SyFi - A package for symbolic finite element computations

Kent-Andre Mardal

February 28, 2008

Outline - Short Description of SyFi

Purpose

- SyFi is a tool for defining polygons, polynomial spaces, degrees of freedom, and finite elements
- SyFi makes it easy to define weak forms (differentiation and integration of polynomials over polygons)
- SyFi generates efficient C++ code for the computation of matrices

Dependencies

- SyFi relies heavily on GiNaC and Swiginac for the symbolic computations and code generation
- SyFi can generate matrices based on either a Dolfin or a Diffpack mesh (we plan to include other meshes soon)
- SyFi can generate either Epetra or PyCC matrices (we plan to include other matrices soon)
- We are working on generating UFC compliant code (Alnæs' talk)

Short Description of GiNaC and Swiginac

GiNaC

- GiNaC is a C++ library for symbolic mathematics
- Authors: C. Bauer, C. Dams A. Frink, V. Kisil, R. Kreckel, A. Sheplyakov, J. Vollinga
- URL: www.ginac.de
- License : GPL

Swiginac

- Swiginac is a Python interface to GiNaC
- Authors: O. Skavhaug and O. Certik
- URL: http://swiginac.berlios.de/
- License: Open

Swiginac Code Example

```
from swiginac import *
x = symbol('x'); y = symbol('y')

f = x*x*y*y
print "f = ", f

dfdx = diff(f,x)
print "df/dx = ", dfdx

intf = integral(x,0,1,f).eval_integ()
print "integral of f from 0 to 1: ", intf

g = cos(x)
print "Taylor series of cos(x) around x == 0.5: ",
print g.series(x == 0.5, 10)
```

SyFi Extends GiNaC/Swiginac

GiNaC/Swiginac supports:

- Polynomials
- Differentiation with respect to one variable
- Integration with respect to one variable

Basic Extensions in SyFi:

- Polynomial spaces (such as Legendre and Bernstein)
- Differentiations with respect to several variables
- Integration over polygons

$\rightarrow$ SyFi extends GiNaC/Swiginac with the ingredients typically needed in finite element methods

Demo: Bernstein polynomials on a Triangle

```
from swiginac import *
from SyFi import *
x = cvar.x; y = cvar.y

t = ReferenceTriangle()

B = bernstein(2, t, 'a')
print "2. order Bernstein polynomial on a triangle ", B

P = B[0]
dPdx = diff(P, x)
print "Derivative with respect to x ", dPdx

int_dPdx = t.integrate(dPdx)
print "Integral of dPdx over an triangle ", int_dPdx

int_line1_dPdx = t.line(1).integrate(dPdx)
print "Integral of dPdx over an edge",  int_line1_dPdx
```

Elements currently implemented in SyFi

Finite Elements

- continuous and discontinuous Lagrangian elements (arbitrary order)
- Nedelec elements (arbitrary order)
- Nedelec H(div) elements (arbitrary order)
- Raviart-Thomas elements (arbitrary order)
- Crouzeix-Raviart elements
- Hermite elements
- Bubble elements
- Arnold-Falk-Winther elasticity element (weak symmetry) (arbitrary order)

Evaluation of Weak Forms in SyFi

We will now demonstrate the computation of various element matrices in SyFi

- The mass matrix on a reference triangle:

$$\boldsymbol{M}_{ij} = \int_T N_i \, N_j \, dx$$

- The stiffness matrix on a mapped tetrahedron:

$$\boldsymbol{A}_{ij} = \int_T (J^{-1} \nabla N_i) \cdot (J^{-1} \nabla N_j) \, dx$$

where J is the Jacobian of the geometry mapping

Demo: Computing the Mass Matrix on the Reference Triangle

```
from swiginac import *
from SyFi import *

t = ReferenceTriangle()
fe = LagrangeFE(t, 3)

for i in range(0, fe.nbf()):
   for j in range(0, fe.nbf()):
      Aij = t.integrate(fe.N(i)*fe.N(i))
```

Demo: Computing the Stiffness Matrix on a Mapped Tethrahedron

```
from swiginac import *
from SyFi import *

t = ReferenceTetrahedron()
fe = LagrangeFE(t, 3)

J = symbolic_matrix(3,3, ‘‘J’’)

for i in range(0, fe.nbf()):
   for j in range(0, fe.nbf()):
      integrand = inner(grad(J, fe.N(i)),
                        grad(J, fe.N(j)))
      Aij = t.integrate(integrand)
```

