Package 'flashier'

October 17, 2023

Type Package

Date 2023-10-16

Title Empirical Bayes Matrix Factorization

Version 1.0.7

URL https://github.com/willwerscheid/flashier

BugReports https://github.com/willwerscheid/flashier/issues

Description Methods for matrix factorization based on Wang and Stephens (2021) <https://jmlr.org/papers/v22/20-589.html>.

Depends R (>= 3.4), ebnm (>= 0.1-21), magrittr

Imports Matrix, parallel, dplyr, stringr, tibble, tidyr, softImpute, irlba, ggplot2

Suggests ashr, cowplot, testthat, knitr, rmarkdown, RcppML, rsvd

License BSD_3_clause + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

VignetteBuilder knitr

NeedsCompilation no

Author Jason Willwerscheid [aut, cre], Peter Carbonetto [aut], Wei Wang [aut], Matthew Stephens [aut], Eric Weine [ctb], Gao Wang [ctb]

Maintainer Jason Willwerscheid <jwillwer@providence.edu>

Repository CRAN

Date/Publication 2023-10-17 09:40:02 UTC

R topics documented:

fitted.flash *Fitted method for flash objects*

Description

Given a flash object, returns the "fitted values" $E(LF') = E(L)E(F)'$.

Usage

```
## S3 method for class 'flash'
fitted(object, ...)
```

Arguments

object	An object inheriting from class flash.
...	Additional parameters are ignored.

Value

The matrix of "fitted values."

fitted.flash_fit *Fitted method for flash fit objects*

Description

Given a flash_fit object, returns the "fitted values" $E(LF') = E(L)E(F)'$.

Usage

```
## S3 method for class 'flash_fit'
fitted(object, ...)
```

Arguments

object	An object inheriting from class flash_fit.
...	Additional parameters are ignored.

Value

The matrix of "fitted values."

`flash` *Empirical Bayes matrix factorization*

Description

Fits an empirical Bayes matrix factorization (see **Details** for a description of the model). The resulting fit is referred to as a "flash" object (short for Factors and Loadings using Adaptive SHrinkage). Two interfaces are provided. The `flash` function provides a simple interface that allows a flash object to be fit in a single pass, while `flash_xxx` functions are pipeable functions that allow for more complex flash objects to be fit incrementally (available functions are listed below under **See Also**). See the vignettes and **Examples** for usage.

Usage

```
flash(
  data,
  S = NULL,
  ebnm_fn = ebnm_point_normal,
  var_type = 0L,
  greedy_Kmax = 50L,
  backfit = FALSE,
  nullcheck = TRUE,
  verbose = 1L
)
```

Arguments

`data`	The observations. Usually a matrix, but can also be a sparse matrix of class `Matrix` or a low-rank matrix representation as returned by, for example, `svd`, `irlba`, `rsvd`, or `softImpute` (in general, any list that includes fields `u`, `d`, and `v` will be interpreted as a low-rank matrix representation).
`S`	The standard errors. Can be `NULL` (in which case all residual variance will be estimated) or a matrix, vector, or scalar. `S` should be a scalar if standard errors are identical across observations. It should be a vector if standard errors either vary across columns but are constant within any given row, or vary across rows but are constant within any given column (`flash` will use the length of the vector to determine whether the supplied values correspond to rows or columns; if the data matrix is square, then the sense must be specified using parameter `S_dim` in function `flash_init`).
`ebnm_fn`	The function or functions used to solve the empirical Bayes normal means (EBNM) subproblems. Most importantly, these functions specify the families of distributions $G_\ell^{(k)}$ and $G_f^{(k)}$ to which the priors on loadings and factors $g_\ell^{(k)}$ and $g_f^{(k)}$ are assumed to belong. If the same function is to be used for both loadings L and factors F, then `ebnm_fn` can be a single function. If one function is to be used for loadings and a second for factors, then `ebnm_fn` should be a list of length two, with the first element giving the function for loadings and the second

the function for factors. If different functions are to be used for different values of k, then factor/loadings pairs must be added successively using multiple calls to either flash_greedy or flash_factors_init.

Any EBNM function provided by package ebnm can be used as input. Non-default arguments to parameters can be supplied using the helper function flash_ebnm. Custom EBNM functions can also be used: for details, see flash_ebnm.

`var_type` Describes the structure of the estimated residual variance. Can be NULL, 0, 1, 2, or c(1, 2). If NULL, then S accounts for all residual variance. If var_type = 0, then the estimated residual variance (which is added to any variance given by S) is assumed to be constant across all observations. Setting var_type = 1 estimates a single variance parameter for each row; var_type = 2 estimates one parameter for each column; and var_type = c(1, 2) optimizes over all rank-one matrices (that is, it assumes that the residual variance parameter s_{ij} can be written $s_{ij} = a_i b_j$, where the n-vector a and the p-vector b are to be estimated).

Note that if any portion of the residual variance is to be estimated, then it is usually faster to set S = NULL and to let flash estimate all of the residual variance. Further, var_type = c(1, 2) is typically much slower than other options, so it should be used with care.

`greedy_Kmax` The maximum number of factors to be added. This will not necessarily be the total number of factors added by flash, since factors are only added as long as they increase the variational lower bound on the log likelihood for the model.

`backfit` A "greedy" fit is performed by adding up to greedy_Kmax factors, optimizing each newly added factor in one go without returning to optimize previously added factors. When backfit = TRUE, flash will additionally perform a final "backfit" where all factors are cyclically updated until convergence. The backfitting procedure typically takes much longer than the greedy algorithm, but it also usually improves the final fit to a significant degree.

`nullcheck` If nullcheck = TRUE, then flash will check that each factor in the final flash object improves the overall fit. Any factor that fails the check will be removed.

