization within the panel.
panel_lsub[]/dense[] pair forms the SPA data structure.
NOTE: There are W of them.

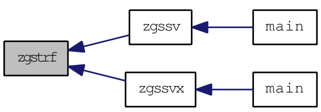

   dense[0:W*m-1]: sparse accumulating (SPA) vector for intermediate values;
        NOTE: there are W of them.


   tempv[0:*]: real temporary used for dense numeric kernels;
The size of this array is defined by NUM_TEMPV() in slu_zdefs.h.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.117.3.33 void zgstrs (trans_t *trans*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, int ∗ *perm_c*, int ∗ *perm_r*, SuperMatrix ∗ *B*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======

ZGSTRS solves a system of linear equations A*X=B or A'*X=B
with A sparse and B dense, using the LU factorization computed by
ZGSTRF.

See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

trans   (input) trans_t
         Specifies the form of the system of equations:
         = NOTRANS: A * X = B  (No transpose)
         = TRANS:   A'* X = B  (Transpose)
         = CONJ:    A**H * X = B  (Conjugate transpose)

L       (input) SuperMatrix*
        The factor L from the factorization Pr*A*Pc=L*U as computed by
        zgstrf(). Use compressed row subscripts storage for supernodes,
        i.e., L has types: Stype = SLU_SC, Dtype = SLU_Z, Mtype = SLU_TRLU.

U       (input) SuperMatrix*
        The factor U from the factorization Pr*A*Pc=L*U as computed by
        zgstrf(). Use column-wise storage scheme, i.e., U has types:
        Stype = SLU_NC, Dtype = SLU_Z, Mtype = SLU_TRU.

perm_c  (input) int*, dimension (L->ncol)
  Column permutation vector, which defines the
        permutation matrix Pc; perm_c[i] = j means column i of A is
        in position j in A*Pc.

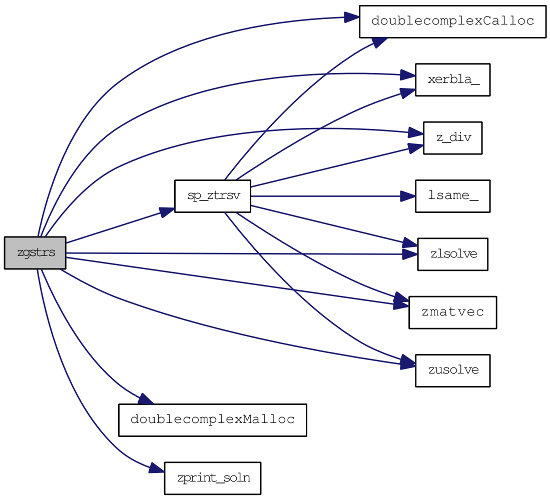
perm_r  (input) int*, dimension (L->nrow)
        Row permutation vector, which defines the permutation matrix Pr;
        perm_r[i] = j means row i of A is in position j in Pr*A.

B       (input/output) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_Z, Mtype = SLU_GE.
        On entry, the right hand side matrix.
        On exit, the solution matrix if info = 0;

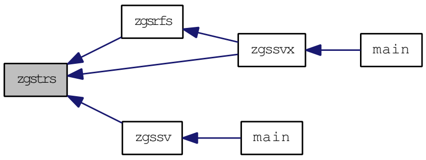
stat     (output) SuperLUStat_t*
         Record the statistics on runtime and floating-point operation count.
         See util.h for the definition of 'SuperLUStat_t'.

info    (output) int*
    = 0: successful exit
  < 0: if info = -i, the i-th argument had an illegal value
```

Here is the call graph for this function:

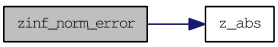


Here is the caller graph for this function:

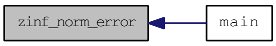


4.117.3.34 void zinf_norm_error (int, SuperMatrix ∗, doublecomplex ∗)

Here is the call graph for this function:



Here is the caller graph for this function:



4.117.3.35 void zlaqgs (SuperMatrix ∗ *A*, double ∗ *r*, double ∗ *c*, double *rowcnd*, double *colcnd*, double *amax*, char ∗ *equed*)

```
Purpose
=======

ZLAQGS equilibrates a general sparse M by N matrix A using the row and
scaling factors in the vectors R and C.
```

```
See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

A       (input/output) SuperMatrix*
        On exit, the equilibrated matrix.  See EQUED for the form of
        the equilibrated matrix. The type of A can be:
 Stype = NC; Dtype = SLU_Z; Mtype = GE.

R       (input) double*, dimension (A->nrow)
        The row scale factors for A.

C       (input) double*, dimension (A->ncol)
        The column scale factors for A.

ROWCND  (input) double
        Ratio of the smallest R(i) to the largest R(i).

COLCND  (input) double
        Ratio of the smallest C(i) to the largest C(i).

AMAX    (input) double
        Absolute value of largest matrix entry.

EQUED   (output) char*
        Specifies the form of equilibration that was done.
        = 'N':  No equilibration
        = 'R':  Row equilibration, i.e., A has been premultiplied by
                diag(R).
        = 'C':  Column equilibration, i.e., A has been postmultiplied
                by diag(C).
        = 'B':  Both row and column equilibration, i.e., A has been
                replaced by diag(R) * A * diag(C).

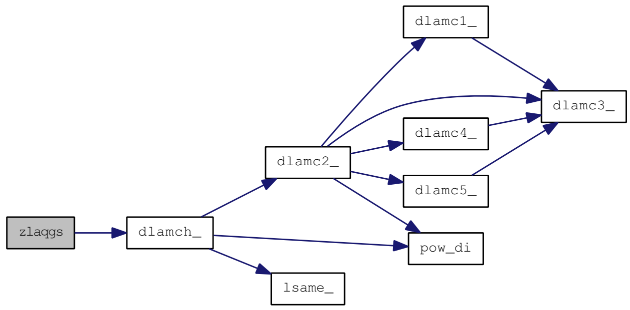
Internal Parameters
===================

THRESH is a threshold value used to decide if row or column scaling
should be done based on the ratio of the row or column scaling
factors.  If ROWCND < THRESH, row scaling is done, and if
COLCND < THRESH, column scaling is done.

LARGE and SMALL are threshold values used to decide if row scaling
should be done based on the absolute size of the largest matrix
element.  If AMAX > LARGE or AMAX < SMALL, row scaling is done.

=====================================================================
```

Here is the call graph for this function:



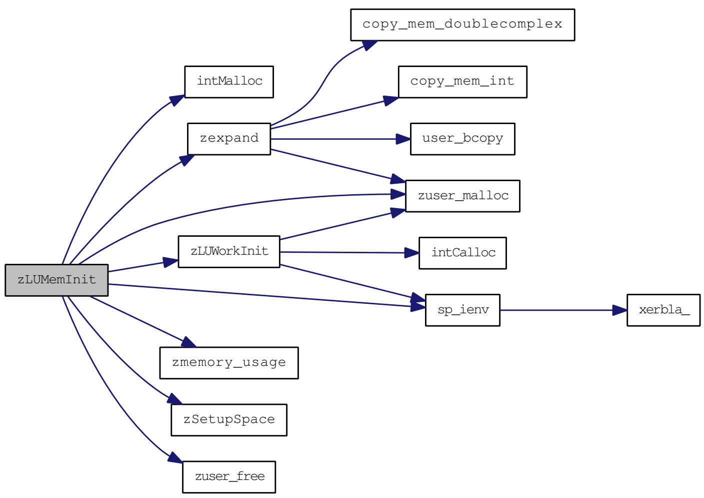
Here is the caller graph for this function:

4.117.3.36 int zLUMemInit (fact_t *fact*, void ∗ *work*, int *lwork*, int *m*, int *n*, int *annz*, int *panel_size*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, GlobalLU_t ∗ *Glu*, int ∗∗ *iwork*, doublecomplex ∗∗ *dwork*)

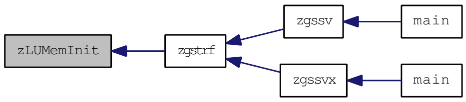
Memory-related.

```
For those unpredictable size, make a guess as FILL * nnz(A).
Return value:
    If lwork = -1, return the estimated amount of space required, plus n;
    otherwise, return the amount of space actually allocated when
    memory allocation failure occurred.
```

Here is the call graph for this function:



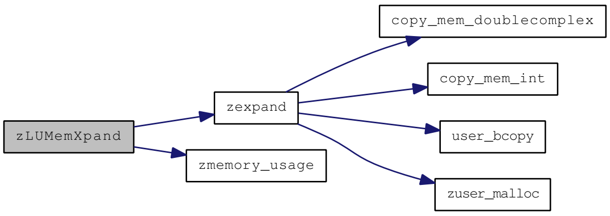
Here is the caller graph for this function:



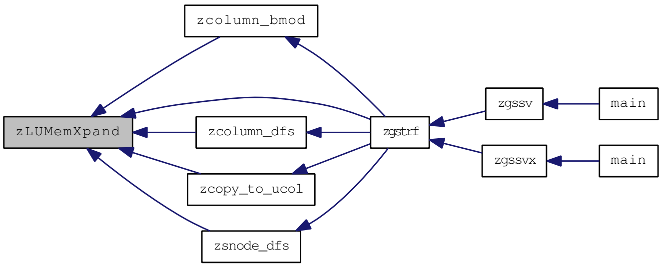
4.117.3.37 int zLUMemXpand (int *jcol*, int *next*, MemType *mem_type*, int ∗ *maxlen*, GlobalLU_t ∗ *Glu*)

```
Return value:   0 - successful return
              > 0 - number of bytes allocated when run out of space
```

Here is the call graph for this function:

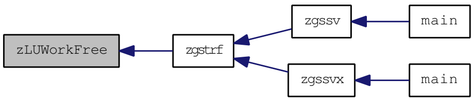


Here is the caller graph for this function:



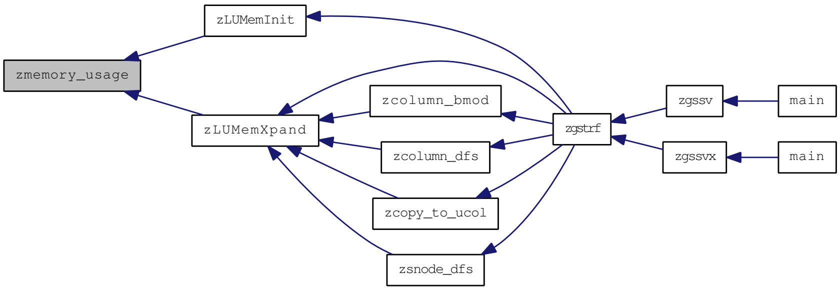
4.117.3.38 void zLUWorkFree (int ∗, doublecomplex ∗, GlobalLU_t ∗)

Here is the caller graph for this function:



4.117.3.39 int zmemory_usage (const *int*, const *int*, const *int*, const *int*)

Here is the caller graph for this function:



4.117.3.40 void zpanel_bmod (const int *m*, const int *w*, const int *jcol*, const int *nseg*, doublecomplex ∗ *dense*, doublecomplex ∗ *tempv*, int ∗ *segrep*, int ∗ *repfnz*, GlobalLU_t ∗ *Glu*, SuperLUStat_t ∗ *stat*)

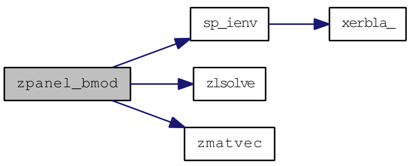
```
Purpose
=======

   Performs numeric block updates (sup-panel) in topological order.
   It features: col-col, 2cols-col, 3cols-col, and sup-col updates.
   Special processing on the supernodal portion of L[*,j]

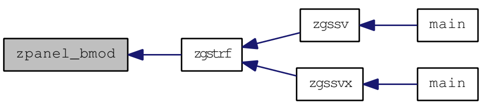
   Before entering this routine, the original nonzeros in the panel
   were already copied into the spa[m,w].

   Updated/Output parameters-
   dense[0:m-1,w]: L[*,j:j+w-1] and U[*,j:j+w-1] are returned
   collectively in the m-by-w vector dense[*].
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.117.3.41 void zpanel_dfs (const int *m*, const int *w*, const int *jcol*, SuperMatrix ∗ *A*, int ∗ *perm_r*, int ∗ *nseg*, doublecomplex ∗ *dense*, int ∗ *panel_lsub*, int ∗ *segrep*, int ∗ *repfnz*, int ∗ *xprune*, int ∗ *marker*, int ∗ *parent*, int ∗ *xplore*, GlobalLU_t ∗ *Glu*)

```
Purpose
=======

  Performs a symbolic factorization on a panel of columns [jcol, jcol+w).

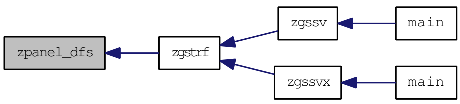
  A supernode representative is the last column of a supernode.
  The nonzeros in U[*,j] are segments that end at supernodal
  representatives.

  The routine returns one list of the supernodal representatives
  in topological order of the dfs that generates them. This list is
  a superset of the topological order of each individual column within
  the panel.
  The location of the first nonzero in each supernodal segment
  (supernodal entry location) is also returned. Each column has a
  separate list for this purpose.

  Two marker arrays are used for dfs:
    marker[i] == jj, if i was visited during dfs of current column jj;
    marker1[i] >= jcol, if i was visited by earlier columns in this panel;

  marker: A-row --> A-row/col (0/1)
  repfnz: SuperA-col --> PA-row
  parent: SuperA-col --> SuperA-col
  xplore: SuperA-col --> index to L-structure
```

Here is the caller graph for this function:



4.117.3.42 double zPivotGrowth (int *ncols*, SuperMatrix ∗ *A*, int ∗ *perm_c*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*)

```
Purpose
=======

Compute the reciprocal pivot growth factor of the leading ncols columns
of the matrix, using the formula:
    min_j ( max_i(abs(A_ij)) / max_i(abs(U_ij)) )

Arguments
=========

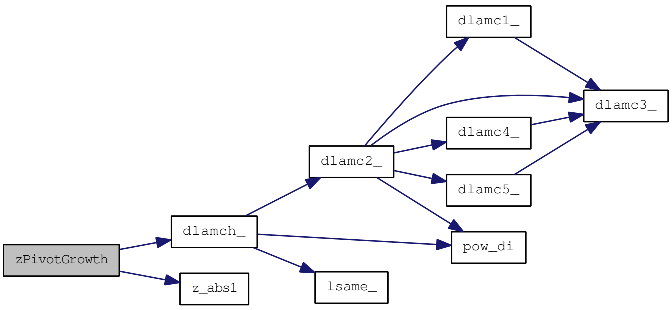
ncols    (input) int
         The number of columns of matrices A, L and U.
```

```
A        (input) SuperMatrix*
   Original matrix A, permuted by columns, of dimension
         (A->nrow, A->ncol). The type of A can be:
         Stype = NC; Dtype = SLU_Z; Mtype = GE.

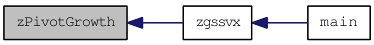
L        (output) SuperMatrix*
         The factor L from the factorization Pr*A=L*U; use compressed row
         subscripts storage for supernodes, i.e., L has type:
         Stype = SC; Dtype = SLU_Z; Mtype = TRLU.

U        (output) SuperMatrix*
   The factor U from the factorization Pr*A*Pc=L*U. Use column-wise
         storage scheme, i.e., U has types: Stype = NC;
         Dtype = SLU_Z; Mtype = TRU.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.117.3.43 int zpivotL (const int *jcol*, const double *u*, int ∗ *usepr*, int ∗ *perm_r*, int ∗ *iperm_r*, int ∗ *iperm_c*, int ∗ *pivrow*, GlobalLU_t ∗ *Glu*, SuperLUStat_t ∗ *stat*)

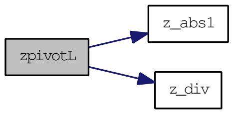
```
Purpose
=======
  Performs the numerical pivoting on the current column of L,
  and the CDIV operation.

  Pivot policy:
  (1) Compute thresh = u * max_(i>=j) abs(A_ij);
  (2) IF user specifies pivot row k and abs(A_kj) >= thresh THEN
          pivot row = k;
      ELSE IF abs(A_jj) >= thresh THEN
          pivot row = j;
      ELSE
          pivot row = m;

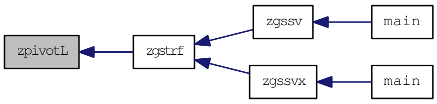
  Note: If you absolutely want to use a given pivot order, then set u=0.0.
```

```
Return value: 0      success;
              i > 0  U(i,i) is exactly zero.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.117.3.44 void zPrint_CompCol_Matrix (char ∗, SuperMatrix ∗)

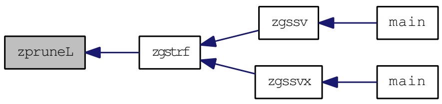
4.117.3.45 void zPrint_Dense_Matrix (char ∗, SuperMatrix ∗)

4.117.3.46 void zPrint_SuperNode_Matrix (char ∗, SuperMatrix ∗)

4.117.3.47 void zpruneL (const int *jcol*, const int ∗ *perm_r*, const int *pivrow*, const int *nseg*, const int ∗ *segrep*, const int ∗ *repfnz*, int ∗ *xprune*, GlobalLU_t ∗ *Glu*)

```
Purpose
=======
  Prunes the L-structure of supernodes whose L-structure
  contains the current pivot row "pivrow"
```

Here is the caller graph for this function:



4.117.3.48 int zQuerySpace (SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, mem_usage_t ∗ *mem_usage*)

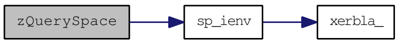
```
mem_usage consists of the following fields:
```

- `for_lu (float)`
  `The amount of space used in bytes for the L data structures.`
- `total_needed (float)`
  `The amount of space needed in bytes to perform factorization.`

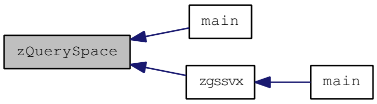
- `expansions (int)`

  `Number of memory expansions during the LU factorization.`

Here is the call graph for this function:

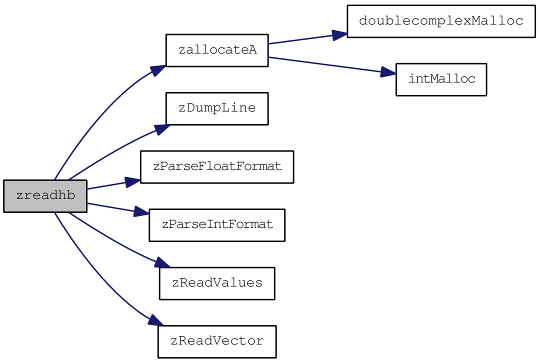


Here is the caller graph for this function:

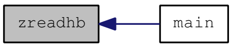


4.117.3.49 void zreadhb (int ∗, int ∗, int ∗, doublecomplex ∗∗, int ∗∗, int ∗∗)

Here is the call graph for this function:



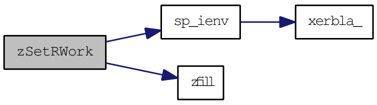
Here is the caller graph for this function:



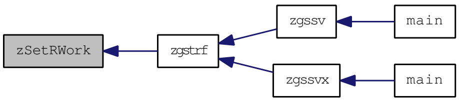
4.117.3.50 void zreadmt (int ∗, int ∗, int ∗, doublecomplex ∗∗, int ∗∗, int ∗∗)

4.117.3.51 void zSetRWork (int, int, doublecomplex ∗, doublecomplex ∗∗, doublecomplex ∗∗)

Here is the call graph for this function:

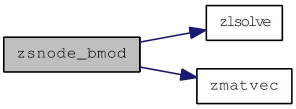


Here is the caller graph for this function:

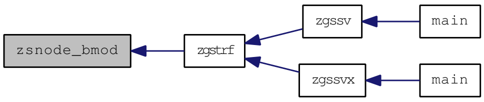


4.117.3.52 int zsnode_bmod (const *int*, const *int*, const *int*, doublecomplex ∗, doublecomplex ∗, GlobalLU_t ∗, SuperLUStat_t ∗)

Here is the call graph for this function:



Here is the caller graph for this function:



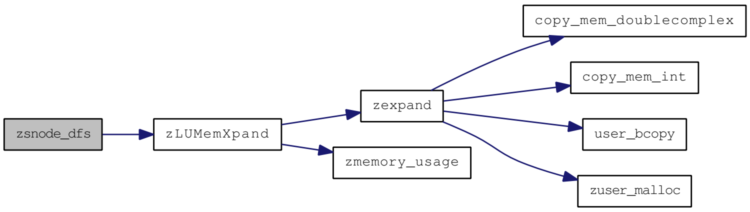
4.117.3.53 int zsnode_dfs (const int *jcol*, const int *kcol*, const int ∗ *asub*, const int ∗ *xa_begin*, const int ∗ *xa_end*, int ∗ *xprune*, int ∗ *marker*, GlobalLU_t ∗ *Glu*)

```
Purpose
=======
   zsnode_dfs() - Determine the union of the row structures of those
   columns within the relaxed snode.
   Note: The relaxed snodes are leaves of the supernodal etree, therefore,
   the portion outside the rectangular supernode must be zero.

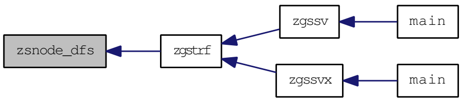

Return value
============
    0   success;
```

```
>0   number of bytes allocated when run out of memory.
```

Here is the call graph for this function:



Here is the caller graph for this function:

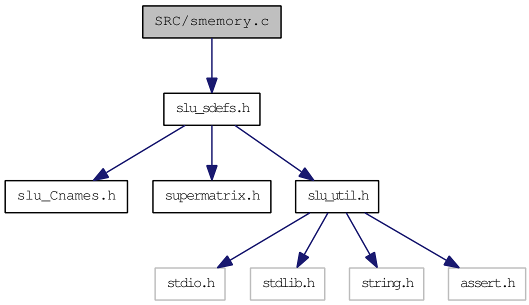


4.118 SRC/smemory.c File Reference

Memory details.

`#include "slu_sdefs.h"`

Include dependency graph for smemory.c:



Data Structures

- struct e_node

    Headers for 4 types of dynamatically managed memory.

- struct LU_stack_t

Defines

- #define NO_MEMTYPE 4
- #define GluIntArray(n) (5 ∗ (n) + 5)
- #define StackFull(x) (x + stack.used >= stack.size)
- #define NotDoubleAlign(addr) ((long int)addr & 7)
- #define DoubleAlign(addr) (((long int)addr + 7) & ∼7L)
- #define TempSpace(m, w)
- #define Reduce(alpha) ((alpha + 1) / 2)

Typedefs

- typedef struct e_node ExpHeader

    Headers for 4 types of dynamatically managed memory.

Functions

- void ∗ sexpand (int ∗prev_len,MemType type,int len_to_copy,int keep_prev,GlobalLU_t ∗Glu)

    Expand the existing storage to accommodate more fill-ins.

- int sLUWorkInit (int m, int n, int panel_size, int ∗∗iworkptr, float ∗∗dworkptr, LU_space_t MemModel)

Allocate known working storage. Returns 0 if success, otherwise returns the number of bytes allocated so far when failure occurred.

- void copy_mem_float (int, void ∗, void ∗)
- void sStackCompress (GlobalLU_t ∗Glu)

  Compress the work[] array to remove fragmentation.

- void sSetupSpace (void ∗work, int lwork, LU_space_t ∗MemModel)

  Setup the memory model to be used for factorization.

- void ∗ suser_malloc (int, int)
- void suser_free (int, int)
- void copy_mem_int (int, void ∗, void ∗)
- void user_bcopy (char ∗, char ∗, int)
- int sQuerySpace (SuperMatrix ∗L, SuperMatrix ∗U, mem_usage_t ∗mem_usage)
- int sLUMemInit (fact_t fact, void ∗work, int lwork, int m, int n, int annz, int panel_size, SuperMatrix ∗L, SuperMatrix ∗U, GlobalLU_t ∗Glu, int ∗∗iwork, float ∗∗dwork)

  Allocate storage for the data structures common to all factor routines.

- void sSetRWork (int m, int panel_size, float ∗dworkptr, float ∗∗dense, float ∗∗tempv)

  Set up pointers for real working arrays.

- void sLUWorkFree (int ∗iwork, float ∗dwork, GlobalLU_t ∗Glu)

  Free the working storage used by factor routines.

- int sLUMemXpand (int jcol, int next, MemType mem_type, int ∗maxlen, GlobalLU_t ∗Glu)

  Expand the data structures for L and U during the factorization.

- void sallocateA (int n, int nnz, float ∗∗a, int ∗∗asub, int ∗∗xa)

  Allocate storage for original matrix A.

- float ∗ floatMalloc (int n)
- float ∗ floatCalloc (int n)
- int smemory_usage (const int nzlmax, const int nzumax, const int nzlumax, const int n)

Variables

- static ExpHeader ∗ expanders = 0
- static LU_stack_t stack
- static int no_expand

4.118.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003
```

4.118.2 Define Documentation

4.118.2.1 #define DoubleAlign(addr) (((long int)addr + 7) & ∼7L)

4.118.2.2 #define GluIntArray(n) (5 ∗ (n) + 5)

4.118.2.3 #define NO_MEMTYPE 4

4.118.2.4 #define NotDoubleAlign(addr) ((long int)addr & 7)

4.118.2.5 #define Reduce(alpha) ((alpha + 1) / 2)

4.118.2.6 #define StackFull(x) (x + stack.used >= stack.size)

4.118.2.7 #define TempSpace(m, w)

Value:

```
( (2*w + 4 + NO_MARKER) * m * sizeof(int) + \
            (w + 1) * m * sizeof(float) )
```

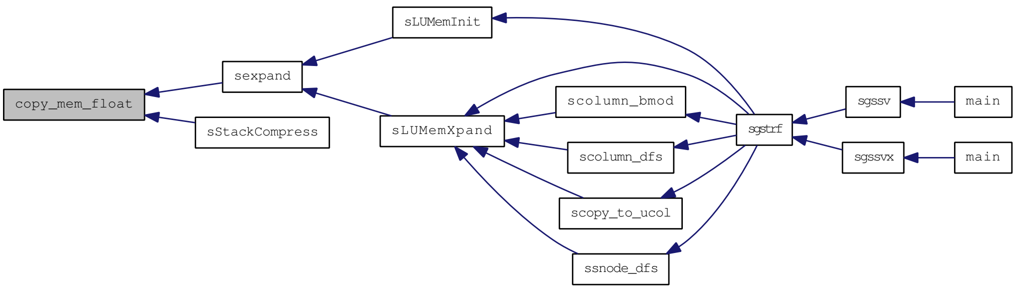
4.118.3 Typedef Documentation

4.118.3.1 typedef struct e_node ExpHeader

4.118.4 Function Documentation

4.118.4.1 void copy_mem_float (int *howmany*, void ∗ *old*, void ∗ *new*)

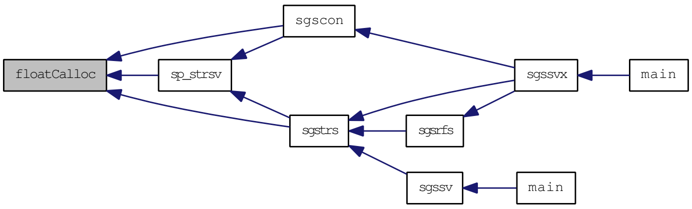
Here is the caller graph for this function:



4.118.4.2 void copy_mem_int (int, void ∗, void ∗)

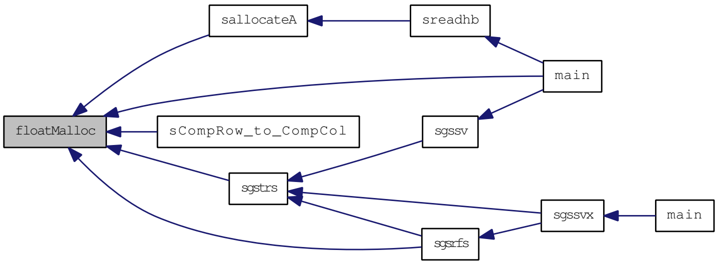
4.118.4.3 float∗ floatCalloc (int *n*)

Here is the caller graph for this function:



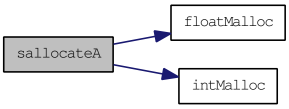
4.118.4.4 float∗ floatMalloc (int *n*)

Here is the caller graph for this function:



4.118.4.5 void sallocateA (int *n*, int *nnz*, float ∗∗ *a*, int ∗∗ *asub*, int ∗∗ *xa*)

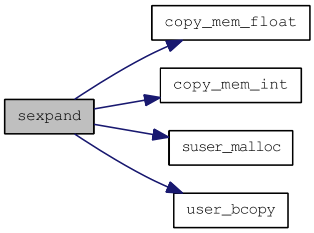
Here is the call graph for this function:



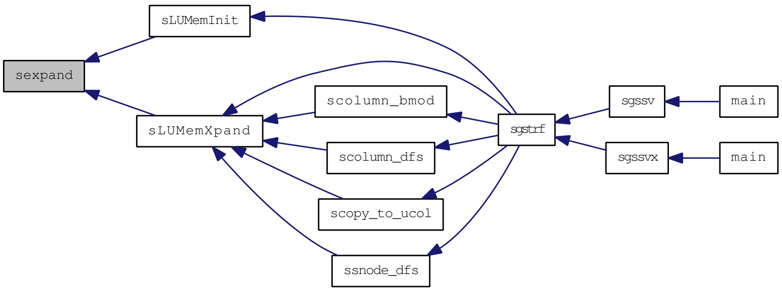
Here is the caller graph for this function:

4.118.4.6 void ∗ sexpand (int ∗ *prev_len*, MemType *type*, int *len_to_copy*, int *keep_prev*, GlobalLU_t ∗ *Glu*)

Here is the call graph for this function:



Here is the caller graph for this function:

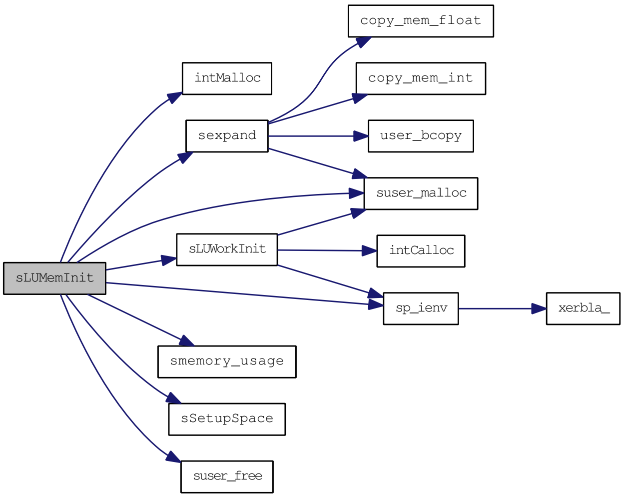


4.118.4.7 int sLUMemInit (fact_t *fact*, void ∗ *work*, int *lwork*, int *m*, int *n*, int *annz*, int *panel_size*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, GlobalLU_t ∗ *Glu*, int ∗∗ *iwork*, float ∗∗ *dwork*)

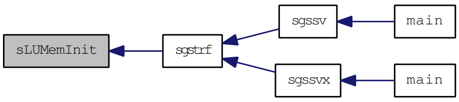
Memory-related.

```
For those unpredictable size, make a guess as FILL * nnz(A).
Return value:
    If lwork = -1, return the estimated amount of space required, plus n;
    otherwise, return the amount of space actually allocated when
    memory allocation failure occurred.
```

Here is the call graph for this function:



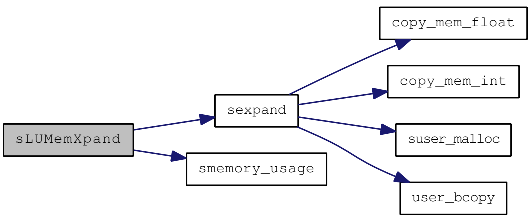
Here is the caller graph for this function:



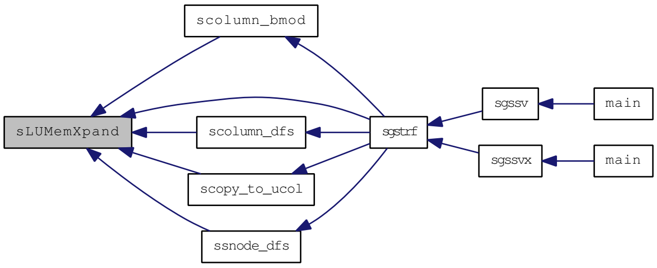
4.118.4.8 int sLUMemXpand (int *jcol*, int *next*, MemType *mem_type*, int ∗ *maxlen*, GlobalLU_t ∗ *Glu*)

```
 Return value:   0 - successful return
               > 0 - number of bytes allocated when run out of space
```

Here is the call graph for this function:

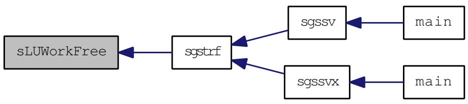


Here is the caller graph for this function:



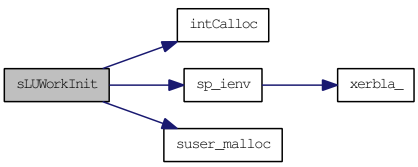
4.118.4.9 void sLUWorkFree (int ∗ *iwork*, float ∗ *dwork*, GlobalLU_t ∗ *Glu*)

Here is the caller graph for this function:

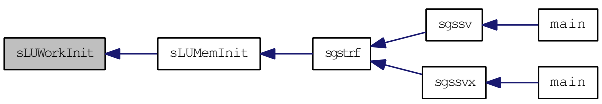


4.118.4.10 int sLUWorkInit (int *m*, int *n*, int *panel_size*, int ∗∗ *iworkptr*, float ∗∗ *dworkptr*, LU_space_t *MemModel*)

Here is the call graph for this function:

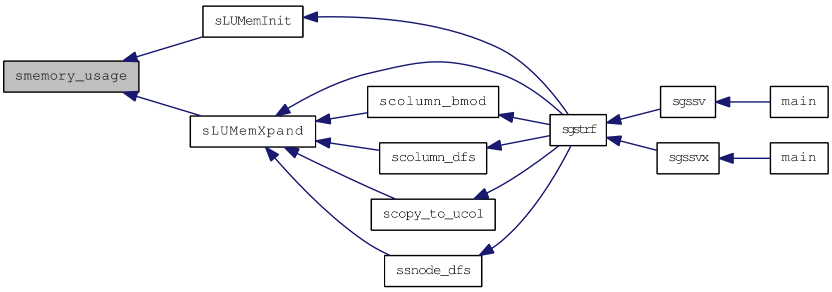


Here is the caller graph for this function:



4.118.4.11 int smemory_usage (const int *nzlmax*, const int *nzumax*, const int *nzlumax*, const int *n*)

Here is the caller graph for this function:

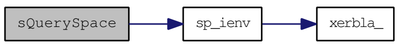


4.118.4.12 int sQuerySpace (SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, mem_usage_t ∗ *mem_usage*)

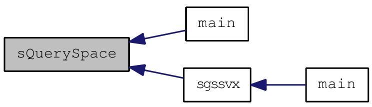
```
mem_usage consists of the following fields:
```

- `for_lu (float)`
  `The amount of space used in bytes for the L data structures.`
- `total_needed (float)`
  `The amount of space needed in bytes to perform factorization.`
- `expansions (int)`
  `Number of memory expansions during the LU factorization.`

Here is the call graph for this function:

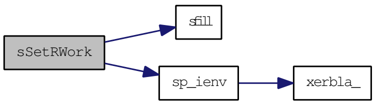


Here is the caller graph for this function:

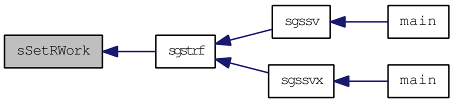


4.118.4.13 void sSetRWork (int *m*, int *panel_size*, float ∗ *dworkptr*, float ∗∗ *dense*, float ∗∗ *tempv*)

Here is the call graph for this function:



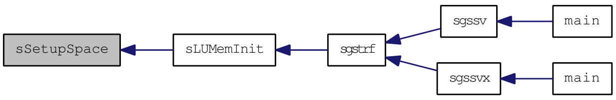
Here is the caller graph for this function:



4.118.4.14 void sSetupSpace (void $*$ *work*, int *lwork*, LU_space_t $*$ *MemModel*)

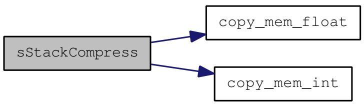
lwork = 0: use system malloc; lwork $>$ 0: use user-supplied work[] space.

Here is the caller graph for this function:



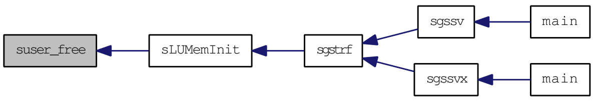
4.118.4.15 void sStackCompress (GlobalLU_t $*$ *Glu*)

Here is the call graph for this function:



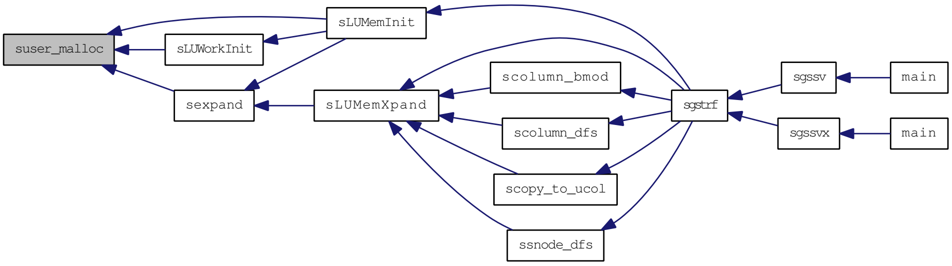
4.118.4.16 void suser_free (int *bytes*, int *which_end*)

Here is the caller graph for this function:



4.118.4.17 void ∗ suser_malloc (int *bytes*, int *which_end*)

Here is the caller graph for this function:



4.118.4.18 void user_bcopy (char ∗, char ∗, int)

4.118.5 Variable Documentation

4.118.5.1 ExpHeader∗ expanders = 0 `[static]`

4.118.5.2 int no_expand `[static]`

4.118.5.3 LU_stack_t stack `[static]`

4.119 SRC/smyblas2.c File Reference

Level 2 Blas operations.

Functions

- void slsolve (int ldm, int ncol, float ∗M, float ∗rhs)

  Solves a dense UNIT lower triangular system.

- void susolve (int ldm, int ncol, float ∗M, float ∗rhs)

  Solves a dense upper triangular system.

- void smatvec (int ldm, int nrow, int ncol, float ∗M, float ∗vec, float ∗Mxvec)

  Performs a dense matrix-vector multiply: Mxvec = Mxvec + M ∗ vec.

4.119.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997
```

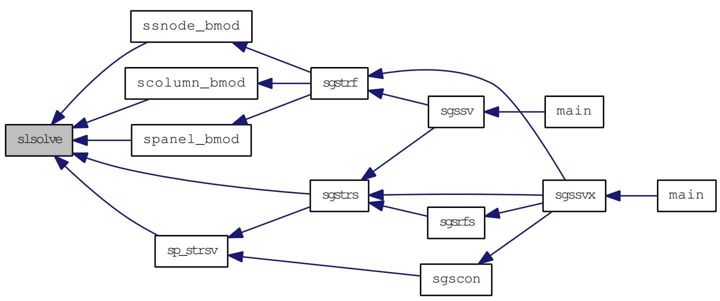
Purpose: Level 2 BLAS operations: solves and matvec, written in C. Note: This is only used when the system lacks an efficient BLAS library.

4.119.2 Function Documentation

4.119.2.1 void slsolve (int *ldm*, int *ncol*, float ∗ *M*, float ∗ *rhs*)

The unit lower triangular matrix is stored in a 2D array M(1:nrow,1:ncol). The solution will be returned in the rhs vector.

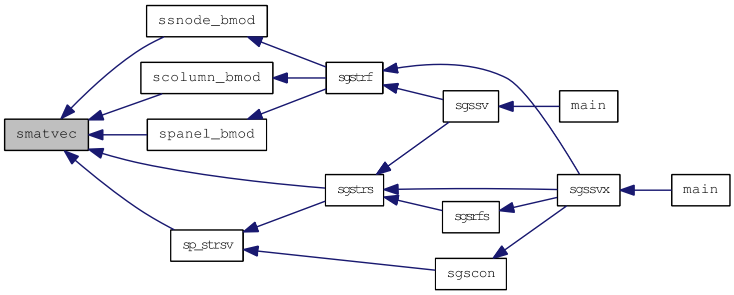
Here is the caller graph for this function:



4.119.2.2 void smatvec (int *ldm*, int *nrow*, int *ncol*, float ∗ *M*, float ∗ *vec*, float ∗ *Mxvec*)

The input matrix is M(1:nrow,1:ncol); The product is returned in Mxvec[].

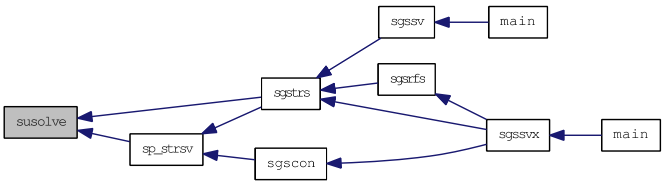
Here is the caller graph for this function:



4.119.2.3 void susolve (int *ldm*, int *ncol*, float ∗ *M*, float ∗ *rhs*)

The upper triangular matrix is stored in a 2-dim array M(1:ldm,1:ncol). The solution will be returned in the rhs vector.

Here is the caller graph for this function:



4.120 SRC/sp_coletree.c File Reference

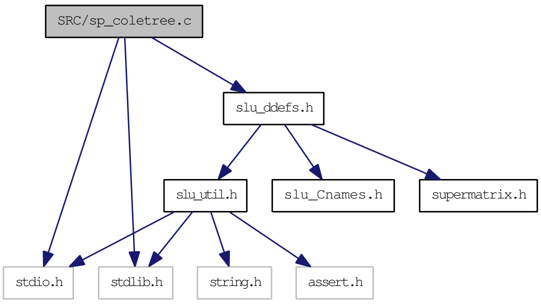
Tree layout and computation routines.