The Computation of the Jacobian matrix

We will now demonstrate the computation the Jacobian matrix for a nonlinear PDE

- $\boldsymbol{u} = \sum_j u_j \boldsymbol{N}_j$
- Nonlinear convection diffusion equation

$$\boldsymbol{F}_i = \int_T (\boldsymbol{u} \cdot \nabla \boldsymbol{u}) \cdot \boldsymbol{N}_i + \nabla \boldsymbol{u} : \nabla \boldsymbol{N}_i \, dx$$

- The Jacobian matrix

$$\boldsymbol{J}_{ij} = \frac{\partial \boldsymbol{F}_i}{\partial u_j}$$

SyFi Code Example : Convection-Diffusion Matrix

```
.. initialize element

for i in range(0,fe.nbf()):

  # compute diffusion term
  fi_diffusion = inner(grad(u), grad(fe.N(i)))

  # compute convection term
  uxgradu = (u.transpose()*grad(u)).evalm()
  fi_convection = inner(uxgradu, fe.N(i), True)

  fi = fi_diffusion + fi_convection
  Fi = polygon.integrate(fi)

  for j in range(0,fe.nbf()):
    # differentiate to get the Jacobian
    Jij = diff(Fi, uj)
    print "J[%d,%d]=%s "%(i,j,Jij)
```

Convection and Power-law Diffusion

Now consider the Power-law model

- Nonlinear convection diffusion equation

$$\boldsymbol{F}_i = \int_T (\boldsymbol{u} \cdot \nabla \boldsymbol{u}) \cdot \boldsymbol{N}_i + \mu(\boldsymbol{u}) \nabla \boldsymbol{u} : \nabla \boldsymbol{N}_i) \, dx$$

- The viscosity:

$$\mu = \mu_0 \|\nabla \boldsymbol{u}\|^{2n}$$

SyFi Code Example : Convection and Power-law Diffusion

```
n = symbol("n")

for i in range(0,fe.nbf()):

  # nonlinear power-law diffusion term
  mu = mu0*inner(grad(u), grad(u))
  fi_diffusion = pow(mu,n)*inner(grad(u), grad(fe.N(i)))

  # nonlinear convection term
  uxgradu = (u.transpose()*grad(u)).evalm()
  fi_convection = inner(uxgradu, fe.N(i), True)

  fi = fi_diffusion + fi_convection

  Fi = polygon.integrate(fi)

  for j in range(0,fe.nbf()):
    # differentiate to get the Jacobian
    Jij = diff(Fi, uj)
    print "J[%d,%d]=%s\n"%(i,j, Jij.evalf().printc())
```

Inlining of C++ in Python

The module Instant

- We have created a Python module for inlining of C++ code in Python
- It employs SWIG for the generation of wrapper code and distutils for compiling extenstion modules

```
from Instant import inline

add_func = inline("double add(double a, double b){ return a+b; }")

print "The sum of 3 and 4.5 is ", add_func(3, 4.5)
```

UFC compliant code generation

UFC

- UFC - Unified Form-assembly Code
- Low level spesification of class declaration and function signatures for finite elements, element tensors etc.
- Together with Logg, Alnæs, Skavhaug and Langtangen we are working on spesifing UFC
- Alnæs will talk more about it later today

Using SyFi/UFC within PyCC

```
import stuff..

n = 10
m = Mesh.UnitSquare(n,n)

mf = UMF.UserMapMatrixFactory()
mf.select_mesh(m)


polygon = SyFi.ReferenceTriangle()
fe = SyFi.LagrangeFE(polygon, 1)

def mass_integrand(u, v, G, Ginv):
    return inner(u, v)

mass_form = UfcCG.BiLinearForm(mass_integrand, name = "mass" )

compiled_fe, compiled_form = UfcCG.gen_extension_module(fe, mass_form)

A = mf.compute_matrix(compiled_form)
```

Concluding Remarks

Present Use of SyFi

- We have implemented a set of simple test examples within SyFi/PyCC for the Poisson problem, Stokes problem, convection-diffusion problems and Navier-Stokes equations
- We have used SyFi/PyCC in rather advanced applications concerning the electrical activity of the heart

Future

- Support other meshes and other matrices
- Implement more elements
- Continue developing solvers for fluid and solid mechanics