`verbose` When and how to display progress updates. Set to 0 for none, 1 for updates after a factor is added or a backfit is completed, 2 for additional notifications about the variational lower bound, and 3 for updates after every iteration. It is also possible to output a single tab-delimited table of values using function flash_set_verbose with verbose = -1.

Details

If Y is an $n \times p$ data matrix, then the rank-one empirical Bayes matrix factorization model is:

$$Y = \ell f' + E,$$

where ℓ is an n-vector of **loadings**, f is a p-vector of **factors**, and E is an $n \times p$ matrix of **residuals** (or "errors"). Additionally:

$$e_{ij} \sim N(0, s_{ij}^2) : i = 1, ..., n; j = 1, ..., p$$

$$\ell \sim g_\ell \in G_\ell$$

$$f \sim g_f \in G_f.$$

The residual variance parameters s_{ij}^2 are constrained to have a simple structure and are fit via maximum likelihood. (For example, one might assume that all standard errors are identical: $s_{ij}^2 = s^2$ for some s^2 and for all i, j). The functions g_ℓ and g_f are assumed to belong to some families of priors G_ℓ and G_f that are specified in advance, and are estimated via variational approximation.

The general rank-K empirical Bayes matrix factorization model is:

$$Y = LF' + E$$

or

$$y_{ij} = \sum_k \ell_{ik} f_{jk} + e_{ij} : i = 1, ..., n; j = 1, ..., p,$$

where L is now a matrix of loadings and F is a matrix of factors.

Separate priors $g_\ell^{(k)}$ and $g_f^{(k)}$ are estimated via empirical Bayes, and different prior families may be used for different values of k. In general, then:

$$e_{ij} \sim N(0, s_{ij}^2) : i = 1, ..., n; j = 1, ..., p$$

$$\ell_{ik} \sim g_\ell^{(k)} \in G_\ell^{(k)} : i = 1, ..., n; k = 1, ..., K$$

$$f_{ik} \sim g_f^{(k)} \in G_f^{(k)} : j = 1, ..., p; k = 1, ..., K.$$

Typically, $G_\ell^{(k)}$ and $G_f^{(k)}$ will be closed under scaling, in which case ℓ_k and f_k are only identifiable up to a **scaling factor** d_k. In other words, we can write:

$$Y = LDF' + E,$$

where D is a diagonal matrix with diagonal entries $d_1, ..., d_K$. The model can then be made identifiable by constraining the scale of ℓ_k and f_k for $k = 1, ..., K$.

Value

A `flash` object. Contains elements:

`n_factors` The total number of factor/loadings pairs K in the fitted model.

`pve` The proportion of variance explained by each factor/loadings pair. Since factors and loadings are not required to be orthogonal, this should be interpreted loosely: for example, the total proportion of variance explained could be larger than 1.

`elbo` The variational lower bound achieved by the fitted model.

`residuals_sd` Estimated residual standard deviations (these include any variance component given as an argument to `S`).

`L_pm`, `L_psd`, `L_lfsr` Posterior means, standard deviations, and local false sign rates for loadings L.

`F_pm`, `F_psd`, `F_lfsr` Posterior means, standard deviations, and local false sign rates for factors F.

`L_ghat` The fitted priors on loadings $\hat{g}_\ell^{(k)}$.

`F_ghat` The fitted priors on factors $\hat{g}_f^{(k)}$.

`sampler` A function that takes a single argument `nsamp` and returns `nsamp` samples from the posterior distributions for factors F and loadings L.

`flash_fit` A `flash_fit` object. Used by `flash` when fitting is not performed all at once, but incrementally via calls to various `flash_xxx` functions.

The following methods are available:

`fitted.flash` Returns the "fitted values" $E(LF') = E(L)E(F)'$.

`residuals.flash` Returns the expected residuals $Y - E(LF') = Y - E(L)E(F)'$.

`ldf.flash` Returns an LDF decomposition (see **Details** above), with columns of L and F scaled as specified by the user.

References

Wei Wang and Matthew Stephens (2021). "Empirical Bayes matrix factorization." *Journal of Machine Learning Research* 22, 1–40.

See Also

`flash_init`, `flash_greedy`, `flash_backfit`, and `flash_nullcheck`. For more advanced functionality, see `flash_factors_init`, `flash_factors_fix`, `flash_factors_set_to_zero`, `flash_factors_remove`, `flash_set_verbose`, and `flash_set_conv_crit`. For extracting useful data from `flash` objects, see `fitted.flash`, `residuals.flash`, and `ldf.flash`.

Examples

```
data(gtex)

# Fit up to 3 factors and backfit.
fl <- flash(gtex, greedy_Kmax = 3L, backfit = TRUE)

# This is equivalent to the series of calls:
fl <- flash_init(gtex) %>%
  flash_greedy(Kmax = 3L) %>%
  flash_backfit() %>%
  flash_nullcheck()

# Fit a unimodal distribution with mean zero to each set of loadings
#   and a scale mixture of normals with mean zero to each factor.
fl <- flash(gtex,
            ebnm_fn = c(ebnm_unimodal,
                        ebnm_normal_scale_mixture),
            greedy_Kmax = 3)

# Fit point-laplace priors using a non-default optimization method.
fl <- flash(gtex,
            ebnm_fn = flash_ebnm(prior_family = "point_laplace",
                                 optmethod = "trust"),
            greedy_Kmax = 3)

# Fit a "Kronecker" (rank-one) variance structure (this can be slow).
```

```
fl <- flash(gtex, var_type = c(1, 2), greedy_Kmax = 3L)
```