`#include <stdio.h>`

`#include <stdlib.h>`

`#include "slu_ddefs.h"`

Include dependency graph for sp_coletree.c:



Functions

- static int * mxCallocInt (int n)
- static void initialize_disjoint_sets (int n, int **pp)
- static int make_set (int i, int *pp)
- static int link (int s, int t, int *pp)
- static int find (int i, int *pp)
- static void finalize_disjoint_sets (int *pp)
- int sp_coletree (int *acolst, int *acolend, int *arow, int nr, int nc, int *parent)
- static void etdfs (int v, int first_kid[ ], int next_kid[ ], int post[ ], int *postnum)
- static void nr_etdfs (int n, int *parent, int *first_kid, int *next_kid, int *post, int postnum)
- int * TreePostorder (int n, int *parent)
- int sp_symetree (int *acolst, int *acolend, int *arow, int n, int *parent)

4.120.1 Detailed Description

```
-- SuperLU routine (version 3.1) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
August 1, 2008

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

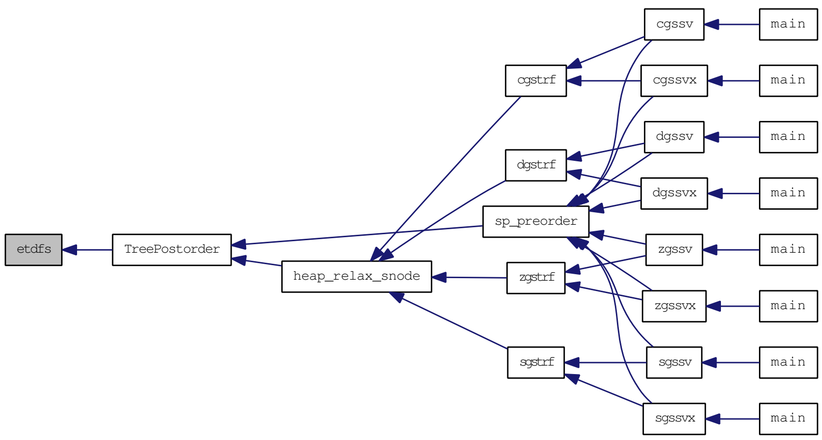
Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
```

```
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.120.2 Function Documentation

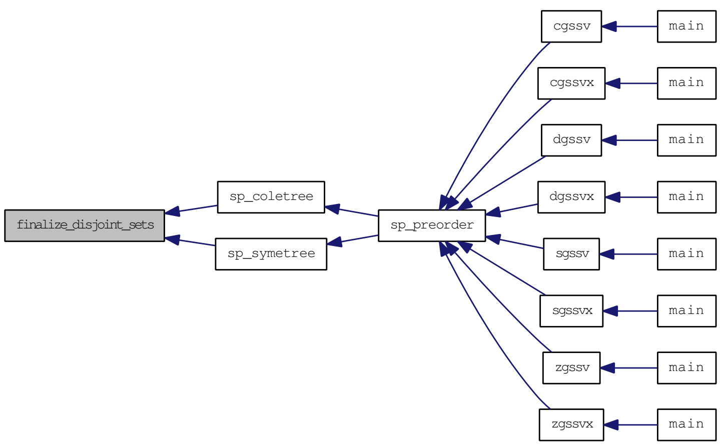
4.120.2.1 static void etdfs (int *v*, int *first_kid*[ ], int *next_kid*[ ], int *post*[ ], int ∗ *postnum*) [static]

Here is the caller graph for this function:



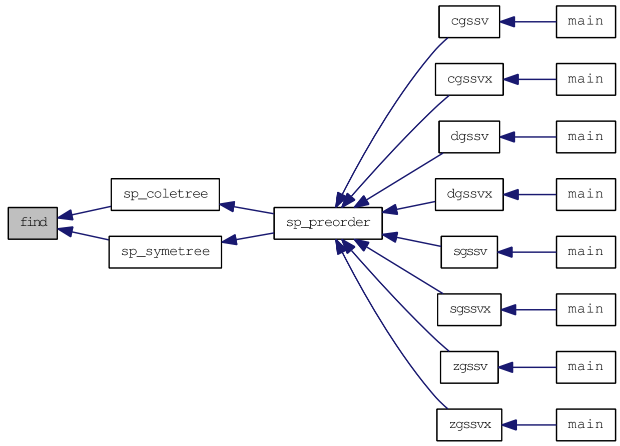
4.120.2.2 static void finalize_disjoint_sets (int ∗ *pp*) [static]

Here is the caller graph for this function:



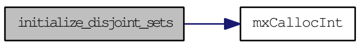
4.120.2.3 static int find (int *i*, int ∗ *pp*) [static]

Here is the caller graph for this function:

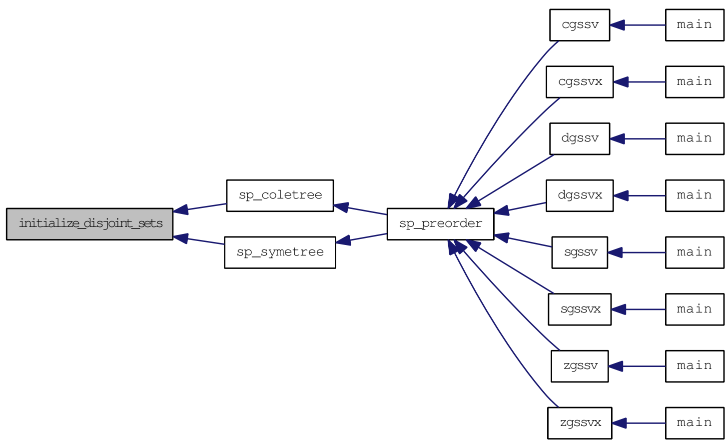


4.120.2.4 static void initialize_disjoint_sets (int *n*, int ∗∗ *pp*) [static]

Here is the call graph for this function:

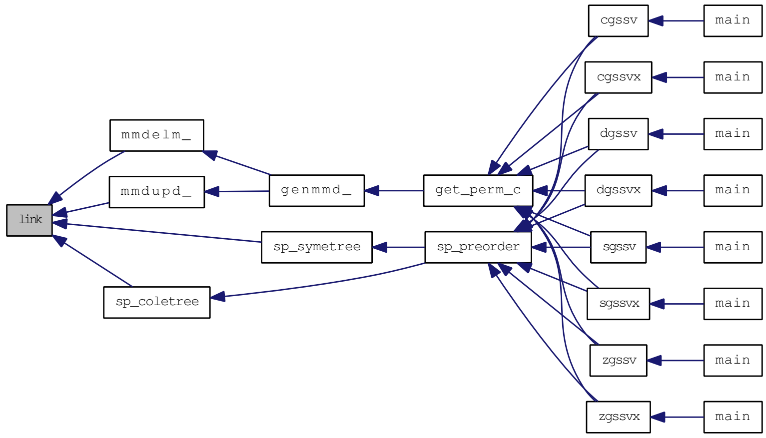


Here is the caller graph for this function:



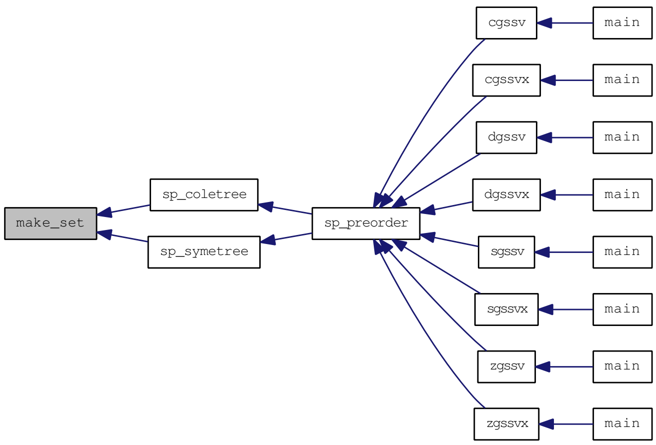
4.120.2.5 static int link (int *s*, int *t*, int ∗ *pp*) [static]

Here is the caller graph for this function:



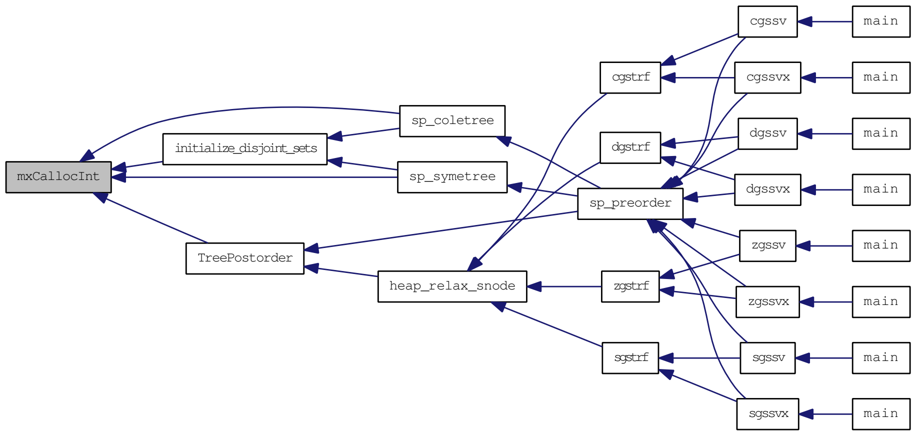
4.120.2.6 static int make_set (int *i*, int ∗ *pp*) [static]

Here is the caller graph for this function:



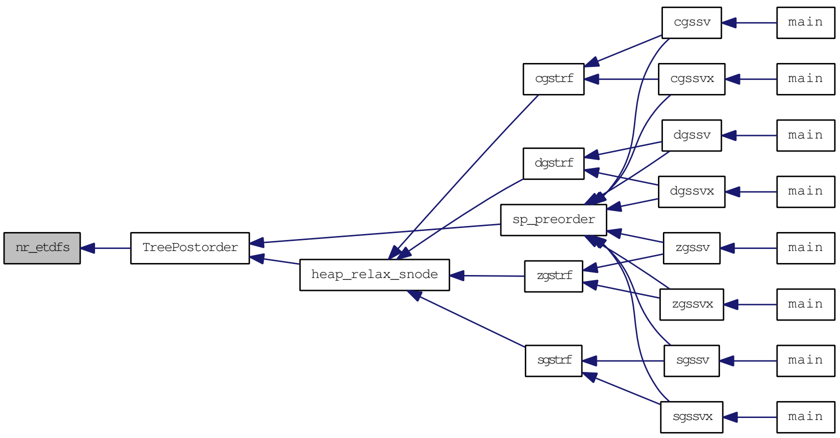
4.120.2.7 static int* mxCallocInt (int *n*) [static]

Here is the caller graph for this function:



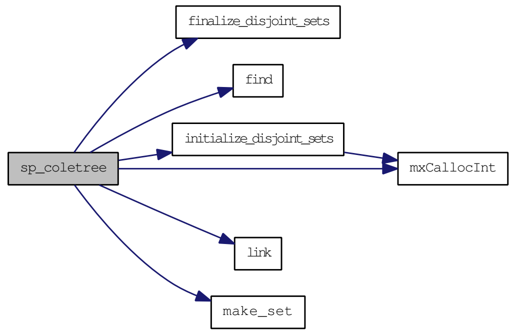
4.120.2.8 static void nr_etdfs (int *n*, int * *parent*, int * *first_kid*, int * *next_kid*, int * *post*, int *postnum*) [static]

Here is the caller graph for this function:

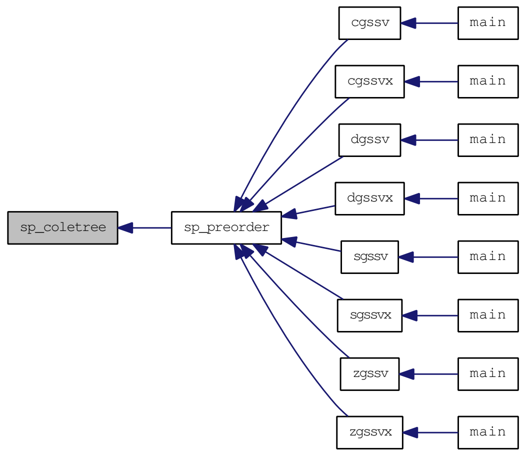


4.120.2.9 int sp_coletree (int ∗ *acolst*, int ∗ *acolend*, int ∗ *arow*, int *nr*, int *nc*, int ∗ *parent*)

Here is the call graph for this function:

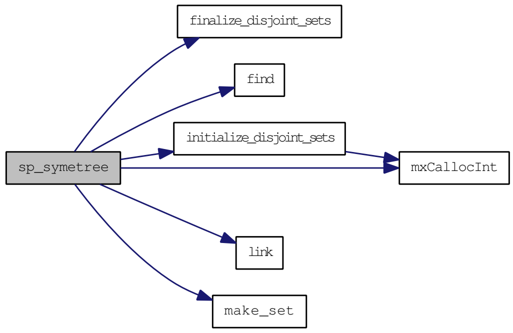


Here is the caller graph for this function:

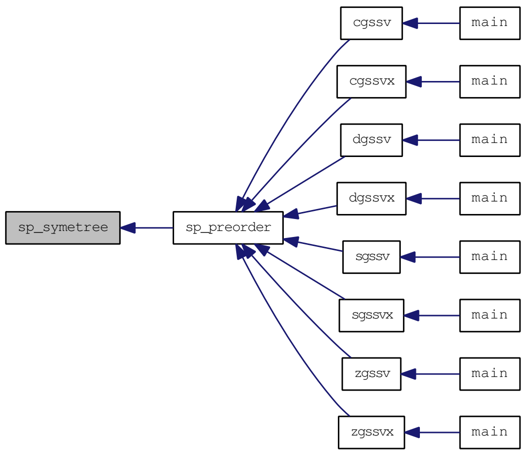


4.120.2.10 int sp_symetree (int ∗ *acolst*, int ∗ *acolend*, int ∗ *arow*, int *n*, int ∗ *parent*)

Here is the call graph for this function:

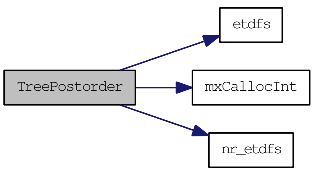


Here is the caller graph for this function:

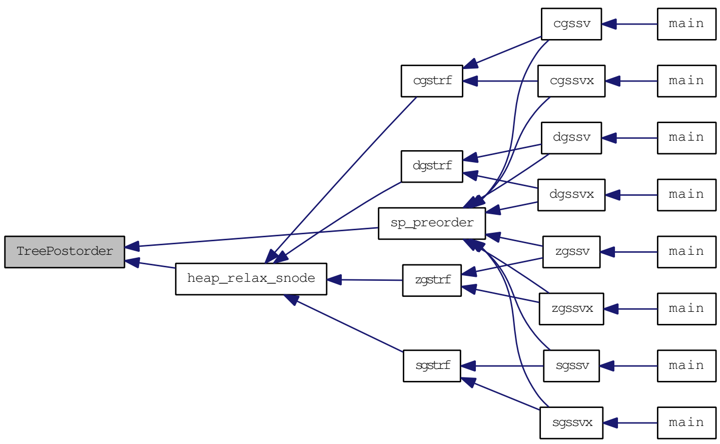


4.120.2.11 int∗ TreePostorder (int *n*, int ∗ *parent*)

Here is the call graph for this function:



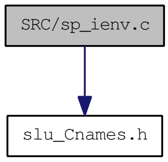
Here is the caller graph for this function:



4.121 SRC/sp_ienv.c File Reference

```
#include "slu_Cnames.h"
```

Include dependency graph for sp_ienv.c:



Functions

- int sp_ienv (int ispec)

4.121.1 Function Documentation

4.121.1.1 int sp_ienv (int *ispec*)

```
Purpose
=======

sp_ienv() is inquired to choose machine-dependent parameters for the
local environment. See ISPEC for a description of the parameters.

This version provides a set of parameters which should give good,
but not optimal, performance on many of the currently available
computers.  Users are encouraged to modify this subroutine to set
the tuning parameters for their particular machine using the option
and problem size information in the arguments.

Arguments
=========

ISPEC   (input) int
        Specifies the parameter to be returned as the value of SP_IENV.
        = 1: the panel size w; a panel consists of w consecutive
     columns of matrix A in the process of Gaussian elimination.
 The best value depends on machine's cache characters.
        = 2: the relaxation parameter relax; if the number of
     nodes (columns) in a subtree of the elimination tree is less
 than relax, this subtree is considered as one supernode,
 regardless of their row structures.
        = 3: the maximum size for a supernode;
    = 4: the minimum row dimension for 2-D blocking to be used;
    = 5: the minimum column dimension for 2-D blocking to be used;
    = 6: the estimated fills factor for L and U, compared with A;

 (SP_IENV) (output) int
        >= 0: the value of the parameter specified by ISPEC
        < 0:  if SP_IENV = -k, the k-th argument had an illegal value.
```

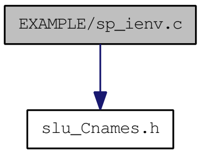
```
=====================================================================
```

Here is the call graph for this function:

4.122 EXAMPLE/sp_ienv.c File Reference

```
#include "slu_Cnames.h"
```

Include dependency graph for sp_ienv.c:



Functions

- int sp_ienv (int ispec)

4.122.1 Function Documentation

4.122.1.1 int sp_ienv (int *ispec*)

```
Purpose
=======

sp_ienv() is inquired to choose machine-dependent parameters for the
local environment. See ISPEC for a description of the parameters.

This version provides a set of parameters which should give good,
but not optimal, performance on many of the currently available
computers.  Users are encouraged to modify this subroutine to set
the tuning parameters for their particular machine using the option
and problem size information in the arguments.

Arguments
=========

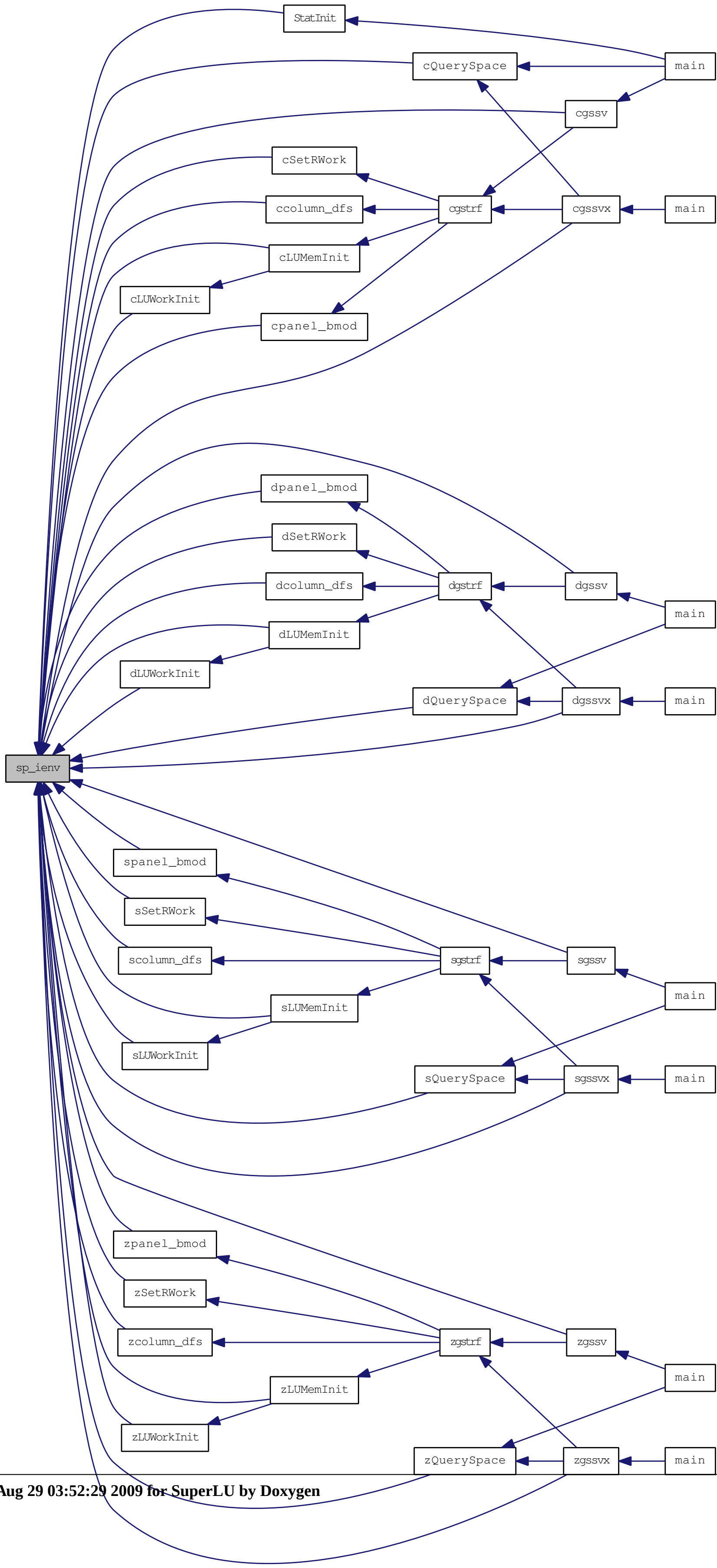
ISPEC   (input) int
        Specifies the parameter to be returned as the value of SP_IENV.
        = 1: the panel size w; a panel consists of w consecutive
     columns of matrix A in the process of Gaussian elimination.
 The best value depends on machine's cache characters.
        = 2: the relaxation parameter relax; if the number of
     nodes (columns) in a subtree of the elimination tree is less
 than relax, this subtree is considered as one supernode,
 regardless of their row structures.
        = 3: the maximum size for a supernode;
= 4: the minimum row dimension for 2-D blocking to be used;
= 5: the minimum column dimension for 2-D blocking to be used;
= 6: the estimated fills factor for L and U, compared with A;

(SP_IENV) (output) int
        >= 0: the value of the parameter specified by ISPEC
        < 0:  if SP_IENV = -k, the k-th argument had an illegal value.
```

```
=====================================================================
```

Here is the call graph for this function:

Here is the caller graph for this function:

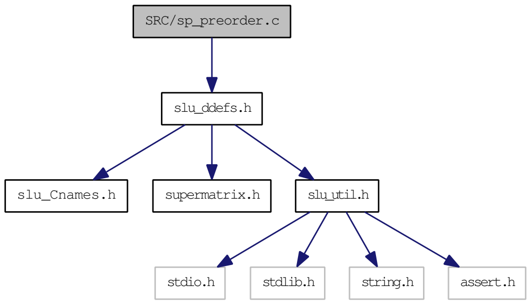


4.123 SRC/sp_preorder.c File Reference

Permute and performs functions on columns of orginal matrix.

`#include "slu_ddefs.h"`

Include dependency graph for sp_preorder.c:



Functions

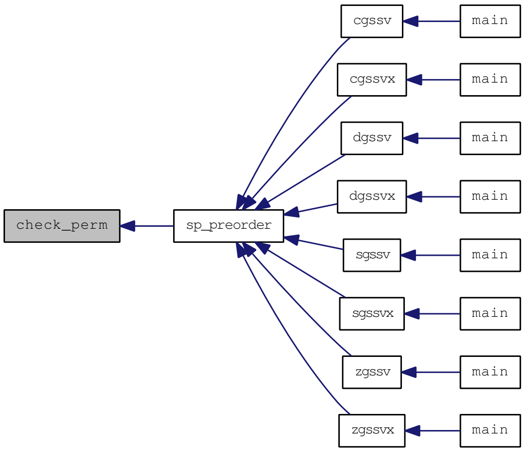
- void sp_preorder (superlu_options_t ∗options, SuperMatrix ∗A, int ∗perm_c, int ∗etree, SuperMatrix ∗AC)
- int check_perm (char ∗what, int n, int ∗perm)

4.123.1 Detailed Description

4.123.2 Function Documentation

4.123.2.1 int check_perm (char ∗ *what*, int *n*, int ∗ *perm*)

Here is the caller graph for this function:



4.123.2.2 void sp_preorder (superlu_options_t * *options*, SuperMatrix * *A*, int * *perm_c*, int * *etree*, SuperMatrix * *AC*)

```
Purpose
=======

sp_preorder() permutes the columns of the original matrix. It performs
the following steps:

   1. Apply column permutation perm_c[] to A's column pointers to form AC;

   2. If options->Fact = DOFACT, then
      (1) Compute column elimination tree etree[] of AC'AC;
      (2) Post order etree[] to get a postordered elimination tree etree[],
          and a postorder permutation post[];
      (3) Apply post[] permutation to columns of AC;
      (4) Overwrite perm_c[] with the product perm_c * post.

Arguments
=========

options (input) superlu_options_t*
        Specifies whether or not the elimination tree will be re-used.
        If options->Fact == DOFACT, this means first time factor A,
        etree is computed, postered, and output.
        Otherwise, re-factor A, etree is input, unchanged on exit.

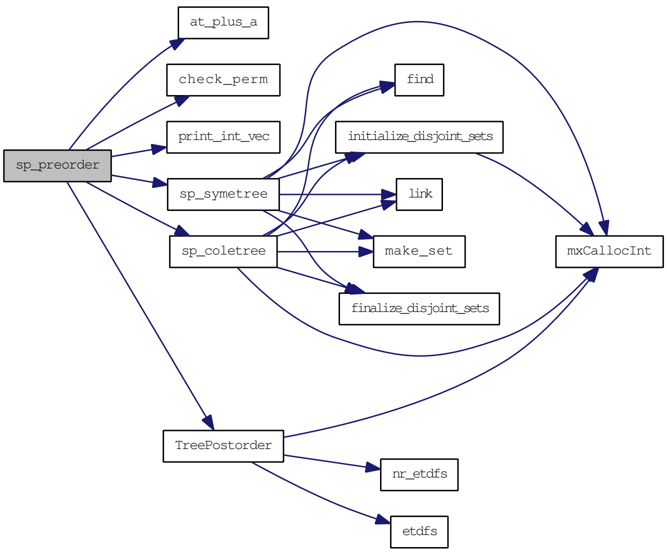
A       (input) SuperMatrix*
        Matrix A in A*X=B, of dimension (A->nrow, A->ncol). The number
        of the linear equations is A->nrow. Currently, the type of A can be:
        Stype = NC or SLU_NCP; Mtype = SLU_GE.
        In the future, more general A may be handled.

perm_c  (input/output) int*
  Column permutation vector of size A->ncol, which defines the
        permutation matrix Pc; perm_c[i] = j means column i of A is
        in position j in A*Pc.
        If options->Fact == DOFACT, perm_c is both input and output.
        On output, it is changed according to a postorder of etree.
        Otherwise, perm_c is input.

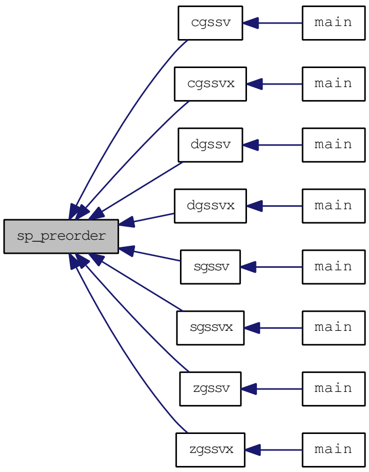
etree   (input/output) int*
        Elimination tree of Pc'*A'*A*Pc, dimension A->ncol.
        If options->Fact == DOFACT, etree is an output argument,
        otherwise it is an input argument.
        Note: etree is a vector of parent pointers for a forest whose
        vertices are the integers 0 to A->ncol-1; etree[root]==A->ncol.

AC      (output) SuperMatrix*
        The resulting matrix after applied the column permutation
        perm_c[] to matrix A. The type of AC can be:
        Stype = SLU_NCP; Dtype = A->Dtype; Mtype = SLU_GE.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.124 SRC/spanel_bmod.c File Reference

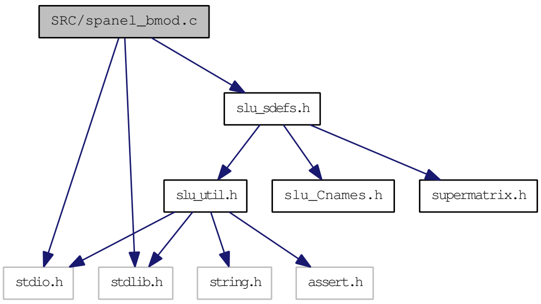
Performs numeric block updates.

`#include <stdio.h>`

`#include <stdlib.h>`

`#include "slu_sdefs.h"`

Include dependency graph for spanel_bmod.c:



Functions

- void slsolve (int, int, float ∗, float ∗)

  Solves a dense UNIT lower triangular system.

- void smatvec (int, int, int, float ∗, float ∗, float ∗)

  Performs a dense matrix-vector multiply: Mxvec = Mxvec + M ∗ vec.

- void scheck_tempv ()
- void spanel_bmod (const int m, const int w, const int jcol, const int nseg, float ∗dense, float ∗tempv, int ∗segrep, int ∗repfnz, GlobalLU_t ∗Glu, SuperLUStat_t ∗stat)

4.124.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.124.2 Function Documentation

4.124.2.1 void scheck_tempv ()

4.124.2.2 void slsolve (int *ldm*, int *ncol*, float ∗ *M*, float ∗ *rhs*)

The unit lower triangular matrix is stored in a 2D array M(1:nrow,1:ncol). The solution will be returned in the rhs vector.

4.124.2.3 void smatvec (int *ldm*, int *nrow*, int *ncol*, float ∗ *M*, float ∗ *vec*, float ∗ *Mxvec*)

The input matrix is M(1:nrow,1:ncol); The product is returned in Mxvec[].

4.124.2.4 void spanel_bmod (const int *m*, const int *w*, const int *jcol*, const int *nseg*, float ∗ *dense*, float ∗ *tempv*, int ∗ *segrep*, int ∗ *repfnz*, GlobalLU_t ∗ *Glu*, SuperLUStat_t ∗ *stat*)

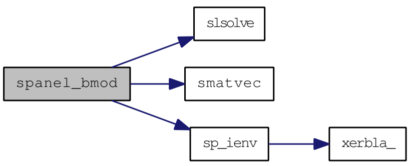
```
Purpose
=======

   Performs numeric block updates (sup-panel) in topological order.
   It features: col-col, 2cols-col, 3cols-col, and sup-col updates.
   Special processing on the supernodal portion of L[*,j]

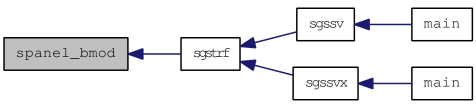
   Before entering this routine, the original nonzeros in the panel
   were already copied into the spa[m,w].

   Updated/Output parameters-
   dense[0:m-1,w]: L[*,j:j+w-1] and U[*,j:j+w-1] are returned
   collectively in the m-by-w vector dense[*].
```

Here is the call graph for this function:



Here is the caller graph for this function:

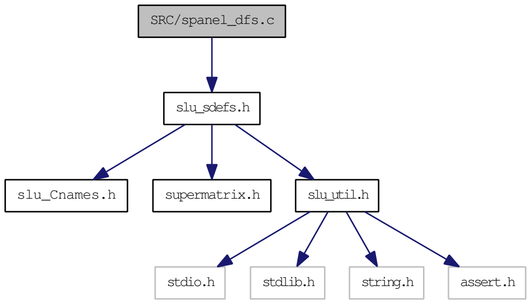


4.125 SRC/spanel_dfs.c File Reference

Peforms a symbolic factorization on a panel of symbols.

`#include "slu_sdefs.h"`

Include dependency graph for spanel_dfs.c:



Functions

- void spanel_dfs (const int m, const int w, const int jcol, SuperMatrix ∗A, int ∗perm_r, int ∗nseg, float ∗dense, int ∗panel_lsub, int ∗segrep, int ∗repfnz, int ∗xprune, int ∗marker, int ∗parent, int ∗xplore, GlobalLU_t ∗Glu)

4.125.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.125.2 Function Documentation

4.125.2.1 void spanel_dfs (const int *m*, const int *w*, const int *jcol*, SuperMatrix ∗ *A*, int ∗ *perm_r*, int ∗ *nseg*, float ∗ *dense*, int ∗ *panel_lsub*, int ∗ *segrep*, int ∗ *repfnz*, int ∗ *xprune*, int ∗ *marker*, int ∗ *parent*, int ∗ *xplore*, GlobalLU_t ∗ *Glu*)

```
Purpose
=======
```

```
Performs a symbolic factorization on a panel of columns [jcol, jcol+w).

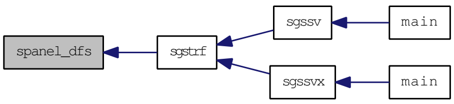
A supernode representative is the last column of a supernode.
The nonzeros in U[*,j] are segments that end at supernodal
representatives.

The routine returns one list of the supernodal representatives
in topological order of the dfs that generates them. This list is
a superset of the topological order of each individual column within
the panel.
The location of the first nonzero in each supernodal segment
(supernodal entry location) is also returned. Each column has a
separate list for this purpose.

Two marker arrays are used for dfs:
  marker[i] == jj, if i was visited during dfs of current column jj;
  marker1[i] >= jcol, if i was visited by earlier columns in this panel;

marker: A-row --> A-row/col (0/1)
repfnz: SuperA-col --> PA-row
parent: SuperA-col --> SuperA-col
xplore: SuperA-col --> index to L-structure
```

Here is the caller graph for this function:



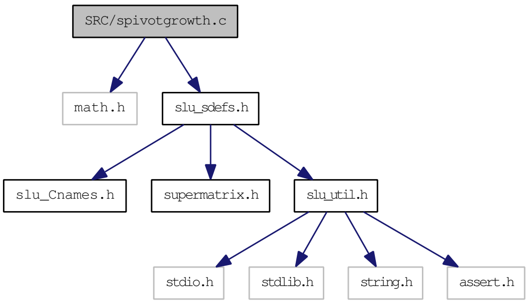
4.126 SRC/spivotgrowth.c File Reference

Computes the reciprocal pivot growth factor.

`#include <math.h>`

`#include "slu_sdefs.h"`

Include dependency graph for spivotgrowth.c:



Functions

- float sPivotGrowth (int ncols, SuperMatrix *A, int *perm_c, SuperMatrix *L, SuperMatrix *U)

4.126.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997
```

4.126.2 Function Documentation

4.126.2.1 float sPivotGrowth (int *ncols*, SuperMatrix * *A*, int * *perm_c*, SuperMatrix * *L*, SuperMatrix * *U*)

```
Purpose
=======

Compute the reciprocal pivot growth factor of the leading ncols columns
of the matrix, using the formula:
    min_j ( max_i(abs(A_ij)) / max_i(abs(U_ij)) )

Arguments
=========

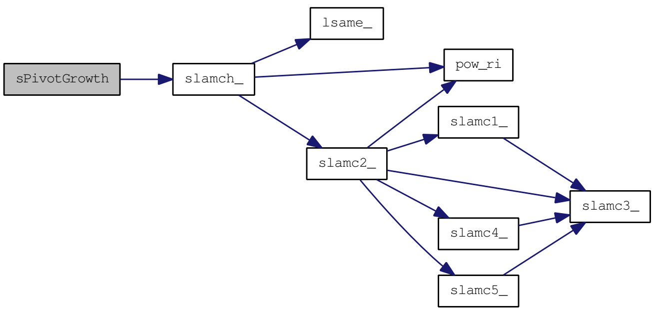
ncols    (input) int
         The number of columns of matrices A, L and U.
```

```
A       (input) SuperMatrix*
    Original matrix A, permuted by columns, of dimension
        (A->nrow, A->ncol). The type of A can be:
        Stype = NC; Dtype = SLU_S; Mtype = GE.

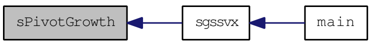
L       (output) SuperMatrix*
        The factor L from the factorization Pr*A=L*U; use compressed row
        subscripts storage for supernodes, i.e., L has type:
        Stype = SC; Dtype = SLU_S; Mtype = TRLU.

U       (output) SuperMatrix*
    The factor U from the factorization Pr*A*Pc=L*U. Use column-wise
        storage scheme, i.e., U has types: Stype = NC;
        Dtype = SLU_S; Mtype = TRU.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.127 SRC/spivotL.c File Reference

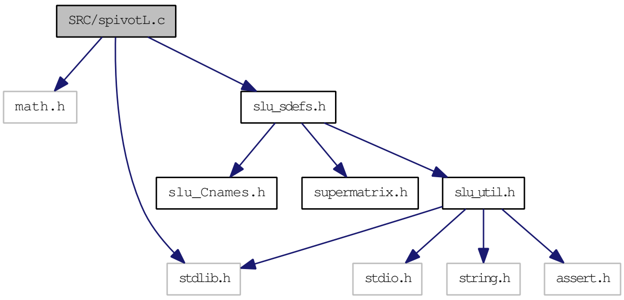
Performs numerical pivoting.

`#include <math.h>`

`#include <stdlib.h>`

`#include "slu_sdefs.h"`

Include dependency graph for spivotL.c:



Functions

- int spivotL (const int jcol, const float u, int ∗usepr, int ∗perm_r, int ∗iperm_r, int ∗iperm_c, int ∗pivrow, GlobalLU_t ∗Glu, SuperLUStat_t ∗stat)

4.127.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.127.2 Function Documentation

4.127.2.1 int spivotL (const int *jcol*, const float *u*, int ∗ *usepr*, int ∗ *perm_r*, int ∗ *iperm_r*, int ∗ *iperm_c*, int ∗ *pivrow*, GlobalLU_t ∗ *Glu*, SuperLUStat_t ∗ *stat*)

```
Purpose
```

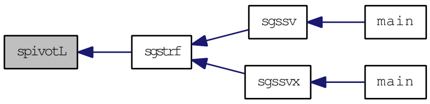
```
=======
  Performs the numerical pivoting on the current column of L,
  and the CDIV operation.

  Pivot policy:
  (1) Compute thresh = u * max_(i>=j) abs(A_ij);
  (2) IF user specifies pivot row k and abs(A_kj) >= thresh THEN
          pivot row = k;
      ELSE IF abs(A_jj) >= thresh THEN
          pivot row = j;
      ELSE
          pivot row = m;

  Note: If you absolutely want to use a given pivot order, then set u=0.0.

  Return value: 0       success;
                i > 0   U(i,i) is exactly zero.
```

Here is the caller graph for this function:

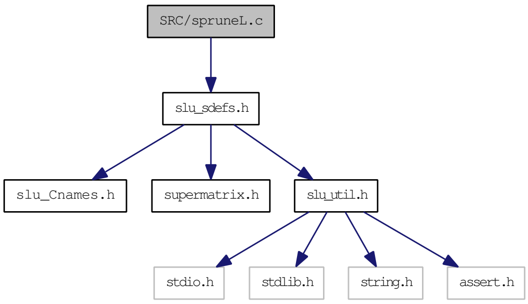


4.128 SRC/spruneL.c File Reference

Prunes the L-structure.

`#include "slu_sdefs.h"`

Include dependency graph for spruneL.c:



Functions

- void spruneL (const int jcol, const int ∗perm_r, const int pivrow, const int nseg, const int ∗segrep, const int ∗repfnz, int ∗xprune, GlobalLU_t ∗Glu)

4.128.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
*
```

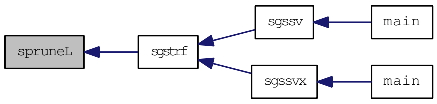
4.128.2 Function Documentation

4.128.2.1 void spruneL (const int *jcol*, const int ∗ *perm_r*, const int *pivrow*, const int *nseg*, const int ∗ *segrep*, const int ∗ *repfnz*, int ∗ *xprune*, GlobalLU_t ∗ *Glu*)

```
Purpose
=======
  Prunes the L-structure of supernodes whose L-structure
```

```
contains the current pivot row "pivrow"
```

Here is the caller graph for this function:



4.129 SRC/sreadhb.c File Reference

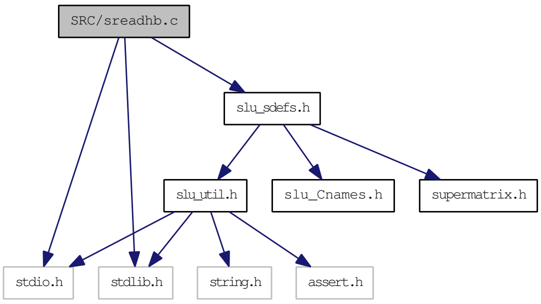
Read a matrix stored in Harwell-Boeing format.

`#include <stdio.h>`

`#include <stdlib.h>`

`#include "slu_sdefs.h"`

Include dependency graph for sreadhb.c:



Functions

- int sDumpLine (FILE ∗fp)

  Eat up the rest of the current line.

- int sParseIntFormat (char ∗buf, int ∗num, int ∗size)
- int sParseFloatFormat (char ∗buf, int ∗num, int ∗size)
- int sReadVector (FILE ∗fp, int n, int ∗where, int perline, int persize)
- int sReadValues (FILE ∗fp, int n, float ∗destination, int perline, int persize)
- void sreadhb (int ∗nrow, int ∗ncol, int ∗nonz, float ∗∗nzval, int ∗∗rowind, int ∗∗colptr)

  Auxiliary routines.

4.129.1 Detailed Description

```
 -- SuperLU routine (version 2.0) --
 Univ. of California Berkeley, Xerox Palo Alto Research Center,
 and Lawrence Berkeley National Lab.
 November 15, 1997

 Purpose
 =======

 Read a FLOAT PRECISION matrix stored in Harwell-Boeing format
 as described below.

 Line 1 (A72,A8)
   Col. 1 - 72   Title (TITLE)
Col. 73 - 80  Key (KEY)
```

```
 Line 2 (5I14)
  Col. 1 - 14   Total number of lines excluding header (TOTCRD)
  Col. 15 - 28  Number of lines for pointers (PTRCRD)
  Col. 29 - 42  Number of lines for row (or variable) indices (INDCRD)
  Col. 43 - 56  Number of lines for numerical values (VALCRD)
Col. 57 - 70  Number of lines for right-hand sides (RHSCRD)
                    (including starting guesses and solution vectors
       if present)
                  (zero indicates no right-hand side data is present)

 Line 3 (A3, 11X, 4I14)
    Col. 1 - 3    Matrix type (see below) (MXTYPE)
  Col. 15 - 28  Number of rows (or variables) (NROW)
  Col. 29 - 42  Number of columns (or elements) (NCOL)
Col. 43 - 56  Number of row (or variable) indices (NNZERO)
              (equal to number of entries for assembled matrices)
  Col. 57 - 70  Number of elemental matrix entries (NELTVL)
              (zero in the case of assembled matrices)
 Line 4 (2A16, 2A20)
  Col. 1 - 16   Format for pointers (PTRFMT)
Col. 17 - 32  Format for row (or variable) indices (INDFMT)
Col. 33 - 52  Format for numerical values of coefficient matrix (VALFMT)
  Col. 53 - 72 Format for numerical values of right-hand sides (RHSFMT)

 Line 5 (A3, 11X, 2I14) Only present if there are right-hand sides present
     Col. 1        Right-hand side type:
            F for full storage or M for same format as matrix
     Col. 2        G if a starting vector(s) (Guess) is supplied. (RHSTYP)
     Col. 3        X if an exact solution vector(s) is supplied.
Col. 15 - 28  Number of right-hand sides (NRHS)
Col. 29 - 42  Number of row indices (NRHSIX)
                 (ignored in case of unassembled matrices)

 The three character type field on line 3 describes the matrix type.
 The following table lists the permitted values for each of the three
 characters. As an example of the type field, RSA denotes that the matrix
 is real, symmetric, and assembled.

 First Character:
R Real matrix
C Complex matrix
P Pattern only (no numerical values supplied)

 Second Character:
S Symmetric
U Unsymmetric
H Hermitian
Z Skew symmetric
R Rectangular

 Third Character:
A Assembled
E Elemental matrices (unassembled)
```

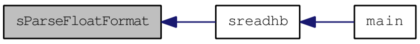
4.129.2 Function Documentation

4.129.2.1 int sDumpLine (FILE * *fp*)

Here is the caller graph for this function:

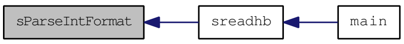
4.129.2.2 int sParseFloatFormat (char * *buf*, int * *num*, int * *size*)

Here is the caller graph for this function:



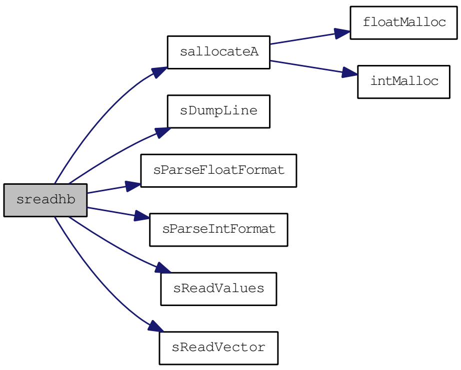
4.129.2.3 int sParseIntFormat (char * *buf*, int * *num*, int * *size*)

Here is the caller graph for this function:

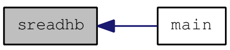


4.129.2.4 void sreadhb (int * *nrow*, int * *ncol*, int * *nonz*, float ** *nzval*, int ** *rowind*, int ** *colptr*)

Here is the call graph for this function:

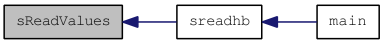


Here is the caller graph for this function:



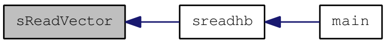
4.129.2.5 int sReadValues (FILE * *fp*, int *n*, float * *destination*, int *perline*, int *persize*)

Here is the caller graph for this function:



4.129.2.6 int sReadVector (FILE * *fp*, int *n*, int * *where*, int *perline*, int *persize*)

Here is the caller graph for this function:

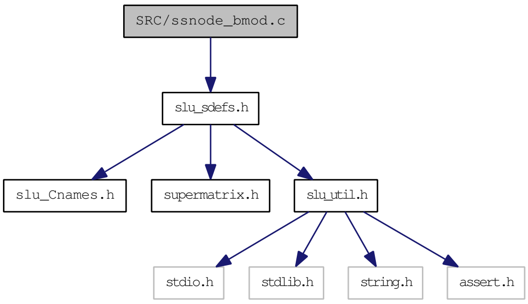


4.130 SRC/ssnode_bmod.c File Reference

Performs numeric block updates within the relaxed snode.

`#include "slu_sdefs.h"`

Include dependency graph for ssnode_bmod.c:



Functions

- int ssnode_bmod (const int jcol, const int jsupno, const int fsupc, float ∗dense, float ∗tempv, GlobalLU_t ∗Glu, SuperLUStat_t ∗stat)

  Performs numeric block updates within the relaxed snode.

4.130.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003


Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

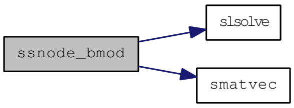

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.


Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

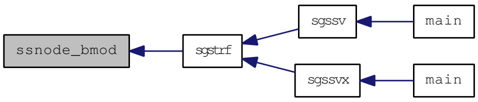
4.130.2 Function Documentation

4.130.2.1 int ssnode_bmod (const int *jcol*, const int *jsupno*, const int *fsupc*, float ∗ *dense*, float ∗ *tempv*, GlobalLU_t ∗ *Glu*, SuperLUStat_t ∗ *stat*)

Here is the call graph for this function:



Here is the caller graph for this function:

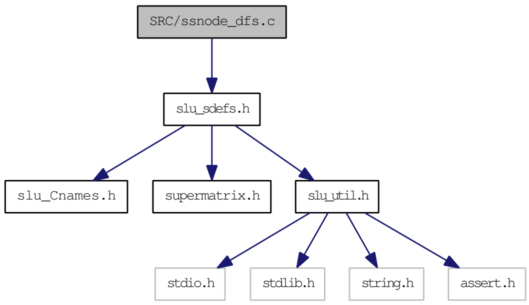


4.131 SRC/ssnode_dfs.c File Reference

Determines the union of row structures of columns within the relaxed node.

`#include "slu_sdefs.h"`

Include dependency graph for ssnode_dfs.c:



Functions

- int ssnode_dfs (const int jcol, const int kcol, const int ∗asub, const int ∗xa_begin, const int ∗xa_end, int ∗xprune, int ∗marker, GlobalLU_t ∗Glu)

4.131.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997


Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.131.2 Function Documentation

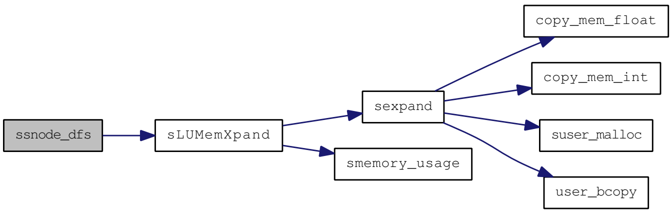
4.131.2.1 int ssnode_dfs (const int *jcol*, const int *kcol*, const int ∗ *asub*, const int ∗ *xa_begin*, const int ∗ *xa_end*, int ∗ *xprune*, int ∗ *marker*, GlobalLU_t ∗ *Glu*)

```
Purpose
=======
   ssnode_dfs() - Determine the union of the row structures of those
```

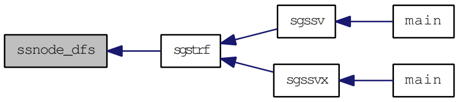
```
    columns within the relaxed snode.
    Note: The relaxed snodes are leaves of the supernodal etree, therefore,
    the portion outside the rectangular supernode must be zero.