`flash_add_intercept` *Add "intercept" to a flash object*

Description

Adds an all-ones vector as a fixed set of loadings (if `rowwise = TRUE`) or fixed factor (if `rowwise = FALSE`). Assuming (without loss of generality) that the fixed factor/loadings is indexed as $k = 1$, a fixed set of loadings gives:

$$\mathbf{y}_{i\cdot} \approx \mathbf{f}_1 + \sum_{k=2}^{K} \ell_{ik}\mathbf{f}_k,$$

so that the (estimated) factor $\mathbf{f}_1 \in \mathbf{R}^p$ is shared by all row-wise observations $\mathbf{y}_{i\cdot} \in \mathbf{R}^p$. A fixed factor gives:

$$\mathbf{y}_{\cdot j} \approx \boldsymbol{\ell}_1 + \sum_{k=2}^{K} f_{jk}\boldsymbol{\ell}_k,$$

so that the (estimated) set of loadings $\ell_1 \in \mathbf{R}^n$ is shared by all column-wise observations $y_{\cdot j} \in \mathbf{R}^n$.

Usage

```
flash_add_intercept(flash, rowwise = TRUE, ebnm_fn = ebnm_point_normal)
```

Arguments

`flash`	A `flash` or `flash_fit` object to which an "intercept" is to be added.
`rowwise`	Should the all-ones vector be added as a fixed set of loadings ("row-wise") or a fixed factor ("column-wise")? See above for details.
`ebnm_fn`	As with other factor/loadings pairs, a prior is put on the estimated factor (if `rowwise = TRUE`) or set of loadings (if `rowwise = FALSE`). Parameter `ebnm_fn` specifies the function used to estimate that prior; see `flash` for details.

Details

The estimated factor (if `rowwise = TRUE`) or set of loadings (if `rowwise = FALSE`) is initialized at the column- or row-wise means of the data (or, if factor/loadings pairs have previously been added, at the column- or row-wise means of the matrix of residuals) and then backfit via function `flash_backfit`.

Value

The `flash` object from argument `flash`, with an "intercept" added.

Examples

```
# The following are equivalent:
init <- list(matrix(rowMeans(gtex), ncol = 1),
             matrix(1, nrow = ncol(gtex)))
fl <- flash_init(gtex) %>%
  flash_factors_init(init) %>%
  flash_factors_fix(kset = 1, which_dim = "factors") %>%
  flash_backfit(kset = 1)

fl <- flash_init(gtex) %>%
  flash_add_intercept(rowwise = FALSE)
```

`flash_backfit` *Backfit a flash object*

Description

Backfits existing flash factor/loadings pairs. Whereas a "greedy" fit optimizes each newly added factor/loadings pair in one go without returning to optimize previously added pairs, a "backfit" updates all existing pairs in a cyclical fashion. See `flash` for examples of usage.

Usage

```
flash_backfit(
  flash,
  kset = NULL,
  extrapolate = TRUE,
  warmstart = TRUE,
  maxiter = 500,
  tol = NULL,
  verbose = NULL
)
```

Arguments

`flash`	A `flash` or `flash_fit` object.
`kset`	A vector of integers specifying which factors to backfit. If `kset = NULL`, then all existing factors will be backfitted.
`extrapolate`	Whether to use an extrapolation technique inspired by Ang and Gillis (2019) to accelerate the fitting process. Control parameters are handled via global options and can be set by calling `options("extrapolate.control") <- control.param`.
`warmstart`	Whether to use "warmstarts" when solving the EBNM subproblems by initializing solutions at the previous value of the fitted prior $\hat{g}$. An important side effect of warmstarts for `ashr`-like prior families is to fix the grid at its initial setting. Fixing the grid can lead to poor fits if there are large changes in the scale of the estimated prior over the course of the fitting process. However, allowing the grid to vary can occasionally result in decreases in ELBO.

`maxiter` The maximum number of backfitting iterations. An "iteration" is defined such that all factors in `kset` get updated at each iteration.

`tol` The convergence tolerance parameter. After each update, the fit is compared to the fit from before the update using a convergence criterion function (by default, the difference in ELBO, but the criterion can be changed via `flash_set_conv_crit`). The backfit is considered to have "converged" when the value of the convergence criterion function over successive updates to *all* factor/loadings pairs is less than or equal to `tol`. If, for example, factor/loadings pairs $1, \ldots, K$ are being sequentially backfitted, then fits are compared before and after the update to factor/loadings 1, before and after the update to factor/loadings 2, and so on through factor/loadings K, and backfitting only terminates when the convergence criterion function returns a value less than or equal to `tol` for all K updates. Note that specifying `tol` here will override any value set by `flash_set_conv_crit`; to use the "global" tolerance parameter, `tol` must be left unspecified (`NULL`). If `tol = NULL` and a global tolerance parameter has not been set, then the default tolerance used is $np\sqrt{\epsilon}$, where n is the number of rows in the dataset, p is the number of columns, and ϵ is equal to `.Machine$double.eps`.

`verbose` When and how to display progress updates. Set to `0` for none, `1` for updates after a factor is added or a backfit is completed, `2` for additional notifications about the variational lower bound, and `3` for updates after every iteration. It is also possible to output a single tab-delimited table of values using function `flash_set_verbose` with `verbose = -1`.

Value

The `flash` object from argument `flash`, backfitted as specified.

`flash_clear_timeout` *Set timeout*

Description

Used in a `flash` pipeline to clear timeout conditions set using `flash_set_timeout`.

Usage

```
flash_clear_timeout(flash)
```

Arguments

`flash` A `flash` or `flash_fit` object.

Value

The `flash` object from argument `flash`, with timeout settings cleared.

flash_conv_crit_elbo_diff *Calculate the difference in ELBO*

Description

The default objective function used to determine convergence when fitting a flash object. Calculates the difference in the variational lower bound ("ELBO") from one iteration to the next.