 Return value
 ============
     0   success;
    >0   number of bytes allocated when run out of memory.
```

Here is the call graph for this function:



Here is the caller graph for this function:

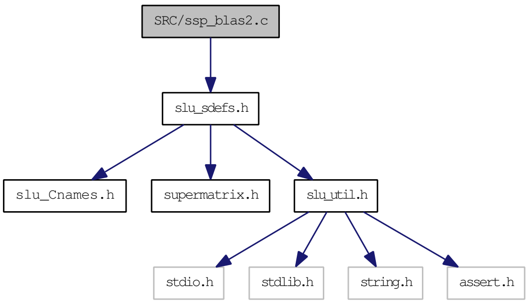


4.132 SRC/ssp_blas2.c File Reference

Sparse BLAS 2, using some dense BLAS 2 operations.

`#include "slu_sdefs.h"`

Include dependency graph for ssp_blas2.c:



Functions

- void susolve (int, int, float ∗, float ∗)

  Solves a dense upper triangular system.

- void slsolve (int, int, float ∗, float ∗)

  Solves a dense UNIT lower triangular system.

- void smatvec (int, int, int, float ∗, float ∗, float ∗)

  Performs a dense matrix-vector multiply: Mxvec = Mxvec + M ∗ vec.

- int sp_strsv (char ∗uplo, char ∗trans, char ∗diag, SuperMatrix ∗L, SuperMatrix ∗U, float ∗x, SuperLUStat_t ∗stat, int ∗info)

  Solves one of the systems of equations A∗x = b, or A'∗x = b.

- int sp_sgemv (char ∗trans, float alpha, SuperMatrix ∗A, float ∗x, int incx, float beta, float ∗y, int incy)

  Performs one of the matrix-vector operations y := alpha∗A∗x + beta∗y, or y := alpha∗A'∗x + beta∗y,.

4.132.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003
```

4.132.2 Function Documentation

4.132.2.1 void slsolve (int *ldm*, int *ncol*, float ∗ *M*, float ∗ *rhs*)

The unit lower triangular matrix is stored in a 2D array M(1:nrow,1:ncol). The solution will be returned in the rhs vector.

4.132.2.2 void smatvec (int *ldm*, int *nrow*, int *ncol*, float ∗ *M*, float ∗ *vec*, float ∗ *Mxvec*)

The input matrix is M(1:nrow,1:ncol); The product is returned in Mxvec[].

4.132.2.3 int sp_sgemv (char ∗ *trans*, float *alpha*, SuperMatrix ∗ *A*, float ∗ *x*, int *incx*, float *beta*, float ∗ *y*, int *incy*)

```
Purpose
=======

sp_sgemv()  performs one of the matrix-vector operations
   y := alpha*A*x + beta*y,   or   y := alpha*A'*x + beta*y,
where alpha and beta are scalars, x and y are vectors and A is a
sparse A->nrow by A->ncol matrix.


Parameters
==========


TRANS   - (input) char*
          On entry, TRANS specifies the operation to be performed as
          follows:
             TRANS = 'N' or 'n'   y := alpha*A*x + beta*y.
             TRANS = 'T' or 't'   y := alpha*A'*x + beta*y.
             TRANS = 'C' or 'c'   y := alpha*A'*x + beta*y.


ALPHA   - (input) float
          On entry, ALPHA specifies the scalar alpha.


A       - (input) SuperMatrix*
          Matrix A with a sparse format, of dimension (A->nrow, A->ncol).
          Currently, the type of A can be:
              Stype = NC or NCP; Dtype = SLU_S; Mtype = GE.
          In the future, more general A can be handled.


X       - (input) float*, array of DIMENSION at least
          ( 1 + ( n - 1 )*abs( INCX ) ) when TRANS = 'N' or 'n'
          and at least
          ( 1 + ( m - 1 )*abs( INCX ) ) otherwise.
          Before entry, the incremented array X must contain the
          vector x.


INCX    - (input) int
          On entry, INCX specifies the increment for the elements of
          X. INCX must not be zero.
```

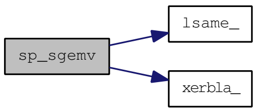
```
BETA    - (input) float
          On entry, BETA specifies the scalar beta. When BETA is
          supplied as zero then Y need not be set on input.

Y       - (output) float*,  array of DIMENSION at least
          ( 1 + ( m - 1 )*abs( INCY ) ) when TRANS = 'N' or 'n'
          and at least
          ( 1 + ( n - 1 )*abs( INCY ) ) otherwise.
          Before entry with BETA non-zero, the incremented array Y
          must contain the vector y. On exit, Y is overwritten by the
          updated vector y.

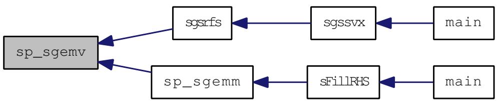
INCY    - (input) int
          On entry, INCY specifies the increment for the elements of
          Y. INCY must not be zero.

==== Sparse Level 2 Blas routine.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.132.2.4 int sp_strsv (char ∗ *uplo*, char ∗ *trans*, char ∗ *diag*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, float ∗ *x*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======

sp_strsv() solves one of the systems of equations
    A*x = b,    or    A'*x = b,
where b and x are n element vectors and A is a sparse unit , or
non-unit, upper or lower triangular matrix.
No test for singularity or near-singularity is included in this
routine. Such tests must be performed before calling this routine.

Parameters
==========

uplo    - (input) char*
          On entry, uplo specifies whether the matrix is an upper or
           lower triangular matrix as follows:
              uplo = 'U' or 'u'   A is an upper triangular matrix.
              uplo = 'L' or 'l'   A is a lower triangular matrix.
```

```
trans  - (input) char*
          On entry, trans specifies the equations to be solved as
          follows:
             trans = 'N' or 'n'    A*x = b.
             trans = 'T' or 't'    A'*x = b.
             trans = 'C' or 'c'    A'*x = b.



diag   - (input) char*
          On entry, diag specifies whether or not A is unit
          triangular as follows:
             diag = 'U' or 'u'   A is assumed to be unit triangular.
             diag = 'N' or 'n'   A is not assumed to be unit
                                 triangular.



L       - (input) SuperMatrix*
    The factor L from the factorization Pr*A*Pc=L*U. Use
          compressed row subscripts storage for supernodes,
          i.e., L has types: Stype = SC, Dtype = SLU_S, Mtype = TRLU.



U       - (input) SuperMatrix*
     The factor U from the factorization Pr*A*Pc=L*U.
     U has types: Stype = NC, Dtype = SLU_S, Mtype = TRU.

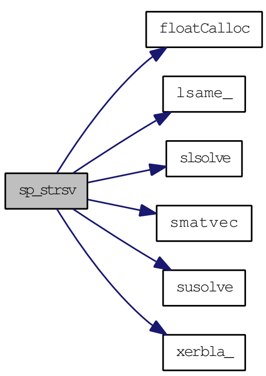


x       - (input/output) float*
          Before entry, the incremented array X must contain the n
          element right-hand side vector b. On exit, X is overwritten
          with the solution vector x.

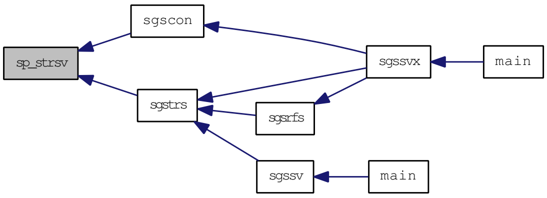


info    - (output) int*
          If *info = -i, the i-th argument had an illegal value.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.132.2.5 void susolve (int *ldm*, int *ncol*, float ∗ *M*, float ∗ *rhs*)

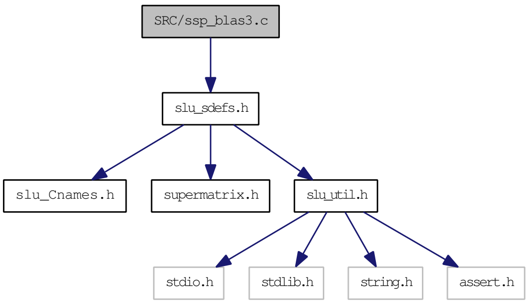
The upper triangular matrix is stored in a 2-dim array M(1:ldm,1:ncol). The solution will be returned in the rhs vector.

4.133 SRC/ssp_blas3.c File Reference

Sparse BLAS3, using some dense BLAS3 operations.

`#include "slu_sdefs.h"`

Include dependency graph for ssp_blas3.c:



Functions

- int sp_sgemm (char *transa, char *transb, int m, int n, int k, float alpha, SuperMatrix *A, float *b, int ldb, float beta, float *c, int ldc)

4.133.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997
```

4.133.2 Function Documentation

4.133.2.1 int sp_sgemm (char * *transa*, char * *transb*, int *m*, int *n*, int *k*, float *alpha*, SuperMatrix * *A*, float * *b*, int *ldb*, float *beta*, float * *c*, int *ldc*)

```
Purpose
  =======

  sp_s performs one of the matrix-matrix operations

     C := alpha*op( A )*op( B ) + beta*C,

  where  op( X ) is one of

     op( X ) = X   or   op( X ) = X'   or   op( X ) = conjg( X' ),

  alpha and beta are scalars, and A, B and C are matrices, with op( A )
  an m by k matrix,  op( B )  a  k by n matrix and  C an m by n matrix.
```

```
Parameters
==========

TRANSA - (input) char*
         On entry, TRANSA specifies the form of op( A ) to be used in
         the matrix multiplication as follows:
            TRANSA = 'N' or 'n',  op( A ) = A.
            TRANSA = 'T' or 't',  op( A ) = A'.
            TRANSA = 'C' or 'c',  op( A ) = conjg( A' ).
         Unchanged on exit.

TRANSB - (input) char*
         On entry, TRANSB specifies the form of op( B ) to be used in
         the matrix multiplication as follows:
            TRANSB = 'N' or 'n',  op( B ) = B.
            TRANSB = 'T' or 't',  op( B ) = B'.
            TRANSB = 'C' or 'c',  op( B ) = conjg( B' ).
         Unchanged on exit.

M      - (input) int
         On entry,  M  specifies  the number of rows of the matrix
  op( A ) and of the matrix C.  M must be at least zero.
  Unchanged on exit.

N      - (input) int
         On entry,  N specifies the number of columns of the matrix
  op( B ) and the number of columns of the matrix C. N must be
  at least zero.
  Unchanged on exit.

K      - (input) int
         On entry, K specifies the number of columns of the matrix
  op( A ) and the number of rows of the matrix op( B ). K must
  be at least  zero.
        Unchanged on exit.

ALPHA  - (input) float
         On entry, ALPHA specifies the scalar alpha.

A      - (input) SuperMatrix*
         Matrix A with a sparse format, of dimension (A->nrow, A->ncol).
         Currently, the type of A can be:
             Stype = NC or NCP; Dtype = SLU_S; Mtype = GE.
         In the future, more general A can be handled.

B      - FLOAT PRECISION array of DIMENSION ( LDB, kb ), where kb is
         n when TRANSB = 'N' or 'n',  and is  k otherwise.
         Before entry with  TRANSB = 'N' or 'n',  the leading k by n
         part of the array B must contain the matrix B, otherwise
         the leading n by k part of the array B must contain the
         matrix B.
         Unchanged on exit.

LDB    - (input) int
         On entry, LDB specifies the first dimension of B as declared
         in the calling (sub) program. LDB must be at least max( 1, n ).
         Unchanged on exit.
```

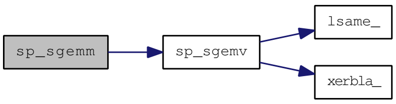
```
BETA    - (input) float
          On entry, BETA specifies the scalar beta. When BETA is
          supplied as zero then C need not be set on input.

C       - FLOAT PRECISION array of DIMENSION ( LDC, n ).
          Before entry, the leading m by n part of the array C must
          contain the matrix C,  except when beta is zero, in which
          case C need not be set on entry.
          On exit, the array C is overwritten by the m by n matrix
  ( alpha*op( A )*B + beta*C ).

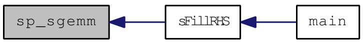
LDC     - (input) int
          On entry, LDC specifies the first dimension of C as declared
          in the calling (sub)program. LDC must be at least max(1,m).
          Unchanged on exit.

==== Sparse Level 3 Blas routine.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.134 SRC/superlu_timer.c File Reference

Returns the time used.

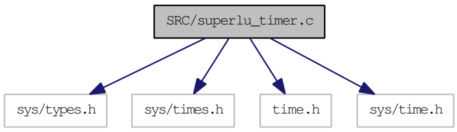
`#include <sys/types.h>`

`#include <sys/times.h>`

`#include <time.h>`

`#include <sys/time.h>`

Include dependency graph for superlu_timer.c:



Defines

- #define CLK_TCK 60

Functions

- double SuperLU_timer_ ()

    Timer function.

4.134.1 Detailed Description

```
Purpose
=======

Returns the time in seconds used by the process.

Note: the timer function call is machine dependent. Use conditional
      compilation to choose the appropriate function.
```

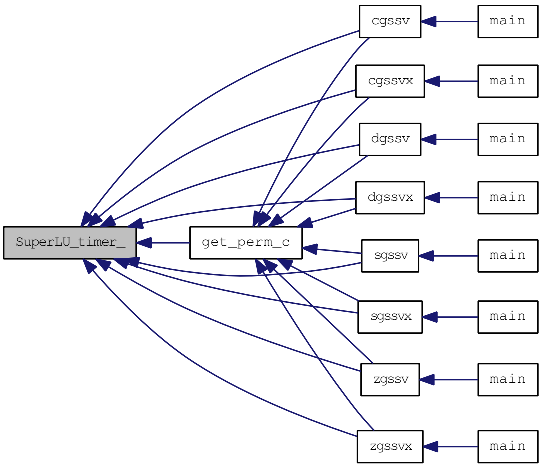
4.134.2 Define Documentation

4.134.2.1 #define CLK_TCK 60

4.134.3 Function Documentation

4.134.3.1 double SuperLU_timer_ ()

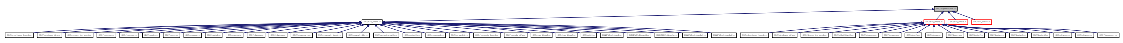
Here is the caller graph for this function:



4.135 SRC/supermatrix.h File Reference

Defines matrix types.

This graph shows which files directly or indirectly include this file:



Data Structures

- struct SuperMatrix
- struct NCformat
- struct NRformat
- struct SCformat
- struct SCPformat
- struct NCPformat
- struct DNformat
- struct NRformat_loc

Enumerations

- enum Stype_t {
  SLU_NC, SLU_NCP, SLU_NR, SLU_SC,
  SLU_SCP, SLU_SR, SLU_DN, SLU_NR_loc }
- enum Dtype_t { SLU_S, SLU_D, SLU_C, SLU_Z }
- enum Mtype_t {
  SLU_GE, SLU_TRLU, SLU_TRUU, SLU_TRL,
  SLU_TRU, SLU_SYL, SLU_SYU, SLU_HEL,
  SLU_HEU }

4.135.1 Detailed Description

4.135.2 Enumeration Type Documentation

4.135.2.1 enum Dtype_t

Enumerator:

SLU_S

SLU_D

SLU_C

SLU_Z

4.135.2.2 enum Mtype_t

Enumerator:

SLU_GE

SLU_TRLU

SLU_TRUU

SLU_TRL

SLU_TRU

SLU_SYL

SLU_SYU

SLU_HEL

SLU_HEU

4.135.2.3 enum Stype_t

Enumerator:

SLU_NC

SLU_NCP

SLU_NR

SLU_SC

SLU_SCP

SLU_SR

SLU_DN

SLU_NR_loc

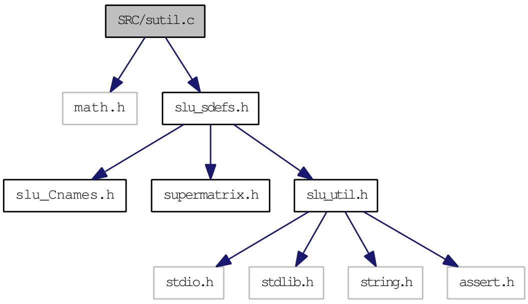
4.136 SRC/sutil.c File Reference

Matrix utility functions.

`#include <math.h>`

`#include "slu_sdefs.h"`

Include dependency graph for sutil.c:



Functions

- void sCreate_CompCol_Matrix (SuperMatrix ∗A, int m, int n, int nnz, float ∗nzval, int ∗rowind, int ∗colptr, Stype_t stype, Dtype_t dtype, Mtype_t mtype)

  Supernodal LU factor related.

- void sCreate_CompRow_Matrix (SuperMatrix ∗A, int m, int n, int nnz, float ∗nzval, int ∗colind, int ∗rowptr, Stype_t stype, Dtype_t dtype, Mtype_t mtype)
- void sCopy_CompCol_Matrix (SuperMatrix ∗A, SuperMatrix ∗B)

  Copy matrix A into matrix B.

- void sCreate_Dense_Matrix (SuperMatrix ∗X, int m, int n, float ∗x, int ldx, Stype_t stype, Dtype_t dtype, Mtype_t mtype)
- void sCopy_Dense_Matrix (int M, int N, float ∗X, int ldx, float ∗Y, int ldy)
- void sCreate_SuperNode_Matrix (SuperMatrix ∗L, int m, int n, int nnz, float ∗nzval, int ∗nzval_-colptr, int ∗rowind, int ∗rowind_colptr, int ∗col_to_sup, int ∗sup_to_col, Stype_t stype, Dtype_t dtype, Mtype_t mtype)
- void sCompRow_to_CompCol (int m, int n, int nnz, float ∗a, int ∗colind, int ∗rowptr, float ∗∗at, int ∗∗rowind, int ∗∗colptr)

  Convert a row compressed storage into a column compressed storage.

- void sPrint_CompCol_Matrix (char ∗what, SuperMatrix ∗A)

  Routines for debugging.

- void sPrint_SuperNode_Matrix (char ∗what, SuperMatrix ∗A)
- void sPrint_Dense_Matrix (char ∗what, SuperMatrix ∗A)
- void sprint_lu_col (char ∗msg, int jcol, int pivrow, int ∗xprune, GlobalLU_t ∗Glu)

  Diagnostic print of column "jcol" in the U/L factor.

- void scheck_tempv (int n, float ∗tempv)

Check whether tempv[] == 0. This should be true before and after calling any numeric routines, i.e., "panel_bmod" and "column_bmod".

- void sGenXtrue (int n, int nrhs, float ∗x, int ldx)
- void sFillRHS (trans_t trans, int nrhs, float ∗x, int ldx, SuperMatrix ∗A, SuperMatrix ∗B)

  Let rhs[i] = sum of i-th row of A, so the solution vector is all 1's.

- void sfill (float ∗a, int alen, float dval)

  Fills a float precision array with a given value.

- void sinf_norm_error (int nrhs, SuperMatrix ∗X, float ∗xtrue)

  Check the inf-norm of the error vector.

- void sPrintPerf (SuperMatrix ∗L, SuperMatrix ∗U, mem_usage_t ∗mem_usage, float rpg, float rcond, float ∗ferr, float ∗berr, char ∗equed, SuperLUStat_t ∗stat)

  Print performance of the code.

- print_float_vec (char ∗what, int n, float ∗vec)

4.136.1 Detailed Description

```
-- SuperLU routine (version 3.1) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
August 1, 2008
```

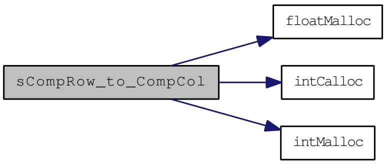
4.136.2 Function Documentation

4.136.2.1 print_float_vec (char ∗ *what*, int *n*, float ∗ *vec*)

4.136.2.2 void scheck_tempv (int *n*, float ∗ *tempv*)

4.136.2.3 void sCompRow_to_CompCol (int *m*, int *n*, int *nnz*, float ∗ *a*, int ∗ *colind*, int ∗ *rowptr*, float ∗∗ *at*, int ∗∗ *rowind*, int ∗∗ *colptr*)

Here is the call graph for this function:



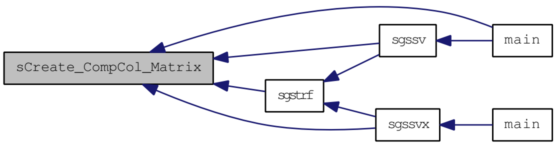
4.136.2.4 void sCopy_CompCol_Matrix (SuperMatrix ∗ *A*, SuperMatrix ∗ *B*)

4.136.2.5 void sCopy_Dense_Matrix (int *M*, int *N*, float ∗ *X*, int *ldx*, float ∗ *Y*, int *ldy*)

Copies a two-dimensional matrix X to another matrix Y.

4.136.2.6 void sCreate_CompCol_Matrix (SuperMatrix ∗ *A*, int *m*, int *n*, int *nnz*, float ∗ *nzval*, int ∗ *rowind*, int ∗ *colptr*, Stype_t *stype*, Dtype_t *dtype*, Mtype_t *mtype*)

Here is the caller graph for this function:



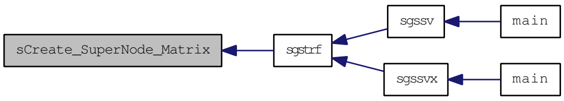
4.136.2.7 void sCreate_CompRow_Matrix (SuperMatrix ∗ *A*, int *m*, int *n*, int *nnz*, float ∗ *nzval*, int ∗ *colind*, int ∗ *rowptr*, Stype_t *stype*, Dtype_t *dtype*, Mtype_t *mtype*)

4.136.2.8 void sCreate_Dense_Matrix (SuperMatrix ∗ *X*, int *m*, int *n*, float ∗ *x*, int *ldx*, Stype_t *stype*, Dtype_t *dtype*, Mtype_t *mtype*)

Here is the caller graph for this function:

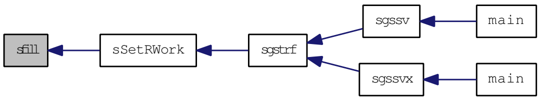
4.136.2.9 void sCreate_SuperNode_Matrix (SuperMatrix ∗ *L*, int *m*, int *n*, int *nnz*, float ∗ *nzval*, int ∗ *nzval_colptr*, int ∗ *rowind*, int ∗ *rowind_colptr*, int ∗ *col_to_sup*, int ∗ *sup_to_col*, Stype_t *stype*, Dtype_t *dtype*, Mtype_t *mtype*)

Here is the caller graph for this function:



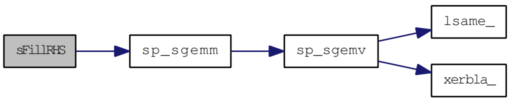
4.136.2.10 void sfill (float ∗ *a*, int *alen*, float *dval*)

Here is the caller graph for this function:

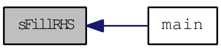


4.136.2.11 void sFillRHS (trans_t *trans*, int *nrhs*, float ∗ *x*, int *ldx*, SuperMatrix ∗ *A*, SuperMatrix ∗ *B*)

Here is the call graph for this function:

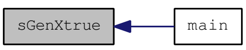


Here is the caller graph for this function:



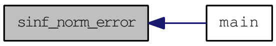
4.136.2.12 void sGenXtrue (int *n*, int *nrhs*, float ∗ *x*, int *ldx*)

Here is the caller graph for this function:



4.136.2.13 void sinf_norm_error (int *nrhs*, SuperMatrix ∗ *X*, float ∗ *xtrue*)

Here is the caller graph for this function:

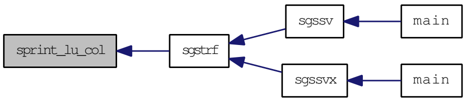


4.136.2.14 void sPrint_CompCol_Matrix (char ∗ *what*, SuperMatrix ∗ *A*)

4.136.2.15 void sPrint_Dense_Matrix (char ∗ *what*, SuperMatrix ∗ *A*)

4.136.2.16 void sprint_lu_col (char ∗ *msg*, int *jcol*, int *pivrow*, int ∗ *xprune*, GlobalLU_t ∗ *Glu*)

Here is the caller graph for this function:



4.136.2.17 void sPrint_SuperNode_Matrix (char ∗ *what*, SuperMatrix ∗ *A*)

4.136.2.18 void sPrintPerf (SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, mem_usage_t ∗ *mem_usage*, float *rpg*, float *rcond*, float ∗ *ferr*, float ∗ *berr*, char ∗ *equed*, SuperLUStat_t ∗ *stat*)

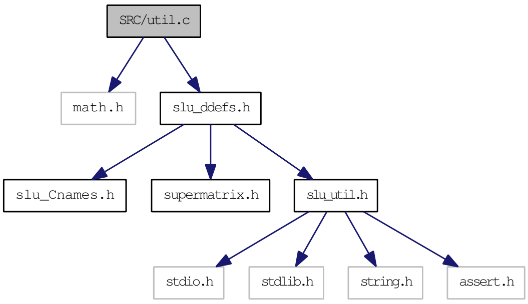
4.137 SRC/util.c File Reference

Utility functions.

`#include <math.h>`

`#include "slu_ddefs.h"`

Include dependency graph for util.c:



Defines

- #define NBUCKS 10

  Get the statistics of the supernodes.

Functions

- void superlu_abort_and_exit (char ∗msg)

  Global statistics variale.

- void set_default_options (superlu_options_t ∗options)

  Set the default values for the options argument.

- void print_options (superlu_options_t ∗options)

  Print the options setting.

- void Destroy_SuperMatrix_Store (SuperMatrix ∗A)

  Deallocate the structure pointing to the actual storage of the matrix.

- void Destroy_CompCol_Matrix (SuperMatrix ∗A)
- void Destroy_CompRow_Matrix (SuperMatrix ∗A)
- void Destroy_SuperNode_Matrix (SuperMatrix ∗A)
- void Destroy_CompCol_Permuted (SuperMatrix ∗A)

  A is of type Stype==NCP.

- void Destroy_Dense_Matrix (SuperMatrix ∗A)

  A is of type Stype==DN.

- void resetrep_col (const int nseg, const int ∗segrep, int ∗repfnz)

Reset repfnz[] for the current column.

- void countnz (const int n, int ∗xprune, int ∗nnzL, int ∗nnzU, GlobalLU_t ∗Glu)

  Count the total number of nonzeros in factors L and U, and in the symmetrically reduced L.

- void fixupL (const int n, const int ∗perm_r, GlobalLU_t ∗Glu)

  Fix up the data storage lsub for L-subscripts. It removes the subscript sets for structural pruning, and applies permuation to the remaining subscripts.

- void print_panel_seg (int n, int w, int jcol, int nseg, int ∗segrep, int ∗repfnz)

  Diagnostic print of segment info after panel_dfs().

- void StatInit (SuperLUStat_t ∗stat)
- void StatPrint (SuperLUStat_t ∗stat)
- void StatFree (SuperLUStat_t ∗stat)
- flops_t LUFactFlops (SuperLUStat_t ∗stat)
- flops_t LUSolveFlops (SuperLUStat_t ∗stat)
- void ifill (int ∗a, int alen, int ival)

  Fills an integer array with a given value.

- void super_stats (int nsuper, int ∗xsup)
- float SpaSize (int n, int np, float sum_npw)
- float DenseSize (int n, float sum_nw)
- void check_repfnz (int n, int w, int jcol, int ∗repfnz)

  Check whether repfnz[] == EMPTY after reset.

- void PrintSumm (char ∗type, int nfail, int nrun, int nerrs)

  Print a summary of the testing results.

- int print_int_vec (char ∗what, int n, int ∗vec)

Variables

- static int max_sup_size

4.137.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.137.2 Define Documentation

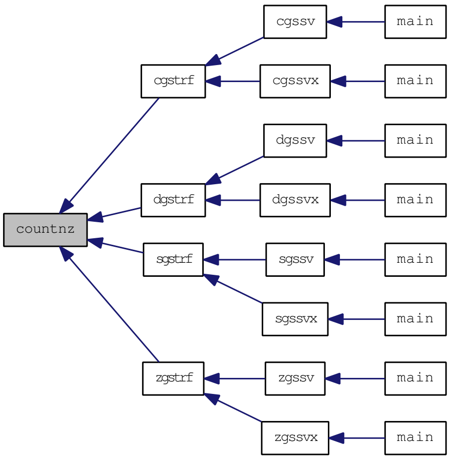
4.137.2.1 #define NBUCKS 10

4.137.3 Function Documentation

4.137.3.1 void check_repfnz (int *n*, int *w*, int *jcol*, int ∗ *repfnz*)

4.137.3.2 void countnz (const int *n*, int ∗ *xprune*, int ∗ *nnzL*, int ∗ *nnzU*, GlobalLU_t ∗ *Glu*)

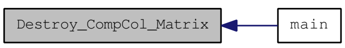
Here is the caller graph for this function:



4.137.3.3 float DenseSize (int *n*, float *sum_nw*)

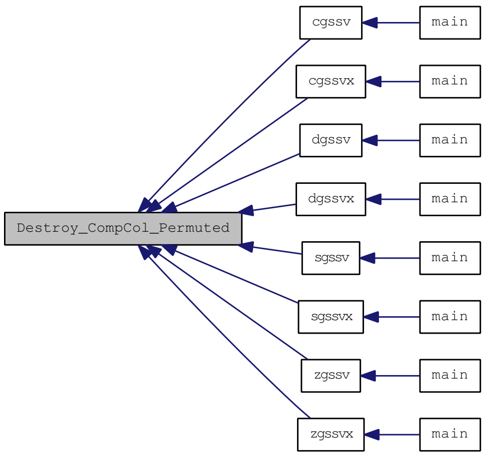
4.137.3.4 void Destroy_CompCol_Matrix (SuperMatrix ∗ *A*)

Here is the caller graph for this function:



4.137.3.5 void Destroy_CompCol_Permuted (SuperMatrix * A)

Here is the caller graph for this function:



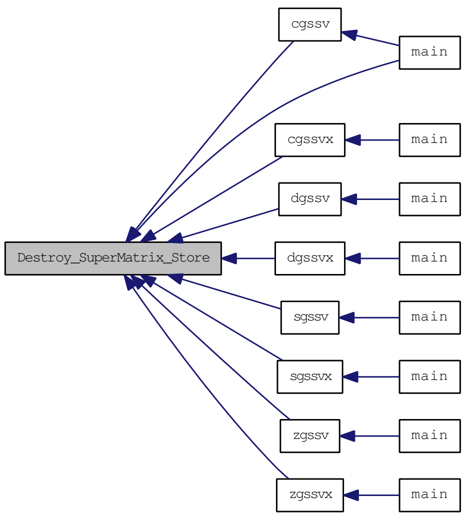
4.137.3.6 void Destroy_CompRow_Matrix (SuperMatrix * A)

4.137.3.7 void Destroy_Dense_Matrix (SuperMatrix * A)

Here is the caller graph for this function:

4.137.3.8 void Destroy_SuperMatrix_Store (SuperMatrix * A)

Here is the caller graph for this function:

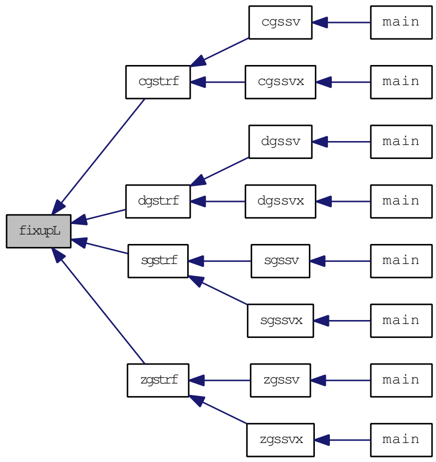


4.137.3.9 void Destroy_SuperNode_Matrix (SuperMatrix * *A*)

Here is the caller graph for this function:

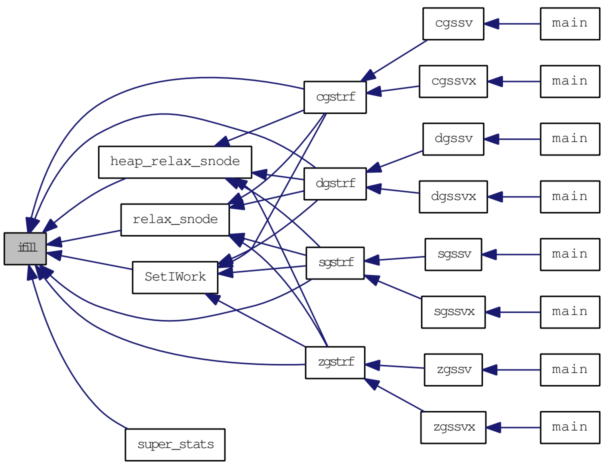
4.137.3.10 void fixupL (const int *n*, const int * *perm_r*, GlobalLU_t * *Glu*)

Here is the caller graph for this function:



4.137.3.11 void ifill (int * *a*, int *alen*, int *ival*)

Here is the caller graph for this function:

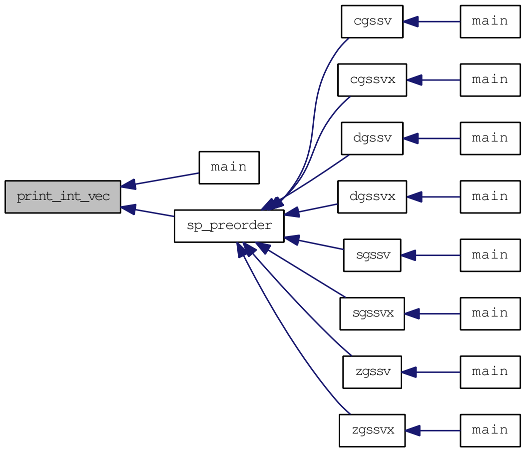


4.137.3.12 flops_t LUFactFlops (SuperLUStat_t ∗ *stat*)

4.137.3.13 flops_t LUSolveFlops (SuperLUStat_t ∗ *stat*)

4.137.3.14 int print_int_vec (char ∗ *what*, int *n*, int ∗ *vec*)

Here is the caller graph for this function:



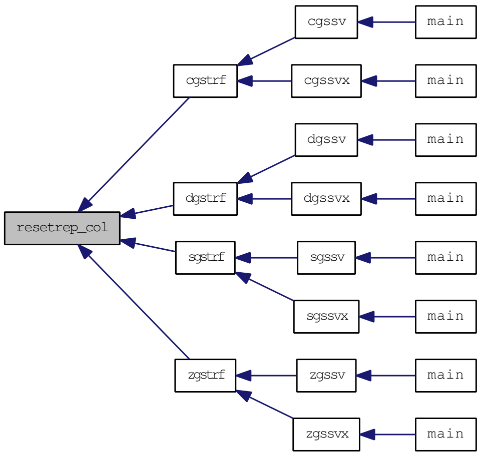
4.137.3.15 void print_options (superlu_options_t ∗ *options*)

4.137.3.16 void print_panel_seg (int *n*, int *w*, int *jcol*, int *nseg*, int ∗ *segrep*, int ∗ *repfnz*)

4.137.3.17 void PrintSumm (char ∗ *type*, int *nfail*, int *nrun*, int *nerrs*)

4.137.3.18 void resetrep_col (const int *nseg*, const int ∗ *segrep*, int ∗ *repfnz*)

Here is the caller graph for this function:



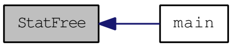
4.137.3.19 void set_default_options (superlu_options_t * *options*)

Here is the caller graph for this function:

4.137.3.20 float SpaSize (int *n*, int *np*, float *sum_npw*)

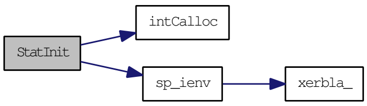
4.137.3.21 void StatFree (SuperLUStat_t * *stat*)

Here is the caller graph for this function:

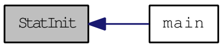


4.137.3.22 void StatInit (SuperLUStat_t * *stat*)

Here is the call graph for this function:

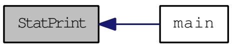


Here is the caller graph for this function:



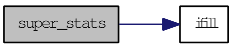
4.137.3.23 void StatPrint (SuperLUStat_t * *stat*)

Here is the caller graph for this function:



4.137.3.24 void super_stats (int *nsuper*, int * *xsup*)

Here is the call graph for this function:



4.137.3.25 void superlu_abort_and_exit (char * *msg*)

4.137.4 Variable Documentation

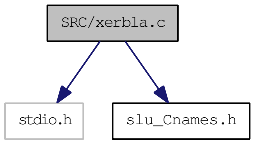
4.137.4.1 int max_sup_size [static]

4.138 SRC/xerbla.c File Reference

`#include <stdio.h>`

`#include "slu_Cnames.h"`

Include dependency graph for xerbla.c:



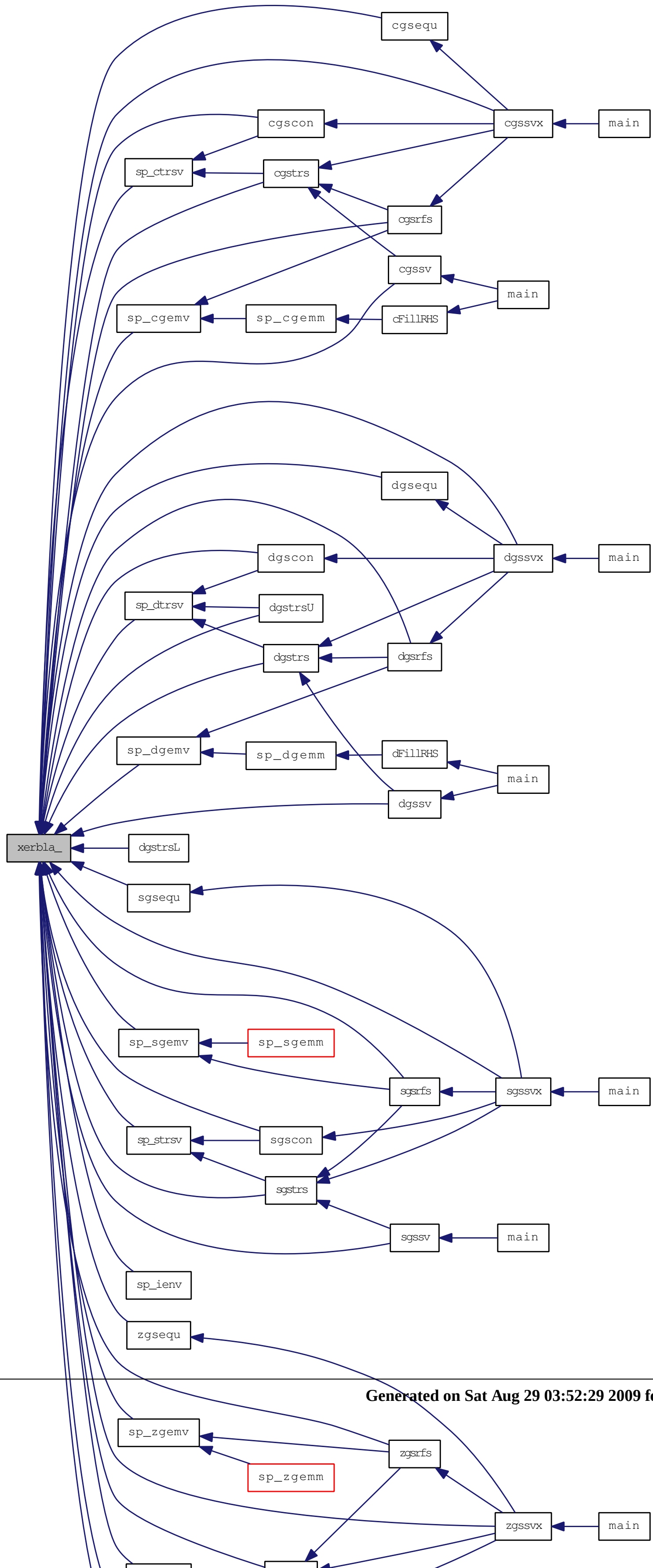
Functions

- int xerbla_ (char *srname, int *info)

4.138.1 Function Documentation

4.138.1.1 int xerbla_ (char * *srname*, int * *info*)

Here is the caller graph for this function:

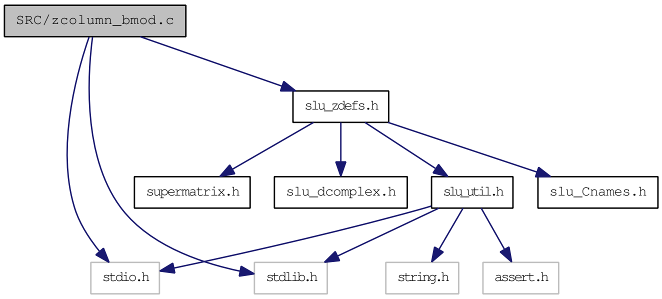


4.139 SRC/zcolumn_bmod.c File Reference

performs numeric block updates

```
#include <stdio.h>
#include <stdlib.h>
#include "slu_zdefs.h"
```

Include dependency graph for zcolumn_bmod.c:



Functions

- void zusolve (int, int, doublecomplex ∗, doublecomplex ∗)

  Solves a dense upper triangular system.

- void zlsolve (int, int, doublecomplex ∗, doublecomplex ∗)

  Solves a dense UNIT lower triangular system.

- void zmatvec (int, int, int, doublecomplex ∗, doublecomplex ∗, doublecomplex ∗)

  Performs a dense matrix-vector multiply: Mxvec = Mxvec + M ∗ vec.

- int zcolumn_bmod (const int jcol, const int nseg, doublecomplex ∗dense, doublecomplex ∗tempv, int ∗segrep, int ∗repfnz, int fpanelc, GlobalLU_t ∗Glu, SuperLUStat_t ∗stat)

4.139.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

 Permission is hereby granted to use or copy this program for any
 purpose, provided the above notices are retained on all copies.
 Permission to modify the code and to distribute modified code is
```

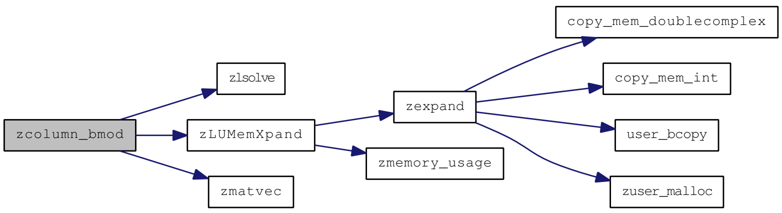
```
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.139.2 Function Documentation

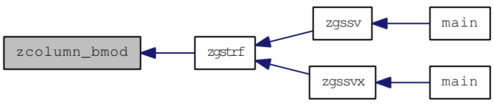
4.139.2.1 int zcolumn_bmod (const int *jcol*, const int *nseg*, doublecomplex ∗ *dense*, doublecomplex ∗ *tempv*, int ∗ *segrep*, int ∗ *repfnz*, int *fpanelc*, GlobalLU_t ∗ *Glu*, SuperLUStat_t ∗ *stat*)

```
Purpose:
========
Performs numeric block updates (sup-col) in topological order.
It features: col-col, 2cols-col, 3cols-col, and sup-col updates.
Special processing on the supernodal portion of L[*,j]
Return value:   0 - successful return
              > 0 - number of bytes allocated when run out of space
```

Here is the call graph for this function:



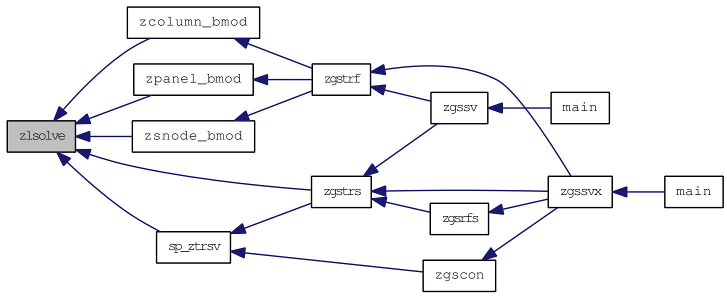
Here is the caller graph for this function:



4.139.2.2 void zlsolve (int *ldm*, int *ncol*, doublecomplex ∗ *M*, doublecomplex ∗ *rhs*)

The unit lower triangular matrix is stored in a 2D array M(1:nrow,1:ncol). The solution will be returned in the rhs vector.

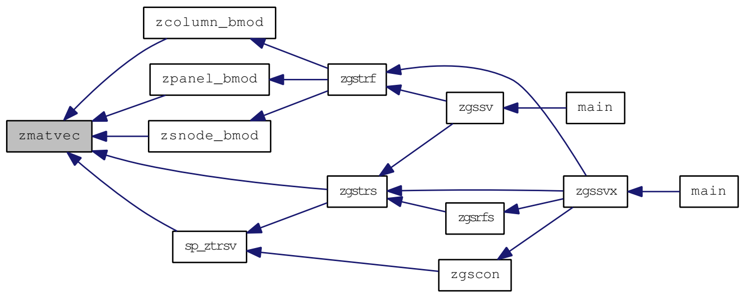
Here is the caller graph for this function:



4.139.2.3 void zmatvec (int *ldm*, int *nrow*, int *ncol*, doublecomplex ∗ *M*, doublecomplex ∗ *vec*, doublecomplex ∗ *Mxvec*)

The input matrix is M(1:nrow,1:ncol); The product is returned in Mxvec[].

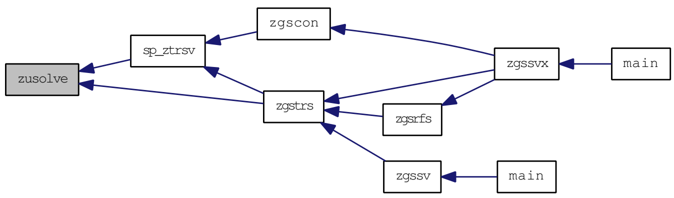
Here is the caller graph for this function:



4.139.2.4 void zusolve (int *ldm*, int *ncol*, doublecomplex ∗ *M*, doublecomplex ∗ *rhs*)

The upper triangular matrix is stored in a 2-dim array M(1:ldm,1:ncol). The solution will be returned in the rhs vector.

Here is the caller graph for this function:

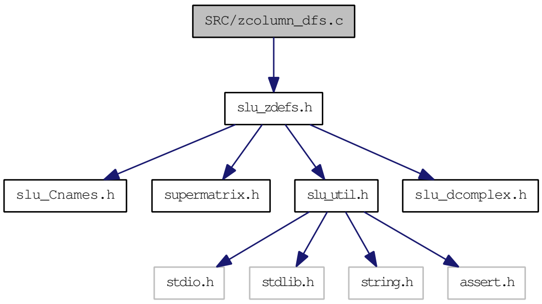


4.140 SRC/zcolumn_dfs.c File Reference

Performs a symbolic factorization.

`#include "slu_zdefs.h"`

Include dependency graph for zcolumn_dfs.c:



Defines

- #define T2_SUPER

  What type of supernodes we want.

Functions

- int zcolumn_dfs (const int m, const int jcol, int ∗perm_r, int ∗nseg, int ∗lsub_col, int ∗segrep, int ∗repfnz, int ∗xprune, int ∗marker, int ∗parent, int ∗xplore, GlobalLU_t ∗Glu)

4.140.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003


Copyright (c) 1994 by Xerox Corporation.  All rights reserved.


THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.


Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.140.2 Define Documentation

4.140.2.1 #define T2_SUPER

4.140.3 Function Documentation

4.140.3.1 int zcolumn_dfs (const int *m*, const int *jcol*, int ∗ *perm_r*, int ∗ *nseg*, int ∗ *lsub_col*, int ∗ *segrep*, int ∗ *repfnz*, int ∗ *xprune*, int ∗ *marker*, int ∗ *parent*, int ∗ *xplore*, GlobalLU_t ∗ *Glu*)

```
Purpose
=======
  "column_dfs" performs a symbolic factorization on column jcol, and
  decide the supernode boundary.