Usage

```
flash_conv_crit_elbo_diff(curr, prev, k)
```

Arguments

curr	The flash_fit object from the current iteration.
prev	The flash_fit object from the previous iteration.
k	Only used during sequential backfits (that is, calls to flash_backfit where extrapolate = FALSE). It then takes the index of the factor/loadings pair currently being optimized.

Details

This function is an example of a function that may be passed to parameter fn in function flash_set_conv_crit to set the convergence criterion for a flash pipeline. See flash_set_conv_crit for details and examples.

Value

A scalar, which is compared against the tolerance parameter tol to determine whether a fit has converged.

See Also

flash_conv_crit_max_chg flash_conv_crit_max_chg_L, flash_conv_crit_max_chg_F

flash_conv_crit_max_chg

Calculate the maximum absolute difference in scaled loadings and factors

Description

An alternative objective function that can be used to determine convergence when fitting a `flash` object. Calculates the maximum (absolute) change over all (posterior expected values for) loadings ℓ_{ik} and factors f_{jk}. At each iteration, the loadings vectors $\ell_{\cdot 1}, \ldots, \ell_{\cdot K}$ and factors $f_{\cdot 1}, \ldots, f_{\cdot K}$ are L^2-normalized.

Usage

```
flash_conv_crit_max_chg(curr, prev, k)
```

Arguments

`curr`	The `flash_fit` object from the current iteration.
`prev`	The `flash_fit` object from the previous iteration.
`k`	Only used during sequential backfits (that is, calls to `flash_backfit` where `extrapolate = FALSE`). It then takes the index of the factor/loadings pair currently being optimized.

Value

A scalar, which is compared against the tolerance parameter `tol` to determine whether a fit has converged.

See Also

`flash_conv_crit_elbo_diff`, `flash_conv_crit_max_chg_L` `flash_conv_crit_max_chg_F`

flash_conv_crit_max_chg_F

Calculate the maximum absolute difference in scaled factors

Description

An alternative objective function that can be used to determine convergence when fitting a `flash` object. Calculates the maximum (absolute) change over all (posterior expected values for) factors f_{jk}. At each iteration, the factors $f_{\cdot 1}, \ldots, f_{\cdot K}$ are L^2-normalized.

Usage

```
flash_conv_crit_max_chg_F(curr, prev, k)
```

Arguments

curr	The flash_fit object from the current iteration.
prev	The flash_fit object from the previous iteration.
k	Only used during sequential backfits (that is, calls to flash_backfit where extrapolate = FALSE). It then takes the index of the factor/loadings pair currently being optimized.

Value

A scalar, which is compared against the tolerance parameter tol to determine whether a fit has converged.

See Also

flash_conv_crit_elbo_diff, flash_conv_crit_max_chg flash_conv_crit_max_chg_L

---

flash_conv_crit_max_chg_L

Calculate the maximum absolute difference in scaled loadings

Description

An alternative objective function that can be used to determine convergence when fitting a flash object. Calculates the maximum (absolute) change over all (posterior expected values for) loadings ℓ_{ik}. At each iteration, the loadings vectors $\ell_{\cdot 1}, \ldots, \ell_{\cdot K}$ are L^2-normalized.

Usage

```
flash_conv_crit_max_chg_L(curr, prev, k)
```

Arguments

curr	The flash_fit object from the current iteration.
prev	The flash_fit object from the previous iteration.
k	Only used during sequential backfits (that is, calls to flash_backfit where extrapolate = FALSE). It then takes the index of the factor/loadings pair currently being optimized.

Value

A scalar, which is compared against the tolerance parameter tol to determine whether a fit has converged.

See Also

flash_conv_crit_elbo_diff, flash_conv_crit_max_chg flash_conv_crit_max_chg_F

`flash_ebnm` *Construct an EBNM function*

Description

`flash_ebnm` is a helper function that provides readable syntax for constructing `ebnm` functions that can serve as arguments to parameter `ebnm_fn` in functions `flash`, `flash_greedy`, and `flash_factors_init` (see **Examples** below). It is also possible to write a custom function from scratch: see **Details** below for a simple example. A more involved example can be found in the "Advanced flashier" vignette.

Usage

```
flash_ebnm(...)
```

Arguments

`...`	Parameters to be passed to function `ebnm` in package `ebnm`. An argument to `prior_family` should be provided unless the default family of point-normal priors is desired. Arguments to parameters `x`, `s`, or `output` must not be included. Finally, if `g_init` is included, then `fix_g = TRUE` must be as well. To fix a prior grid, use parameter `scale` rather than `g_init`.

Details

As input to parameter `ebnm_fn` in functions `flash`, `flash_greedy`, and `flash_factors_init`, it should suffice for many purposes to provide functions from package `ebnm` as is (for example, one might set `ebnm_fn = ebnm_point_laplace`). To use non-default arguments, function `flash_ebnm` may be used (see **Examples**). Custom functions may also be written. In general, any function that is used as an argument to `ebnm_fn` must accept parameters:

`x` A vector of observations.

`s` A vector of standard errors, or a scalar if all standard errors are equal.

`g_init` The prior g. Usually, this is left unspecified (`NULL`) and estimated from the data. If it is supplied and `fix_g = TRUE`, then the prior is fixed at `g_init`; if `fix_g = FALSE`, then `g_init` gives the initial value of g used during optimization.

In `flashier`, g is fixed during the wrap-up phase when estimating local false sign rates and constructing a sampler; and `g_init` is used with `fix_g = FALSE` to "warmstart" backfits (see `flash_backfit`). If none of these features (local false sign rates, samplers, or warmstarts) are needed, then both `g_init` and `fix_g` can be ignored (the EBNM function must still accept them as parameters, but it need not do anything with their arguments).