  This routine does not use numeric values, but only use the RHS
  row indices to start the dfs.

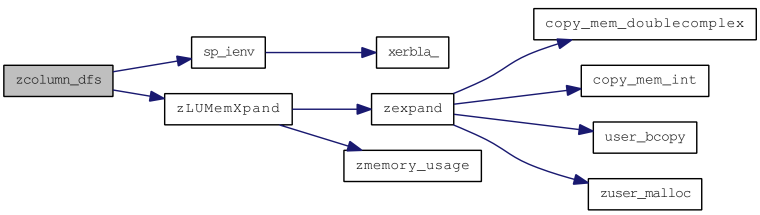

  A supernode representative is the last column of a supernode.
  The nonzeros in U[*,j] are segments that end at supernodal
  representatives. The routine returns a list of such supernodal
  representatives in topological order of the dfs that generates them.
  The location of the first nonzero in each such supernodal segment
  (supernodal entry location) is also returned.


Local parameters
================
  nseg: no of segments in current U[*,j]
  jsuper: jsuper=EMPTY if column j does not belong to the same
supernode as j-1. Otherwise, jsuper=nsuper.

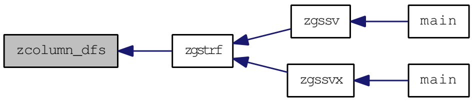

  marker2: A-row --> A-row/col (0/1)
  repfnz: SuperA-col --> PA-row
  parent: SuperA-col --> SuperA-col
  xplore: SuperA-col --> index to L-structure


Return value
============
    0  success;
  > 0  number of bytes allocated when run out of space.
```

Here is the call graph for this function:



Here is the caller graph for this function:

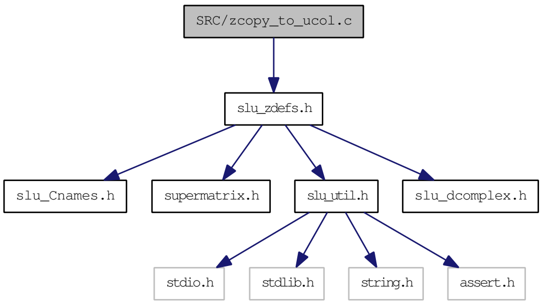


4.141 SRC/zcopy_to_ucol.c File Reference

Copy a computed column of U to the compressed data structure.

`#include "slu_zdefs.h"`

Include dependency graph for zcopy_to_ucol.c:



Functions

- int zcopy_to_ucol (int jcol, int nseg, int ∗segrep, int ∗repfnz, int ∗perm_r, doublecomplex ∗dense, GlobalLU_t ∗Glu)

4.141.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997
Copyright (c) 1994 by Xerox Corporation.  All rights reserved.



THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

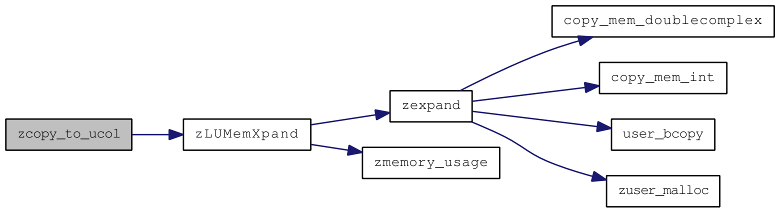


Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

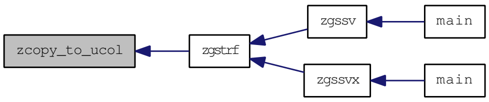
4.141.2 Function Documentation

4.141.2.1 int zcopy_to_ucol (int *jcol*, int *nseg*, int ∗ *segrep*, int ∗ *repfnz*, int ∗ *perm_r*, doublecomplex ∗ *dense*, GlobalLU_t ∗ *Glu*)

Here is the call graph for this function:



Here is the caller graph for this function:



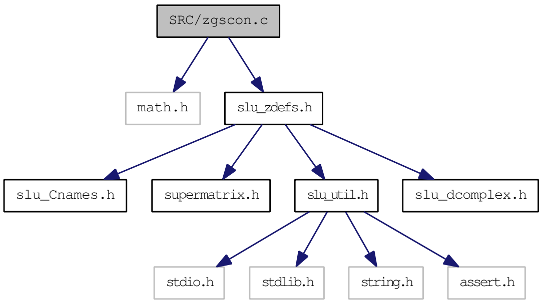
4.142 SRC/zgscon.c File Reference

Estimates reciprocal of the condition number of a general matrix.

`#include <math.h>`

`#include "slu_zdefs.h"`

Include dependency graph for zgscon.c:



Functions

- void zgscon (char *norm, SuperMatrix *L, SuperMatrix *U, double anorm, double *rcond, SuperLUStat_t *stat, int *info)

4.142.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003


Modified from lapack routines ZGECON.
```

4.142.2 Function Documentation

4.142.2.1 void zgscon (char * *norm*, SuperMatrix * *L*, SuperMatrix * *U*, double *anorm*, double * *rcond*, SuperLUStat_t * *stat*, int * *info*)

```
Purpose
=======

ZGSCON estimates the reciprocal of the condition number of a general
real matrix A, in either the 1-norm or the infinity-norm, using
the LU factorization computed by ZGETRF.   *

An estimate is obtained for norm(inv(A)), and the reciprocal of the
condition number is computed as
   RCOND = 1 / ( norm(A) * norm(inv(A)) ).
```

```
See supermatrix.h for the definition of 'SuperMatrix' structure.


Arguments
=========


 NORM    (input) char*
         Specifies whether the 1-norm condition number or the
         infinity-norm condition number is required:
         = '1' or 'O':  1-norm;
         = 'I':         Infinity-norm.


 L       (input) SuperMatrix*
         The factor L from the factorization Pr*A*Pc=L*U as computed by
         zgstrf(). Use compressed row subscripts storage for supernodes,
         i.e., L has types: Stype = SLU_SC, Dtype = SLU_Z, Mtype = SLU_TRLU.


 U       (input) SuperMatrix*
         The factor U from the factorization Pr*A*Pc=L*U as computed by
         zgstrf(). Use column-wise storage scheme, i.e., U has types:
         Stype = SLU_NC, Dtype = SLU_Z, Mtype = SLU_TRU.

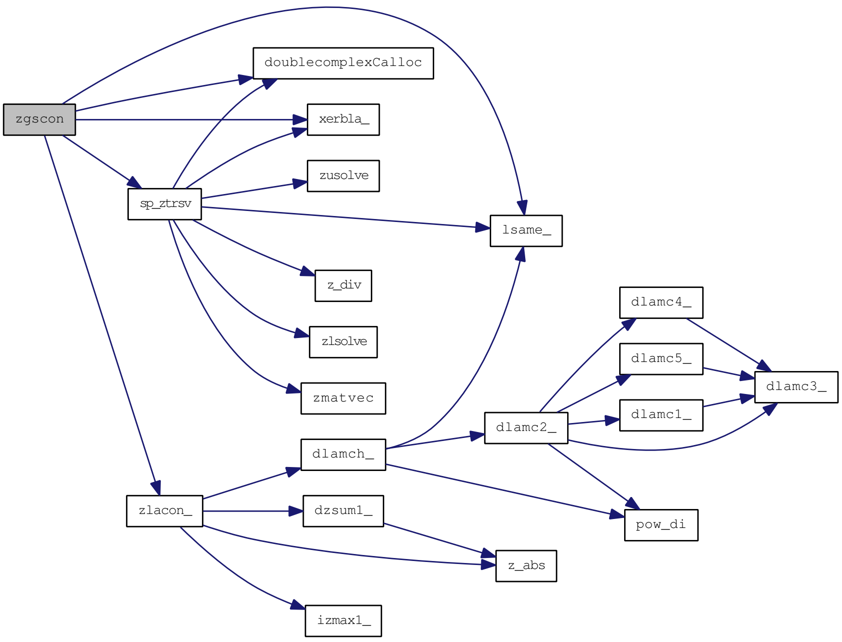

 ANORM   (input) double
         If NORM = '1' or 'O', the 1-norm of the original matrix A.
         If NORM = 'I', the infinity-norm of the original matrix A.


 RCOND   (output) double*
        The reciprocal of the condition number of the matrix A,
        computed as RCOND = 1/(norm(A) * norm(inv(A))).


 INFO    (output) int*
        = 0:  successful exit
        < 0:  if INFO = -i, the i-th argument had an illegal value


 ====================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:

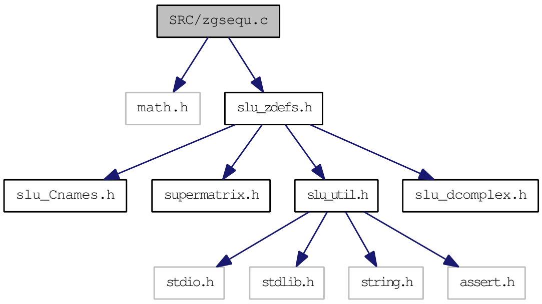
4.143 SRC/zgsequ.c File Reference

Computes row and column scalings.

`#include <math.h>`

`#include "slu_zdefs.h"`

Include dependency graph for zgsequ.c:



Functions

- void zgsequ (SuperMatrix ∗A, double ∗r, double ∗c, double ∗rowcnd, double ∗colcnd, double ∗amax, int ∗info)

  Driver related.

4.143.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997


Modified from LAPACK routine ZGEEQU
```

4.143.2 Function Documentation

4.143.2.1 void zgsequ (SuperMatrix ∗ *A*, double ∗ *r*, double ∗ *c*, double ∗ *rowcnd*, double ∗ *colcnd*, double ∗ *amax*, int ∗ *info*)

```
Purpose
  =======

  ZGSEQU computes row and column scalings intended to equilibrate an
  M-by-N sparse matrix A and reduce its condition number. R returns the row
  scale factors and C the column scale factors, chosen to try to make
  the largest element in each row and column of the matrix B with
  elements B(i,j)=R(i)*A(i,j)*C(j) have absolute value 1.
```

```
R(i) and C(j) are restricted to be between SMLNUM = smallest safe
number and BIGNUM = largest safe number.  Use of these scaling
factors is not guaranteed to reduce the condition number of A but
works well in practice.


See supermatrix.h for the definition of 'SuperMatrix' structure.


Arguments
=========


A       (input) SuperMatrix*
        The matrix of dimension (A->nrow, A->ncol) whose equilibration
        factors are to be computed. The type of A can be:
        Stype = SLU_NC; Dtype = SLU_Z; Mtype = SLU_GE.


R       (output) double*, size A->nrow
        If INFO = 0 or INFO > M, R contains the row scale factors
        for A.


C       (output) double*, size A->ncol
        If INFO = 0,  C contains the column scale factors for A.


ROWCND  (output) double*
        If INFO = 0 or INFO > M, ROWCND contains the ratio of the
        smallest R(i) to the largest R(i).  If ROWCND >= 0.1 and
        AMAX is neither too large nor too small, it is not worth
        scaling by R.

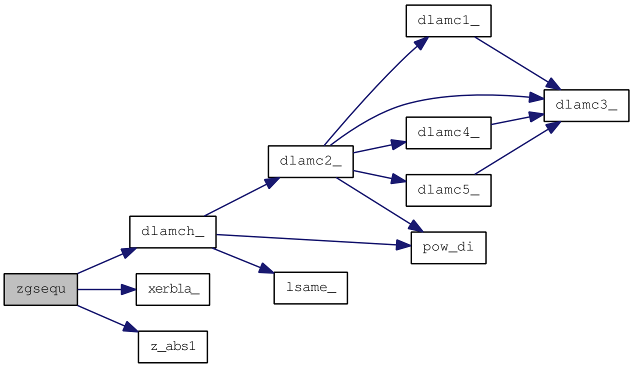

COLCND  (output) double*
        If INFO = 0, COLCND contains the ratio of the smallest
        C(i) to the largest C(i).  If COLCND >= 0.1, it is not
        worth scaling by C.


AMAX    (output) double*
        Absolute value of largest matrix element.  If AMAX is very
        close to overflow or very close to underflow, the matrix
        should be scaled.


INFO    (output) int*
        = 0:  successful exit
        < 0:  if INFO = -i, the i-th argument had an illegal value
        > 0:  if INFO = i,  and i is
              <= A->nrow:  the i-th row of A is exactly zero
              >  A->ncol:  the (i-M)-th column of A is exactly zero


=====================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:

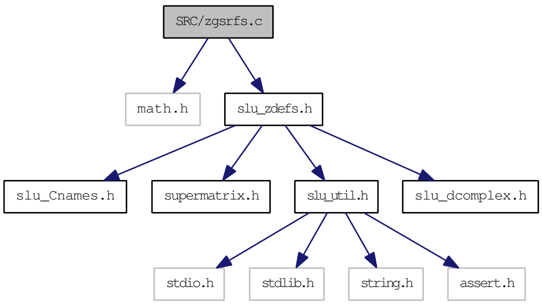
4.144 SRC/zgsrfs.c File Reference

Improves computed solution to a system of inear equations.

`#include <math.h>`

`#include "slu_zdefs.h"`

Include dependency graph for zgsrfs.c:



Defines

- #define ITMAX 5

Functions

- void zgsrfs (trans_t trans, SuperMatrix ∗A, SuperMatrix ∗L, SuperMatrix ∗U, int ∗perm_c, int ∗perm_r, char ∗equed, double ∗R, double ∗C, SuperMatrix ∗B, SuperMatrix ∗X, double ∗ferr, double ∗berr, SuperLUStat_t ∗stat, int ∗info)

4.144.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003
```

```
Modified from lapack routine ZGERFS
```

4.144.2 Define Documentation

4.144.2.1 #define ITMAX 5

4.144.3 Function Documentation

4.144.3.1 void zgsrfs (trans_t *trans*, SuperMatrix ∗ *A*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, int ∗ *perm_c*, int ∗ *perm_r*, char ∗ *equed*, double ∗ *R*, double ∗ *C*, SuperMatrix ∗ *B*, SuperMatrix ∗ *X*, double ∗ *ferr*, double ∗ *berr*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======

ZGSRFS improves the computed solution to a system of linear
equations and provides error bounds and backward error estimates for
the solution.

If equilibration was performed, the system becomes:
        (diag(R)*A_original*diag(C)) * X = diag(R)*B_original.

See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

trans   (input) trans_t
         Specifies the form of the system of equations:
         = NOTRANS: A * X = B  (No transpose)
         = TRANS:   A'* X = B  (Transpose)
         = CONJ:    A**H * X = B  (Conjugate transpose)

  A       (input) SuperMatrix*
          The original matrix A in the system, or the scaled A if
          equilibration was done. The type of A can be:
          Stype = SLU_NC, Dtype = SLU_Z, Mtype = SLU_GE.

  L       (input) SuperMatrix*
    The factor L from the factorization Pr*A*Pc=L*U. Use
          compressed row subscripts storage for supernodes,
          i.e., L has types: Stype = SLU_SC, Dtype = SLU_Z, Mtype = SLU_TRLU.

  U       (input) SuperMatrix*
          The factor U from the factorization Pr*A*Pc=L*U as computed by
          zgstrf(). Use column-wise storage scheme,
          i.e., U has types: Stype = SLU_NC, Dtype = SLU_Z, Mtype = SLU_TRU.

  perm_c  (input) int*, dimension (A->ncol)
    Column permutation vector, which defines the
          permutation matrix Pc; perm_c[i] = j means column i of A is
          in position j in A*Pc.

  perm_r  (input) int*, dimension (A->nrow)
          Row permutation vector, which defines the permutation matrix Pr;
          perm_r[i] = j means row i of A is in position j in Pr*A.
```

```
equed    (input) Specifies the form of equilibration that was done.
         = 'N': No equilibration.
         = 'R': Row equilibration, i.e., A was premultiplied by diag(R).
         = 'C': Column equilibration, i.e., A was postmultiplied by
                diag(C).
         = 'B': Both row and column equilibration, i.e., A was replaced
                by diag(R)*A*diag(C).

R        (input) double*, dimension (A->nrow)
         The row scale factors for A.
         If equed = 'R' or 'B', A is premultiplied by diag(R).
         If equed = 'N' or 'C', R is not accessed.

C        (input) double*, dimension (A->ncol)
         The column scale factors for A.
         If equed = 'C' or 'B', A is postmultiplied by diag(C).
         If equed = 'N' or 'R', C is not accessed.

B        (input) SuperMatrix*
         B has types: Stype = SLU_DN, Dtype = SLU_Z, Mtype = SLU_GE.
         The right hand side matrix B.
         if equed = 'R' or 'B', B is premultiplied by diag(R).

X        (input/output) SuperMatrix*
         X has types: Stype = SLU_DN, Dtype = SLU_Z, Mtype = SLU_GE.
         On entry, the solution matrix X, as computed by zgstrs().
         On exit, the improved solution matrix X.
         if *equed = 'C' or 'B', X should be premultiplied by diag(C)
             in order to obtain the solution to the original system.

FERR     (output) double*, dimension (B->ncol)
         The estimated forward error bound for each solution vector
         X(j) (the j-th column of the solution matrix X).
         If XTRUE is the true solution corresponding to X(j), FERR(j)
         is an estimated upper bound for the magnitude of the largest
         element in (X(j) - XTRUE) divided by the magnitude of the
         largest element in X(j).  The estimate is as reliable as
         the estimate for RCOND, and is almost always a slight
         overestimate of the true error.

BERR     (output) double*, dimension (B->ncol)
         The componentwise relative backward error of each solution
         vector X(j) (i.e., the smallest relative change in
         any element of A or B that makes X(j) an exact solution).

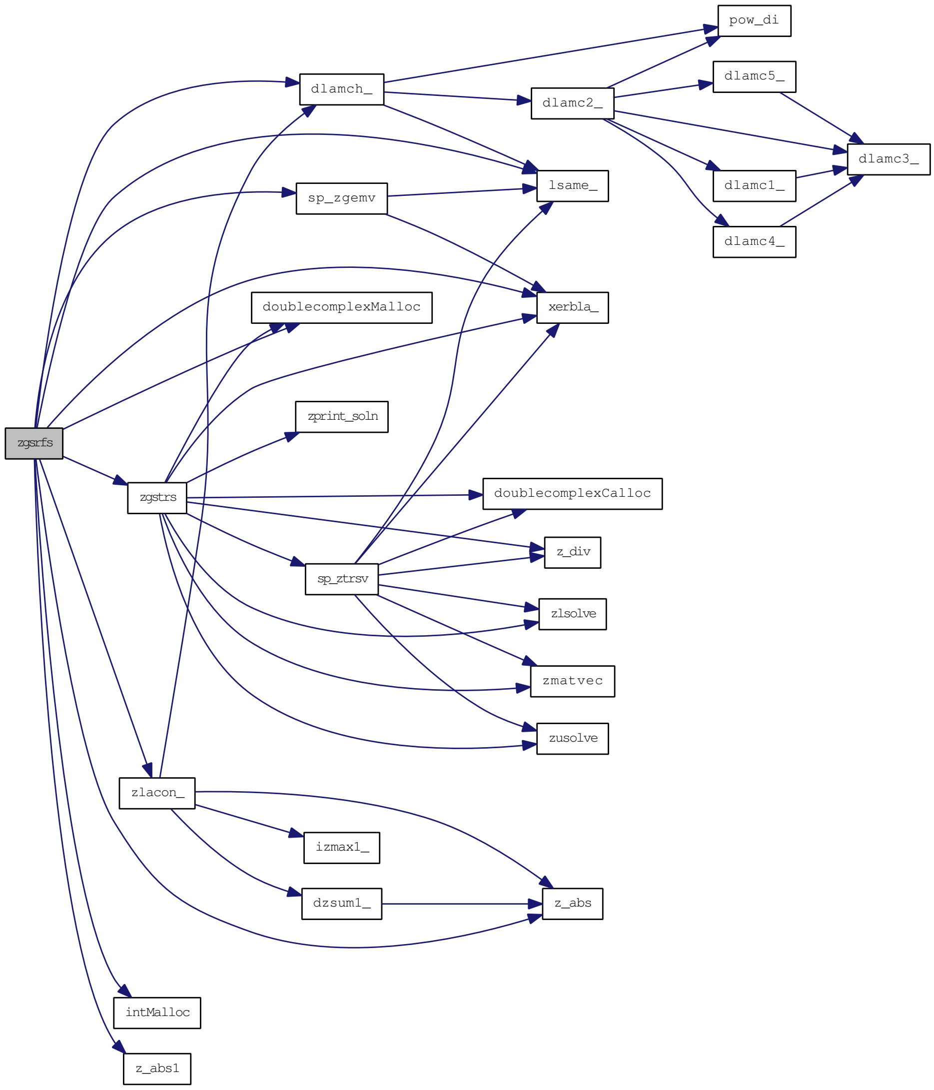
stat      (output) SuperLUStat_t*
          Record the statistics on runtime and floating-point operation count.
          See util.h for the definition of 'SuperLUStat_t'.

info     (output) int*
         = 0:  successful exit
          < 0:  if INFO = -i, the i-th argument had an illegal value

 Internal Parameters
 ===================
```

```
ITMAX is the maximum number of steps of iterative refinement.
```

Here is the call graph for this function:



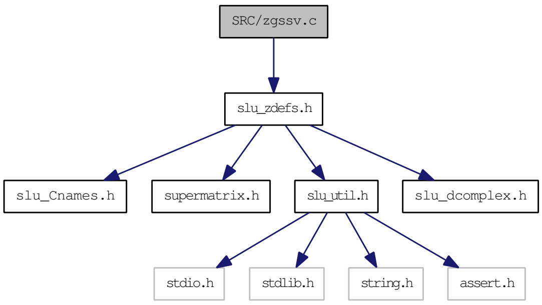
Here is the caller graph for this function:

4.145 SRC/zgssv.c File Reference

Solves the system of linear equations A∗X=B.

`#include "slu_zdefs.h"`

Include dependency graph for zgssv.c:



Functions

- void zgssv (superlu_options_t ∗options, SuperMatrix ∗A, int ∗perm_c, int ∗perm_r, SuperMatrix ∗L, SuperMatrix ∗U, SuperMatrix ∗B, SuperLUStat_t ∗stat, int ∗info)

  Driver routines.

4.145.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003
```

4.145.2 Function Documentation

4.145.2.1 void zgssv (superlu_options_t ∗ *options*, SuperMatrix ∗ *A*, int ∗ *perm_c*, int ∗ *perm_r*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, SuperMatrix ∗ *B*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======

ZGSSV solves the system of linear equations A*X=B, using the
LU factorization from ZGSTRF. It performs the following steps:

  1. If A is stored column-wise (A->Stype = SLU_NC):

     1.1. Permute the columns of A, forming A*Pc, where Pc
          is a permutation matrix. For more details of this step,
          see sp_preorder.c.
```

```
    1.2. Factor A as Pr*A*Pc=L*U with the permutation Pr determined
         by Gaussian elimination with partial pivoting.
         L is unit lower triangular with offdiagonal entries
         bounded by 1 in magnitude, and U is upper triangular.

    1.3. Solve the system of equations A*X=B using the factored
         form of A.

 2. If A is stored row-wise (A->Stype = SLU_NR), apply the
    above algorithm to the transpose of A:

    2.1. Permute columns of transpose(A) (rows of A),
         forming transpose(A)*Pc, where Pc is a permutation matrix.
         For more details of this step, see sp_preorder.c.

    2.2. Factor A as Pr*transpose(A)*Pc=L*U with the permutation Pr
         determined by Gaussian elimination with partial pivoting.
         L is unit lower triangular with offdiagonal entries
         bounded by 1 in magnitude, and U is upper triangular.

    2.3. Solve the system of equations A*X=B using the factored
         form of A.

 See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

options (input) superlu_options_t*
        The structure defines the input parameters to control
        how the LU decomposition will be performed and how the
        system will be solved.

A       (input) SuperMatrix*
        Matrix A in A*X=B, of dimension (A->nrow, A->ncol). The number
        of linear equations is A->nrow. Currently, the type of A can be:
        Stype = SLU_NC or SLU_NR; Dtype = SLU_Z; Mtype = SLU_GE.
        In the future, more general A may be handled.

perm_c  (input/output) int*
        If A->Stype = SLU_NC, column permutation vector of size A->ncol
        which defines the permutation matrix Pc; perm_c[i] = j means
        column i of A is in position j in A*Pc.
        If A->Stype = SLU_NR, column permutation vector of size A->nrow
        which describes permutation of columns of transpose(A)
        (rows of A) as described above.

        If options->ColPerm = MY_PERMC or options->Fact = SamePattern or
           options->Fact = SamePattern_SameRowPerm, it is an input argument.
           On exit, perm_c may be overwritten by the product of the input
           perm_c and a permutation that postorders the elimination tree
           of Pc'*A'*Pc; perm_c is not changed if the elimination tree
           is already in postorder.
        Otherwise, it is an output argument.
```

```
perm_r  (input/output) int*
        If A->Stype = SLU_NC, row permutation vector of size A->nrow,
        which defines the permutation matrix Pr, and is determined
        by partial pivoting.  perm_r[i] = j means row i of A is in
        position j in Pr*A.
        If A->Stype = SLU_NR, permutation vector of size A->ncol, which
        determines permutation of rows of transpose(A)
        (columns of A) as described above.



        If options->RowPerm = MY_PERMR or
           options->Fact = SamePattern_SameRowPerm, perm_r is an
           input argument.
        otherwise it is an output argument.



L       (output) SuperMatrix*
        The factor L from the factorization
            Pr*A*Pc=L*U              (if A->Stype = SLU_NC) or
            Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
        Uses compressed row subscripts storage for supernodes, i.e.,
        L has types: Stype = SLU_SC, Dtype = SLU_Z, Mtype = SLU_TRLU.



U       (output) SuperMatrix*
  The factor U from the factorization
            Pr*A*Pc=L*U              (if A->Stype = SLU_NC) or
            Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
        Uses column-wise storage scheme, i.e., U has types:
        Stype = SLU_NC, Dtype = SLU_Z, Mtype = SLU_TRU.



B       (input/output) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_Z, Mtype = SLU_GE.
        On entry, the right hand side matrix.
        On exit, the solution matrix if info = 0;

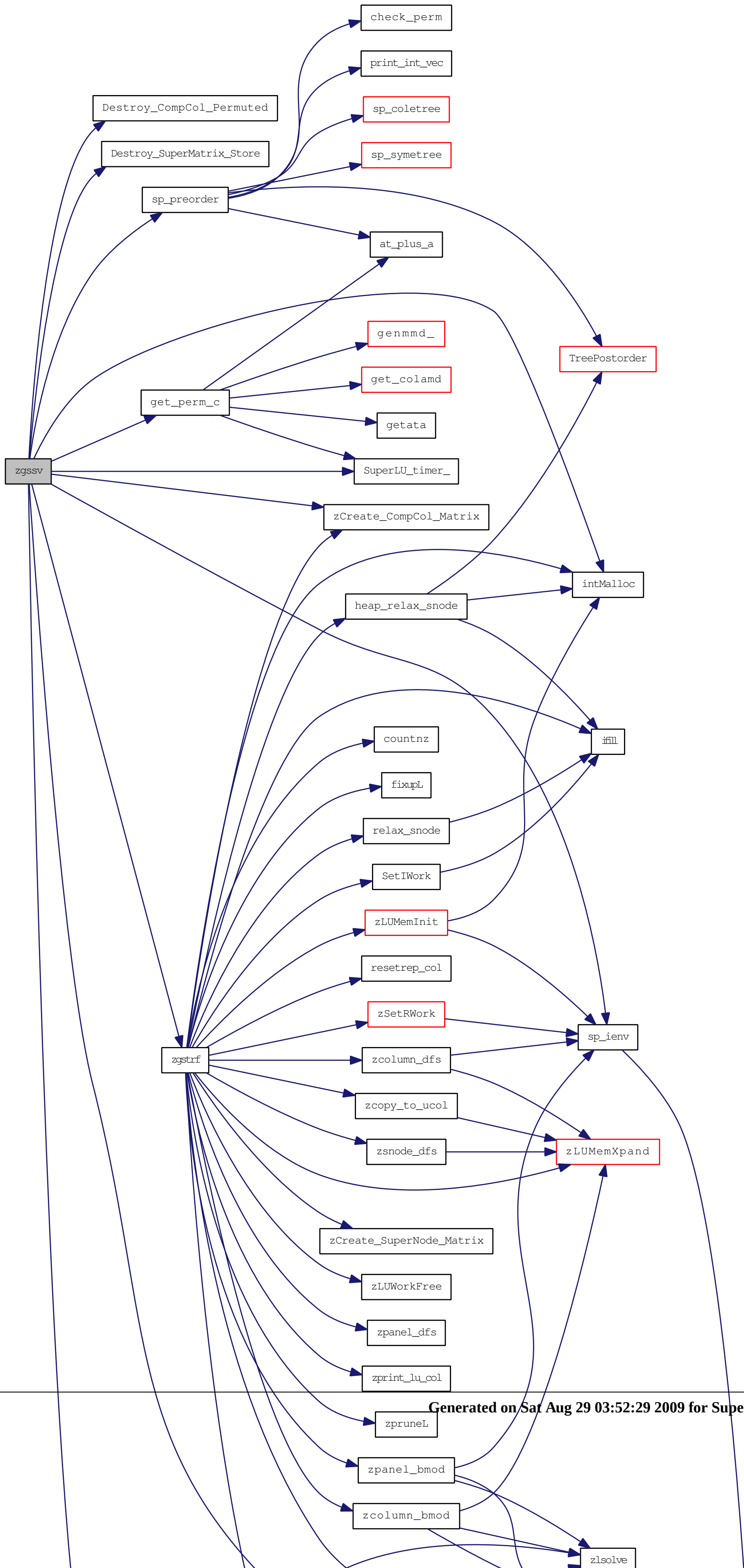


stat   (output) SuperLUStat_t*
       Record the statistics on runtime and floating-point operation count.
       See util.h for the definition of 'SuperLUStat_t'.

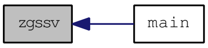


info    (output) int*
  = 0: successful exit
        > 0: if info = i, and i is
            <= A->ncol: U(i,i) is exactly zero. The factorization has
               been completed, but the factor U is exactly singular,
               so the solution could not be computed.
            > A->ncol: number of bytes allocated when memory allocation
               failure occurred, plus A->ncol.
```

Here is the call graph for this function:



Here is the caller graph for this function:

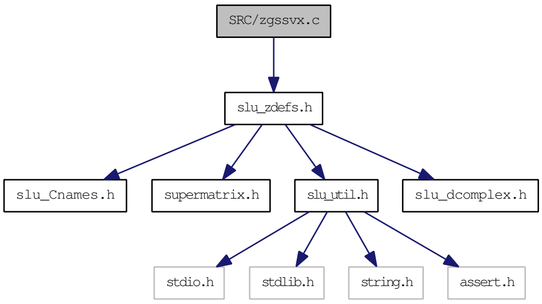


4.146 SRC/zgssvx.c File Reference

Solves the system of linear equations A∗X=B or A'∗X=B.

`#include "slu_zdefs.h"`

Include dependency graph for zgssvx.c:



Functions

- void zgssvx (superlu_options_t ∗options, SuperMatrix ∗A, int ∗perm_c, int ∗perm_r, int ∗etree, char ∗equed, double ∗R, double ∗C, SuperMatrix ∗L, SuperMatrix ∗U, void ∗work, int lwork, SuperMatrix ∗B, SuperMatrix ∗X, double ∗recip_pivot_growth, double ∗rcond, double ∗ferr, double ∗berr, mem_usage_t ∗mem_usage, SuperLUStat_t ∗stat, int ∗info)

4.146.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003
```

4.146.2 Function Documentation

4.146.2.1 void zgssvx (superlu_options_t ∗ *options*, SuperMatrix ∗ *A*, int ∗ *perm_c*, int ∗ *perm_r*, int ∗ *etree*, char ∗ *equed*, double ∗ *R*, double ∗ *C*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, void ∗ *work*, int *lwork*, SuperMatrix ∗ *B*, SuperMatrix ∗ *X*, double ∗ *recip_pivot_growth*, double ∗ *rcond*, double ∗ *ferr*, double ∗ *berr*, mem_usage_t ∗ *mem_usage*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======

ZGSSVX solves the system of linear equations A*X=B or A'*X=B, using
the LU factorization from zgstrf(). Error bounds on the solution and
a condition estimate are also provided. It performs the following steps:

  1. If A is stored column-wise (A->Stype = SLU_NC):
```

```
   1.1. If options->Equil = YES, scaling factors are computed to
        equilibrate the system:
        options->Trans = NOTRANS:
            diag(R)*A*diag(C) *inv(diag(C))*X = diag(R)*B
        options->Trans = TRANS:
            (diag(R)*A*diag(C))**T *inv(diag(R))*X = diag(C)*B
        options->Trans = CONJ:
            (diag(R)*A*diag(C))**H *inv(diag(R))*X = diag(C)*B
        Whether or not the system will be equilibrated depends on the
        scaling of the matrix A, but if equilibration is used, A is
        overwritten by diag(R)*A*diag(C) and B by diag(R)*B
        (if options->Trans=NOTRANS) or diag(C)*B (if options->Trans
        = TRANS or CONJ).

   1.2. Permute columns of A, forming A*Pc, where Pc is a permutation
        matrix that usually preserves sparsity.
        For more details of this step, see sp_preorder.c.

   1.3. If options->Fact != FACTORED, the LU decomposition is used to
        factor the matrix A (after equilibration if options->Equil = YES)
        as Pr*A*Pc = L*U, with Pr determined by partial pivoting.

   1.4. Compute the reciprocal pivot growth factor.

   1.5. If some U(i,i) = 0, so that U is exactly singular, then the
        routine returns with info = i. Otherwise, the factored form of
        A is used to estimate the condition number of the matrix A. If
        the reciprocal of the condition number is less than machine
        precision, info = A->ncol+1 is returned as a warning, but the
        routine still goes on to solve for X and computes error bounds
        as described below.

   1.6. The system of equations is solved for X using the factored form
        of A.

   1.7. If options->IterRefine != NOREFINE, iterative refinement is
        applied to improve the computed solution matrix and calculate
        error bounds and backward error estimates for it.

   1.8. If equilibration was used, the matrix X is premultiplied by
        diag(C) (if options->Trans = NOTRANS) or diag(R)
        (if options->Trans = TRANS or CONJ) so that it solves the
        original system before equilibration.

2. If A is stored row-wise (A->Stype = SLU_NR), apply the above algorithm
   to the transpose of A:

   2.1. If options->Equil = YES, scaling factors are computed to
        equilibrate the system:
        options->Trans = NOTRANS:
            diag(R)*A*diag(C) *inv(diag(C))*X = diag(R)*B
        options->Trans = TRANS:
            (diag(R)*A*diag(C))**T *inv(diag(R))*X = diag(C)*B
        options->Trans = CONJ:
            (diag(R)*A*diag(C))**H *inv(diag(R))*X = diag(C)*B
```

```
        Whether or not the system will be equilibrated depends on the
        scaling of the matrix A, but if equilibration is used, A' is
        overwritten by diag(R)*A'*diag(C) and B by diag(R)*B
        (if trans='N') or diag(C)*B (if trans = 'T' or 'C').

   2.2. Permute columns of transpose(A) (rows of A),
        forming transpose(A)*Pc, where Pc is a permutation matrix that
        usually preserves sparsity.
        For more details of this step, see sp_preorder.c.

   2.3. If options->Fact != FACTORED, the LU decomposition is used to
        factor the transpose(A) (after equilibration if
        options->Fact = YES) as Pr*transpose(A)*Pc = L*U with the
        permutation Pr determined by partial pivoting.

   2.4. Compute the reciprocal pivot growth factor.

   2.5. If some U(i,i) = 0, so that U is exactly singular, then the
        routine returns with info = i. Otherwise, the factored form
        of transpose(A) is used to estimate the condition number of the
        matrix A. If the reciprocal of the condition number
        is less than machine precision, info = A->nrow+1 is returned as
        a warning, but the routine still goes on to solve for X and
        computes error bounds as described below.

   2.6. The system of equations is solved for X using the factored form
        of transpose(A).

   2.7. If options->IterRefine != NOREFINE, iterative refinement is
        applied to improve the computed solution matrix and calculate
        error bounds and backward error estimates for it.

   2.8. If equilibration was used, the matrix X is premultiplied by
        diag(C) (if options->Trans = NOTRANS) or diag(R)
        (if options->Trans = TRANS or CONJ) so that it solves the
        original system before equilibration.

  See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

options (input) superlu_options_t*
        The structure defines the input parameters to control
        how the LU decomposition will be performed and how the
        system will be solved.

A       (input/output) SuperMatrix*
        Matrix A in A*X=B, of dimension (A->nrow, A->ncol). The number
        of the linear equations is A->nrow. Currently, the type of A can be:
        Stype = SLU_NC or SLU_NR, Dtype = SLU_D, Mtype = SLU_GE.
        In the future, more general A may be handled.
```

```
        On entry, If options->Fact = FACTORED and equed is not 'N',
        then A must have been equilibrated by the scaling factors in
        R and/or C.
        On exit, A is not modified if options->Equil = NO, or if
        options->Equil = YES but equed = 'N' on exit.
        Otherwise, if options->Equil = YES and equed is not 'N',
        A is scaled as follows:
        If A->Stype = SLU_NC:
          equed = 'R':  A := diag(R) * A
          equed = 'C':  A := A * diag(C)
          equed = 'B':  A := diag(R) * A * diag(C).
        If A->Stype = SLU_NR:
          equed = 'R':  transpose(A) := diag(R) * transpose(A)
          equed = 'C':  transpose(A) := transpose(A) * diag(C)
          equed = 'B':  transpose(A) := diag(R) * transpose(A) * diag(C).

perm_c  (input/output) int*
  If A->Stype = SLU_NC, Column permutation vector of size A->ncol,
        which defines the permutation matrix Pc; perm_c[i] = j means
        column i of A is in position j in A*Pc.
        On exit, perm_c may be overwritten by the product of the input
        perm_c and a permutation that postorders the elimination tree
        of Pc'*A'*A*Pc; perm_c is not changed if the elimination tree
        is already in postorder.

        If A->Stype = SLU_NR, column permutation vector of size A->nrow,
        which describes permutation of columns of transpose(A)
        (rows of A) as described above.

perm_r  (input/output) int*
        If A->Stype = SLU_NC, row permutation vector of size A->nrow,
        which defines the permutation matrix Pr, and is determined
        by partial pivoting.  perm_r[i] = j means row i of A is in
        position j in Pr*A.

        If A->Stype = SLU_NR, permutation vector of size A->ncol, which
        determines permutation of rows of transpose(A)
        (columns of A) as described above.

        If options->Fact = SamePattern_SameRowPerm, the pivoting routine
        will try to use the input perm_r, unless a certain threshold
        criterion is violated. In that case, perm_r is overwritten by a
        new permutation determined by partial pivoting or diagonal
        threshold pivoting.
        Otherwise, perm_r is output argument.

etree   (input/output) int*,  dimension (A->ncol)
        Elimination tree of Pc'*A'*A*Pc.
        If options->Fact != FACTORED and options->Fact != DOFACT,
        etree is an input argument, otherwise it is an output argument.
        Note: etree is a vector of parent pointers for a forest whose
        vertices are the integers 0 to A->ncol-1; etree[root]==A->ncol.

equed   (input/output) char*
        Specifies the form of equilibration that was done.
        = 'N': No equilibration.
```

```
        = 'R': Row equilibration, i.e., A was premultiplied by diag(R).
        = 'C': Column equilibration, i.e., A was postmultiplied by diag(C).
        = 'B': Both row and column equilibration, i.e., A was replaced
               by diag(R)*A*diag(C).
        If options->Fact = FACTORED, equed is an input argument,
        otherwise it is an output argument.

R       (input/output) double*, dimension (A->nrow)
        The row scale factors for A or transpose(A).
        If equed = 'R' or 'B', A (if A->Stype = SLU_NC) or transpose(A)
            (if A->Stype = SLU_NR) is multiplied on the left by diag(R).
        If equed = 'N' or 'C', R is not accessed.
        If options->Fact = FACTORED, R is an input argument,
            otherwise, R is output.
        If options->zFact = FACTORED and equed = 'R' or 'B', each element
            of R must be positive.

C       (input/output) double*, dimension (A->ncol)
        The column scale factors for A or transpose(A).
        If equed = 'C' or 'B', A (if A->Stype = SLU_NC) or transpose(A)
            (if A->Stype = SLU_NR) is multiplied on the right by diag(C).
        If equed = 'N' or 'R', C is not accessed.
        If options->Fact = FACTORED, C is an input argument,
            otherwise, C is output.
        If options->Fact = FACTORED and equed = 'C' or 'B', each element
            of C must be positive.

L       (output) SuperMatrix*
  The factor L from the factorization
            Pr*A*Pc=L*U              (if A->Stype SLU_= NC) or
            Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
        Uses compressed row subscripts storage for supernodes, i.e.,
        L has types: Stype = SLU_SC, Dtype = SLU_Z, Mtype = SLU_TRLU.

U       (output) SuperMatrix*
  The factor U from the factorization
            Pr*A*Pc=L*U              (if A->Stype = SLU_NC) or
            Pr*transpose(A)*Pc=L*U   (if A->Stype = SLU_NR).
        Uses column-wise storage scheme, i.e., U has types:
        Stype = SLU_NC, Dtype = SLU_Z, Mtype = SLU_TRU.

work    (workspace/output) void*, size (lwork) (in bytes)
        User supplied workspace, should be large enough
        to hold data structures for factors L and U.
        On exit, if fact is not 'F', L and U point to this array.

lwork   (input) int
        Specifies the size of work array in bytes.
        = 0:  allocate space internally by system malloc;
        > 0:  use user-supplied work array of length lwork in bytes,
              returns error if space runs out.
        = -1: the routine guesses the amount of space needed without
              performing the factorization, and returns it in
              mem_usage->total_needed; no other side effects.

        See argument 'mem_usage' for memory usage statistics.
```

```
B       (input/output) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_Z, Mtype = SLU_GE.
        On entry, the right hand side matrix.
        If B->ncol = 0, only LU decomposition is performed, the triangular
                        solve is skipped.
        On exit,
           if equed = 'N', B is not modified; otherwise
           if A->Stype = SLU_NC:
              if options->Trans = NOTRANS and equed = 'R' or 'B',
                 B is overwritten by diag(R)*B;
              if options->Trans = TRANS or CONJ and equed = 'C' of 'B',
                 B is overwritten by diag(C)*B;
           if A->Stype = SLU_NR:
              if options->Trans = NOTRANS and equed = 'C' or 'B',
                 B is overwritten by diag(C)*B;
              if options->Trans = TRANS or CONJ and equed = 'R' of 'B',
                 B is overwritten by diag(R)*B.

X       (output) SuperMatrix*
        X has types: Stype = SLU_DN, Dtype = SLU_Z, Mtype = SLU_GE.
        If info = 0 or info = A->ncol+1, X contains the solution matrix
        to the original system of equations. Note that A and B are modified
        on exit if equed is not 'N', and the solution to the equilibrated
        system is inv(diag(C))*X if options->Trans = NOTRANS and
        equed = 'C' or 'B', or inv(diag(R))*X if options->Trans = 'T' or 'C'
        and equed = 'R' or 'B'.

recip_pivot_growth (output) double*
        The reciprocal pivot growth factor max_j( norm(A_j)/norm(U_j) ).
        The infinity norm is used. If recip_pivot_growth is much less
        than 1, the stability of the LU factorization could be poor.

rcond   (output) double*
        The estimate of the reciprocal condition number of the matrix A
        after equilibration (if done). If rcond is less than the machine
        precision (in particular, if rcond = 0), the matrix is singular
        to working precision. This condition is indicated by a return
        code of info > 0.

FERR    (output) double*, dimension (B->ncol)
        The estimated forward error bound for each solution vector
        X(j) (the j-th column of the solution matrix X).
        If XTRUE is the true solution corresponding to X(j), FERR(j)
        is an estimated upper bound for the magnitude of the largest
        element in (X(j) - XTRUE) divided by the magnitude of the
        largest element in X(j).  The estimate is as reliable as
        the estimate for RCOND, and is almost always a slight
        overestimate of the true error.
        If options->IterRefine = NOREFINE, ferr = 1.0.