`fix_g` If `TRUE`, the prior is fixed at `g_init` instead of estimated. See the description of `g_init` above.

`output` A character vector indicating which values are to be returned. Custom EBNM functions can safely ignore this parameter (again, they must accept it as a parameter, but they do not need to do anything with its argument).

The return object must be a list that includes fields:

`posterior` A data frame that includes columns `mean` and `second_moment` (the first and second moments for each posterior distribution $p(\theta_i \mid s_i, \hat{g}), i = 1, ..., n$). Optionally, a column `lfsr` giving local false sign rates may also be included.

`fitted_g` The estimated prior $\hat{g}$. Within `flashier`, `fitted_g` is only ever used as an argument to `g_init` in subsequent calls to the same EBNM function, so the manner in which it is represented is unimportant.

`log_likelihood` The optimal log likelihood $L(\hat{g}) := \sum_i \log p(x_i \mid \hat{g}, s_i)$.

`posterior_sampler` An optional field containing a function that samples from the posterior distributions of the "means" θ_i. If included, the function should take a single parameter `nsamp` and return a matrix where rows correspond to samples and columns correspond to observations (that is, there should be `nsamp` rows and n columns).

Value

A function that can be passed as argument to parameter `ebnm_fn` in functions `flash`, `flash_greedy`, and `flash_factors_init`.

See Also

`ebnm`

Examples

```
# A custom EBNM function might be written as follows:
my_ebnm_fn <- function(x, s, g_init, fix_g, output) {
  ebnm_res <- ebnm_point_laplace(
    x = x,
    s = s,
    g_init = g_init,
    fix_g = fix_g,
    output = output,
    control = list(iterlim = 10)
  )
  return(ebnm_res)
}

# The following are equivalent:
fl1 <- flash(
  gtex,
  ebnm_fn = my_ebnm_fn,
  greedy_Kmax = 2
)
fl2 <- flash(
  gtex,
  ebnm_fn = flash_ebnm(
    prior_family = "point_laplace",
    control = list(iterlim = 10)
  ),
  greedy_Kmax = 2
)
```

`flash_factors_fix` *Fix flash factors*

Description

Fixes loadings $\ell_{\cdot k}$ or factors $f_{\cdot k}$ for one or more factor/loadings pairs, so that their values are not updated during subsequent backfits. For a given pair, either the loadings or factor can be fixed, but not both, and either all entries or a subset can be fixed. To unfix, use function `flash_factors_unfix`. See `flash_factors_init` for an example of usage.

Usage

```
flash_factors_fix(
  flash,
  kset,
  which_dim = c("factors", "loadings"),
  fixed_idx = NULL,
  use_fixed_in_ebnm = NULL
)
```

Arguments

`flash`	A `flash` or `flash_fit` object.
`kset`	A vector of integers indexing the factor/loadings pairs whose loadings or factors are to be fixed.
`which_dim`	Whether to fix factors or loadings.
`fixed_idx`	If `fixed_idx = NULL`, then all loadings or factor values will be fixed. If only a subset are to be fixed, then `fixed_idx` should be an appropriately-sized vector or matrix of values that can be coerced to logical. For example, if a subset of loadings for two factor/loadings pairs are to be fixed, then `fixed_idx` should be a length-n vector or an n by 2 matrix (where n is the number of rows in the data matrix).
`use_fixed_in_ebnm`	By default, fixed elements are ignored when solving the EBNM subproblem in order to estimate the prior $\hat{g}$. This behavior can be changed by setting `use_fixed_in_ebnm = TRUE`. This is a global setting which applies to all factor/loadings pairs; behavior cannot vary from one factor/loadings pair to another.

Value

The `flash` object from argument `flash`, with factors or loadings fixed as specified.

`flash_factors_init` *Initialize flash factors at specified values*

Description

Initializes factor/loadings pairs at values specified by `init`. This function has two primary uses: 1. One can initialize multiple factor/loadings pairs at once using an SVD-like function and then optimize them via function `flash_backfit`. Sometimes this results in a better fit than adding them one at a time via `flash_greedy`. 2. One can initialize factor/loadings pairs and then fix the factor (or loadings) via function `flash_factors_fix` to incorporate "known" factors into a `flash` object. See below for examples of both use cases.

Usage

```
flash_factors_init(flash, init, ebnm_fn = ebnm_point_normal)
```

Arguments

`flash`	A `flash` or `flash_fit` object to which factors are to be added.
`init`	An SVD-like object (specifically, a list containing fields `u`, `d`, and `v`), a `flash` or `flash_fit` object, or a list of matrices specifying the values at which factors and loadings are to be initialized (for a data matrix of size $n \times p$, this should be a list of length two, with the first element a matrix of size $n \times k$ and the second a matrix of size $p \times k$). If a flash fit is supplied, then it will be used to initialize both the first and second moments of posteriors on loadings and factors. Otherwise, the supplied values will be used to initialize posterior means, with posterior second moments initialized as the squared values of the first moments. Missing entries are not allowed.
`ebnm_fn`	The function or functions used to solve the empirical Bayes normal means (EBNM) subproblems. Most importantly, these functions specify the families of distributions $G_\ell^{(k)}$ and $G_f^{(k)}$ to which the priors on loadings and factors $g_\ell^{(k)}$ and $g_f^{(k)}$ are assumed to belong. If the same function is to be used for both loadings L and factors F, then `ebnm_fn` can be a single function. If one function is to be used for loadings and a second for factors, then `ebnm_fn` should be a list of length two, with the first element giving the function for loadings and the second the function for factors. If different functions are to be used for different values of k, then factor/loadings pairs must be added successively using multiple calls to either `flash_greedy` or `flash_factors_init`. Any EBNM function provided by package `ebnm` can be used as input. Non-default arguments to parameters can be supplied using the helper function `flash_ebnm`. Custom EBNM functions can also be used: for details, see `flash_ebnm`.