BERR    (output) double*, dimension (B->ncol)
        The componentwise relative backward error of each solution
        vector X(j) (i.e., the smallest relative change in
        any element of A or B that makes X(j) an exact solution).
        If options->IterRefine = NOREFINE, berr = 1.0.

mem_usage (output) mem_usage_t*
```

```
Record the memory usage statistics, consisting of following fields:
```

- for_lu (float)

```
The amount of space used in bytes for L data structures.
```

- total_needed (float)

```
The amount of space needed in bytes to perform factorization.
```

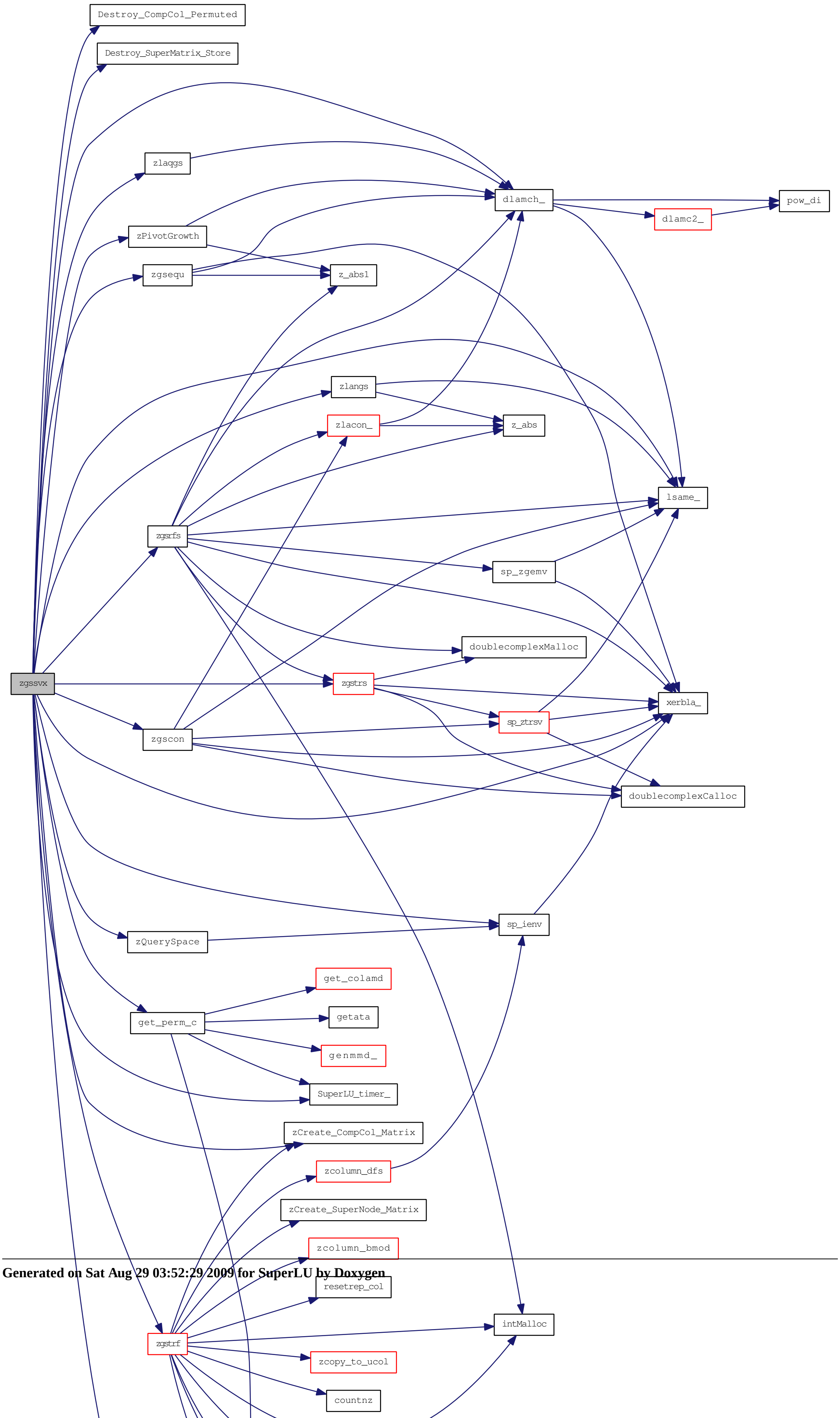
- expansions (int)

```
The number of memory expansions during the LU factorization.
```

```
stat   (output) SuperLUStat_t*
       Record the statistics on runtime and floating-point operation count.
       See util.h for the definition of 'SuperLUStat_t'.
```

```
info    (output) int*
        = 0: successful exit
        < 0: if info = -i, the i-th argument had an illegal value
        > 0: if info = i, and i is
             <= A->ncol: U(i,i) is exactly zero. The factorization has
                   been completed, but the factor U is exactly
                   singular, so the solution and error bounds
                   could not be computed.
             = A->ncol+1: U is nonsingular, but RCOND is less than machine
                   precision, meaning that the matrix is singular to
                   working precision. Nevertheless, the solution and
                   error bounds are computed because there are a number
                   of situations where the computed solution can be more
                   accurate than the value of RCOND would suggest.
             > A->ncol+1: number of bytes allocated when memory allocation
                   failure occurred, plus A->ncol.
```

Here is the call graph for this function:



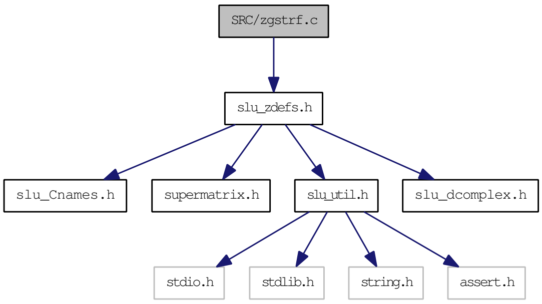
Here is the caller graph for this function:

4.147 SRC/zgstrf.c File Reference

Computes an LU factorization of a general sparse matrix.

`#include "slu_zdefs.h"`

Include dependency graph for zgstrf.c:



Functions

- void zgstrf (superlu_options_t ∗options, SuperMatrix ∗A, double drop_tol, int relax, int panel_size, int ∗etree, void ∗work, int lwork, int ∗perm_c, int ∗perm_r, SuperMatrix ∗L, SuperMatrix ∗U, SuperLUStat_t ∗stat, int ∗info)

4.147.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.147.2 Function Documentation

4.147.2.1 void zgstrf (superlu_options_t ∗ *options*, SuperMatrix ∗ *A*, double *drop_tol*, int *relax*, int *panel_size*, int ∗ *etree*, void ∗ *work*, int *lwork*, int ∗ *perm_c*, int ∗ *perm_r*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======
```

```
ZGSTRF computes an LU factorization of a general sparse m-by-n
matrix A using partial pivoting with row interchanges.
The factorization has the form
    Pr * A = L * U
where Pr is a row permutation matrix, L is lower triangular with unit
diagonal elements (lower trapezoidal if A->nrow > A->ncol), and U is upper
triangular (upper trapezoidal if A->nrow < A->ncol).

See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

options (input) superlu_options_t*
        The structure defines the input parameters to control
        how the LU decomposition will be performed.

A        (input) SuperMatrix*
   Original matrix A, permuted by columns, of dimension
         (A->nrow, A->ncol). The type of A can be:
         Stype = SLU_NCP; Dtype = SLU_Z; Mtype = SLU_GE.

drop_tol (input) double (NOT IMPLEMENTED)
   Drop tolerance parameter. At step j of the Gaussian elimination,
         if abs(A_ij)/(max_i abs(A_ij)) < drop_tol, drop entry A_ij.
         0 <= drop_tol <= 1. The default value of drop_tol is 0.

relax    (input) int
         To control degree of relaxing supernodes. If the number
         of nodes (columns) in a subtree of the elimination tree is less
         than relax, this subtree is considered as one supernode,
         regardless of the row structures of those columns.

panel_size (input) int
         A panel consists of at most panel_size consecutive columns.

etree    (input) int*, dimension (A->ncol)
         Elimination tree of A'*A.
         Note: etree is a vector of parent pointers for a forest whose
         vertices are the integers 0 to A->ncol-1; etree[root]==A->ncol.
         On input, the columns of A should be permuted so that the
         etree is in a certain postorder.

work     (input/output) void*, size (lwork) (in bytes)
         User-supplied work space and space for the output data structures.
         Not referenced if lwork = 0;

lwork   (input) int
        Specifies the size of work array in bytes.
        = 0:  allocate space internally by system malloc;
        > 0:  use user-supplied work array of length lwork in bytes,
              returns error if space runs out.
        = -1: the routine guesses the amount of space needed without
              performing the factorization, and returns it in
              *info; no other side effects.
```

```
perm_c   (input) int*, dimension (A->ncol)
   Column permutation vector, which defines the
         permutation matrix Pc; perm_c[i] = j means column i of A is
         in position j in A*Pc.
         When searching for diagonal, perm_c[*] is applied to the
         row subscripts of A, so that diagonal threshold pivoting
         can find the diagonal of A, rather than that of A*Pc.


perm_r   (input/output) int*, dimension (A->nrow)
         Row permutation vector which defines the permutation matrix Pr,
         perm_r[i] = j means row i of A is in position j in Pr*A.
         If options->Fact = SamePattern_SameRowPerm, the pivoting routine
            will try to use the input perm_r, unless a certain threshold
            criterion is violated. In that case, perm_r is overwritten by
            a new permutation determined by partial pivoting or diagonal
            threshold pivoting.
         Otherwise, perm_r is output argument;


L        (output) SuperMatrix*
         The factor L from the factorization Pr*A=L*U; use compressed row
         subscripts storage for supernodes, i.e., L has type:
         Stype = SLU_SC, Dtype = SLU_Z, Mtype = SLU_TRLU.


U        (output) SuperMatrix*
   The factor U from the factorization Pr*A*Pc=L*U. Use column-wise
         storage scheme, i.e., U has types: Stype = SLU_NC,
         Dtype = SLU_Z, Mtype = SLU_TRU.


stat     (output) SuperLUStat_t*
         Record the statistics on runtime and floating-point operation count.
         See util.h for the definition of 'SuperLUStat_t'.


info     (output) int*
         = 0: successful exit
         < 0: if info = -i, the i-th argument had an illegal value
         > 0: if info = i, and i is
            <= A->ncol: U(i,i) is exactly zero. The factorization has
               been completed, but the factor U is exactly singular,
               and division by zero will occur if it is used to solve a
               system of equations.
            > A->ncol: number of bytes allocated when memory allocation
               failure occurred, plus A->ncol. If lwork = -1, it is
               the estimated amount of space needed, plus A->ncol.


======================================================================


Local Working Arrays:
======================
  m = number of rows in the matrix
  n = number of columns in the matrix


  xprune[0:n-1]: xprune[*] points to locations in subscript
vector lsub[*]. For column i, xprune[i] denotes the point where
structural pruning begins. I.e. only xlsub[i],..,xprune[i]-1 need
to be traversed for symbolic factorization.
```

```
   marker[0:3*m-1]: marker[i] = j means that node i has been
reached when working on column j.
Storage: relative to original row subscripts
NOTE: There are 3 of them: marker/marker1 are used for panel dfs,
      see zpanel_dfs.c; marker2 is used for inner-factorization,
            see zcolumn_dfs.c.
```

```
   parent[0:m-1]: parent vector used during dfs
      Storage: relative to new row subscripts
```

```
   xplore[0:m-1]: xplore[i] gives the location of the next (dfs)
unexplored neighbor of i in lsub[*]
```

```
   segrep[0:nseg-1]: contains the list of supernodal representatives
in topological order of the dfs. A supernode representative is the
last column of a supernode.
      The maximum size of segrep[] is n.
```

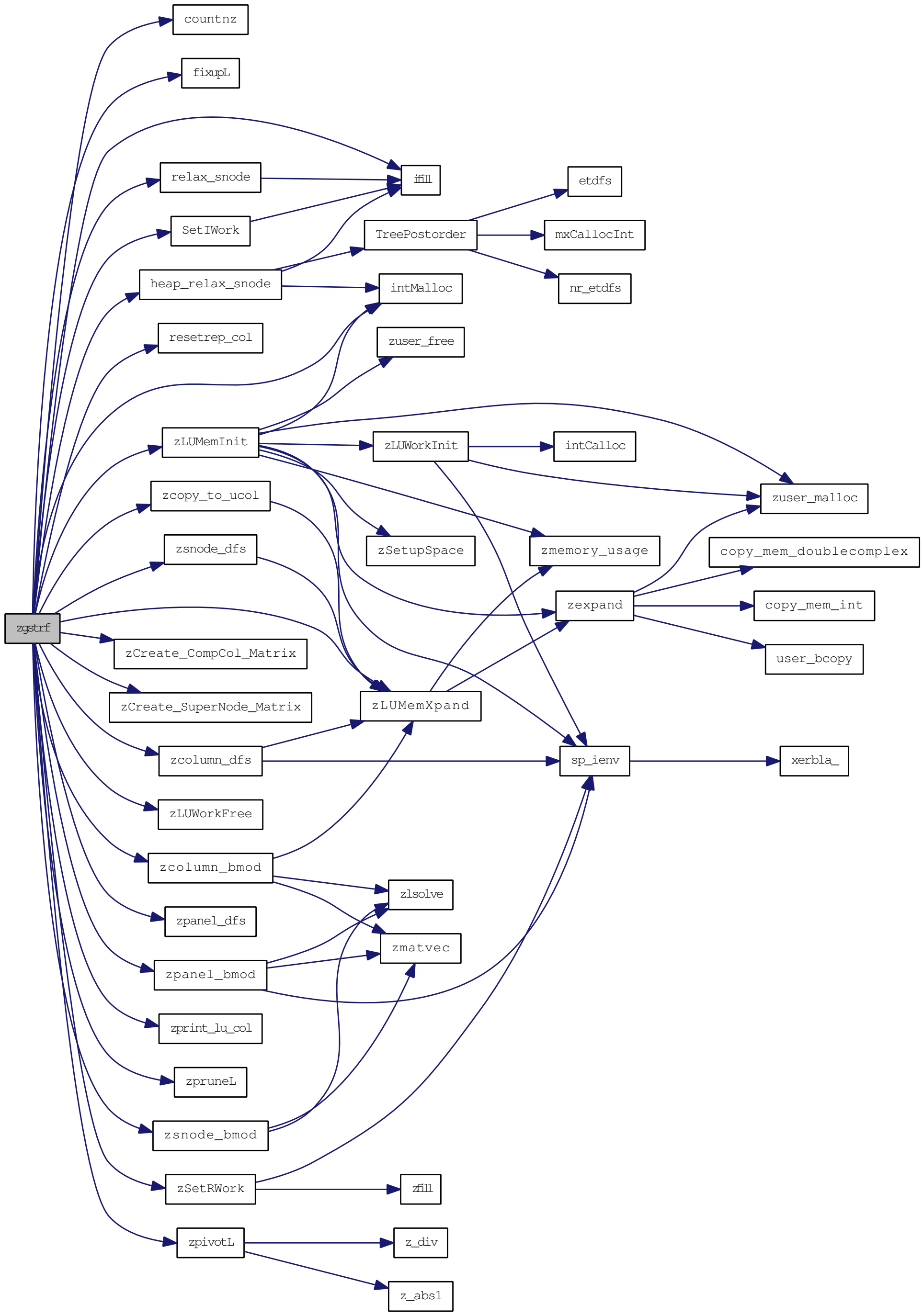
```
   repfnz[0:W*m-1]: for a nonzero segment U[*,j] that ends at a
supernodal representative r, repfnz[r] is the location of the first
nonzero in this segment.  It is also used during the dfs: repfnz[r]>0
indicates the supernode r has been explored.
NOTE: There are W of them, each used for one column of a panel.
```

```
   panel_lsub[0:W*m-1]: temporary for the nonzeros row indices below
      the panel diagonal. These are filled in during zpanel_dfs(), and are
      used later in the inner LU factorization within the panel.
panel_lsub[]/dense[] pair forms the SPA data structure.
NOTE: There are W of them.
```

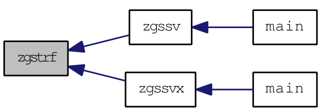
```
   dense[0:W*m-1]: sparse accumulating (SPA) vector for intermediate values;
        NOTE: there are W of them.
```

```
   tempv[0:*]: real temporary used for dense numeric kernels;
The size of this array is defined by NUM_TEMPV() in slu_zdefs.h.
```

Here is the call graph for this function:



Here is the caller graph for this function:

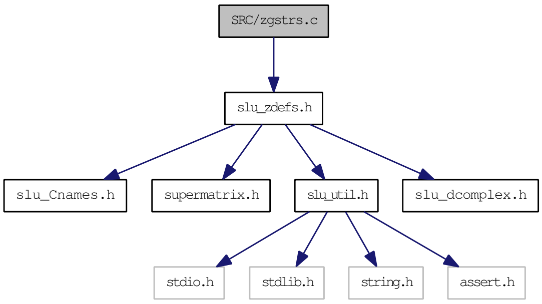


4.148 SRC/zgstrs.c File Reference

Solves a system using LU factorization.

`#include "slu_zdefs.h"`

Include dependency graph for zgstrs.c:



Functions

- void zusolve (int, int, doublecomplex ∗, doublecomplex ∗)

  Solves a dense upper triangular system.

- void zlsolve (int, int, doublecomplex ∗, doublecomplex ∗)

  Solves a dense UNIT lower triangular system.

- void zmatvec (int, int, int, doublecomplex ∗, doublecomplex ∗, doublecomplex ∗)

  Performs a dense matrix-vector multiply: Mxvec = Mxvec + M ∗ vec.

- void zgstrs (trans_t trans, SuperMatrix ∗L, SuperMatrix ∗U, int ∗perm_c, int ∗perm_r, SuperMatrix ∗B, SuperLUStat_t ∗stat, int ∗info)
- void zprint_soln (int n, int nrhs, doublecomplex ∗soln)

4.148.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.148.2 Function Documentation

4.148.2.1 void zgstrs (trans_t *trans*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, int ∗ *perm_c*, int ∗ *perm_r*, SuperMatrix ∗ *B*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======

ZGSTRS solves a system of linear equations A*X=B or A'*X=B
with A sparse and B dense, using the LU factorization computed by
ZGSTRF.

See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

trans   (input) trans_t
         Specifies the form of the system of equations:
         = NOTRANS: A * X = B  (No transpose)
         = TRANS:   A'* X = B  (Transpose)
         = CONJ:    A**H * X = B  (Conjugate transpose)

L       (input) SuperMatrix*
        The factor L from the factorization Pr*A*Pc=L*U as computed by
        zgstrf(). Use compressed row subscripts storage for supernodes,
        i.e., L has types: Stype = SLU_SC, Dtype = SLU_Z, Mtype = SLU_TRLU.

U       (input) SuperMatrix*
        The factor U from the factorization Pr*A*Pc=L*U as computed by
        zgstrf(). Use column-wise storage scheme, i.e., U has types:
        Stype = SLU_NC, Dtype = SLU_Z, Mtype = SLU_TRU.

perm_c  (input) int*, dimension (L->ncol)
  Column permutation vector, which defines the
        permutation matrix Pc; perm_c[i] = j means column i of A is
        in position j in A*Pc.

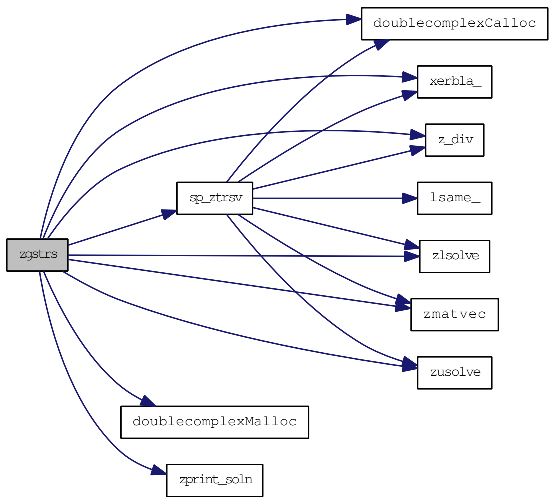
perm_r  (input) int*, dimension (L->nrow)
        Row permutation vector, which defines the permutation matrix Pr;
        perm_r[i] = j means row i of A is in position j in Pr*A.

B       (input/output) SuperMatrix*
        B has types: Stype = SLU_DN, Dtype = SLU_Z, Mtype = SLU_GE.
        On entry, the right hand side matrix.
        On exit, the solution matrix if info = 0;

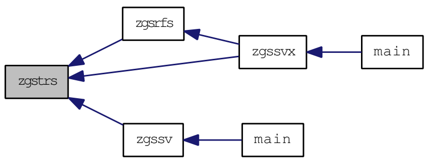
stat     (output) SuperLUStat_t*
         Record the statistics on runtime and floating-point operation count.
         See util.h for the definition of 'SuperLUStat_t'.

info    (output) int*
    = 0: successful exit
  < 0: if info = -i, the i-th argument had an illegal value
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.148.2.2 void zlsolve (int *ldm*, int *ncol*, doublecomplex $*$ *M*, doublecomplex $*$ *rhs*)

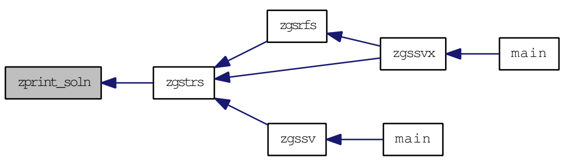
The unit lower triangular matrix is stored in a 2D array M(1:nrow,1:ncol). The solution will be returned in the rhs vector.

4.148.2.3 void zmatvec (int *ldm*, int *nrow*, int *ncol*, doublecomplex $*$ *M*, doublecomplex $*$ *vec*, doublecomplex $*$ *Mxvec*)

The input matrix is M(1:nrow,1:ncol); The product is returned in Mxvec[].

4.148.2.4 void zprint_soln (int *n*, int *nrhs*, doublecomplex $*$ *soln*)

Here is the caller graph for this function:



4.148.2.5 void zusolve (int *ldm*, int *ncol*, doublecomplex $*$ *M*, doublecomplex $*$ *rhs*)

The upper triangular matrix is stored in a 2-dim array M(1:ldm,1:ncol). The solution will be returned in the rhs vector.

4.149 SRC/zlacon.c File Reference

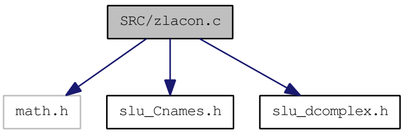
Estimates the 1-norm.

`#include <math.h>`

`#include "slu_Cnames.h"`

`#include "slu_dcomplex.h"`

Include dependency graph for zlacon.c:



Functions

- int zlacon_ (int ∗n, doublecomplex ∗v, doublecomplex ∗x, double ∗est, int ∗kase)

4.149.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997
```

4.149.2 Function Documentation

4.149.2.1 int zlacon_ (int ∗ *n*, doublecomplex ∗ *v*, doublecomplex ∗ *x*, double ∗ *est*, int ∗ *kase*)

```
Purpose
=======

ZLACON estimates the 1-norm of a square matrix A.
Reverse communication is used for evaluating matrix-vector products.

Arguments
=========

N       (input) INT
        The order of the matrix.  N >= 1.

V       (workspace) DOUBLE COMPLEX PRECISION array, dimension (N)
        On the final return, V = A*W,  where  EST = norm(V)/norm(W)
        (W is not returned).

X       (input/output) DOUBLE COMPLEX PRECISION array, dimension (N)
        On an intermediate return, X should be overwritten by
```

```
                A * X,    if KASE=1,
                A' * X,   if KASE=2,
          where A' is the conjugate transpose of A,
         and ZLACON must be re-called with all the other parameters
          unchanged.

   EST     (output) DOUBLE PRECISION
           An estimate (a lower bound) for norm(A).

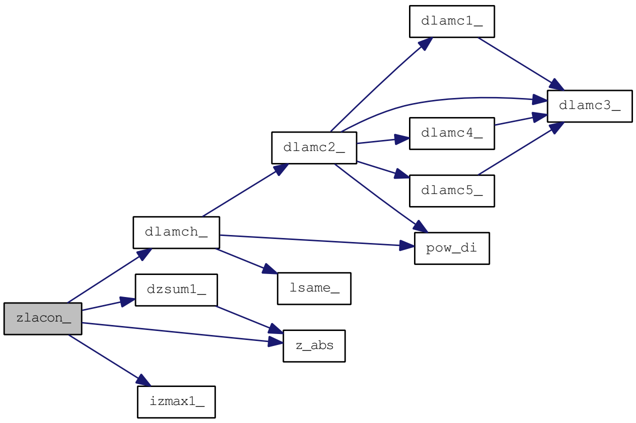
   KASE    (input/output) INT
           On the initial call to ZLACON, KASE should be 0.
           On an intermediate return, KASE will be 1 or 2, indicating
           whether X should be overwritten by A * X  or A' * X.
           On the final return from ZLACON, KASE will again be 0.

   Further Details
   ======= =======

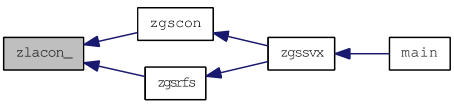
   Contributed by Nick Higham, University of Manchester.
   Originally named CONEST, dated March 16, 1988.

   Reference: N.J. Higham, "FORTRAN codes for estimating the one-norm of
   a real or complex matrix, with applications to condition estimation",
   ACM Trans. Math. Soft., vol. 14, no. 4, pp. 381-396, December 1988.
   ====================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:



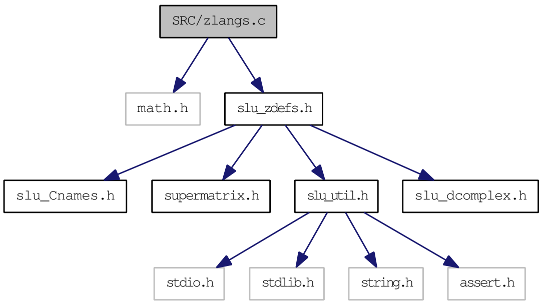
4.150 SRC/zlangs.c File Reference

Returns the value of the one norm.

`#include <math.h>`

`#include "slu_zdefs.h"`

Include dependency graph for zlangs.c:



Functions

- double zlangs (char ∗norm, SuperMatrix ∗A)

4.150.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997


Modified from lapack routine ZLANGE
```

4.150.2 Function Documentation

4.150.2.1 double zlangs (char ∗ *norm*, SuperMatrix ∗ *A*)

```
Purpose
  =======


  ZLANGS returns the value of the one norm, or the Frobenius norm, or
  the infinity norm, or the element of largest absolute value of a
  real matrix A.


  Description
  ===========


  ZLANGE returns the value
```

```
    ZLANGE = ( max(abs(A(i,j))), NORM = 'M' or 'm'
             (
             ( norm1(A),         NORM = '1', 'O' or 'o'
             (
             ( normI(A),         NORM = 'I' or 'i'
             (
             ( normF(A),         NORM = 'F', 'f', 'E' or 'e'

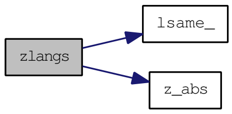
 where  norm1  denotes the  one norm of a matrix (maximum column sum),
 normI  denotes the  infinity norm  of a matrix  (maximum row sum) and
 normF  denotes the  Frobenius norm of a matrix (square root of sum of
 squares).  Note that  max(abs(A(i,j)))  is not a  matrix norm.

 Arguments
 =========

 NORM    (input) CHARACTER*1
         Specifies the value to be returned in ZLANGE as described above.
 A       (input) SuperMatrix*
         The M by N sparse matrix A.

=====================================================================
```

Here is the call graph for this function:



Here is the caller graph for this function:

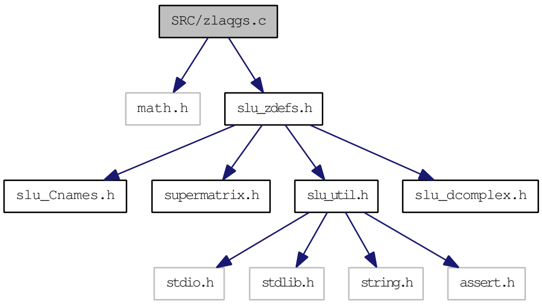
4.151 SRC/zlaqgs.c File Reference

Equlibrates a general sprase matrix.

`#include <math.h>`

`#include "slu_zdefs.h"`

Include dependency graph for zlaqgs.c:



Defines

- #define THRESH (0.1)

Functions

- void zlaqgs (SuperMatrix ∗A, double ∗r, double ∗c, double rowcnd, double colcnd, double amax, char ∗equed)

4.151.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997

Modified from LAPACK routine ZLAQGE
```

4.151.2 Define Documentation

4.151.2.1 #define THRESH (0.1)

4.151.3 Function Documentation

4.151.3.1 void zlaqgs (SuperMatrix ∗ *A*, double ∗ *r*, double ∗ *c*, double *rowcnd*, double *colcnd*, double *amax*, char ∗ *equed*)

```
Purpose
=======
```

```
ZLAQGS equilibrates a general sparse M by N matrix A using the row and
scaling factors in the vectors R and C.

See supermatrix.h for the definition of 'SuperMatrix' structure.

Arguments
=========

A       (input/output) SuperMatrix*
        On exit, the equilibrated matrix.  See EQUED for the form of
        the equilibrated matrix. The type of A can be:
 Stype = NC; Dtype = SLU_Z; Mtype = GE.

R       (input) double*, dimension (A->nrow)
        The row scale factors for A.

C       (input) double*, dimension (A->ncol)
        The column scale factors for A.

ROWCND  (input) double
        Ratio of the smallest R(i) to the largest R(i).

COLCND  (input) double
        Ratio of the smallest C(i) to the largest C(i).

AMAX    (input) double
        Absolute value of largest matrix entry.

EQUED   (output) char*
        Specifies the form of equilibration that was done.
        = 'N':  No equilibration
        = 'R':  Row equilibration, i.e., A has been premultiplied by
                diag(R).
        = 'C':  Column equilibration, i.e., A has been postmultiplied
                by diag(C).
        = 'B':  Both row and column equilibration, i.e., A has been
                replaced by diag(R) * A * diag(C).

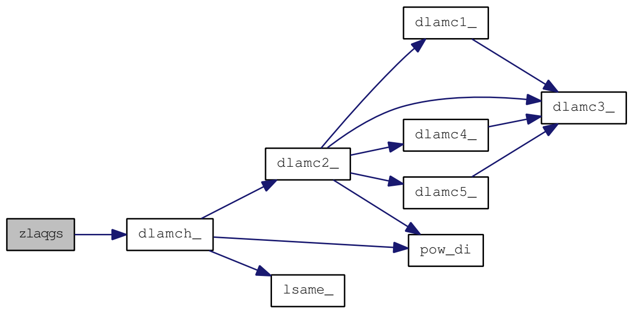
Internal Parameters
===================

THRESH is a threshold value used to decide if row or column scaling
should be done based on the ratio of the row or column scaling
factors.  If ROWCND < THRESH, row scaling is done, and if
COLCND < THRESH, column scaling is done.

LARGE and SMALL are threshold values used to decide if row scaling
should be done based on the absolute size of the largest matrix
element.  If AMAX > LARGE or AMAX < SMALL, row scaling is done.

=====================================================================
```

Here is the call graph for this function:



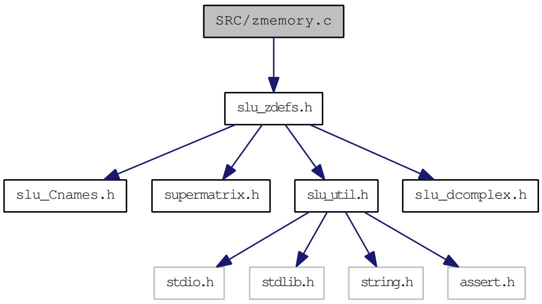
Here is the caller graph for this function:

4.152 SRC/zmemory.c File Reference

Memory details.

`#include "slu_zdefs.h"`

Include dependency graph for zmemory.c:



Data Structures

- struct e_node

  Headers for 4 types of dynamatically managed memory.

- struct LU_stack_t

Defines

- #define NO_MEMTYPE 4
- #define GluIntArray(n) (5 ∗ (n) + 5)
- #define StackFull(x) (x + stack.used >= stack.size)
- #define NotDoubleAlign(addr) ((long int)addr & 7)
- #define DoubleAlign(addr) (((long int)addr + 7) & ∼7L)
- #define TempSpace(m, w)
- #define Reduce(alpha) ((alpha + 1) / 2)

Typedefs

- typedef struct e_node ExpHeader

  Headers for 4 types of dynamatically managed memory.

Functions

- void ∗ zexpand (int ∗prev_len,MemType type,int len_to_copy,int keep_prev,GlobalLU_t ∗Glu)

  Expand the existing storage to accommodate more fill-ins.

- int zLUWorkInit (int m, int n, int panel_size, int ∗∗iworkptr, doublecomplex ∗∗dworkptr, LU_space_t MemModel)

Allocate known working storage. Returns 0 if success, otherwise returns the number of bytes allocated so far when failure occurred.

- void copy_mem_doublecomplex (int, void ∗, void ∗)
- void zStackCompress (GlobalLU_t ∗Glu)

  Compress the work[] array to remove fragmentation.

- void zSetupSpace (void ∗work, int lwork, LU_space_t ∗MemModel)

  Setup the memory model to be used for factorization.

- void ∗ zuser_malloc (int, int)
- void zuser_free (int, int)
- void copy_mem_int (int, void ∗, void ∗)
- void user_bcopy (char ∗, char ∗, int)
- int zQuerySpace (SuperMatrix ∗L, SuperMatrix ∗U, mem_usage_t ∗mem_usage)
- int zLUMemInit (fact_t fact, void ∗work, int lwork, int m, int n, int annz, int panel_size, SuperMatrix ∗L, SuperMatrix ∗U, GlobalLU_t ∗Glu, int ∗∗iwork, doublecomplex ∗∗dwork)

  Allocate storage for the data structures common to all factor routines.

- void zSetRWork (int m, int panel_size, doublecomplex ∗dworkptr, doublecomplex ∗∗dense, doublecomplex ∗∗tempv)

  Set up pointers for real working arrays.

- void zLUWorkFree (int ∗iwork, doublecomplex ∗dwork, GlobalLU_t ∗Glu)

  Free the working storage used by factor routines.

- int zLUMemXpand (int jcol, int next, MemType mem_type, int ∗maxlen, GlobalLU_t ∗Glu)

  Expand the data structures for L and U during the factorization.

- void zallocateA (int n, int nnz, doublecomplex ∗∗a, int ∗∗asub, int ∗∗xa)

  Allocate storage for original matrix A.

- doublecomplex ∗ doublecomplexMalloc (int n)
- doublecomplex ∗ doublecomplexCalloc (int n)
- int zmemory_usage (const int nzlmax, const int nzumax, const int nzlumax, const int n)

Variables

- static ExpHeader ∗ expanders = 0
- static LU_stack_t stack
- static int no_expand

4.152.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003
```

4.152.2 Define Documentation

4.152.2.1 #define DoubleAlign(addr) (((long int)addr + 7) & ∼7L)

4.152.2.2 #define GluIntArray(n) (5 ∗ (n) + 5)

4.152.2.3 #define NO_MEMTYPE 4

4.152.2.4 #define NotDoubleAlign(addr) ((long int)addr & 7)

4.152.2.5 #define Reduce(alpha) ((alpha + 1) / 2)

4.152.2.6 #define StackFull(x) (x + stack.used >= stack.size)

4.152.2.7 #define TempSpace(m, w)

Value:

```
( (2*w + 4 + NO_MARKER) * m * sizeof(int) + \
            (w + 1) * m * sizeof(doublecomplex) )
```

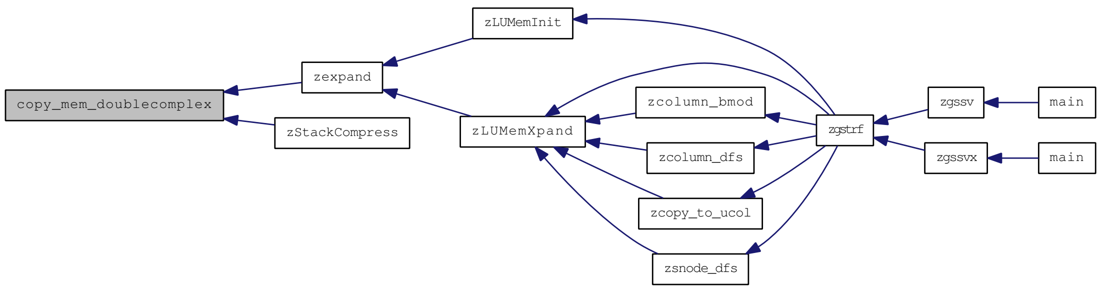
4.152.3 Typedef Documentation

4.152.3.1 typedef struct e_node ExpHeader

4.152.4 Function Documentation

4.152.4.1 void copy_mem_doublecomplex (int *howmany*, void ∗ *old*, void ∗ *new*)

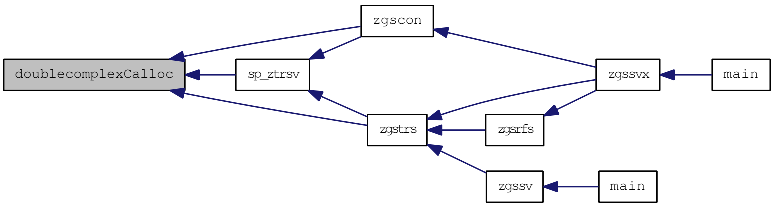
Here is the caller graph for this function:



4.152.4.2 void copy_mem_int (int, void ∗, void ∗)

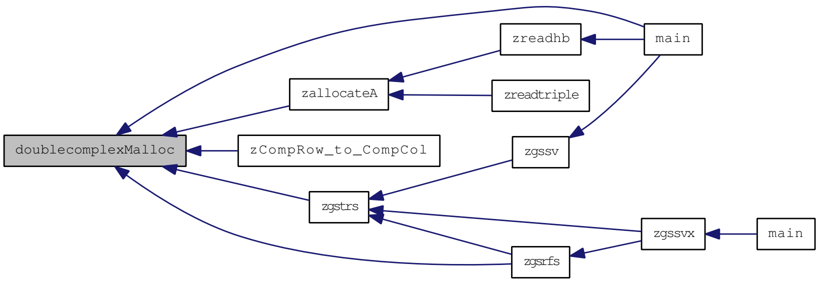
4.152.4.3 doublecomplex∗ doublecomplexCalloc (int *n*)

Here is the caller graph for this function:



4.152.4.4 doublecomplex∗ doublecomplexMalloc (int *n*)

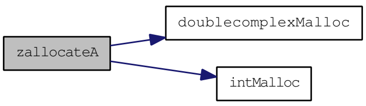
Here is the caller graph for this function:



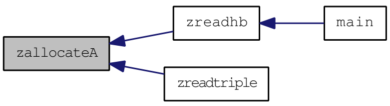
4.152.4.5 void user_bcopy (char ∗, char ∗, int)

4.152.4.6 void zallocateA (int *n*, int *nnz*, doublecomplex ∗∗ *a*, int ∗∗ *asub*, int ∗∗ *xa*)

Here is the call graph for this function:

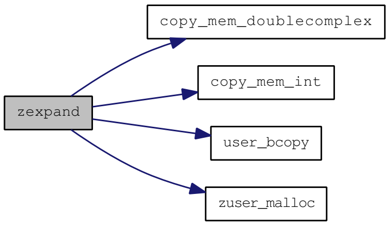


Here is the caller graph for this function:

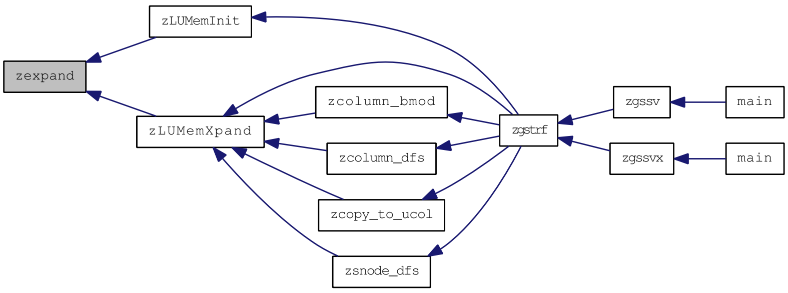


4.152.4.7 void ∗ zexpand (int ∗ *prev_len*, MemType *type*, int *len_to_copy*, int *keep_prev*, GlobalLU_t ∗ *Glu*)

Here is the call graph for this function:



Here is the caller graph for this function:

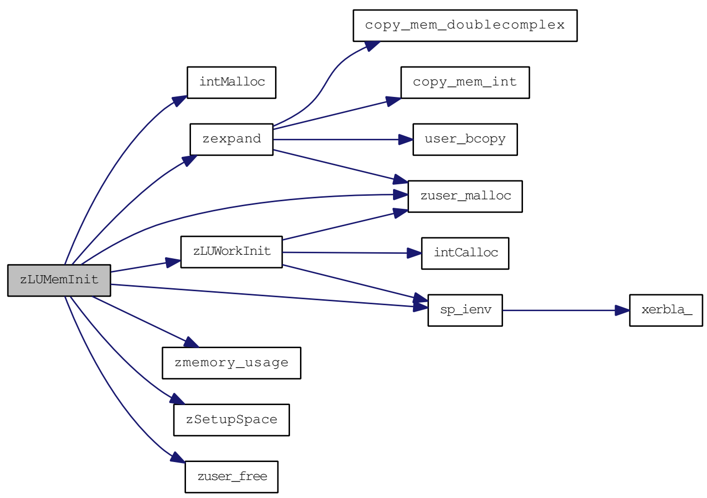


4.152.4.8 int zLUMemInit (fact_t *fact*, void ∗ *work*, int *lwork*, int *m*, int *n*, int *annz*, int *panel_size*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, GlobalLU_t ∗ *Glu*, int ∗∗ *iwork*, doublecomplex ∗∗ *dwork*)

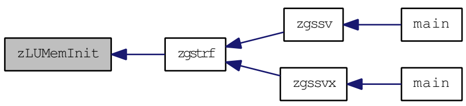
Memory-related.

```
For those unpredictable size, make a guess as FILL * nnz(A).
Return value:
    If lwork = -1, return the estimated amount of space required, plus n;
    otherwise, return the amount of space actually allocated when
    memory allocation failure occurred.
```

Here is the call graph for this function:



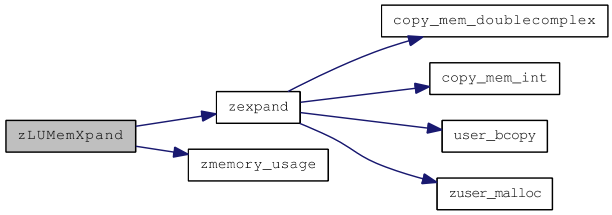
Here is the caller graph for this function:



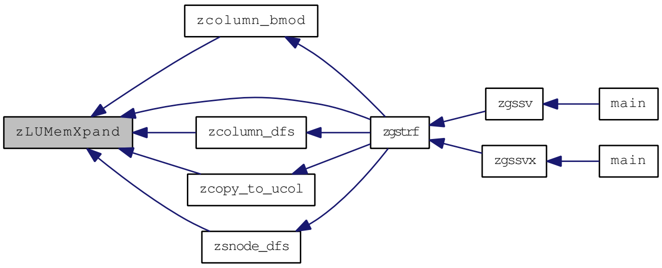
4.152.4.9 int zLUMemXpand (int *jcol*, int *next*, MemType *mem_type*, int ∗ *maxlen*, GlobalLU_t ∗ *Glu*)

```
Return value:   0 - successful return
              > 0 - number of bytes allocated when run out of space
```

Here is the call graph for this function:

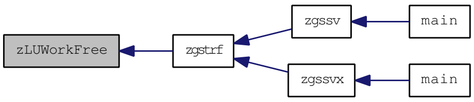


Here is the caller graph for this function:



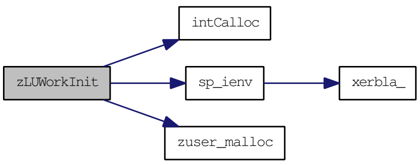
4.152.4.10 void zLUWorkFree (int ∗ *iwork*, doublecomplex ∗ *dwork*, GlobalLU_t ∗ *Glu*)

Here is the caller graph for this function:

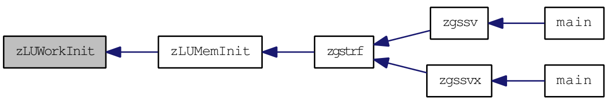


4.152.4.11 int zLUWorkInit (int *m*, int *n*, int *panel_size*, int ∗∗ *iworkptr*, doublecomplex ∗∗ *dworkptr*, LU_space_t *MemModel*)

Here is the call graph for this function:

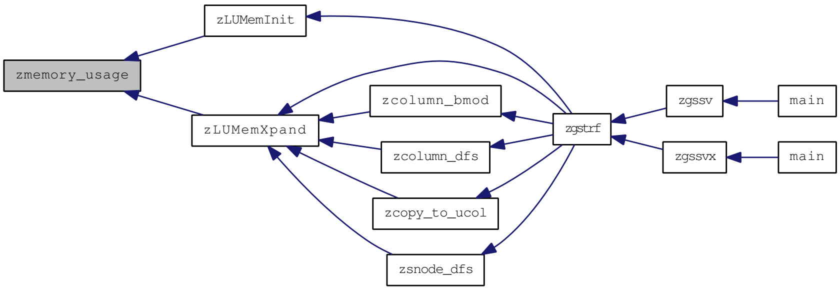


Here is the caller graph for this function:



4.152.4.12 int zmemory_usage (const int *nzlmax*, const int *nzumax*, const int *nzlumax*, const int *n*)

Here is the caller graph for this function:

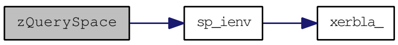


4.152.4.13 int zQuerySpace (SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, mem_usage_t ∗ *mem_usage*)

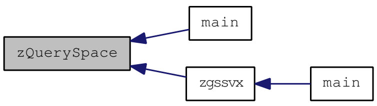
```
mem_usage consists of the following fields:
```

- `for_lu (float)`
  `The amount of space used in bytes for the L data structures.`
- `total_needed (float)`
  `The amount of space needed in bytes to perform factorization.`
- `expansions (int)`
  `Number of memory expansions during the LU factorization.`

Here is the call graph for this function:

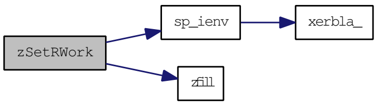


Here is the caller graph for this function:

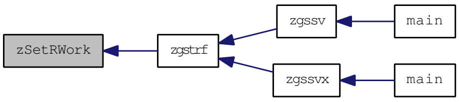


4.152.4.14 void zSetRWork (int *m*, int *panel_size*, doublecomplex ∗ *dworkptr*, doublecomplex ∗∗ *dense*, doublecomplex ∗∗ *tempv*)

Here is the call graph for this function:



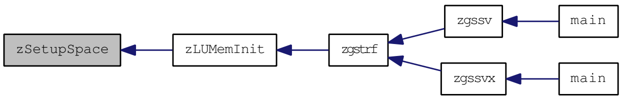
Here is the caller graph for this function:



4.152.4.15 void zSetupSpace (void ∗ *work*, int *lwork*, LU_space_t ∗ *MemModel*)

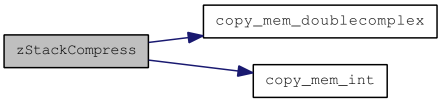
lwork = 0: use system malloc; lwork > 0: use user-supplied work[] space.

Here is the caller graph for this function:



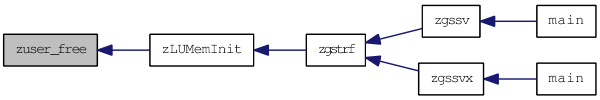
4.152.4.16 void zStackCompress (GlobalLU_t ∗ *Glu*)

Here is the call graph for this function:



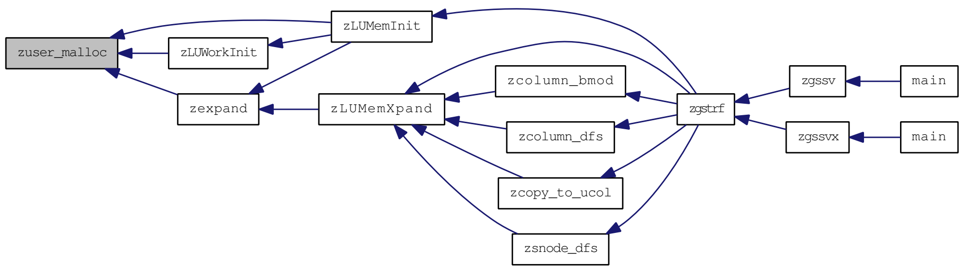
4.152.4.17 void zuser_free (int *bytes*, int *which_end*)

Here is the caller graph for this function:



4.152.4.18 void * zuser_malloc (int *bytes*, int *which_end*)

Here is the caller graph for this function:



4.152.5 Variable Documentation

4.152.5.1 ExpHeader* expanders = 0 [static]

4.152.5.2 int no_expand [static]

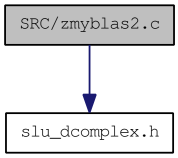
4.152.5.3 LU_stack_t stack [static]

4.153 SRC/zmyblas2.c File Reference

Level 2 Blas operations.