Value

The `flash` object from argument `flash`, with factors and loadings initialized as specified.

Examples

```
# Initialize several factors at once and backfit.
fl <- flash_init(gtex) %>%
  flash_factors_init(init = svd(gtex, nu = 5, nv = 5)) %>%
  flash_backfit()

# Add fixed loadings with \ell_i identically equal to one. This can be
#   interpreted as giving a "mean" factor that accounts for different
#   row-wise means.
ones <- matrix(1, nrow = nrow(gtex), ncol = 1)
# Initialize the factor at the least squares solution.
ls_soln <- t(solve(crossprod(ones), crossprod(ones, gtex)))
fl <- flash_init(gtex) %>%
  flash_factors_init(init = list(ones, ls_soln)) %>%
  flash_factors_fix(kset = 1, which_dim = "loadings") %>%
  flash_backfit() %>%
  flash_greedy(Kmax = 5L)
```

---

flash_factors_remove *Remove factors from a flash object*

Description

Sets factor/loadings pairs to zero and then removes them from the flash object. Note that this will change the indices of existing pairs.

Usage

```
flash_factors_remove(flash, kset)
```

Arguments

flash	A flash or flash_fit object.
kset	A vector of integers specifying which factor/loadings pairs to remove.

Value

The flash object from argument flash, with the factors specified by kset removed.

See Also

flash_factors_set_to_zero

flash_factors_reorder *Reorder factors in a flash object*

Description

Reorders the factor/loadings pairs in a flash object.

Usage

```
flash_factors_reorder(flash, kset)
```

Arguments

flash	A flash or flash_fit object.
kset	A vector of integers specifying the new order of the factor/loadings pairs. All existing factors must be included in kset; to drop factors, use flash_factors_remove.

Value

The flash object from argument flash, with the factors reordered according to argument kset.

flash_factors_set_to_zero *Set flash factors to zero*

Description

Sets factor/loadings pairs to zero but does not remove them from the flash object (so as to keep the indices of existing pairs the same).

Usage

```
flash_factors_set_to_zero(flash, kset)
```

Arguments

flash	A flash or flash_fit object.
kset	A vector of integers specifying which factor/loadings pairs to set to zero.

Value

The flash object from argument flash, with the factors specified by kset set to zero.

See Also

flash_factors_remove

flash_factors_unfix *Unfix flash factors*

Description

If loadings $\ell_{\cdot k}$ or factors $f_{\cdot k}$ for one or more factor/loadings pairs have been "fixed" using function flash_factors_fix, then they can be unfixed using function flash_factors_unfix.

Usage

```
flash_factors_unfix(flash, kset)
```

Arguments

flash	A flash or flash_fit object.
kset	A vector of integers indexing the factor/loadings pairs whose values are to be unfixed.

Value

The flash object from argument flash, with values for the factor/loadings pairs specified by kset unfixed.

flash_fit *Extract a flash_fit object*

Description

flash_fit objects are the "internal" objects used by flash functions to fit an EBMF model. Whereas flash objects (the end results of the fitting process) include user-friendly fields and methods, flash_fit objects were not designed for public consumption and can be unwieldy. Nonetheless, some advanced flash functionality requires the wielding of flash_fit objects. In particular, initialization, convergence, and "verbose" display functions all take one or more flash_fit objects as input (see parameter init_fn in function flash_greedy; parameter fn in flash_set_conv_crit; and parameter fns in flash_set_verbose). For users who would like to write custom functions, the accessor functions and methods enumerated below may prove useful. See flash_set_verbose for an example.

Usage

```
flash_fit(flash)

flash_fit_get_pm(f, n)

flash_fit_get_p2m(f, n)
```

```
flash_fit_get_est_tau(f)

flash_fit_get_fixed_tau(f)

flash_fit_get_tau(f)

flash_fit_get_elbo(f)

flash_fit_get_KL(f, n)

flash_fit_get_g(f, n)
```

Arguments

flash	A flash object.
f	A flash_fit object.
n	Set n = 1 to access loadings L and n = 2 to access factors F).

Details

The following S3 methods are available for flash_fit objects at all times except while optimizing new factor/loadings pairs as part of a "greedy" fit:

fitted.flash_fit Returns the "fitted values" $E(LF') = E(L)E(F)'$.

residuals.flash_fit Returns the expected residuals $Y - E(LF') = Y - E(L)E(F)'$.

ldf.flash_fit Returns an LDF decomposition, with columns of L and F scaled as specified by the user.

Value

See function descriptions below.

Functions

- flash_fit_get_pm(): The posterior means for the loadings matrix L (when parameter n is equal to 1) or factor matrix F (when n = 2). While optimizing new factor/loadings pairs as part of a "greedy" fit, only the posterior means for the new loadings $\ell_{\cdot k}$ or factor $f_{\cdot k}$ will be returned.
- flash_fit_get_p2m(): The posterior second moments for the loadings matrix L (when parameter n is equal to 1) or factor matrix F (when n = 2). While optimizing new factor/loadings pairs, only the posterior second moments for the new loadings $\ell_{\cdot k}$ or factor $f_{\cdot k}$ will be returned.
- flash_fit_get_est_tau(): Equal to $1/\sigma^2$, where σ^2 is the estimated portion of the residual variance (total, by row, or by column, depending on the variance type).
- flash_fit_get_fixed_tau(): Equal to $1/s^2$, where s^2 is the fixed portion of the residual variance (total, by row, or by column).
- flash_fit_get_tau(): The overall precision $1/(\sigma^2 + s^2)$.

- `flash_fit_get_elbo()`: The variational lower bound (ELBO).
- `flash_fit_get_KL()`: A vector containing the KL-divergence portions of the ELBO, with one value for each factor (when `n = 2`) or set of loadings (when `n = 1`). While optimizing new factor/loadings pairs, only the KL-divergence for the new factor or loadings will be returned.
- `flash_fit_get_g()`: A list containing estimated priors on loadings $\hat{g}_\ell$ (when `n = 1`) or factors $\hat{g}_f$ (when `n = 2`). While optimizing new factor/loadings pairs, only the estimated prior on the new factor or loadings will be returned.

`flash_greedy` *Greedily add factors to a flash object*

Description

Adds factor/loadings pairs to a flash object in a "greedy" manner. Up to `Kmax` pairs are added one at a time. At each step, `flash_greedy` attempts to find an optimal additional (rank-one) factor given all previously added factors. The additional factor is retained if it increases the variational lower bound (ELBO); otherwise, fitting terminates. See `flash` for examples of usage.

Usage

```
flash_greedy(
  flash,
  Kmax = 1,
  ebnm_fn = ebnm_point_normal,
  init_fn = NULL,
  extrapolate = FALSE,
  warmstart = FALSE,
  maxiter = 500,
  tol = NULL,
  verbose = NULL
)
```

Arguments

`flash`	A `flash` or `flash_fit` object to which factors are to be added.
`Kmax`	The maximum number of factors to be added. This will not necessarily be the total number of factors added by `flash_greedy`, since factors are only added as long as they increase the ELBO.
`ebnm_fn`	The function or functions used to solve the empirical Bayes normal means (EBNM) subproblems. Most importantly, these functions specify the families of distributions $G_\ell^{(k)}$ and $G_f^{(k)}$ to which the priors on loadings and factors $g_\ell^{(k)}$ and $g_f^{(k)}$ are assumed to belong. If the same function is to be used for both loadings L and factors F, then `ebnm_fn` can be a single function. If one function is to be used for loadings and a second for factors, then `ebnm_fn` should be a list of length two, with the first element giving the function for loadings and the second

the function for factors. If different functions are to be used for different values of k, then factor/loadings pairs must be added successively using multiple calls to either `flash_greedy` or `flash_factors_init`.

Any EBNM function provided by package `ebnm` can be used as input. Non-default arguments to parameters can be supplied using the helper function `flash_ebnm`. Custom EBNM functions can also be used: for details, see `flash_ebnm`.

`init_fn` The function used to initialize factor/loadings pairs. Functions `flash_greedy_init_default`, `flash_greedy_init_softImpute`, and `flash_greedy_init_irlba` have been supplied; note, in particular, that `flash_greedy_init_softImpute` can yield better results than the default initialization function when there is missing data. Custom initialization functions may also be used. If `init_fn = NULL` then `flash_greedy_init_default` will be used, with an attempt made to set argument `sign_constraints` appropriately via test calls to the EBNM function(s) specified by parameter `ebnm_fn`. If factors or loadings are constrained in some other fashion (e.g., bounded support), then the initialization function should be modified to account for the constraints — otherwise, the greedy algorithm can stop adding factor/loadings pairs too early. Custom initialization functions should accept a single parameter referring to a `flash_fit` object and should output a list consisting of two vectors, which will be used as initial values for the new loadings $\ell_{\cdot k}$ and the new factor $f_{\cdot k}$. Typically, a custom initialization function will extract the matrix of residuals from the `flash_fit` object using method `residuals.flash_fit` and then return a (possibly constrained) rank-one approximation to the matrix of residuals. See **Examples** below.

`extrapolate` Whether to use an extrapolation technique inspired by Ang and Gillis (2019) to accelerate the fitting process. Control parameters are handled via global options and can be set by calling `options("extrapolate.control") <- control.param`.

`warmstart` Whether to use "warmstarts" when solving the EBNM subproblems by initializing solutions at the previous value of the fitted prior $\hat{g}$. An important side effect of warmstarts for `ashr`-like prior families is to fix the grid at its initial setting. Fixing the grid can lead to poor fits if there are large changes in the scale of the estimated prior over the course of the fitting process. However, allowing the grid to vary can occasionally result in decreases in ELBO.

`maxiter` The maximum number of iterations when optimizing a greedily added factor/loadings pair.

`tol` The convergence tolerance parameter. At each iteration, the fit is compared to the fit from the previous iteration using a convergence criterion function (by default, the difference in ELBO, but the criterion can be changed via `flash_set_conv_crit`). When the value returned by this function is less than or equal to `tol`, the newly added factor/loadings pair is considered to have "converged," so that `flash_greedy` moves on and attempts to add another new factor (or, if the maximum number of factors `Kmax` has been reached, the process terminates). Note that specifying `tol` here will override any value set by `flash_set_conv_crit`; to use the "global" tolerance parameter, `tol` must be left unspecified (`NULL`). If `tol = NULL` and a global tolerance parameter has not been set, then the default tolerance used is $np\sqrt{\epsilon}$, where n is the number of rows in the dataset, p is the number of columns, and ϵ is equal to `.Machine$double.eps`.

verbose — When and how to display progress updates. Set to `0` for none, `1` for updates after a factor is added or a backfit is completed, `2` for additional notifications about the variational lower bound, and `3` for updates after every iteration. It is also possible to output a single tab-delimited table of values using function `flash_set_verbose` with `verbose = -1`.