```
#include "slu_dcomplex.h"
```

Include dependency graph for zmyblas2.c:



Functions

- void zlsolve (int ldm, int ncol, doublecomplex ∗M, doublecomplex ∗rhs)

  Solves a dense UNIT lower triangular system.

- void zusolve (int ldm, int ncol, doublecomplex ∗M, doublecomplex ∗rhs)

  Solves a dense upper triangular system.

- void zmatvec (int ldm, int nrow, int ncol, doublecomplex ∗M, doublecomplex ∗vec, doublecomplex ∗Mxvec)

  Performs a dense matrix-vector multiply: Mxvec = Mxvec + M ∗ vec.

4.153.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997
```

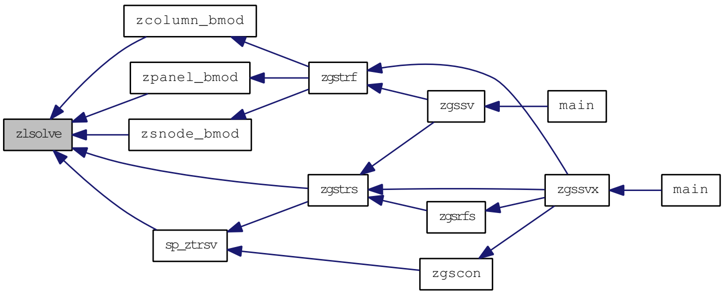
Purpose: Level 2 BLAS operations: solves and matvec, written in C. Note: This is only used when the system lacks an efficient BLAS library.

4.153.2 Function Documentation

4.153.2.1 void zlsolve (int *ldm*, int *ncol*, doublecomplex ∗ *M*, doublecomplex ∗ *rhs*)

The unit lower triangular matrix is stored in a 2D array M(1:nrow,1:ncol). The solution will be returned in the rhs vector.

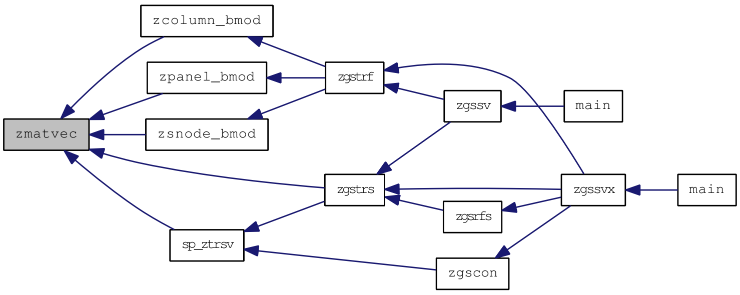
Here is the caller graph for this function:



4.153.2.2 void zmatvec (int *ldm*, int *nrow*, int *ncol*, doublecomplex $*$ *M*, doublecomplex $*$ *vec*, doublecomplex $*$ *Mxvec*)

The input matrix is M(1:nrow,1:ncol); The product is returned in Mxvec[].

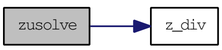
Here is the caller graph for this function:



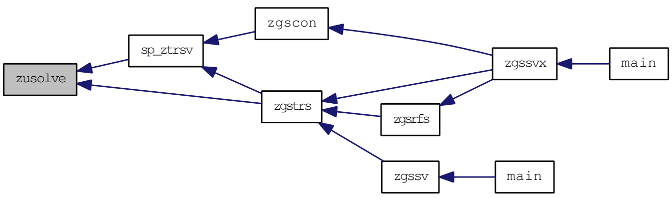
4.153.2.3 void zusolve (int *ldm*, int *ncol*, doublecomplex $*$ *M*, doublecomplex $*$ *rhs*)

The upper triangular matrix is stored in a 2-dim array M(1:ldm,1:ncol). The solution will be returned in the rhs vector.

Here is the call graph for this function:



Here is the caller graph for this function:



4.154 SRC/zpanel_bmod.c File Reference

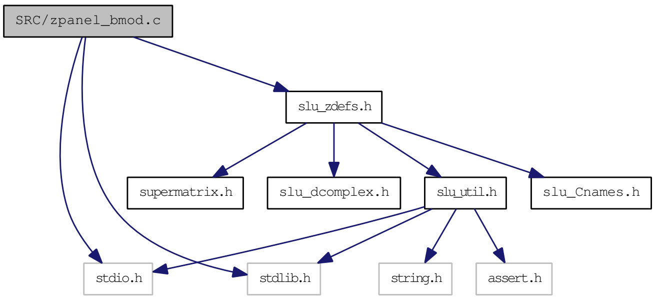
Performs numeric block updates.

`#include <stdio.h>`

`#include <stdlib.h>`

`#include "slu_zdefs.h"`

Include dependency graph for zpanel_bmod.c:



Functions

- void zlsolve (int, int, doublecomplex ∗, doublecomplex ∗)

  Solves a dense UNIT lower triangular system.

- void zmatvec (int, int, int, doublecomplex ∗, doublecomplex ∗, doublecomplex ∗)

  Performs a dense matrix-vector multiply: Mxvec = Mxvec + M ∗ vec.

- void zcheck_tempv ()
- void zpanel_bmod (const int m, const int w, const int jcol, const int nseg, doublecomplex ∗dense, doublecomplex ∗tempv, int ∗segrep, int ∗repfnz, GlobalLU_t ∗Glu, SuperLUStat_t ∗stat)

4.154.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.154.2 Function Documentation

4.154.2.1 void zcheck_tempv ()

4.154.2.2 void zlsolve (int *ldm*, int *ncol*, doublecomplex ∗ *M*, doublecomplex ∗ *rhs*)

The unit lower triangular matrix is stored in a 2D array M(1:nrow,1:ncol). The solution will be returned in the rhs vector.

4.154.2.3 void zmatvec (int *ldm*, int *nrow*, int *ncol*, doublecomplex ∗ *M*, doublecomplex ∗ *vec*, doublecomplex ∗ *Mxvec*)

The input matrix is M(1:nrow,1:ncol); The product is returned in Mxvec[].

4.154.2.4 void zpanel_bmod (const int *m*, const int *w*, const int *jcol*, const int *nseg*, doublecomplex ∗ *dense*, doublecomplex ∗ *tempv*, int ∗ *segrep*, int ∗ *repfnz*, GlobalLU_t ∗ *Glu*, SuperLUStat_t ∗ *stat*)

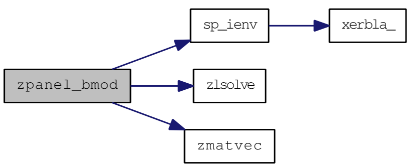
```
Purpose
=======

   Performs numeric block updates (sup-panel) in topological order.
   It features: col-col, 2cols-col, 3cols-col, and sup-col updates.
   Special processing on the supernodal portion of L[*,j]

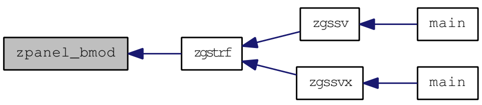
   Before entering this routine, the original nonzeros in the panel
   were already copied into the spa[m,w].

   Updated/Output parameters-
   dense[0:m-1,w]: L[*,j:j+w-1] and U[*,j:j+w-1] are returned
   collectively in the m-by-w vector dense[*].
```

Here is the call graph for this function:



Here is the caller graph for this function:

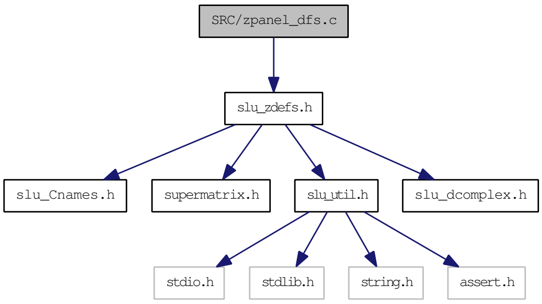


4.155 SRC/zpanel_dfs.c File Reference

Peforms a symbolic factorization on a panel of symbols.

`#include "slu_zdefs.h"`

Include dependency graph for zpanel_dfs.c:



Functions

- void zpanel_dfs (const int m, const int w, const int jcol, SuperMatrix ∗A, int ∗perm_r, int ∗nseg, doublecomplex ∗dense, int ∗panel_lsub, int ∗segrep, int ∗repfnz, int ∗xprune, int ∗marker, int ∗parent, int ∗xplore, GlobalLU_t ∗Glu)

4.155.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.155.2 Function Documentation

4.155.2.1 void zpanel_dfs (const int *m*, const int *w*, const int *jcol*, SuperMatrix ∗ *A*, int ∗ *perm_r*, int ∗ *nseg*, doublecomplex ∗ *dense*, int ∗ *panel_lsub*, int ∗ *segrep*, int ∗ *repfnz*, int ∗ *xprune*, int ∗ *marker*, int ∗ *parent*, int ∗ *xplore*, GlobalLU_t ∗ *Glu*)

```
Purpose
=======
```

```
Performs a symbolic factorization on a panel of columns [jcol, jcol+w).

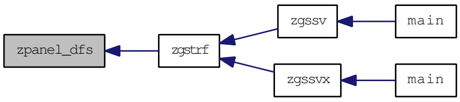
A supernode representative is the last column of a supernode.
The nonzeros in U[*,j] are segments that end at supernodal
representatives.

The routine returns one list of the supernodal representatives
in topological order of the dfs that generates them. This list is
a superset of the topological order of each individual column within
the panel.
The location of the first nonzero in each supernodal segment
(supernodal entry location) is also returned. Each column has a
separate list for this purpose.

Two marker arrays are used for dfs:
  marker[i] == jj, if i was visited during dfs of current column jj;
  marker1[i] >= jcol, if i was visited by earlier columns in this panel;

marker: A-row --> A-row/col (0/1)
repfnz: SuperA-col --> PA-row
parent: SuperA-col --> SuperA-col
xplore: SuperA-col --> index to L-structure
```

Here is the caller graph for this function:



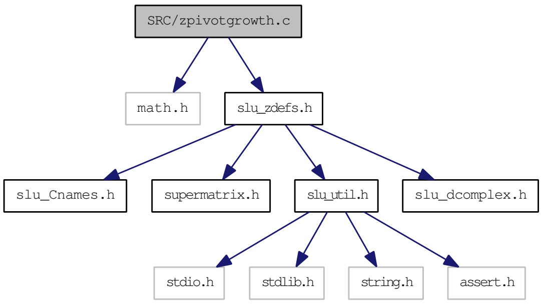
4.156 SRC/zpivotgrowth.c File Reference

Computes the reciprocal pivot growth factor.

`#include <math.h>`

`#include "slu_zdefs.h"`

Include dependency graph for zpivotgrowth.c:



Functions

- double zPivotGrowth (int ncols, SuperMatrix *A, int *perm_c, SuperMatrix *L, SuperMatrix *U)

4.156.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997
```

4.156.2 Function Documentation

4.156.2.1 double zPivotGrowth (int *ncols*, SuperMatrix * *A*, int * *perm_c*, SuperMatrix * *L*, SuperMatrix * *U*)

```
Purpose
=======


Compute the reciprocal pivot growth factor of the leading ncols columns
of the matrix, using the formula:
    min_j ( max_i(abs(A_ij)) / max_i(abs(U_ij)) )


Arguments
=========

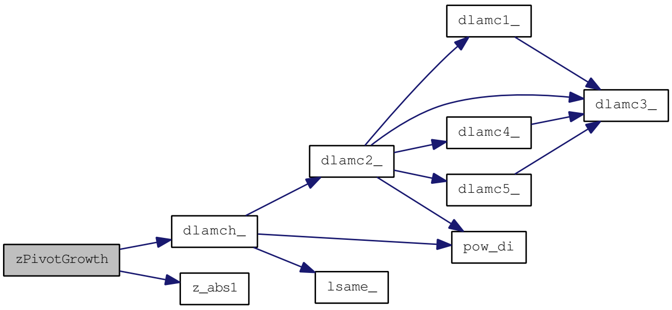

ncols    (input) int
         The number of columns of matrices A, L and U.
```

```
A        (input) SuperMatrix*
   Original matrix A, permuted by columns, of dimension
         (A->nrow, A->ncol). The type of A can be:
         Stype = NC; Dtype = SLU_Z; Mtype = GE.

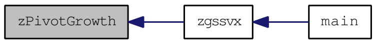
L        (output) SuperMatrix*
         The factor L from the factorization Pr*A=L*U; use compressed row
         subscripts storage for supernodes, i.e., L has type:
         Stype = SC; Dtype = SLU_Z; Mtype = TRLU.

U        (output) SuperMatrix*
   The factor U from the factorization Pr*A*Pc=L*U. Use column-wise
         storage scheme, i.e., U has types: Stype = NC;
         Dtype = SLU_Z; Mtype = TRU.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.157 SRC/zpivotL.c File Reference

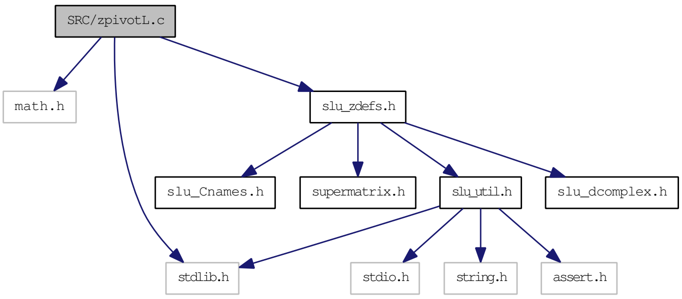
Performs numerical pivoting.

`#include <math.h>`

`#include <stdlib.h>`

`#include "slu_zdefs.h"`

Include dependency graph for zpivotL.c:



Functions

- int zpivotL (const int jcol, const double u, int ∗usepr, int ∗perm_r, int ∗iperm_r, int ∗iperm_c, int ∗pivrow, GlobalLU_t ∗Glu, SuperLUStat_t ∗stat)

4.157.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003

Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.157.2 Function Documentation

4.157.2.1 int zpivotL (const int *jcol*, const double *u*, int ∗ *usepr*, int ∗ *perm_r*, int ∗ *iperm_r*, int ∗ *iperm_c*, int ∗ *pivrow*, GlobalLU_t ∗ *Glu*, SuperLUStat_t ∗ *stat*)

```
Purpose
```

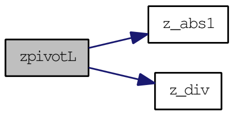
```
=======
  Performs the numerical pivoting on the current column of L,
  and the CDIV operation.

  Pivot policy:
  (1) Compute thresh = u * max_(i>=j) abs(A_ij);
  (2) IF user specifies pivot row k and abs(A_kj) >= thresh THEN
          pivot row = k;
      ELSE IF abs(A_jj) >= thresh THEN
          pivot row = j;
      ELSE
          pivot row = m;

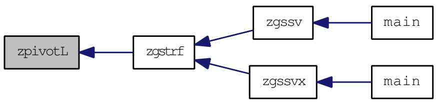
  Note: If you absolutely want to use a given pivot order, then set u=0.0.

  Return value: 0      success;
                i > 0  U(i,i) is exactly zero.
```

Here is the call graph for this function:



Here is the caller graph for this function:

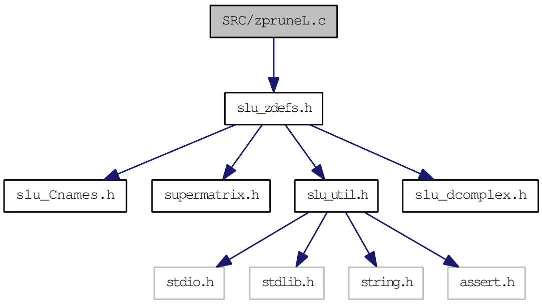


4.158 SRC/zpruneL.c File Reference

Prunes the L-structure.

`#include "slu_zdefs.h"`

Include dependency graph for zpruneL.c:



Functions

- void zpruneL (const int jcol, const int ∗perm_r, const int pivrow, const int nseg, const int ∗segrep, const int ∗repfnz, int ∗xprune, GlobalLU_t ∗Glu)

4.158.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997


Copyright (c) 1994 by Xerox Corporation.  All rights reserved.


THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.


Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
*
```

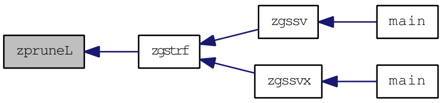
4.158.2 Function Documentation

4.158.2.1 void zpruneL (const int *jcol*, const int ∗ *perm_r*, const int *pivrow*, const int *nseg*, const int ∗ *segrep*, const int ∗ *repfnz*, int ∗ *xprune*, GlobalLU_t ∗ *Glu*)

```
Purpose
=======
  Prunes the L-structure of supernodes whose L-structure
```

```
contains the current pivot row "pivrow"
```

Here is the caller graph for this function:

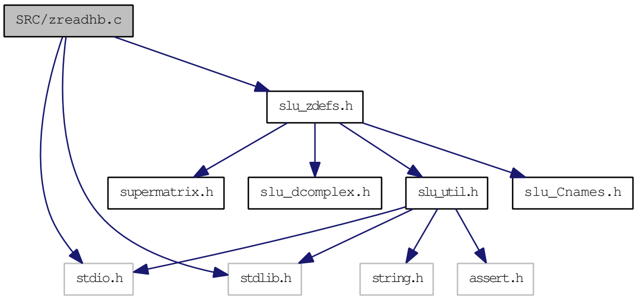


4.159 SRC/zreadhb.c File Reference

Read a matrix stored in Harwell-Boeing format.

```
#include <stdio.h>
#include <stdlib.h>
#include "slu_zdefs.h"
```

Include dependency graph for zreadhb.c:



Functions

- int zDumpLine (FILE ∗fp)

  Eat up the rest of the current line.

- int zParseIntFormat (char ∗buf, int ∗num, int ∗size)
- int zParseFloatFormat (char ∗buf, int ∗num, int ∗size)
- int zReadVector (FILE ∗fp, int n, int ∗where, int perline, int persize)
- int zReadValues (FILE ∗fp, int n, doublecomplex ∗destination, int perline, int persize)

  Read complex numbers as pairs of (real, imaginary).

- void zreadhb (int ∗nrow, int ∗ncol, int ∗nonz, doublecomplex ∗∗nzval, int ∗∗rowind, int ∗∗colptr)

  Auxiliary routines.

4.159.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997


Purpose
=======


Read a DOUBLE COMPLEX PRECISION matrix stored in Harwell-Boeing format
as described below.
```

```
 Line 1 (A72,A8)
   Col. 1 - 72   Title (TITLE)
Col. 73 - 80  Key (KEY)

 Line 2 (5I14)
  Col. 1 - 14   Total number of lines excluding header (TOTCRD)
  Col. 15 - 28  Number of lines for pointers (PTRCRD)
  Col. 29 - 42  Number of lines for row (or variable) indices (INDCRD)
  Col. 43 - 56  Number of lines for numerical values (VALCRD)
Col. 57 - 70  Number of lines for right-hand sides (RHSCRD)
                    (including starting guesses and solution vectors
       if present)
                  (zero indicates no right-hand side data is present)

 Line 3 (A3, 11X, 4I14)
    Col. 1 - 3    Matrix type (see below) (MXTYPE)
  Col. 15 - 28  Number of rows (or variables) (NROW)
  Col. 29 - 42  Number of columns (or elements) (NCOL)
Col. 43 - 56  Number of row (or variable) indices (NNZERO)
              (equal to number of entries for assembled matrices)
  Col. 57 - 70  Number of elemental matrix entries (NELTVL)
              (zero in the case of assembled matrices)
 Line 4 (2A16, 2A20)
  Col. 1 - 16   Format for pointers (PTRFMT)
Col. 17 - 32  Format for row (or variable) indices (INDFMT)
Col. 33 - 52  Format for numerical values of coefficient matrix (VALFMT)
  Col. 53 - 72 Format for numerical values of right-hand sides (RHSFMT)

 Line 5 (A3, 11X, 2I14) Only present if there are right-hand sides present
     Col. 1        Right-hand side type:
           F for full storage or M for same format as matrix
     Col. 2        G if a starting vector(s) (Guess) is supplied. (RHSTYP)
     Col. 3        X if an exact solution vector(s) is supplied.
Col. 15 - 28  Number of right-hand sides (NRHS)
Col. 29 - 42  Number of row indices (NRHSIX)
                 (ignored in case of unassembled matrices)

 The three character type field on line 3 describes the matrix type.
 The following table lists the permitted values for each of the three
 characters. As an example of the type field, RSA denotes that the matrix
 is real, symmetric, and assembled.

 First Character:
R Real matrix
C Complex matrix
P Pattern only (no numerical values supplied)

 Second Character:
S Symmetric
U Unsymmetric
H Hermitian
Z Skew symmetric
R Rectangular

 Third Character:
A Assembled
E Elemental matrices (unassembled)
```

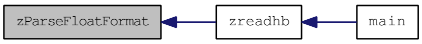
4.159.2 Function Documentation

4.159.2.1 int zDumpLine (FILE ∗ *fp*)

Here is the caller graph for this function:

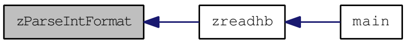
4.159.2.2 int zParseFloatFormat (char ∗ *buf*, int ∗ *num*, int ∗ *size*)

Here is the caller graph for this function:



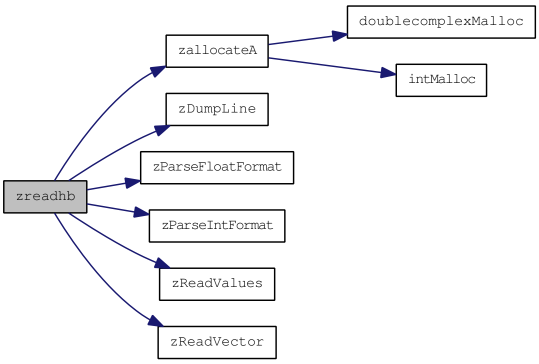
4.159.2.3 int zParseIntFormat (char ∗ *buf*, int ∗ *num*, int ∗ *size*)

Here is the caller graph for this function:

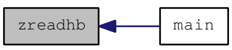


4.159.2.4 void zreadhb (int ∗ *nrow*, int ∗ *ncol*, int ∗ *nonz*, doublecomplex ∗∗ *nzval*, int ∗∗ *rowind*, int ∗∗ *colptr*)

Here is the call graph for this function:

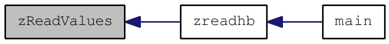


Here is the caller graph for this function:



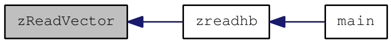
4.159.2.5 int zReadValues (FILE * *fp*, int *n*, doublecomplex * *destination*, int *perline*, int *persize*)

Here is the caller graph for this function:



4.159.2.6 int zReadVector (FILE * *fp*, int *n*, int * *where*, int *perline*, int *persize*)

Here is the caller graph for this function:

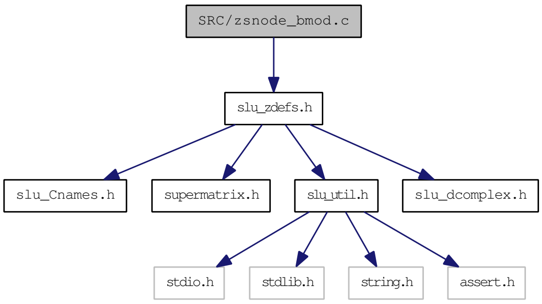


4.160 SRC/zsnode_bmod.c File Reference

Performs numeric block updates within the relaxed snode.

`#include "slu_zdefs.h"`

Include dependency graph for zsnode_bmod.c:



Functions

- int zsnode_bmod (const int jcol, const int jsupno, const int fsupc, doublecomplex ∗dense, doublecomplex ∗tempv, GlobalLU_t ∗Glu, SuperLUStat_t ∗stat)

  Performs numeric block updates within the relaxed snode.

4.160.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003


Copyright (c) 1994 by Xerox Corporation.  All rights reserved.

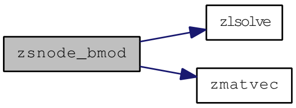

THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.


Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

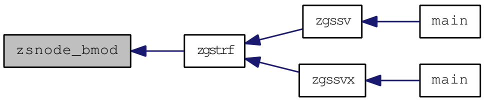
4.160.2 Function Documentation

4.160.2.1 int zsnode_bmod (const int *jcol*, const int *jsupno*, const int *fsupc*, doublecomplex ∗ *dense*, doublecomplex ∗ *tempv*, GlobalLU_t ∗ *Glu*, SuperLUStat_t ∗ *stat*)

Here is the call graph for this function:



Here is the caller graph for this function:

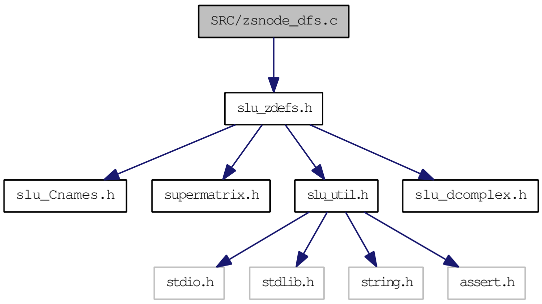


4.161 SRC/zsnode_dfs.c File Reference

Determines the union of row structures of columns within the relaxed node.

```
#include "slu_zdefs.h"
```

Include dependency graph for zsnode_dfs.c:



Functions

- int zsnode_dfs (const int jcol, const int kcol, const int ∗asub, const int ∗xa_begin, const int ∗xa_end, int ∗xprune, int ∗marker, GlobalLU_t ∗Glu)

4.161.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997


Copyright (c) 1994 by Xerox Corporation.  All rights reserved.


THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.


Permission is hereby granted to use or copy this program for any
purpose, provided the above notices are retained on all copies.
Permission to modify the code and to distribute modified code is
granted, provided the above notices are retained, and a notice that
the code was modified is included with the above copyright notice.
```

4.161.2 Function Documentation

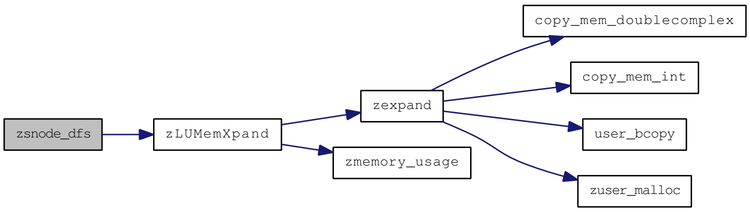
4.161.2.1 int zsnode_dfs (const int *jcol*, const int *kcol*, const int ∗ *asub*, const int ∗ *xa_begin*, const int ∗ *xa_end*, int ∗ *xprune*, int ∗ *marker*, GlobalLU_t ∗ *Glu*)

```
Purpose
=======
   zsnode_dfs() - Determine the union of the row structures of those
```

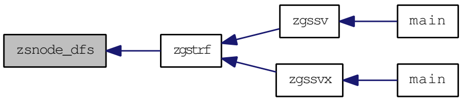
```
    columns within the relaxed snode.
    Note: The relaxed snodes are leaves of the supernodal etree, therefore,
    the portion outside the rectangular supernode must be zero.

 Return value
 ============
     0   success;
    >0   number of bytes allocated when run out of memory.
```

Here is the call graph for this function:



Here is the caller graph for this function:

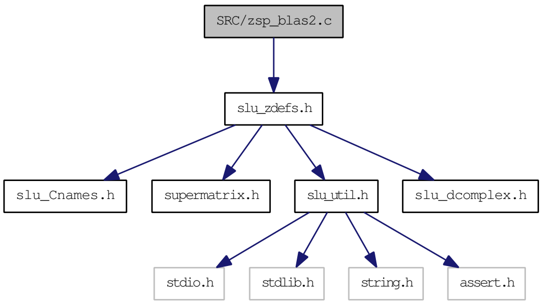


4.162 SRC/zsp_blas2.c File Reference

Sparse BLAS 2, using some dense BLAS 2 operations.

`#include "slu_zdefs.h"`

Include dependency graph for zsp_blas2.c:



Functions

- void zusolve (int, int, doublecomplex ∗, doublecomplex ∗)

  Solves a dense upper triangular system.

- void zlsolve (int, int, doublecomplex ∗, doublecomplex ∗)

  Solves a dense UNIT lower triangular system.

- void zmatvec (int, int, int, doublecomplex ∗, doublecomplex ∗, doublecomplex ∗)

  Performs a dense matrix-vector multiply: Mxvec = Mxvec + M ∗ vec.

- int sp_ztrsv (char ∗uplo, char ∗trans, char ∗diag, SuperMatrix ∗L, SuperMatrix ∗U, doublecomplex ∗x, SuperLUStat_t ∗stat, int ∗info)

  Solves one of the systems of equations A∗x = b, or A'∗x = b.

- int sp_zgemv (char ∗trans, doublecomplex alpha, SuperMatrix ∗A, doublecomplex ∗x, int incx, doublecomplex beta, doublecomplex ∗y, int incy)

  Performs one of the matrix-vector operations y := alpha∗A∗x + beta∗y, or y := alpha∗A'∗x + beta∗y.

4.162.1 Detailed Description

```
-- SuperLU routine (version 3.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
October 15, 2003
```

4.162.2 Function Documentation

4.162.2.1 int sp_zgemv (char ∗ *trans*, doublecomplex *alpha*, SuperMatrix ∗ *A*, doublecomplex ∗ *x*, int *incx*, doublecomplex *beta*, doublecomplex ∗ *y*, int *incy*)

```
Purpose
=======

sp_zgemv()  performs one of the matrix-vector operations
   y := alpha*A*x + beta*y,   or   y := alpha*A'*x + beta*y,
where alpha and beta are scalars, x and y are vectors and A is a
sparse A->nrow by A->ncol matrix.

Parameters
==========

TRANS  - (input) char*
         On entry, TRANS specifies the operation to be performed as
         follows:
            TRANS = 'N' or 'n'   y := alpha*A*x + beta*y.
            TRANS = 'T' or 't'   y := alpha*A'*x + beta*y.
            TRANS = 'C' or 'c'   y := alpha*A'*x + beta*y.

ALPHA  - (input) doublecomplex
         On entry, ALPHA specifies the scalar alpha.

A      - (input) SuperMatrix*
         Before entry, the leading m by n part of the array A must
         contain the matrix of coefficients.

X      - (input) doublecomplex*, array of DIMENSION at least
         ( 1 + ( n - 1 )*abs( INCX ) ) when TRANS = 'N' or 'n'
        and at least
         ( 1 + ( m - 1 )*abs( INCX ) ) otherwise.
         Before entry, the incremented array X must contain the
         vector x.

INCX   - (input) int
         On entry, INCX specifies the increment for the elements of
         X. INCX must not be zero.

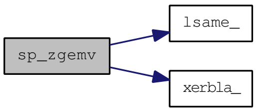
BETA   - (input) doublecomplex
         On entry, BETA specifies the scalar beta. When BETA is
         supplied as zero then Y need not be set on input.

Y      - (output) doublecomplex*,  array of DIMENSION at least
         ( 1 + ( m - 1 )*abs( INCY ) ) when TRANS = 'N' or 'n'
         and at least
         ( 1 + ( n - 1 )*abs( INCY ) ) otherwise.
         Before entry with BETA non-zero, the incremented array Y
         must contain the vector y. On exit, Y is overwritten by the
         updated vector y.

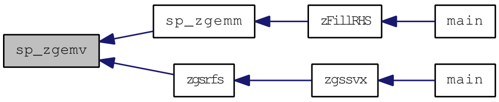
INCY   - (input) int
         On entry, INCY specifies the increment for the elements of
         Y. INCY must not be zero.
```

```
==== Sparse Level 2 Blas routine.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.162.2.2 int sp_ztrsv (char ∗ *uplo*, char ∗ *trans*, char ∗ *diag*, SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, doublecomplex ∗ *x*, SuperLUStat_t ∗ *stat*, int ∗ *info*)

```
Purpose
=======

sp_ztrsv() solves one of the systems of equations
    A*x = b,    or    A'*x = b,
where b and x are n element vectors and A is a sparse unit , or
non-unit, upper or lower triangular matrix.
No test for singularity or near-singularity is included in this
routine. Such tests must be performed before calling this routine.

Parameters
==========

uplo   - (input) char*
         On entry, uplo specifies whether the matrix is an upper or
          lower triangular matrix as follows:
             uplo = 'U' or 'u'   A is an upper triangular matrix.
             uplo = 'L' or 'l'   A is a lower triangular matrix.

trans  - (input) char*
          On entry, trans specifies the equations to be solved as
          follows:
             trans = 'N' or 'n'   A*x = b.
             trans = 'T' or 't'   A'*x = b.
             trans = 'C' or 'c'   A^H*x = b.

diag   - (input) char*
          On entry, diag specifies whether or not A is unit
          triangular as follows:
             diag = 'U' or 'u'   A is assumed to be unit triangular.
             diag = 'N' or 'n'   A is not assumed to be unit
                                 triangular.
```

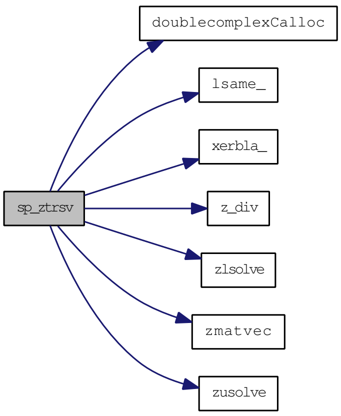
```
L       - (input) SuperMatrix*
    The factor L from the factorization Pr*A*Pc=L*U. Use
          compressed row subscripts storage for supernodes,
          i.e., L has types: Stype = SC, Dtype = SLU_Z, Mtype = TRLU.

U       - (input) SuperMatrix*
     The factor U from the factorization Pr*A*Pc=L*U.
     U has types: Stype = NC, Dtype = SLU_Z, Mtype = TRU.

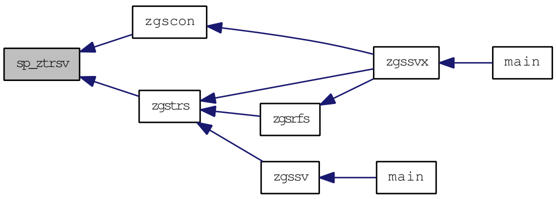
x       - (input/output) doublecomplex*
          Before entry, the incremented array X must contain the n
          element right-hand side vector b. On exit, X is overwritten
          with the solution vector x.

info    - (output) int*
          If *info = -i, the i-th argument had an illegal value.
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.162.2.3 void zlsolve (int *ldm*, int *ncol*, doublecomplex ∗ *M*, doublecomplex ∗ *rhs*)

The unit lower triangular matrix is stored in a 2D array M(1:nrow,1:ncol). The solution will be returned in the rhs vector.

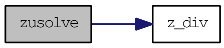
4.162.2.4 void zmatvec (int *ldm*, int *nrow*, int *ncol*, doublecomplex ∗ *M*, doublecomplex ∗ *vec*, doublecomplex ∗ *Mxvec*)

The input matrix is M(1:nrow,1:ncol); The product is returned in Mxvec[].

4.162.2.5 void zusolve (int *ldm*, int *ncol*, doublecomplex ∗ *M*, doublecomplex ∗ *rhs*)

The upper triangular matrix is stored in a 2-dim array M(1:ldm,1:ncol). The solution will be returned in the rhs vector.

Here is the call graph for this function:

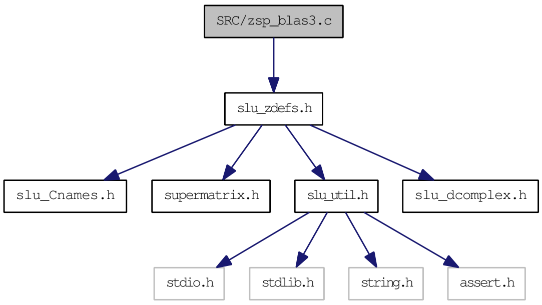


4.163 SRC/zsp_blas3.c File Reference

Sparse BLAS3, using some dense BLAS3 operations.

`#include "slu_zdefs.h"`

Include dependency graph for zsp_blas3.c:



Functions

- int sp_zgemm (char ∗transa, char ∗transb, int m, int n, int k, doublecomplex alpha, SuperMatrix ∗A, doublecomplex ∗b, int ldb, doublecomplex beta, doublecomplex ∗c, int ldc)

4.163.1 Detailed Description

```
-- SuperLU routine (version 2.0) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
November 15, 1997
```

4.163.2 Function Documentation

4.163.2.1 int sp_zgemm (char ∗ *transa*, char ∗ *transb*, int *m*, int *n*, int *k*, doublecomplex *alpha*, SuperMatrix ∗ *A*, doublecomplex ∗ *b*, int *ldb*, doublecomplex *beta*, doublecomplex ∗ *c*, int *ldc*)

```
Purpose
  =======

  sp_z performs one of the matrix-matrix operations

     C := alpha*op( A )*op( B ) + beta*C,

  where  op( X ) is one of

     op( X ) = X   or   op( X ) = X'   or   op( X ) = conjg( X' ),

  alpha and beta are scalars, and A, B and C are matrices, with op( A )
  an m by k matrix,  op( B )  a  k by n matrix and  C an m by n matrix.
```

```
Parameters
==========

TRANSA - (input) char*
         On entry, TRANSA specifies the form of op( A ) to be used in
         the matrix multiplication as follows:
            TRANSA = 'N' or 'n',  op( A ) = A.
            TRANSA = 'T' or 't',  op( A ) = A'.
            TRANSA = 'C' or 'c',  op( A ) = conjg( A' ).
         Unchanged on exit.

TRANSB - (input) char*
         On entry, TRANSB specifies the form of op( B ) to be used in
         the matrix multiplication as follows:
            TRANSB = 'N' or 'n',  op( B ) = B.
            TRANSB = 'T' or 't',  op( B ) = B'.
            TRANSB = 'C' or 'c',  op( B ) = conjg( B' ).
         Unchanged on exit.

M      - (input) int
         On entry,  M  specifies  the number of rows of the matrix
  op( A ) and of the matrix C.  M must be at least zero.
  Unchanged on exit.

N      - (input) int
         On entry,  N specifies the number of columns of the matrix
  op( B ) and the number of columns of the matrix C. N must be
  at least zero.
  Unchanged on exit.

K      - (input) int
         On entry, K specifies the number of columns of the matrix
  op( A ) and the number of rows of the matrix op( B ). K must
  be at least  zero.
        Unchanged on exit.

ALPHA  - (input) doublecomplex
         On entry, ALPHA specifies the scalar alpha.

A      - (input) SuperMatrix*
         Matrix A with a sparse format, of dimension (A->nrow, A->ncol).
         Currently, the type of A can be:
             Stype = NC or NCP; Dtype = SLU_Z; Mtype = GE.
         In the future, more general A can be handled.

B      - DOUBLE COMPLEX PRECISION array of DIMENSION ( LDB, kb ), where kb is
         n when TRANSB = 'N' or 'n',  and is  k otherwise.
         Before entry with  TRANSB = 'N' or 'n',  the leading k by n
         part of the array B must contain the matrix B, otherwise
         the leading n by k part of the array B must contain the
         matrix B.
         Unchanged on exit.

LDB    - (input) int
         On entry, LDB specifies the first dimension of B as declared
         in the calling (sub) program. LDB must be at least max( 1, n ).
         Unchanged on exit.
```

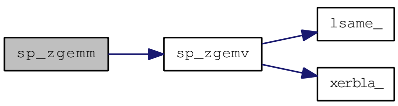
```
BETA    - (input) doublecomplex
          On entry, BETA specifies the scalar beta. When BETA is
          supplied as zero then C need not be set on input.

C       - DOUBLE COMPLEX PRECISION array of DIMENSION ( LDC, n ).
          Before entry, the leading m by n part of the array C must
          contain the matrix C,  except when beta is zero, in which
          case C need not be set on entry.
          On exit, the array C is overwritten by the m by n matrix
  ( alpha*op( A )*B + beta*C ).

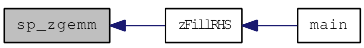
LDC     - (input) int
          On entry, LDC specifies the first dimension of C as declared
          in the calling (sub)program. LDC must be at least max(1,m).
          Unchanged on exit.

==== Sparse Level 3 Blas routine.
```

Here is the call graph for this function:



Here is the caller graph for this function:



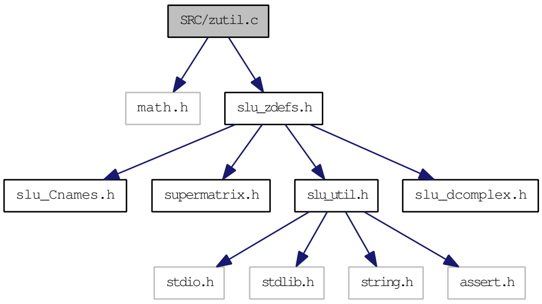
4.164 SRC/zutil.c File Reference

Matrix utility functions.

`#include <math.h>`

`#include "slu_zdefs.h"`

Include dependency graph for zutil.c:



Functions

- void zCreate_CompCol_Matrix (SuperMatrix ∗A, int m, int n, int nnz, doublecomplex ∗nzval, int ∗rowind, int ∗colptr, Stype_t stype, Dtype_t dtype, Mtype_t mtype)

  Supernodal LU factor related.

- void zCreate_CompRow_Matrix (SuperMatrix ∗A, int m, int n, int nnz, doublecomplex ∗nzval, int ∗colind, int ∗rowptr, Stype_t stype, Dtype_t dtype, Mtype_t mtype)
- void zCopy_CompCol_Matrix (SuperMatrix ∗A, SuperMatrix ∗B)

  Copy matrix A into matrix B.

- void zCreate_Dense_Matrix (SuperMatrix ∗X, int m, int n, doublecomplex ∗x, int ldx, Stype_t stype, Dtype_t dtype, Mtype_t mtype)
- void zCopy_Dense_Matrix (int M, int N, doublecomplex ∗X, int ldx, doublecomplex ∗Y, int ldy)
- void zCreate_SuperNode_Matrix (SuperMatrix ∗L, int m, int n, int nnz, doublecomplex ∗nzval, int ∗nzval_colptr, int ∗rowind, int ∗rowind_colptr, int ∗col_to_sup, int ∗sup_to_col, Stype_t stype, Dtype_t dtype, Mtype_t mtype)
- void zCompRow_to_CompCol (int m, int n, int nnz, doublecomplex ∗a, int ∗colind, int ∗rowptr, doublecomplex ∗∗at, int ∗∗rowind, int ∗∗colptr)

  Convert a row compressed storage into a column compressed storage.

- void zPrint_CompCol_Matrix (char ∗what, SuperMatrix ∗A)

  Routines for debugging.

- void zPrint_SuperNode_Matrix (char ∗what, SuperMatrix ∗A)
- void zPrint_Dense_Matrix (char ∗what, SuperMatrix ∗A)
- void zprint_lu_col (char ∗msg, int jcol, int pivrow, int ∗xprune, GlobalLU_t ∗Glu)

  Diagnostic print of column "jcol" in the U/L factor.

- void zcheck_tempv (int n, doublecomplex ∗tempv)

Check whether tempv[] == 0. This should be true before and after calling any numeric routines, i.e., "panel_bmod" and "column_bmod".

- void zGenXtrue (int n, int nrhs, doublecomplex ∗x, int ldx)
- void zFillRHS (trans_t trans, int nrhs, doublecomplex ∗x, int ldx, SuperMatrix ∗A, SuperMatrix ∗B)

  Let rhs[i] = sum of i-th row of A, so the solution vector is all 1's.

- void zfill (doublecomplex ∗a, int alen, doublecomplex dval)

  Fills a doublecomplex precision array with a given value.

- void zinf_norm_error (int nrhs, SuperMatrix ∗X, doublecomplex ∗xtrue)

  Check the inf-norm of the error vector.

- void zPrintPerf (SuperMatrix ∗L, SuperMatrix ∗U, mem_usage_t ∗mem_usage, double rpg, double rcond, double ∗ferr, double ∗berr, char ∗equed, SuperLUStat_t ∗stat)

  Print performance of the code.

- print_doublecomplex_vec (char ∗what, int n, doublecomplex ∗vec)

4.164.1 Detailed Description

```
-- SuperLU routine (version 3.1) --
Univ. of California Berkeley, Xerox Palo Alto Research Center,
and Lawrence Berkeley National Lab.
August 1, 2008
```

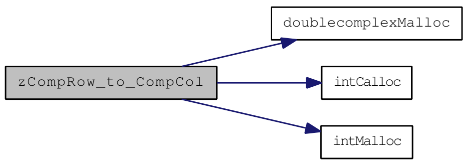
4.164.2 Function Documentation

4.164.2.1 print_doublecomplex_vec (char ∗ *what*, int *n*, doublecomplex ∗ *vec*)

4.164.2.2 void zcheck_tempv (int *n*, doublecomplex ∗ *tempv*)

4.164.2.3 void zCompRow_to_CompCol (int *m*, int *n*, int *nnz*, doublecomplex ∗ *a*, int ∗ *colind*, int ∗ *rowptr*, doublecomplex ∗∗ *at*, int ∗∗ *rowind*, int ∗∗ *colptr*)

Here is the call graph for this function:



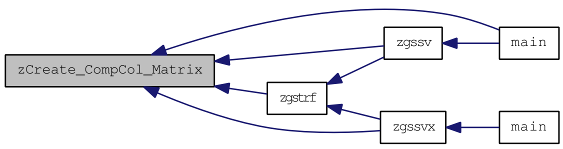
4.164.2.4 void zCopy_CompCol_Matrix (SuperMatrix ∗ *A*, SuperMatrix ∗ *B*)

4.164.2.5 void zCopy_Dense_Matrix (int *M*, int *N*, doublecomplex ∗ *X*, int *ldx*, doublecomplex ∗ *Y*, int *ldy*)

Copies a two-dimensional matrix X to another matrix Y.

4.164.2.6 void zCreate_CompCol_Matrix (SuperMatrix ∗ *A*, int *m*, int *n*, int *nnz*, doublecomplex ∗ *nzval*, int ∗ *rowind*, int ∗ *colptr*, Stype_t *stype*, Dtype_t *dtype*, Mtype_t *mtype*)

Here is the caller graph for this function:



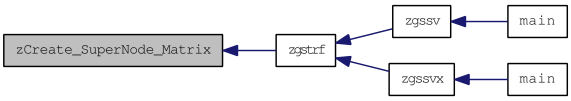
4.164.2.7 void zCreate_CompRow_Matrix (SuperMatrix ∗ *A*, int *m*, int *n*, int *nnz*, doublecomplex ∗ *nzval*, int ∗ *colind*, int ∗ *rowptr*, Stype_t *stype*, Dtype_t *dtype*, Mtype_t *mtype*)

4.164.2.8 void zCreate_Dense_Matrix (SuperMatrix ∗ *X*, int *m*, int *n*, doublecomplex ∗ *x*, int *ldx*, Stype_t *stype*, Dtype_t *dtype*, Mtype_t *mtype*)

Here is the caller graph for this function:

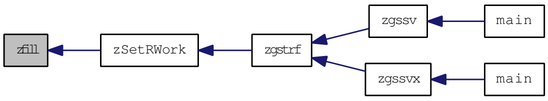
4.164.2.9 void zCreate_SuperNode_Matrix (SuperMatrix ∗ *L*, int *m*, int *n*, int *nnz*, doublecomplex ∗ *nzval*, int ∗ *nzval_colptr*, int ∗ *rowind*, int ∗ *rowind_colptr*, int ∗ *col_to_sup*, int ∗ *sup_to_col*, Stype_t *stype*, Dtype_t *dtype*, Mtype_t *mtype*)

Here is the caller graph for this function:



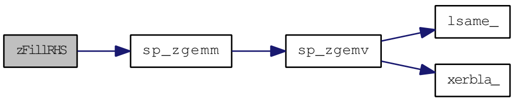
4.164.2.10 void zfill (doublecomplex ∗ *a*, int *alen*, doublecomplex *dval*)

Here is the caller graph for this function:

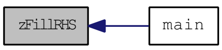


4.164.2.11 void zFillRHS (trans_t *trans*, int *nrhs*, doublecomplex ∗ *x*, int *ldx*, SuperMatrix ∗ *A*, SuperMatrix ∗ *B*)

Here is the call graph for this function:

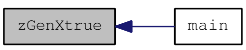


Here is the caller graph for this function:



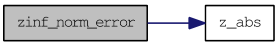
4.164.2.12 void zGenXtrue (int *n*, int *nrhs*, doublecomplex ∗ *x*, int *ldx*)

Here is the caller graph for this function:

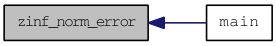


4.164.2.13 void zinf_norm_error (int *nrhs*, SuperMatrix ∗ *X*, doublecomplex ∗ *xtrue*)

Here is the call graph for this function:



Here is the caller graph for this function:

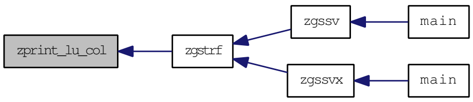


4.164.2.14 void zPrint_CompCol_Matrix (char ∗ *what*, SuperMatrix ∗ *A*)

4.164.2.15 void zPrint_Dense_Matrix (char ∗ *what*, SuperMatrix ∗ *A*)

4.164.2.16 void zprint_lu_col (char ∗ *msg*, int *jcol*, int *pivrow*, int ∗ *xprune*, GlobalLU_t ∗ *Glu*)

Here is the caller graph for this function:



4.164.2.17 void zPrint_SuperNode_Matrix (char ∗ *what*, SuperMatrix ∗ *A*)

4.164.2.18 void zPrintPerf (SuperMatrix ∗ *L*, SuperMatrix ∗ *U*, mem_usage_t ∗ *mem_usage*, double *rpg*, double *rcond*, double ∗ *ferr*, double ∗ *berr*, char ∗ *equed*, SuperLUStat_t ∗ *stat*)

Index

ABORT
- slu_util.h, 588

abs
- dlamch.c, 286
- slamch.c, 424

ADD_
- slu_Cnames.h, 486

ADD__
- slu_Cnames.h, 486

ALIVE
- colamd.c, 187
- old_colamd.c, 371

array
- LU_stack_t, 19

ASSERT
- colamd.c, 187

at_plus_a
- get_perm_c.c, 345

BOTH
- slu_util.h, 591

c_abs
- scomplex.c, 384
- slu_scomplex.h, 539

c_abs1
- scomplex.c, 384
- slu_scomplex.h, 539

c_add
- slu_scomplex.h, 538

C_CALL
- slu_Cnames.h, 486

c_div
- scomplex.c, 384
- slu_scomplex.h, 539

c_eq
- slu_scomplex.h, 538

c_exp
- scomplex.c, 384
- slu_scomplex.h, 540

c_sub
- slu_scomplex.h, 538

callocateA
- cmemory.c, 161
- slu_cdefs.h, 443

cc_conj
- slu_scomplex.h, 538

cc_mult
- slu_scomplex.h, 538

ccheck_tempv
- cpanel_bmod.c, 201
- cutil.c, 227

ccolumn_bmod
- ccolumn_bmod.c, 111
- slu_cdefs.h, 443

ccolumn_bmod.c
- ccolumn_bmod, 111
- clsolve, 111
- cmatvec, 112
- cusolve, 112

ccolumn_dfs
- ccolumn_dfs.c, 114
- slu_cdefs.h, 443

ccolumn_dfs.c
- ccolumn_dfs, 114
- T2_SUPER, 114

cCompRow_to_CompCol
- cutil.c, 227
- slu_cdefs.h, 444

cCopy_CompCol_Matrix
- cutil.c, 228
- slu_cdefs.h, 445

cCopy_Dense_Matrix
- cutil.c, 228
- slu_cdefs.h, 445

ccopy_to_ucol
- ccopy_to_ucol.c, 117
- slu_cdefs.h, 445

ccopy_to_ucol.c
- ccopy_to_ucol, 117

cCreate_CompCol_Matrix
- cutil.c, 228
- slu_cdefs.h, 445

cCreate_CompRow_Matrix
- cutil.c, 228
- slu_cdefs.h, 445

cCreate_Dense_Matrix
- cutil.c, 228
- slu_cdefs.h, 446

cCreate_SuperNode_Matrix

cutil.c, 228
slu_cdefs.h, 446
cDumpLine
creadhb.c, 212
cexpand
cmemory.c, 161
cfill
cutil.c, 228
slu_cdefs.h, 446
cFillRHS
cutil.c, 229
slu_cdefs.h, 446
cGenXtrue
cutil.c, 229
slu_cdefs.h, 446
cgscon
cgscon.c, 118
slu_cdefs.h, 447
cgscon.c
cgscon, 118
cgsequ
cgsequ.c, 121
slu_cdefs.h, 448
cgsequ.c
cgsequ, 121
cgsrfs
cgsrfs.c, 125
slu_cdefs.h, 450
cgsrfs.c
cgsrfs, 125
ITMAX, 125
cgssv
cgssv.c, 128
slu_cdefs.h, 452
cgssv.c
cgssv, 128
cgssvx
cgssvx.c, 133
slu_cdefs.h, 456
cgssvx.c
cgssvx, 133
cgstrf
cgstrf.c, 142
slu_cdefs.h, 463
cgstrf.c
cgstrf, 142
cgstrs
cgstrs.c, 149
slu_cdefs.h, 467
cgstrs.c
cgstrs, 149
clsolve, 150
cmatvec, 150
cprint_soln, 150
cusolve, 150
CHECK_MALLOC
slu_util.h, 588
check_perm
sp_preorder.c, 684
check_repfnz
slu_util.h, 593
util.c, 724
check_tempv
slu_cdefs.h, 468
slu_ddefs.h, 495
slu_sdefs.h, 545
slu_zdefs.h, 617
cinf_norm_error
cutil.c, 229
slu_cdefs.h, 468
clacon.c
clacon_, 152
clacon_
clacon.c, 152
clangs
clangs.c, 154
clangs.c
clangs, 154
claqgs
claqgs.c, 156
slu_cdefs.h, 469
claqgs.c
claqgs, 156
THRESH, 156
clear_mark
colamd.c, 187
old_colamd.c, 371
clinsol.c
main, 35
clinsol1.c
main, 38
clinsolx.c
main, 41
parse_command_line, 42
clinsolx1.c
main, 45
parse_command_line, 46
clinsolx2.c
main, 49
parse_command_line, 50
CLK_TCK
superlu_timer.c, 714
clsolve
ccolumn_bmod.c, 111
cgstrs.c, 150
cmyblas2.c, 171
cpanel_bmod.c, 201
csp_blas2.c, 219

cLUMemInit
cmemory.c, 162
slu_cdefs.h, 470
cLUMemXpand
cmemory.c, 163
slu_cdefs.h, 471
cLUWorkFree
cmemory.c, 164
slu_cdefs.h, 472
cLUWorkInit
cmemory.c, 164
cmatvec
ccolumn_bmod.c, 112
cgstrs.c, 150
cmyblas2.c, 172
cpanel_bmod.c, 201
csp_blas2.c, 219
cmemory.c
callocateA, 161
cexpand, 161
cLUMemInit, 162
cLUMemXpand, 163
cLUWorkFree, 164
cLUWorkInit, 164
cmemory_usage, 164
complexCalloc, 165
complexMalloc, 165
copy_mem_complex, 165
copy_mem_int, 166
cQuerySpace, 167
cSetRWork, 168
cSetupSpace, 168
cStackCompress, 168
cuser_free, 169
cuser_malloc, 169
DoubleAlign, 161
expanders, 170
ExpHeader, 161
GluIntArray, 161
no_expand, 170
NO_MEMTYPE, 161
NotDoubleAlign, 161
Reduce, 161
stack, 170
StackFull, 161
TempSpace, 161
user_bcopy, 169
cmemory_usage
cmemory.c, 164
slu_cdefs.h, 472
cmyblas2.c
clsolve, 171
cmatvec, 172
cusolve, 172
COL
slu_util.h, 591
COL_IS_ALIVE
colamd.c, 187
old_colamd.c, 371
COL_IS_DEAD
colamd.c, 187
old_colamd.c, 371
COL_IS_DEAD_PRINCIPAL
colamd.c, 187
old_colamd.c, 371
col_to_sup
SCformat, 26
SCPformat, 27
COLAMD
slu_util.h, 591
colamd
colamd.c, 188
colamd.h, 198
old_colamd.c, 371
old_colamd.h, 373
colamd.c
ALIVE, 187
ASSERT, 187
clear_mark, 187
COL_IS_ALIVE, 187
COL_IS_DEAD, 187
COL_IS_DEAD_PRINCIPAL, 187
colamd, 188
colamd_recommended, 188
colamd_report, 189
colamd_set_defaults, 189
DEAD, 187
DEAD_NON_PRINCIPAL, 187
DEAD_PRINCIPAL, 187
DEBUG0, 187
DEBUG1, 187
DEBUG2, 187
DEBUG3, 187
DEBUG4, 187
detect_super_cols, 189
EMPTY, 187
FALSE, 187
find_ordering, 190
garbage_collection, 190
INDEX, 187
init_rows_cols, 191
init_scoring, 191
KILL_NON_PRINCIPAL_COL, 187
KILL_PRINCIPAL_COL, 187
KILL_ROW, 187
MAX, 187
MIN, 187
ONES_COMPLEMENT, 187

order_children, 192
print_report, 192
PRINTF, 187
PRIVATE, 187
PUBLIC, 187
ROW_IS_ALIVE, 187
ROW_IS_DEAD, 187
ROW_IS_MARKED_DEAD, 187
symamd, 192
symamd_report, 193
TRUE, 187
colamd.h
colamd, 198
COLAMD_C, 197
Colamd_Col, 198
COLAMD_DEFRAG_COUNT, 197
COLAMD_DENSE_COL, 197
COLAMD_DENSE_ROW, 197
COLAMD_ERROR_A_not_present, 197
COLAMD_ERROR_A_too_small, 197
COLAMD_ERROR_col_length_negative, 197
COLAMD_ERROR_internal_error, 197
COLAMD_ERROR_ncol_negative, 197
COLAMD_ERROR_nnz_negative, 197
COLAMD_ERROR_nrow_negative, 197
COLAMD_ERROR_out_of_memory, 197
COLAMD_ERROR_p0_nonzero, 197
COLAMD_ERROR_p_not_present, 197
COLAMD_ERROR_row_index_out_of_-bounds, 197
COLAMD_INFO1, 197
COLAMD_INFO2, 197
COLAMD_INFO3, 197
COLAMD_KNOBS, 197
COLAMD_OK, 197
COLAMD_OK_BUT_JUMBLED, 197
COLAMD_R, 197
COLAMD_RECOMMENDED, 197
colamd_recommended, 198
colamd_report, 199
Colamd_Row, 198
colamd_set_defaults, 199
COLAMD_STATS, 198
COLAMD_STATUS, 198
symamd, 199
symamd_report, 199
COLAMD_C
colamd.h, 197
Colamd_Col
colamd.h, 198
Colamd_Col_struct, 7
degree_next, 8
hash, 8
hash_next, 8
headhash, 8
length, 8
order, 8
parent, 8
prev, 8
score, 8
shared1, 8
shared2, 8
shared3, 8
shared4, 8
start, 8
thickness, 8
COLAMD_DEFRAG_COUNT
colamd.h, 197
old_colamd.h, 373
COLAMD_DENSE_COL
colamd.h, 197
old_colamd.h, 373
COLAMD_DENSE_ROW
colamd.h, 197
old_colamd.h, 373
COLAMD_ERROR_A_not_present
colamd.h, 197
COLAMD_ERROR_A_too_small
colamd.h, 197
COLAMD_ERROR_col_length_negative
colamd.h, 197
COLAMD_ERROR_internal_error
colamd.h, 197
COLAMD_ERROR_ncol_negative
colamd.h, 197
COLAMD_ERROR_nnz_negative
colamd.h, 197
COLAMD_ERROR_nrow_negative
colamd.h, 197
COLAMD_ERROR_out_of_memory
colamd.h, 197
COLAMD_ERROR_p0_nonzero
colamd.h, 197
COLAMD_ERROR_p_not_present
colamd.h, 197
COLAMD_ERROR_row_index_out_of_bounds
colamd.h, 197
COLAMD_INFO1
colamd.h, 197
COLAMD_INFO2
colamd.h, 197
COLAMD_INFO3
colamd.h, 197
COLAMD_JUMBLED_COLS
old_colamd.h, 373
COLAMD_KNOBS
colamd.h, 197

old_colamd.h, 373
COLAMD_OK
colamd.h, 197
COLAMD_OK_BUT_JUMBLED
colamd.h, 197
COLAMD_R
colamd.h, 197
COLAMD_RECOMMENDED
colamd.h, 197
colamd_recommended
colamd.c, 188
colamd.h, 198
old_colamd.c, 372
old_colamd.h, 373
colamd_report
colamd.c, 189
colamd.h, 199
Colamd_Row
colamd.h, 198
Colamd_Row_struct, 9
degree, 9
first_column, 9
length, 9
mark, 9
p, 9
shared1, 9
shared2, 9
start, 9
colamd_set_defaults
colamd.c, 189
colamd.h, 199
old_colamd.c, 372
old_colamd.h, 374
COLAMD_STATS
colamd.h, 198
old_colamd.h, 373
COLAMD_STATUS
colamd.h, 198
colbeg
NCPformat, 22
colend
NCPformat, 22
colind
NRformat, 23
NRformat_loc, 24
ColInfo
old_colamd.c, 371
ColInfo_struct, 10
degree_next, 11
hash, 11
hash_next, 11
headhash, 11
length, 11
order, 11
parent, 11
prev, 11
score, 11
shared1, 11
shared2, 11
shared3, 11
shared4, 11
start, 11
thickness, 11
COLPERM
slu_util.h, 592
ColPerm
superlu_options_t, 29
colperm_t
slu_util.h, 590
colptr
NCformat, 21
complex, 12
i, 12
r, 12
complexCalloc
cmemory.c, 165
slu_cdefs.h, 472
complexMalloc
cmemory.c, 165
slu_cdefs.h, 472
ConditionNumber
superlu_options_t, 29
CONJ
slu_util.h, 592
copy_mem_complex
cmemory.c, 165
copy_mem_double
dmemory.c, 302
copy_mem_doublecomplex
zmemory.c, 783
copy_mem_float
smemory.c, 662
copy_mem_int
cmemory.c, 166
dmemory.c, 302
memory.c, 359
smemory.c, 662
zmemory.c, 783
countnz
slu_cdefs.h, 473
slu_ddefs.h, 495
slu_sdefs.h, 545
slu_zdefs.h, 617
util.c, 724
cpanel_bmod
cpanel_bmod.c, 201
slu_cdefs.h, 473
cpanel_bmod.c

ccheck_tempv, 201
clsolve, 201
cmatvec, 201
cpanel_bmod, 201
cpanel_dfs
cpanel_dfs.c, 202
slu_cdefs.h, 474
cpanel_dfs.c
cpanel_dfs, 202
cParseFloatFormat
creadhb.c, 212
cParseIntFormat
creadhb.c, 212
cPivotGrowth
cpivotgrowth.c, 204
slu_cdefs.h, 475
cpivotgrowth.c
cPivotGrowth, 204
cpivotL
cpivotL.c, 206
slu_cdefs.h, 476
cpivotL.c
cpivotL, 206
cPrint_CompCol_Matrix
cutil.c, 229
slu_cdefs.h, 476
cPrint_Dense_Matrix
cutil.c, 229
slu_cdefs.h, 477
cprint_lu_col
cutil.c, 229
cprint_soln
cgstrs.c, 150
cPrint_SuperNode_Matrix
cutil.c, 230
slu_cdefs.h, 477
cPrintPerf
cutil.c, 230
cpruneL
cpruneL.c, 208
slu_cdefs.h, 477
cpruneL.c
cpruneL, 208
cQuerySpace
cmemory.c, 167
slu_cdefs.h, 477
creadhb
creadhb.c, 212
slu_cdefs.h, 477
creadhb.c
cDumpLine, 212
cParseFloatFormat, 212
cParseIntFormat, 212
creadhb, 212
cReadValues, 212
cReadVector, 213
creadmt
slu_cdefs.h, 478
cReadValues
creadhb.c, 212
cReadVector
creadhb.c, 213
cs_mult
slu_scomplex.h, 539
cSetRWork
cmemory.c, 168
slu_cdefs.h, 478
cSetupSpace
cmemory.c, 168
csnode_bmod
csnode_bmod.c, 215
slu_cdefs.h, 478
csnode_bmod.c
csnode_bmod, 215
csnode_dfs
csnode_dfs.c, 216
slu_cdefs.h, 479
csnode_dfs.c
csnode_dfs, 216
csp_blas2.c
clsolve, 219
cmatvec, 219
cusolve, 219
sp_cgemv, 219
sp_ctrsv, 220
csp_blas3.c
sp_cgemm, 223
cStackCompress
cmemory.c, 168
cuser_free
cmemory.c, 169
cuser_malloc
cmemory.c, 169
cusolve
ccolumn_bmod.c, 112
cgstrs.c, 150
cmyblas2.c, 172
csp_blas2.c, 219
cutil.c
ccheck_tempv, 227
cCompRow_to_CompCol, 227
cCopy_CompCol_Matrix, 228
cCopy_Dense_Matrix, 228
cCreate_CompCol_Matrix, 228
cCreate_CompRow_Matrix, 228
cCreate_Dense_Matrix, 228
cCreate_SuperNode_Matrix, 228
cfill, 228

cFillRHS, 229
cGenXtrue, 229
cinf_norm_error, 229
cPrint_CompCol_Matrix, 229
cPrint_Dense_Matrix, 229
cprint_lu_col, 229
cPrint_SuperNode_Matrix, 230
cPrintPerf, 230
print_complex_vec, 230
CX
dzsum1.c, 343
icmax1.c, 352
izmax1.c, 354
scsum1.c, 387

d_cnjg
dcomplex.c, 238
slu_dcomplex.h, 489
d_imag
dcomplex.c, 238
slu_dcomplex.h, 489
d_sign
dlacon.c, 283
slacon.c, 421
dabs
slamch.c, 424
dallocateA
dmemory.c, 303
slu_ddefs.h, 495
dcheck_tempv
dpanel_bmod.c, 313
dutil.c, 340
dcolumn_bmod
dcolumn_bmod.c, 232
slu_ddefs.h, 495
dcolumn_bmod.c
dcolumn_bmod, 232
dlsolve, 232
dmatvec, 233
dusolve, 233
dcolumn_dfs
dcolumn_dfs.c, 235
slu_ddefs.h, 496
dcolumn_dfs.c
dcolumn_dfs, 235
T2_SUPER, 235
dcomplex.c
d_cnjg, 238
d_imag, 238
z_abs, 238
z_abs1, 238
z_div, 238
z_exp, 238
dCompRow_to_CompCol
dutil.c, 340
slu_ddefs.h, 497
dCopy_CompCol_Matrix
dutil.c, 340
slu_ddefs.h, 497
dCopy_Dense_Matrix
dutil.c, 340
slu_ddefs.h, 497
dcopy_to_ucol
dcopy_to_ucol.c, 240
slu_ddefs.h, 497
dcopy_to_ucol.c
dcopy_to_ucol, 240
dCreate_CompCol_Matrix
dutil.c, 340
slu_ddefs.h, 497
dCreate_CompRow_Matrix
dutil.c, 340
slu_ddefs.h, 498
dCreate_Dense_Matrix
dutil.c, 340
slu_ddefs.h, 498
dCreate_SuperNode_Matrix
dutil.c, 340
slu_ddefs.h, 498
dDumpLine
dreadhb.c, 324
DEAD
colamd.c, 187
old_colamd.c, 371
DEAD_NON_PRINCIPAL
colamd.c, 187
old_colamd.c, 371
DEAD_PRINCIPAL
colamd.c, 187
old_colamd.c, 371
DEBUG0
colamd.c, 187
old_colamd.c, 371
DEBUG1
colamd.c, 187
old_colamd.c, 371
DEBUG2
colamd.c, 187
old_colamd.c, 371
DEBUG3
colamd.c, 187
old_colamd.c, 371
DEBUG4
colamd.c, 187
old_colamd.c, 371
degree
Colamd_Row_struct, 9
RowInfo_struct, 25

degree_next
    Colamd_Col_struct, 8
    ColInfo_struct, 11
DenseSize
    util.c, 724
Destroy_CompCol_Matrix
    slu_util.h, 593
    util.c, 724
Destroy_CompCol_Permuted
    slu_util.h, 593
    util.c, 724
Destroy_CompRow_Matrix
    slu_util.h, 593
    util.c, 725
Destroy_Dense_Matrix
    slu_util.h, 593
    util.c, 725
Destroy_SuperMatrix_Store
    slu_util.h, 593
    util.c, 725
Destroy_SuperNode_Matrix
    slu_util.h, 594
    util.c, 726
detect_super_cols
    colamd.c, 189
    old_colamd.c, 372
dexpand
    dmemory.c, 303
dfill
    dutil.c, 341
    slu_ddefs.h, 498
dFillRHS
    dutil.c, 341
    slu_ddefs.h, 498
dGenXtrue
    dutil.c, 341
    slu_ddefs.h, 499
dGetDiagU
    dGetDiagU.c, 242
dGetDiagU.c
    dGetDiagU, 242
dgscon
    dgscon.c, 243
    slu_ddefs.h, 499
dgscon.c
    dgscon, 243
dgsequ
    dgsequ.c, 246
    slu_ddefs.h, 500
dgsequ.c
    dgsequ, 246
dgsrfs
    dgsrfs.c, 250
    slu_ddefs.h, 502
dgsrfs.c
    dgsrfs, 250
    ITMAX, 250
dgssv
    dgssv.c, 253
    slu_ddefs.h, 504
dgssv.c
    dgssv, 253
dgssvx
    dgssvx.c, 258
    slu_ddefs.h, 509
dgssvx.c
    dgssvx, 258
dgstrf
    dgstrf.c, 267
    slu_ddefs.h, 516
dgstrf.c
    dgstrf, 267
dgstrs
    dgstrs.c, 274
    slu_ddefs.h, 520
dgstrs.c
    dgstrs, 274
    dlsolve, 275
    dmatvec, 275
    dprint_soln, 275
    dusolve, 275
dgstrsL
    dgstrsL.c, 278
dgstrsL.c
    dgstrsL, 278
    dlsolve, 279
    dmatvec, 279
    dprint_soln, 279
    dusolve, 279
dgstrsU
    dgstrsU.c, 281
dgstrsU.c
    dgstrsU, 281
    dlsolve, 282
    dmatvec, 282
    dusolve, 282
DiagPivotThresh
    superlu_options_t, 29
DiagScale_t
    slu_util.h, 591
dinf_norm_error
    dutil.c, 341
    slu_ddefs.h, 521
dlacon.c
    d_sign, 283
    dlacon_, 283
    i_dnnt, 283
dlacon_

dlacon.c, 283
dlamc1_
dlamch.c, 286
dlamc2_
dlamch.c, 287
dlamc3_
dlamch.c, 289
dlamc4_
dlamch.c, 290
dlamc5_
dlamch.c, 291
dlamch.c
abs, 286
dlamc1_, 286
dlamc2_, 287
dlamc3_, 289
dlamc4_, 290
dlamc5_, 291
dlamch_, 292
FALSE_, 286
max, 286
min, 286
pow_di, 294
TRUE_, 286
dlamch_
dlamch.c, 292
dlangs
dlangs.c, 295
dlangs.c
dlangs, 295
dlaqgs
dlaqgs.c, 297
slu_ddefs.h, 521
dlaqgs.c
dlaqgs, 297
THRESH, 297
dlinsol.c
main, 53
dlinsol1.c
main, 56
dlinsolx.c
main, 59
parse_command_line, 60
dlinsolx1.c
main, 63
parse_command_line, 64
dlinsolx2.c
main, 67
parse_command_line, 68
dlsolve
dcolumn_bmod.c, 232
dgstrs.c, 275
dgstrsL.c, 279
dgstrsU.c, 282
dmyblas2.c, 310
dpanel_bmod.c, 313
dsp_blas2.c, 331
dLUMemInit
dmemory.c, 303
slu_ddefs.h, 523
dLUMemXpand
dmemory.c, 304
slu_ddefs.h, 524
dLUWorkFree
dmemory.c, 305
slu_ddefs.h, 524
dLUWorkInit
dmemory.c, 305
dmatvec
dcolumn_bmod.c, 233
dgstrs.c, 275
dgstrsL.c, 279
dgstrsU.c, 282
dmyblas2.c, 310
dpanel_bmod.c, 313
dsp_blas2.c, 331
dmemory.c
copy_mem_double, 302
copy_mem_int, 302
dallocateA, 303
dexpand, 303
dLUMemInit, 303
dLUMemXpand, 304
dLUWorkFree, 305
dLUWorkInit, 305
dmemory_usage, 305
DoubleAlign, 302
doubleCalloc, 306
doubleMalloc, 306
dQuerySpace, 306
dSetRWork, 307
dSetupSpace, 307
dStackCompress, 308
duser_free, 308
duser_malloc, 308
expanders, 309
ExpHeader, 302
GluIntArray, 302
no_expand, 309
NO_MEMTYPE, 302
NotDoubleAlign, 302
Reduce, 302
stack, 309
StackFull, 302
TempSpace, 302
user_bcopy, 308
dmemory_usage
dmemory.c, 305

slu_ddefs.h, 524
dmyblas2.c
dlsolve, 310
dmatvec, 310
dusolve, 311
DNformat, 13
lda, 13
nzval, 13
DOFACT
slu_util.h, 591
DOUBLE
slu_util.h, 591
DoubleAlign
cmemory.c, 161
dmemory.c, 302
smemory.c, 662
zmemory.c, 783
doubleCalloc
dmemory.c, 306
slu_ddefs.h, 525
slu_zdefs.h, 617
doublecomplex, 14
i, 14
r, 14
doublecomplexCalloc
slu_zdefs.h, 617
zmemory.c, 784
doublecomplexMalloc
slu_zdefs.h, 617
zmemory.c, 784
doubleMalloc
dmemory.c, 306
slu_ddefs.h, 525
slu_zdefs.h, 617
dpanel_bmod
dpanel_bmod.c, 313
slu_ddefs.h, 525
dpanel_bmod.c
dcheck_tempv, 313
dlsolve, 313
dmatvec, 313
dpanel_bmod, 313
dpanel_dfs
dpanel_dfs.c, 314
slu_ddefs.h, 526
dpanel_dfs.c
dpanel_dfs, 314
dParseFloatFormat
dreadhb.c, 324
dParseIntFormat
dreadhb.c, 324
dPivotGrowth
dpivotgrowth.c, 316
slu_ddefs.h, 526
dpivotgrowth.c
dPivotGrowth, 316
dpivotL
dpivotL.c, 318
slu_ddefs.h, 527
dpivotL.c
dpivotL, 318
dPrint_CompCol_Matrix
dutil.c, 342
slu_ddefs.h, 528
dPrint_Dense_Matrix
dutil.c, 342
slu_ddefs.h, 528
dprint_lu_col
dutil.c, 342
dprint_soln
dgstrs.c, 275
dgstrsL.c, 279
dPrint_SuperNode_Matrix
dutil.c, 342
slu_ddefs.h, 528
dPrintPerf
dutil.c, 342
dpruneL
dpruneL.c, 320
slu_ddefs.h, 528
dpruneL.c
dpruneL, 320
dQuerySpace
dmemory.c, 306
slu_ddefs.h, 528
dreadhb
dreadhb.c, 324
slu_ddefs.h, 529
dreadhb.c
dDumpLine, 324
dParseFloatFormat, 324
dParseIntFormat, 324
dreadhb, 324
dReadValues, 324
dReadVector, 325
dreadmt
slu_ddefs.h, 529
dreadrhs
dreadtriple.c, 69
dreadtriple
dreadtriple.c, 69
dreadtriple.c
dreadrhs, 69
dreadtriple, 69
dReadValues
dreadhb.c, 324
dReadVector
dreadhb.c, 325

dSetRWork
    dmemory.c, 307
    slu_ddefs.h, 530
dSetupSpace
    dmemory.c, 307
dsnode_bmod
    dsnode_bmod.c, 327
    slu_ddefs.h, 530
dsnode_bmod.c
    dsnode_bmod, 327
dsnode_dfs
    dsnode_dfs.c, 328
    slu_ddefs.h, 530
dsnode_dfs.c
    dsnode_dfs, 328
dsp_blas2.c
    dlsolve, 331
    dmatvec, 331
    dusolve, 331
    sp_dgemv, 331
    sp_dtrsv, 332
dsp_blas3.c
    sp_dgemm, 335
dStackCompress
    dmemory.c, 308
Dtype
    SuperMatrix, 31
Dtype_t
    supermatrix.h, 715
duser_free
    dmemory.c, 308
duser_malloc
    dmemory.c, 308
dusolve
    dcolumn_bmod.c, 233
    dgstrs.c, 275
    dgstrsL.c, 279
    dgstrsU.c, 282
    dmyblas2.c, 311
    dsp_blas2.c, 331
dutil.c
    dcheck_tempv, 340
    dCompRow_to_CompCol, 340
    dCopy_CompCol_Matrix, 340
    dCopy_Dense_Matrix, 340
    dCreate_CompCol_Matrix, 340
    dCreate_CompRow_Matrix, 340
    dCreate_Dense_Matrix, 340
    dCreate_SuperNode_Matrix, 340
    dfill, 341
    dFillRHS, 341
    dGenXtrue, 341
    dinf_norm_error, 341
    dPrint_CompCol_Matrix, 342
    dPrint_Dense_Matrix, 342
    dprint_lu_col, 342
    dPrint_SuperNode_Matrix, 342
    dPrintPerf, 342
    print_double_vec, 342
dzsum1.c
    CX, 343
    dzsum1_, 343
dzsum1_
    dzsum1.c, 343

e_node, 15
    mem, 15
    size, 15
EMPTY
    colamd.c, 187
    old_colamd.c, 371
    slu_util.h, 589
EQUIL
    slu_util.h, 592
Equil
    superlu_options_t, 29
etdfs
    sp_coletree.c, 673
ETREE
    slu_util.h, 592
EXAMPLE/clinsol.c, 33
EXAMPLE/clinsol1.c, 36
EXAMPLE/clinsolx.c, 39
EXAMPLE/clinsolx1.c, 43
EXAMPLE/clinsolx2.c, 47
EXAMPLE/dlinsol.c, 51
EXAMPLE/dlinsol1.c, 54
EXAMPLE/dlinsolx.c, 57
EXAMPLE/dlinsolx1.c, 61
EXAMPLE/dlinsolx2.c, 65
EXAMPLE/dreadtriple.c, 69
EXAMPLE/slinsol.c, 70
EXAMPLE/slinsol1.c, 73
EXAMPLE/slinsolx.c, 76
EXAMPLE/slinsolx1.c, 80
EXAMPLE/slinsolx2.c, 84
EXAMPLE/sp_ienv.c, 681
    sp_ienv, 681
EXAMPLE/superlu.c, 88
EXAMPLE/zlinsol.c, 91
EXAMPLE/zlinsol1.c, 94
EXAMPLE/zlinsolx.c, 97
EXAMPLE/zlinsolx1.c, 101
EXAMPLE/zlinsolx2.c, 105
EXAMPLE/zreadtriple.c, 109
expanders
    cmemory.c, 170
    dmemory.c, 309

smemory.c, 669
zmemory.c, 790
expansions
mem_usage_t, 20
ExpHeader
cmemory.c, 161
dmemory.c, 302
smemory.c, 662
zmemory.c, 783
EXTRA
slu_util.h, 591

F77_CALL_C
slu_Cnames.h, 486
FACT
slu_util.h, 592
Fact
superlu_options_t, 29
fact_t
slu_util.h, 591
FACTORED
slu_util.h, 591
FALSE
colamd.c, 187
old_colamd.c, 371
slu_util.h, 590
FALSE_
dlamch.c, 286
slamch.c, 424
FERR
slu_util.h, 592
finalize_disjoint_sets
sp_coletree.c, 673
find
sp_coletree.c, 673
find_ordering
colamd.c, 190
old_colamd.c, 372
first_column
Colamd_Row_struct, 9
RowInfo_struct, 25
FIRSTCOL_OF_SNODE
slu_util.h, 590
fixupL
slu_cdefs.h, 479
slu_ddefs.h, 531
slu_sdefs.h, 545
slu_zdefs.h, 618
util.c, 726
floatCalloc
slu_cdefs.h, 480
slu_sdefs.h, 545
smemory.c, 663
floatMalloc
slu_cdefs.h, 480
slu_sdefs.h, 545
smemory.c, 663
flops_t
slu_util.h, 590
for_lu
mem_usage_t, 20
fst_row
NRformat_loc, 24

garbage_collection
colamd.c, 190
old_colamd.c, 372
GEMV
slu_util.h, 592
genmmd_
get_perm_c.c, 346
mmd.c, 364
get_colamd
get_perm_c.c, 346
get_perm_c
get_perm_c.c, 347
slu_util.h, 594
get_perm_c.c
at_plus_a, 345
genmmd_, 346
get_colamd, 346
get_perm_c, 347
getata, 348
getata
get_perm_c.c, 348
GlobalLU_t, 16
lsub, 18
lusup, 18
MemModel, 18
n, 18
nzlmax, 18
nzlumax, 18
nzumax, 18
supno, 18
ucol, 18
usub, 18
xlsub, 18
xlusup, 18
xsup, 18
xusub, 18
GluIntArray
cmemory.c, 161
dmemory.c, 302
smemory.c, 662
zmemory.c, 783

hash
Colamd_Col_struct, 8

ColInfo_struct, 11
hash_next
Colamd_Col_struct, 8
ColInfo_struct, 11
HEAD
slu_util.h, 592
headhash
Colamd_Col_struct, 8
ColInfo_struct, 11
heap_relax_snode
heap_relax_snode.c, 350
slu_util.h, 595
heap_relax_snode.c
heap_relax_snode, 350

i
complex, 12
doublecomplex, 14
i_dnnt
dlacon.c, 283
slacon.c, 421
icmax1.c
CX, 352
icmax1_, 352
icmax1_
icmax1.c, 352
ifill
slu_util.h, 596
util.c, 726
INDEX
colamd.c, 187
init_rows_cols
colamd.c, 191
old_colamd.c, 372
init_scoring
colamd.c, 191
old_colamd.c, 372
initialize_disjoint_sets
sp_coletree.c, 674
int_t
slu_cdefs.h, 443
slu_ddefs.h, 495
slu_sdefs.h, 545
slu_zdefs.h, 617
intCalloc
memory.c, 359
slu_util.h, 597
intMalloc
memory.c, 360
slu_util.h, 597
IterRefine
superlu_options_t, 29
IterRefine_t
slu_util.h, 591
ITMAX
cgsrfs.c, 125
dgsrfs.c, 250
sgsrfs.c, 395
zgsrfs.c, 748
izmax1.c
CX, 354
izmax1_, 354
izmax1_
izmax1.c, 354

KILL_NON_PRINCIPAL_COL
colamd.c, 187
old_colamd.c, 371
KILL_PRINCIPAL_COL
colamd.c, 187
old_colamd.c, 371
KILL_ROW
colamd.c, 187
old_colamd.c, 371

L_FST_SUPC
slu_util.h, 590
L_NZ_START
slu_util.h, 590
L_SUB
slu_util.h, 590
L_SUB_START
slu_util.h, 590
LargeDiag
slu_util.h, 592
lda
DNformat, 13
length
Colamd_Col_struct, 8
Colamd_Row_struct, 9
ColInfo_struct, 11
RowInfo_struct, 25
link
sp_coletree.c, 674
Logical
slu_util.h, 590
lsame.c
lsame_, 356
lsame_
lsame.c, 356
slu_util.h, 598
LSUB
slu_util.h, 591
lsub
GlobalLU_t, 18
LU_space_t
slu_util.h, 591
LU_stack_t, 19

array, 19
size, 19
top1, 19
top2, 19
used, 19
LUFactFlops
util.c, 726
LUSolveFlops
util.c, 727
LUSUP
slu_util.h, 591
lusup
GlobalLU_t, 18

m_loc
NRformat_loc, 24
main
clinsol.c, 35
clinsol1.c, 38
clinsolx.c, 41
clinsolx1.c, 45
clinsolx2.c, 49
dlinsol.c, 53
dlinsol1.c, 56
dlinsolx.c, 59
dlinsolx1.c, 63
dlinsolx2.c, 67
slinsol.c, 72
slinsol1.c, 75
slinsolx.c, 78
slinsolx1.c, 82
slinsolx2.c, 86
superlu.c, 90
zlinsol.c, 93
zlinsol1.c, 96
zlinsolx.c, 99
zlinsolx1.c, 103
zlinsolx2.c, 107
make_set
sp_coletree.c, 675
mark
Colamd_Row_struct, 9
RowInfo_struct, 25
MAX
colamd.c, 187
old_colamd.c, 371
max
dlamch.c, 286
slamch.c, 424
max_sup_size
util.c, 729
mem
e_node, 15
mem_usage_t, 20
expansions, 20
for_lu, 20
total_needed, 20
MemModel
GlobalLU_t, 18
memory.c
copy_mem_int, 359
intCalloc, 359
intMalloc, 360
SetIWork, 361
superlu_free, 362
superlu_malloc, 362
user_bcopy, 362
MemType
slu_util.h, 591
MIN
colamd.c, 187
old_colamd.c, 371
min
dlamch.c, 286
slamch.c, 424
mmd.c
genmmd_, 364
mmdelm_, 365
mmdint_, 365
mmdnum_, 366
mmdupd_, 366
shortint, 364
MMD_AT_PLUS_A
slu_util.h, 590
MMD_ATA
slu_util.h, 590
mmdelm_
mmd.c, 365
mmdint_
mmd.c, 365
mmdnum_
mmd.c, 366
mmdupd_
mmd.c, 366
Mtype
SuperMatrix, 31
Mtype_t
supermatrix.h, 715
mxCallocInt
sp_coletree.c, 675
MY_PERMC
slu_util.h, 591
MY_PERMR
slu_util.h, 592

n
GlobalLU_t, 18
NATURAL

slu_util.h, 590
NBUCKS
util.c, 724
NCformat, 21
colptr, 21
nnz, 21
nzval, 21
rowind, 21
ncol
SuperMatrix, 31
NCPformat, 22
colbeg, 22
colend, 22
nnz, 22
nzval, 22
rowind, 22
nnz
NCformat, 21
NCPformat, 22
NRformat, 23
SCformat, 26
SCPformat, 27
nnz_loc
NRformat_loc, 24
NO
slu_util.h, 592
no_expand
cmemory.c, 170
dmemory.c, 309
smemory.c, 669
zmemory.c, 790
NO_MARKER
slu_util.h, 590
NO_MEMTYPE
cmemory.c, 161
dmemory.c, 302
smemory.c, 662
zmemory.c, 783
NOCHANGE
slu_Cnames.h, 486
NOEQUIL
slu_util.h, 591
NOREFINE
slu_util.h, 591
NOROWPERM
slu_util.h, 592
NotDoubleAlign
cmemory.c, 161
dmemory.c, 302
smemory.c, 662
zmemory.c, 783
NOTRANS
slu_util.h, 592
NPHASES
slu_util.h, 592
nr_etdfs
sp_coletree.c, 676
NRformat, 23
colind, 23
nnz, 23
nzval, 23
rowptr, 23
NRformat_loc, 24
colind, 24
fst_row, 24
m_loc, 24
nnz_loc, 24
nzval, 24
rowptr, 24
nrow
SuperMatrix, 31
nsuper
SCformat, 26
SCPformat, 27
NUM_TEMPV
slu_util.h, 590
nzlmax
GlobalLU_t, 18
nzlumax
GlobalLU_t, 18
nzumax
GlobalLU_t, 18
nzval
DNformat, 13
NCformat, 21
NCPformat, 22
NRformat, 23
NRformat_loc, 24
SCformat, 26
SCPformat, 27
nzval_colbeg
SCPformat, 27
nzval_colend
SCPformat, 27
nzval_colptr
SCformat, 26

old_colamd.c
ALIVE, 371
clear_mark, 371
COL_IS_ALIVE, 371
COL_IS_DEAD, 371
COL_IS_DEAD_PRINCIPAL, 371
colamd, 371
colamd_recommended, 372
colamd_set_defaults, 372
ColInfo, 371
DEAD, 371

DEAD_NON_PRINCIPAL, 371
DEAD_PRINCIPAL, 371
DEBUG0, 371
DEBUG1, 371
DEBUG2, 371
DEBUG3, 371
DEBUG4, 371
detect_super_cols, 372
EMPTY, 371
FALSE, 371
find_ordering, 372
garbage_collection, 372
init_rows_cols, 372
init_scoring, 372
KILL_NON_PRINCIPAL_COL, 371
KILL_PRINCIPAL_COL, 371
KILL_ROW, 371
MAX, 371
MIN, 371
ONES_COMPLEMENT, 371
order_children, 372
PRIVATE, 371
PUBLIC, 371
ROW_IS_ALIVE, 371
ROW_IS_DEAD, 371
ROW_IS_MARKED_DEAD, 371
RowInfo, 371
TRUE, 371
old_colamd.h
colamd, 373
COLAMD_DEFRAG_COUNT, 373
COLAMD_DENSE_COL, 373
COLAMD_DENSE_ROW, 373
COLAMD_JUMBLED_COLS, 373
COLAMD_KNOBS, 373
colamd_recommended, 373
colamd_set_defaults, 374
COLAMD_STATS, 373
ONES_COMPLEMENT
colamd.c, 187
old_colamd.c, 371
ops
SuperLUStat_t, 30
order
Colamd_Col_struct, 8
ColInfo_struct, 11
order_children
colamd.c, 192
old_colamd.c, 372

p
Colamd_Row_struct, 9
RowInfo_struct, 25
panel_histo
SuperLUStat_t, 30
parent
Colamd_Col_struct, 8
ColInfo_struct, 11
parse_command_line
clinsolx.c, 42
clinsolx1.c, 46
clinsolx2.c, 50
dlinsolx.c, 60
dlinsolx1.c, 64
dlinsolx2.c, 68
slinsolx.c, 79
slinsolx1.c, 83
slinsolx2.c, 87
zlinsolx.c, 100
zlinsolx1.c, 104
zlinsolx2.c, 108
PhaseType
slu_util.h, 591
PivotGrowth
superlu_options_t, 29
pow_di
dlamch.c, 294
pow_ri
slamch.c, 424
prev
Colamd_Col_struct, 8
ColInfo_struct, 11
print_complex_vec
cutil.c, 230
print_double_vec
dutil.c, 342
print_doublecomplex_vec
zutil.c, 821
print_float_vec
sutil.c, 719
print_int_vec
util.c, 727
print_lu_col
slu_cdefs.h, 480
slu_ddefs.h, 531
slu_sdefs.h, 545
slu_zdefs.h, 618
print_options
util.c, 727
print_panel_seg
slu_util.h, 601
util.c, 727
print_report
colamd.c, 192
PRINTF
colamd.c, 187
PrintPerf
slu_cdefs.h, 481

slu_ddefs.h, 531
slu_sdefs.h, 545
slu_zdefs.h, 618
PrintStat
superlu_options_t, 29
PrintSumm
slu_util.h, 601
util.c, 727
PRIVATE
colamd.c, 187
old_colamd.c, 371
PUBLIC
colamd.c, 187
old_colamd.c, 371

r
complex, 12
doublecomplex, 14
r_cnjg
scomplex.c, 384
slu_scomplex.h, 540
r_imag
scomplex.c, 384
slu_scomplex.h, 540
RCOND
slu_util.h, 592
Reduce
cmemory.c, 161
dmemory.c, 302
smemory.c, 662
zmemory.c, 783
REFINE
slu_util.h, 592
RefineInitialized
superlu_options_t, 29
RefineSteps
SuperLUStat_t, 30
RELAX
slu_util.h, 592
relax_snode
relax_snode.c, 375
slu_util.h, 601
relax_snode.c
relax_snode, 375
ReplaceTinyPivot
superlu_options_t, 29
resetrep_col
slu_util.h, 601
util.c, 727
ROW
slu_util.h, 591
ROW_IS_ALIVE
colamd.c, 187
old_colamd.c, 371
ROW_IS_DEAD
colamd.c, 187
old_colamd.c, 371
ROW_IS_MARKED_DEAD
colamd.c, 187
old_colamd.c, 371
rowind
NCformat, 21
NCPformat, 22
SCformat, 26
SCPformat, 27
rowind_colbeg
SCPformat, 27
rowind_colend
SCPformat, 27
rowind_colptr
SCformat, 26
RowInfo
old_colamd.c, 371
RowInfo_struct, 25
degree, 25
first_column, 25
length, 25
mark, 25
p, 25
shared1, 25
shared2, 25
start, 25
RowPerm
superlu_options_t, 29
rowperm_t
slu_util.h, 592
rowptr
NRformat, 23
NRformat_loc, 24

sallocateA
slu_sdefs.h, 545
smemory.c, 663
SamePattern
slu_util.h, 591
SamePattern_SameRowPerm
slu_util.h, 591
SCformat, 26
col_to_sup, 26
nnz, 26
nsuper, 26
nzval, 26
nzval_colptr, 26
rowind, 26
rowind_colptr, 26
sup_to_col, 26
scheck_tempv
spanel_bmod.c, 688

sutil.c, 719
scolumn_bmod
scolumn_bmod.c, 378
slu_sdefs.h, 545
scolumn_bmod.c
scolumn_bmod, 378
slsolve, 378
smatvec, 379
susolve, 379
scolumn_dfs
scolumn_dfs.c, 381
slu_sdefs.h, 546
scolumn_dfs.c
scolumn_dfs, 381
T2_SUPER, 381
scomplex.c
c_abs, 384
c_abs1, 384
c_div, 384
c_exp, 384
r_cnjg, 384
r_imag, 384
sCompRow_to_CompCol
slu_sdefs.h, 547
sutil.c, 719
sCopy_CompCol_Matrix
slu_sdefs.h, 547
sutil.c, 719
sCopy_Dense_Matrix
slu_sdefs.h, 547
sutil.c, 719
scopy_to_ucol
scopy_to_ucol.c, 386
slu_sdefs.h, 547
scopy_to_ucol.c
scopy_to_ucol, 386
score
Colamd_Col_struct, 8
ColInfo_struct, 11
SCPformat, 27
col_to_sup, 27
nnz, 27
nsuper, 27
nzval, 27
nzval_colbeg, 27
nzval_colend, 27
rowind, 27
rowind_colbeg, 27
rowind_colend, 27
sup_to_colbeg, 27
sup_to_colend, 27
sCreate_CompCol_Matrix
slu_sdefs.h, 548
sutil.c, 719
sCreate_CompRow_Matrix
slu_sdefs.h, 548
sutil.c, 719
sCreate_Dense_Matrix
slu_sdefs.h, 548
sutil.c, 719
sCreate_SuperNode_Matrix
slu_sdefs.h, 548
sutil.c, 719
scsum1.c
CX, 387
scsum1_, 387
scsum1_
scsum1.c, 387
sDumpLine
sreadhb.c, 699
set_default_options
slu_util.h, 602
util.c, 727
SetIWork
memory.c, 361
slu_util.h, 602
sexpand
smemory.c, 663
sfill
slu_sdefs.h, 548
sutil.c, 720
sFillRHS
slu_sdefs.h, 549
sutil.c, 720
sGenXtrue
slu_sdefs.h, 549
sutil.c, 720
sgscon
sgscon.c, 389
slu_sdefs.h, 549
sgscon.c
sgscon, 389
sgsequ
sgsequ.c, 391
slu_sdefs.h, 550
sgsequ.c
sgsequ, 391
sgsrfs
sgsrfs.c, 395
slu_sdefs.h, 552
sgsrfs.c
ITMAX, 395
sgsrfs, 395
sgssv
sgssv.c, 398
slu_sdefs.h, 554
sgssv.c
sgssv, 398

sgssvx
sgssvx.c, 403
slu_sdefs.h, 559
sgssvx.c
sgssvx, 403
sgstrf
sgstrf.c, 412
slu_sdefs.h, 566
sgstrf.c
sgstrf, 412
sgstrs
sgstrs.c, 418
slu_sdefs.h, 569
sgstrs.c
sgstrs, 418
slsolve, 419
smatvec, 419
sprint_soln, 419
susolve, 419
shared1
Colamd_Col_struct, 8
Colamd_Row_struct, 9
ColInfo_struct, 11
RowInfo_struct, 25
shared2
Colamd_Col_struct, 8
Colamd_Row_struct, 9
ColInfo_struct, 11
RowInfo_struct, 25
shared3
Colamd_Col_struct, 8
ColInfo_struct, 11
shared4
Colamd_Col_struct, 8
ColInfo_struct, 11
shortint
mmd.c, 364
sinf_norm_error
slu_sdefs.h, 571
sutil.c, 720
SINGLE
slu_util.h, 591
size
e_node, 15
LU_stack_t, 19
slacon.c
d_sign, 421
i_dnnt, 421
slacon_, 421
slacon_
slacon.c, 421
slamc1_
slamch.c, 424
slamc2_
slamch.c, 426
slamc3_
slamch.c, 428
slamc4_
slamch.c, 429
slamc5_
slamch.c, 430
slamch.c
abs, 424
dabs, 424
FALSE_, 424
max, 424
min, 424
pow_ri, 424
slamc1_, 424
slamc2_, 426
slamc3_, 428
slamc4_, 429
slamc5_, 430
slamch_, 431
TRUE_, 424
slamch_
slamch.c, 431
slangs
slangs.c, 434
slangs.c
slangs, 434
slaqgs
slaqgs.c, 436
slu_sdefs.h, 571
slaqgs.c
slaqgs, 436
THRESH, 436
slinsol.c
main, 72
slinsol1.c
main, 75
slinsolx.c
main, 78
parse_command_line, 79
slinsolx1.c
main, 82
parse_command_line, 83
slinsolx2.c
main, 86
parse_command_line, 87
slsolve
scolumn_bmod.c, 378
sgstrs.c, 419
smyblas2.c, 670
spanel_bmod.c, 688
ssp_blas2.c, 706
SLU_C
supermatrix.h, 715

SLU_D
supermatrix.h, 715
SLU_DN
supermatrix.h, 716
SLU_GE
supermatrix.h, 716
SLU_HEL
supermatrix.h, 716
SLU_HEU
supermatrix.h, 716
SLU_NC
supermatrix.h, 716
SLU_NCP
supermatrix.h, 716
SLU_NR
supermatrix.h, 716
SLU_NR_loc
supermatrix.h, 716
SLU_S
supermatrix.h, 715
SLU_SC
supermatrix.h, 716
SLU_SCP
supermatrix.h, 716
SLU_SR
supermatrix.h, 716
SLU_SYL
supermatrix.h, 716
SLU_SYU
supermatrix.h, 716
SLU_TRL
supermatrix.h, 716
SLU_TRLU
supermatrix.h, 716
SLU_TRU
supermatrix.h, 716
SLU_TRUU
supermatrix.h, 716
slu_util.h
BOTH, 591
COL, 591
COLAMD, 591
COLPERM, 592
CONJ, 592
DOFACT, 591
DOUBLE, 591
EQUIL, 592
ETREE, 592
EXTRA, 591
FACT, 592
FACTORED, 591
FERR, 592
GEMV, 592
HEAD, 592
LargeDiag, 592
LSUB, 591
LUSUP, 591
MMD_AT_PLUS_A, 590
MMD_ATA, 590
MY_PERMC, 591
MY_PERMR, 592
NATURAL, 590
NO, 592
NOEQUIL, 591
NOREFINE, 591
NOROWPERM, 592
NOTRANS, 592
NPHASES, 592
RCOND, 592
REFINE, 592
RELAX, 592
ROW, 591
SamePattern, 591
SamePattern_SameRowPerm, 591
SINGLE, 591
SOLVE, 592
SYSTEM, 591
TAIL, 592
TRANS, 592
TRSV, 592
UCOL, 591
USER, 591
USUB, 591
YES, 592
SLU_Z
supermatrix.h, 715
slu_cdefs.h
callocateA, 443
ccolumn_bmod, 443
ccolumn_dfs, 443
cCompRow_to_CompCol, 444
cCopy_CompCol_Matrix, 445
cCopy_Dense_Matrix, 445
ccopy_to_ucol, 445
cCreate_CompCol_Matrix, 445
cCreate_CompRow_Matrix, 445
cCreate_Dense_Matrix, 446
cCreate_SuperNode_Matrix, 446
cfill, 446
cFillRHS, 446
cGenXtrue, 446
cgscon, 447
cgsequ, 448
cgsrfs, 450
cgssv, 452
cgssvx, 456
cgstrf, 463
cgstrs, 467

check_tempv, 468
cinf_norm_error, 468
claqgs, 469
cLUMemInit, 470
cLUMemXpand, 471
cLUWorkFree, 472
cmemory_usage, 472
complexCalloc, 472
complexMalloc, 472
countnz, 473
cpanel_bmod, 473
cpanel_dfs, 474
cPivotGrowth, 475
cpivotL, 476
cPrint_CompCol_Matrix, 476
cPrint_Dense_Matrix, 477
cPrint_SuperNode_Matrix, 477
cpruneL, 477
cQuerySpace, 477
creadhb, 477
creadmt, 478
cSetRWork, 478
csnode_bmod, 478
csnode_dfs, 479
fixupL, 479
floatCalloc, 480
floatMalloc, 480
int_t, 443
print_lu_col, 480
PrintPerf, 481
sp_cgemm, 481
sp_cgemv, 482
sp_ctrsv, 484
slu_Cnames.h
ADD_, 486
ADD__, 486
C_CALL, 486
F77_CALL_C, 486
NOCHANGE, 486
UPCASE, 486
slu_dcomplex.h
d_cnjg, 489
d_imag, 489
z_abs, 489
z_abs1, 489
z_add, 488
z_div, 489
z_eq, 488
z_exp, 490
z_sub, 488
zd_mult, 488
zz_conj, 488
zz_mult, 488
slu_ddefs.h
check_tempv, 495
countnz, 495
dallocateA, 495
dcolumn_bmod, 495
dcolumn_dfs, 496
dCompRow_to_CompCol, 497
dCopy_CompCol_Matrix, 497
dCopy_Dense_Matrix, 497
dcopy_to_ucol, 497
dCreate_CompCol_Matrix, 497
dCreate_CompRow_Matrix, 498
dCreate_Dense_Matrix, 498
dCreate_SuperNode_Matrix, 498
dfill, 498
dFillRHS, 498
dGenXtrue, 499
dgscon, 499
dgsequ, 500
dgsrfs, 502
dgssv, 504
dgssvx, 509
dgstrf, 516
dgstrs, 520
dinf_norm_error, 521
dlaqgs, 521
dLUMemInit, 523
dLUMemXpand, 524
dLUWorkFree, 524
dmemory_usage, 524
doubleCalloc, 525
doubleMalloc, 525
dpanel_bmod, 525
dpanel_dfs, 526
dPivotGrowth, 526
dpivotL, 527
dPrint_CompCol_Matrix, 528
dPrint_Dense_Matrix, 528
dPrint_SuperNode_Matrix, 528
dpruneL, 528
dQuerySpace, 528
dreadhb, 529
dreadmt, 529
dSetRWork, 530
dsnode_bmod, 530
dsnode_dfs, 530
fixupL, 531
int_t, 495
print_lu_col, 531
PrintPerf, 531
sp_dgemm, 531
sp_dgemv, 533
sp_dtrsv, 534
slu_scomplex.h
c_abs, 539

c_abs1, 539
c_add, 538
c_div, 539
c_eq, 538
c_exp, 540
c_sub, 538
cc_conj, 538
cc_mult, 538
cs_mult, 539
r_cnjg, 540
r_imag, 540
slu_sdefs.h
check_tempv, 545
countnz, 545
fixupL, 545
floatCalloc, 545
floatMalloc, 545
int_t, 545
print_lu_col, 545
PrintPerf, 545
sallocateA, 545
scolumn_bmod, 545
scolumn_dfs, 546
sCompRow_to_CompCol, 547
sCopy_CompCol_Matrix, 547
sCopy_Dense_Matrix, 547
scopy_to_ucol, 547
sCreate_CompCol_Matrix, 548
sCreate_CompRow_Matrix, 548
sCreate_Dense_Matrix, 548
sCreate_SuperNode_Matrix, 548
sfill, 548
sFillRHS, 549
sGenXtrue, 549
sgscon, 549
sgsequ, 550
sgsrfs, 552
sgssv, 554
sgssvx, 559
sgstrf, 566
sgstrs, 569
sinf_norm_error, 571
slaqgs, 571
sLUMemInit, 573
sLUMemXpand, 573
sLUWorkFree, 574
smemory_usage, 574
sp_sgemm, 574
sp_sgemv, 576
sp_strsv, 578
spanel_bmod, 579
spanel_dfs, 580
sPivotGrowth, 581
spivotL, 582
sPrint_CompCol_Matrix, 582
sPrint_Dense_Matrix, 582
sPrint_SuperNode_Matrix, 582
spruneL, 582
sQuerySpace, 583
sreadhb, 583
sreadmt, 584
sSetRWork, 584
ssnode_bmod, 584
ssnode_dfs, 584
slu_util.h
ABORT, 588
CHECK_MALLOC, 588
check_repfnz, 593
colperm_t, 590
Destroy_CompCol_Matrix, 593
Destroy_CompCol_Permuted, 593
Destroy_CompRow_Matrix, 593
Destroy_Dense_Matrix, 593
Destroy_SuperMatrix_Store, 593
Destroy_SuperNode_Matrix, 594
DiagScale_t, 591
EMPTY, 589
fact_t, 591
FALSE, 590
FIRSTCOL_OF_SNODE, 590
flops_t, 590
get_perm_c, 594
heap_relax_snode, 595
ifill, 596
intCalloc, 597
intMalloc, 597
IterRefine_t, 591
L_FST_SUPC, 590
L_NZ_START, 590
L_SUB, 590
L_SUB_START, 590
Logical, 590
lsame_, 598
LU_space_t, 591
MemType, 591
NO_MARKER, 590
NUM_TEMPV, 590
PhaseType, 591
print_panel_seg, 601
PrintSumm, 601
relax_snode, 601
resetrep_col, 601
rowperm_t, 592
set_default_options, 602
SetIWork, 602
snode_profile, 603
sp_coletree, 603
sp_ienv, 604

sp_preorder, 607
spcoletree, 608
stack_end_t, 592
StatFree, 608
StatInit, 608
StatPrint, 609
super_stats, 609
superlu_abort_and_exit, 609
SUPERLU_FREE, 590
superlu_free, 609
SUPERLU_MALLOC, 590
superlu_malloc, 609
SUPERLU_MAX, 590
SUPERLU_MIN, 590
SuperLU_timer_, 609
trans_t, 592
TreePostorder, 610
TRUE, 590
U_NZ_START, 590
U_SUB, 590
USER_ABORT, 590
USER_FREE, 590
USER_MALLOC, 590
xerbla_, 610
yes_no_t, 592
slu_zdefs.h
check_tempv, 617
countnz, 617
doubleCalloc, 617
doublecomplexCalloc, 617
doublecomplexMalloc, 617
doubleMalloc, 617
fixupL, 618
int_t, 617
print_lu_col, 618
PrintPerf, 618
sp_zgemm, 618
sp_zgemv, 620
sp_ztrsv, 621
zallocateA, 623
zcolumn_bmod, 623
zcolumn_dfs, 624
zCompRow_to_CompCol, 625
zCopy_CompCol_Matrix, 625
zCopy_Dense_Matrix, 625
zcopy_to_ucol, 625
zCreate_CompCol_Matrix, 626
zCreate_CompRow_Matrix, 626
zCreate_Dense_Matrix, 626
zCreate_SuperNode_Matrix, 626
zfill, 627
zFillRHS, 627
zGenXtrue, 627
zgscon, 627
zgsequ, 629
zgsrfs, 631
zgssv, 633
zgssvx, 637
zgstrf, 644
zgstrs, 648
zinf_norm_error, 649
zlaqgs, 649
zLUMemInit, 651
zLUMemXpand, 652
zLUWorkFree, 652
zmemory_usage, 652
zpanel_bmod, 653
zpanel_dfs, 653
zPivotGrowth, 654
zpivotL, 655
zPrint_CompCol_Matrix, 656
zPrint_Dense_Matrix, 656
zPrint_SuperNode_Matrix, 656
zpruneL, 656
zQuerySpace, 656
zreadhb, 657
zreadmt, 657
zSetRWork, 658
zsnode_bmod, 658
zsnode_dfs, 658
sLUMemInit
slu_sdefs.h, 573
smemory.c, 664
sLUMemXpand
slu_sdefs.h, 573
smemory.c, 665
sLUWorkFree
slu_sdefs.h, 574
smemory.c, 666
sLUWorkInit
smemory.c, 666
smatvec
scolumn_bmod.c, 379
sgstrs.c, 419
smyblas2.c, 670
spanel_bmod.c, 688
ssp_blas2.c, 706
smemory.c
copy_mem_float, 662
copy_mem_int, 662
DoubleAlign, 662
expanders, 669
ExpHeader, 662
floatCalloc, 663
floatMalloc, 663
GluIntArray, 662
no_expand, 669
NO_MEMTYPE, 662

NotDoubleAlign, 662
Reduce, 662
sallocateA, 663
sexpand, 663
sLUMemInit, 664
sLUMemXpand, 665
sLUWorkFree, 666
sLUWorkInit, 666
smemory_usage, 666
sQuerySpace, 667
sSetRWork, 667
sSetupSpace, 668
sStackCompress, 668
stack, 669
StackFull, 662
suser_free, 668
suser_malloc, 668
TempSpace, 662
user_bcopy, 669
smemory_usage
slu_sdefs.h, 574
smemory.c, 666
smyblas2.c
slsolve, 670
smatvec, 670
susolve, 671
snode_profile
slu_util.h, 603
SOLVE
slu_util.h, 592
SolveInitialized
superlu_options_t, 29
sp_cgemm
csp_blas3.c, 223
slu_cdefs.h, 481
sp_cgemv
csp_blas2.c, 219
slu_cdefs.h, 482
sp_coletree
slu_util.h, 603
sp_coletree.c, 676
sp_coletree.c
etdfs, 673
finalize_disjoint_sets, 673
find, 673
initialize_disjoint_sets, 674
link, 674
make_set, 675
mxCallocInt, 675
nr_etdfs, 676
sp_coletree, 676
sp_symetree, 677
TreePostorder, 678
sp_ctrsv
csp_blas2.c, 220
slu_cdefs.h, 484
sp_dgemm
dsp_blas3.c, 335
slu_ddefs.h, 531
sp_dgemv
dsp_blas2.c, 331
slu_ddefs.h, 533
sp_dtrsv
dsp_blas2.c, 332
slu_ddefs.h, 534
sp_ienv
EXAMPLE/sp_ienv.c, 681
slu_util.h, 604
SRC/sp_ienv.c, 679
sp_preorder
slu_util.h, 607
sp_preorder.c, 684
sp_preorder.c
check_perm, 684
sp_preorder, 684
sp_sgemm
slu_sdefs.h, 574
ssp_blas3.c, 710
sp_sgemv
slu_sdefs.h, 576
ssp_blas2.c, 706
sp_strsv
slu_sdefs.h, 578
ssp_blas2.c, 707
sp_symetree
sp_coletree.c, 677
sp_zgemm
slu_zdefs.h, 618
zsp_blas3.c, 816
sp_zgemv
slu_zdefs.h, 620
zsp_blas2.c, 812
sp_ztrsv
slu_zdefs.h, 621
zsp_blas2.c, 813
spanel_bmod
slu_sdefs.h, 579
spanel_bmod.c, 688
spanel_bmod.c
scheck_tempv, 688
slsolve, 688
smatvec, 688
spanel_bmod, 688
spanel_dfs
slu_sdefs.h, 580
spanel_dfs.c, 689
spanel_dfs.c
spanel_dfs, 689

sParseFloatFormat
sreadhb.c, 699
sParseIntFormat
sreadhb.c, 699
SpaSize
util.c, 728
spcoletree
slu_util.h, 608
sPivotGrowth
slu_sdefs.h, 581
spivotgrowth.c, 691
spivotgrowth.c
sPivotGrowth, 691
spivotL
slu_sdefs.h, 582
spivotL.c, 693
spivotL.c
spivotL, 693
sPrint_CompCol_Matrix
slu_sdefs.h, 582
sutil.c, 721
sPrint_Dense_Matrix
slu_sdefs.h, 582
sutil.c, 721
sprint_lu_col
sutil.c, 721
sprint_soln
sgstrs.c, 419
sPrint_SuperNode_Matrix
slu_sdefs.h, 582
sutil.c, 721
sPrintPerf
sutil.c, 721
spruneL
slu_sdefs.h, 582
spruneL.c, 695
spruneL.c
spruneL, 695
sQuerySpace
slu_sdefs.h, 583
smemory.c, 667
SRC/ccolumn_bmod.c, 110
SRC/ccolumn_dfs.c, 113
SRC/ccopy_to_ucol.c, 116
SRC/cgscon.c, 118
SRC/cgsequ.c, 121
SRC/cgsrfs.c, 124
SRC/cgssv.c, 128
SRC/cgssvx.c, 133
SRC/cgstrf.c, 142
SRC/cgstrs.c, 148
SRC/clacon.c, 152
SRC/clangs.c, 154
SRC/claqgs.c, 156
SRC/cmemory.c, 159
SRC/cmyblas2.c, 171
SRC/colamd.c, 173
SRC/colamd.h, 194
SRC/cpanel_bmod.c, 200
SRC/cpanel_dfs.c, 202
SRC/cpivotgrowth.c, 204
SRC/cpivotL.c, 206
SRC/cpruneL.c, 208
SRC/creadhb.c, 210
SRC/csnode_bmod.c, 214
SRC/csnode_dfs.c, 216
SRC/csp_blas2.c, 218
SRC/csp_blas3.c, 223
SRC/cutil.c, 226
SRC/dcolumn_bmod.c, 231
SRC/dcolumn_dfs.c, 234
SRC/dcomplex.c, 237
SRC/dcopy_to_ucol.c, 239
SRC/dGetDiagU.c, 241
SRC/dgscon.c, 243
SRC/dgsequ.c, 246
SRC/dgsrfs.c, 249
SRC/dgssv.c, 253
SRC/dgssvx.c, 258
SRC/dgstrf.c, 267
SRC/dgstrs.c, 273
SRC/dgstrsL.c, 277
SRC/dgstrsU.c, 280
SRC/dlacon.c, 283
SRC/dlamch.c, 285
SRC/dlangs.c, 295
SRC/dlaqgs.c, 297
SRC/dmemory.c, 300
SRC/dmyblas2.c, 310
SRC/dpanel_bmod.c, 312
SRC/dpanel_dfs.c, 314
SRC/dpivotgrowth.c, 316
SRC/dpivotL.c, 318
SRC/dpruneL.c, 320
SRC/dreadhb.c, 322
SRC/dsnode_bmod.c, 326
SRC/dsnode_dfs.c, 328
SRC/dsp_blas2.c, 330
SRC/dsp_blas3.c, 335
SRC/dutil.c, 338
SRC/dzsum1.c, 343
SRC/get_perm_c.c, 345
SRC/heap_relax_snode.c, 350
SRC/icmax1.c, 352
SRC/izmax1.c, 354
SRC/lsame.c, 356
SRC/memory.c, 358
SRC/mmd.c, 364

SRC/old_colamd.c, 368
SRC/old_colamd.h, 373
SRC/relax_snode.c, 375
SRC/scolumn_bmod.c, 377
SRC/scolumn_dfs.c, 380
SRC/scomplex.c, 383
SRC/scopy_to_ucol.c, 385
SRC/scsum1.c, 387
SRC/sgscon.c, 389
SRC/sgsequ.c, 391
SRC/sgsrfs.c, 394
SRC/sgssv.c, 398
SRC/sgssvx.c, 403
SRC/sgstrf.c, 412
SRC/sgstrs.c, 417
SRC/slacon.c, 421
SRC/slamch.c, 423
SRC/slangs.c, 434
SRC/slaqgs.c, 436
SRC/slu_cdefs.h, 439
SRC/slu_Cnames.h, 486
SRC/slu_dcomplex.h, 487
SRC/slu_ddefs.h, 491
SRC/slu_scomplex.h, 537
SRC/slu_sdefs.h, 541
SRC/slu_util.h, 586
SRC/slu_zdefs.h, 613
SRC/smemory.c, 660
SRC/smyblas2.c, 670
SRC/sp_coletree.c, 672
SRC/sp_ienv.c, 679
    sp_ienv, 679
SRC/sp_preorder.c, 684
SRC/spanel_bmod.c, 687
SRC/spanel_dfs.c, 689
SRC/spivotgrowth.c, 691
SRC/spivotL.c, 693
SRC/spruneL.c, 695
SRC/sreadhb.c, 697
SRC/ssnode_bmod.c, 701
SRC/ssnode_dfs.c, 703
SRC/ssp_blas2.c, 705
SRC/ssp_blas3.c, 710
SRC/superlu_timer.c, 713
SRC/supermatrix.h, 715
SRC/sutil.c, 717
SRC/util.c, 722
SRC/xerbla.c, 730
SRC/zcolumn_bmod.c, 733
SRC/zcolumn_dfs.c, 736
SRC/zcopy_to_ucol.c, 739
SRC/zgscon.c, 741
SRC/zgsequ.c, 744
SRC/zgsrfs.c, 747
SRC/zgssv.c, 751
SRC/zgssvx.c, 756
SRC/zgstrf.c, 765
SRC/zgstrs.c, 770
SRC/zlacon.c, 774
SRC/zlangs.c, 776
SRC/zlaqgs.c, 778
SRC/zmemory.c, 781
SRC/zmyblas2.c, 791
SRC/zpanel_bmod.c, 793
SRC/zpanel_dfs.c, 795
SRC/zpivotgrowth.c, 797
SRC/zpivotL.c, 799
SRC/zpruneL.c, 801
SRC/zreadhb.c, 803
SRC/zsnode_bmod.c, 807
SRC/zsnode_dfs.c, 809
SRC/zsp_blas2.c, 811
SRC/zsp_blas3.c, 816
SRC/zutil.c, 819
sreadhb
    slu_sdefs.h, 583
    sreadhb.c, 699
sreadhb.c
    sDumpLine, 699
    sParseFloatFormat, 699
    sParseIntFormat, 699
    sreadhb, 699
    sReadValues, 699
    sReadVector, 700
sreadmt
    slu_sdefs.h, 584
sReadValues
    sreadhb.c, 699
sReadVector
    sreadhb.c, 700
sSetRWork
    slu_sdefs.h, 584
    smemory.c, 667
sSetupSpace
    smemory.c, 668
ssnode_bmod
    slu_sdefs.h, 584
    ssnode_bmod.c, 702
ssnode_bmod.c
    ssnode_bmod, 702
ssnode_dfs
    slu_sdefs.h, 584
    ssnode_dfs.c, 703
ssnode_dfs.c
    ssnode_dfs, 703
ssp_blas2.c
    slsolve, 706
    smatvec, 706

sp_sgemv, 706
sp_strsv, 707
susolve, 709
ssp_blas3.c
sp_sgemm, 710
sStackCompress
smemory.c, 668
stack
cmemory.c, 170
dmemory.c, 309
smemory.c, 669
zmemory.c, 790
stack_end_t
slu_util.h, 592
StackFull
cmemory.c, 161
dmemory.c, 302
smemory.c, 662
zmemory.c, 783
start
Colamd_Col_struct, 8
Colamd_Row_struct, 9
ColInfo_struct, 11
RowInfo_struct, 25
StatFree
slu_util.h, 608
util.c, 728
StatInit
slu_util.h, 608
util.c, 728
StatPrint
slu_util.h, 609
util.c, 728
Store
SuperMatrix, 31
Stype
SuperMatrix, 31
Stype_t
supermatrix.h, 716
sup_to_col
SCformat, 26
sup_to_colbeg
SCPformat, 27
sup_to_colend
SCPformat, 27
super_stats
slu_util.h, 609
util.c, 728
superlu.c
main, 90
superlu_abort_and_exit
slu_util.h, 609
util.c, 728
SUPERLU_FREE
slu_util.h, 590
superlu_free
memory.c, 362
slu_util.h, 609
SUPERLU_MALLOC
slu_util.h, 590
superlu_malloc
memory.c, 362
slu_util.h, 609
SUPERLU_MAX
slu_util.h, 590
SUPERLU_MIN
slu_util.h, 590
superlu_options_t, 28
ColPerm, 29
ConditionNumber, 29
DiagPivotThresh, 29
Equil, 29
Fact, 29
IterRefine, 29
PivotGrowth, 29
PrintStat, 29
RefineInitialized, 29
ReplaceTinyPivot, 29
RowPerm, 29
SolveInitialized, 29
SymmetricMode, 29
Trans, 29
superlu_timer.c
CLK_TCK, 714
SuperLU_timer_, 714
SuperLU_timer_
slu_util.h, 609
superlu_timer.c, 714
SuperLUStat_t, 30
ops, 30
panel_histo, 30
RefineSteps, 30
TinyPivots, 30
utime, 30
SuperMatrix, 31
Dtype, 31
Mtype, 31
ncol, 31
nrow, 31
Store, 31
Stype, 31
supermatrix.h
Dtype_t, 715
Mtype_t, 715
SLU_C, 715
SLU_D, 715
SLU_DN, 716
SLU_GE, 716

SLU_HEL, 716
SLU_HEU, 716
SLU_NC, 716
SLU_NCP, 716
SLU_NR, 716
SLU_NR_loc, 716
SLU_S, 715
SLU_SC, 716
SLU_SCP, 716
SLU_SR, 716
SLU_SYL, 716
SLU_SYU, 716
SLU_TRL, 716
SLU_TRLU, 716
SLU_TRU, 716
SLU_TRUU, 716
SLU_Z, 715
Stype_t, 716
supno
GlobalLU_t, 18
suser_free
smemory.c, 668
suser_malloc
smemory.c, 668
susolve
scolumn_bmod.c, 379
sgstrs.c, 419
smyblas2.c, 671
ssp_blas2.c, 709
sutil.c
print_float_vec, 719
scheck_tempv, 719
sCompRow_to_CompCol, 719
sCopy_CompCol_Matrix, 719
sCopy_Dense_Matrix, 719
sCreate_CompCol_Matrix, 719
sCreate_CompRow_Matrix, 719
sCreate_Dense_Matrix, 719
sCreate_SuperNode_Matrix, 719
sfill, 720
sFillRHS, 720
sGenXtrue, 720
sinf_norm_error, 720
sPrint_CompCol_Matrix, 721
sPrint_Dense_Matrix, 721
sprint_lu_col, 721
sPrint_SuperNode_Matrix, 721
sPrintPerf, 721
symamd
colamd.c, 192
colamd.h, 199
symamd_report
colamd.c, 193
colamd.h, 199
SymmetricMode
superlu_options_t, 29
SYSTEM
slu_util.h, 591

T2_SUPER
ccolumn_dfs.c, 114
dcolumn_dfs.c, 235
scolumn_dfs.c, 381
zcolumn_dfs.c, 737
TAIL
slu_util.h, 592
TempSpace
cmemory.c, 161
dmemory.c, 302
smemory.c, 662
zmemory.c, 783
thickness
Colamd_Col_struct, 8
ColInfo_struct, 11
THRESH
claqgs.c, 156
dlaqgs.c, 297
slaqgs.c, 436
zlaqgs.c, 778
TinyPivots
SuperLUStat_t, 30
top1
LU_stack_t, 19
top2
LU_stack_t, 19
total_needed
mem_usage_t, 20
TRANS
slu_util.h, 592
Trans
superlu_options_t, 29
trans_t
slu_util.h, 592
TreePostorder
slu_util.h, 610
sp_coletree.c, 678
TRSV
slu_util.h, 592
TRUE
colamd.c, 187
old_colamd.c, 371
slu_util.h, 590
TRUE_
dlamch.c, 286
slamch.c, 424

U_NZ_START
slu_util.h, 590

U_SUB
slu_util.h, 590
UCOL
slu_util.h, 591
ucol
GlobalLU_t, 18
UPCASE
slu_Cnames.h, 486
used
LU_stack_t, 19
USER
slu_util.h, 591
USER_ABORT
slu_util.h, 590
user_bcopy
cmemory.c, 169
dmemory.c, 308
memory.c, 362
smemory.c, 669
zmemory.c, 784
USER_FREE
slu_util.h, 590
USER_MALLOC
slu_util.h, 590
USUB
slu_util.h, 591
usub
GlobalLU_t, 18
util.c
check_repfnz, 724
countnz, 724
DenseSize, 724
Destroy_CompCol_Matrix, 724
Destroy_CompCol_Permuted, 724
Destroy_CompRow_Matrix, 725
Destroy_Dense_Matrix, 725
Destroy_SuperMatrix_Store, 725
Destroy_SuperNode_Matrix, 726
fixupL, 726
ifill, 726
LUFactFlops, 726
LUSolveFlops, 727
max_sup_size, 729
NBUCKS, 724
print_int_vec, 727
print_options, 727
print_panel_seg, 727
PrintSumm, 727
resetrep_col, 727
set_default_options, 727
SpaSize, 728
StatFree, 728
StatInit, 728
StatPrint, 728
super_stats, 728
superlu_abort_and_exit, 728
utime
SuperLUStat_t, 30

xerbla.c
xerbla_, 732
xerbla_
slu_util.h, 610
xerbla.c, 732
xlsub
GlobalLU_t, 18
xlusup
GlobalLU_t, 18
xsup
GlobalLU_t, 18
xusub
GlobalLU_t, 18

YES
slu_util.h, 592
yes_no_t
slu_util.h, 592

z_abs
dcomplex.c, 238
slu_dcomplex.h, 489
z_abs1
dcomplex.c, 238
slu_dcomplex.h, 489
z_add
slu_dcomplex.h, 488
z_div
dcomplex.c, 238
slu_dcomplex.h, 489
z_eq
slu_dcomplex.h, 488
z_exp
dcomplex.c, 238
slu_dcomplex.h, 490
z_sub
slu_dcomplex.h, 488
zallocateA
slu_zdefs.h, 623
zmemory.c, 784
zcheck_tempv
zpanel_bmod.c, 794
zutil.c, 821
zcolumn_bmod
slu_zdefs.h, 623
zcolumn_bmod.c, 734
zcolumn_bmod.c
zcolumn_bmod, 734
zlsolve, 734

zmatvec, 735
zusolve, 735
zcolumn_dfs
slu_zdefs.h, 624
zcolumn_dfs.c, 737
zcolumn_dfs.c
T2_SUPER, 737
zcolumn_dfs, 737
zCompRow_to_CompCol
slu_zdefs.h, 625
zutil.c, 821
zCopy_CompCol_Matrix
slu_zdefs.h, 625
zutil.c, 821
zCopy_Dense_Matrix
slu_zdefs.h, 625
zutil.c, 821
zcopy_to_ucol
slu_zdefs.h, 625
zcopy_to_ucol.c, 740
zcopy_to_ucol.c
zcopy_to_ucol, 740
zCreate_CompCol_Matrix
slu_zdefs.h, 626
zutil.c, 821
zCreate_CompRow_Matrix
slu_zdefs.h, 626
zutil.c, 821
zCreate_Dense_Matrix
slu_zdefs.h, 626
zutil.c, 821
zCreate_SuperNode_Matrix
slu_zdefs.h, 626
zutil.c, 821
zd_mult
slu_dcomplex.h, 488
zDumpLine
zreadhb.c, 805
zexpand
zmemory.c, 784
zfill
slu_zdefs.h, 627
zutil.c, 822
zFillRHS
slu_zdefs.h, 627
zutil.c, 822
zGenXtrue
slu_zdefs.h, 627
zutil.c, 822
zgscon
slu_zdefs.h, 627
zgscon.c, 741
zgscon.c
zgscon, 741
zgsequ
slu_zdefs.h, 629
zgsequ.c, 744
zgsequ.c
zgsequ, 744
zgsrfs
slu_zdefs.h, 631
zgsrfs.c, 748
zgsrfs.c
ITMAX, 748
zgsrfs, 748
zgssv
slu_zdefs.h, 633
zgssv.c, 751
zgssv.c
zgssv, 751
zgssvx
slu_zdefs.h, 637
zgssvx.c, 756
zgssvx.c
zgssvx, 756
zgstrf
slu_zdefs.h, 644
zgstrf.c, 765
zgstrf.c
zgstrf, 765
zgstrs
slu_zdefs.h, 648
zgstrs.c, 771
zgstrs.c
zgstrs, 771
zlsolve, 772
zmatvec, 772
zprint_soln, 772
zusolve, 772
zinf_norm_error
slu_zdefs.h, 649
zutil.c, 822
zlacon.c
zlacon_, 774
zlacon_
zlacon.c, 774
zlangs
zlangs.c, 776
zlangs.c
zlangs, 776
zlaqgs
slu_zdefs.h, 649
zlaqgs.c, 778
zlaqgs.c
THRESH, 778
zlaqgs, 778
zlinsol.c
main, 93

zlinsol1.c
main, 96
zlinsolx.c
main, 99
parse_command_line, 100
zlinsolx1.c
main, 103
parse_command_line, 104
zlinsolx2.c
main, 107
parse_command_line, 108
zlsolve
zcolumn_bmod.c, 734
zgstrs.c, 772
zmyblas2.c, 791
zpanel_bmod.c, 794
zsp_blas2.c, 814
zLUMemInit
slu_zdefs.h, 651
zmemory.c, 785
zLUMemXpand
slu_zdefs.h, 652
zmemory.c, 786
zLUWorkFree
slu_zdefs.h, 652
zmemory.c, 787
zLUWorkInit
zmemory.c, 787
zmatvec
zcolumn_bmod.c, 735
zgstrs.c, 772
zmyblas2.c, 792
zpanel_bmod.c, 794
zsp_blas2.c, 814
zmemory.c
copy_mem_doublecomplex, 783
copy_mem_int, 783
DoubleAlign, 783
doublecomplexCalloc, 784
doublecomplexMalloc, 784
expanders, 790
ExpHeader, 783
GluIntArray, 783
no_expand, 790
NO_MEMTYPE, 783
NotDoubleAlign, 783
Reduce, 783
stack, 790
StackFull, 783
TempSpace, 783
user_bcopy, 784
zallocateA, 784
zexpand, 784
zLUMemInit, 785
zLUMemXpand, 786
zLUWorkFree, 787
zLUWorkInit, 787
zmemory_usage, 787
zQuerySpace, 788
zSetRWork, 788
zSetupSpace, 789
zStackCompress, 789
zuser_free, 789
zuser_malloc, 789
zmemory_usage
slu_zdefs.h, 652
zmemory.c, 787
zmyblas2.c
zlsolve, 791
zmatvec, 792
zusolve, 792
zpanel_bmod
slu_zdefs.h, 653
zpanel_bmod.c, 794
zpanel_bmod.c
zcheck_tempv, 794
zlsolve, 794
zmatvec, 794
zpanel_bmod, 794
zpanel_dfs
slu_zdefs.h, 653
zpanel_dfs.c, 795
zpanel_dfs.c
zpanel_dfs, 795
zParseFloatFormat
zreadhb.c, 805
zParseIntFormat
zreadhb.c, 805
zPivotGrowth
slu_zdefs.h, 654
zpivotgrowth.c, 797
zpivotgrowth.c
zPivotGrowth, 797
zpivotL
slu_zdefs.h, 655
zpivotL.c, 799
zpivotL.c
zpivotL, 799
zPrint_CompCol_Matrix
slu_zdefs.h, 656
zutil.c, 823
zPrint_Dense_Matrix
slu_zdefs.h, 656
zutil.c, 823
zprint_lu_col
zutil.c, 823
zprint_soln
zgstrs.c, 772

zPrint_SuperNode_Matrix
slu_zdefs.h, 656
zutil.c, 823
zPrintPerf
zutil.c, 823
zpruneL
slu_zdefs.h, 656
zpruneL.c, 801
zpruneL.c
zpruneL, 801
zQuerySpace
slu_zdefs.h, 656
zmemory.c, 788
zreadhb
slu_zdefs.h, 657
zreadhb.c, 805
zreadhb.c
zDumpLine, 805
zParseFloatFormat, 805
zParseIntFormat, 805
zreadhb, 805
zReadValues, 805
zReadVector, 806
zreadmt
slu_zdefs.h, 657
zreadtriple
zreadtriple.c, 109
zreadtriple.c
zreadtriple, 109
zReadValues
zreadhb.c, 805
zReadVector
zreadhb.c, 806
zSetRWork
slu_zdefs.h, 658
zmemory.c, 788
zSetupSpace
zmemory.c, 789
zsnode_bmod
slu_zdefs.h, 658
zsnode_bmod.c, 808
zsnode_bmod.c
zsnode_bmod, 808
zsnode_dfs
slu_zdefs.h, 658
zsnode_dfs.c, 809
zsnode_dfs.c
zsnode_dfs, 809
zsp_blas2.c
sp_zgemv, 812
sp_ztrsv, 813
zlsolve, 814
zmatvec, 814
zusolve, 815
zsp_blas3.c
sp_zgemm, 816
zStackCompress
zmemory.c, 789
zuser_free
zmemory.c, 789
zuser_malloc
zmemory.c, 789
zusolve
zcolumn_bmod.c, 735
zgstrs.c, 772
zmyblas2.c, 792
zsp_blas2.c, 815
zutil.c
print_doublecomplex_vec, 821
zcheck_tempv, 821
zCompRow_to_CompCol, 821
zCopy_CompCol_Matrix, 821
zCopy_Dense_Matrix, 821
zCreate_CompCol_Matrix, 821
zCreate_CompRow_Matrix, 821
zCreate_Dense_Matrix, 821
zCreate_SuperNode_Matrix, 821
zfill, 822
zFillRHS, 822
zGenXtrue, 822
zinf_norm_error, 822
zPrint_CompCol_Matrix, 823
zPrint_Dense_Matrix, 823
zprint_lu_col, 823
zPrint_SuperNode_Matrix, 823
zPrintPerf, 823
zz_conj
slu_dcomplex.h, 488
zz_mult
slu_dcomplex.h, 488