Value

The `flash` object from argument `flash`, with up to `Kmax` new factor/loadings pairs "greedily" added.

See Also

`flash_greedy_init_default`, `flash_greedy_init_softImpute`, `flash_greedy_init_irlba`

Examples

```
# The following are examples of advanced usage. See ?flash for basic usage.

# Increase the maximum number of iterations in the default initialization
#   method.
my_init_fn <- function(f) flash_greedy_init_default(f, maxiter = 500)
fl <- flash_init(gtex) %>%
  flash_greedy(init_fn = my_init_fn)

# Use a custom initialization function that wraps function nmf from
#   package RcppML.
nmf_init_fn <- function(f) {
  nmf_res <- RcppML::nmf(resid(f), k = 1, verbose = FALSE)
  return(list(as.vector(nmf_res$w), as.vector(nmf_res$h)))
}
fl.nmf <- flash_init(gtex) %>%
  flash_greedy(ebnm_fn = ebnm_unimodal_nonnegative,
               init_fn = nmf_init_fn)
```

---

flash_greedy_init_default

Initialize a flash factor

Description

The default method for initializing the loadings $\ell_{\cdot k}$ and factor values $f_{\cdot k}$ of a new ("greedy") flash factor. It is essentially an implementation of the power method, but unlike many existing implementations, it can handle missing data and sign constraints. For details, see Chapter 2.2.3 in the reference below.

Usage

```
flash_greedy_init_default(
  flash,
  sign_constraints = NULL,
  tol = NULL,
  maxiter = 100,
  seed = 666
)
```

Arguments

flash	A flash_fit object.
sign_constraints	This parameter can be used to constrain the sign of the initial factor and loadings. It should be a vector of length two with entries equal to -1, 0, or 1. The first entry constrains the sign of the loadings $\ell_{\cdot k}$, with -1 yielding nonpositive loadings, +1 yielding nonnegative loadings, and 0 indicating that loadings should not be constrained. The second entry of sign_constraints similarly constrains the sign of factor values $f_{\cdot k}$. If sign_constraints = NULL, then no constraints will be applied.
tol	Convergence tolerance parameter. When the maximum (absolute) change over all values ℓ_{ik} and f_{jk} is less than or equal to tol, initialization terminates. At each iteration, the factor and loadings are L^2-normalized. The default tolerance parameter is $\min(1/n, 1/p)$, where n is the number of rows in the data matrix and p is the number of columns.
maxiter	Maximum number of power iterations.
seed	Since initialization is random, a default seed is set for reproducibility.

Value

A list of length two consisting of, respectively, the vector of initial values for loadings $\ell_{\cdot k}$ and the vector of initial factor values $f_{\cdot k}$.

References

Jason Willwerscheid (2021), *Empirical Bayes Matrix Factorization: Methods and Applications*. Ph.D. thesis, University of Chicago.

See Also

flash_greedy, flash_greedy_init_softImpute, flash_greedy_init_irlba

flash_greedy_init_irlba

Initialize a flash factor using IRLBA

Description

Initializes a new ("greedy") flash factor using `irlba`. This can be somewhat faster than `flash_greedy_init_default` for large, dense data matrices. For sparse matrices of class `Matrix`, the default initialization should generally be preferred.

Usage

```
flash_greedy_init_irlba(flash, seed = 666, ...)
```

Arguments

`flash`	A `flash_fit` object.
`seed`	Since initialization is random, a default seed is set for reproducibility.
`...`	Additional parameters to be passed to `irlba`.

Value

A list of length two consisting of, respectively, the vector of initial values for loadings $\ell_{\cdot k}$ and the vector of initial factor values $f_{\cdot k}$.

See Also

`flash_greedy`, `flash_greedy_init_default`, `flash_greedy_init_softImpute`

flash_greedy_init_softImpute

Initialize a flash factor using softImpute

Description

Initializes a new ("greedy") flash factor using `softImpute`. When there is missing data, this can yield better results than `flash_greedy_init_default` without sacrificing much (if any) speed.

Usage

```
flash_greedy_init_softImpute(flash, seed = 666, ...)
```

Arguments

flash	A flash_fit object.
seed	Since initialization is random, a default seed is set for reproducibility.
...	Additional parameters to be passed to softImpute.

Value

A list of length two consisting of, respectively, the vector of initial values for loadings $\ell_{\cdot k}$ and the vector of initial factor values $f_{\cdot k}$.

See Also

flash_greedy, flash_greedy_init_default, flash_greedy_init_irlba

---

flash_init *Initialize flash object*

Description

Sets up a flash object with no factors. Since all other flash_xxx functions take a flash or flash_fit object as their first argument, calling flash_init should be the first step in any flash pipeline. See flash for examples of usage.

Usage

```
flash_init(data, S = NULL, var_type = 0L, S_dim = NULL)
```

Arguments

data	The observations. Usually a matrix, but can also be a sparse matrix of class Matrix or a low-rank matrix representation as returned by, for example, svd, irlba, rsvd, or softImpute (in general, any list that includes fields u, d, and v will be interpreted as a low-rank matrix representation).
S	The standard errors. Can be NULL (in which case all residual variance will be estimated) or a matrix, vector, or scalar. S should be a scalar if standard errors are identical across observations. It should be a vector if standard errors either vary across columns but are constant within any given row, or vary across rows but are constant within any given column (flash will use the length of the vector to determine whether the supplied values correspond to rows or columns; if the data matrix is square, then the sense must be specified using parameter S_dim in function flash_init).
var_type	Describes the structure of the estimated residual variance. Can be NULL, 0, 1, 2, or c(1, 2). If NULL, then S accounts for all residual variance. If var_type = 0, then the estimated residual variance (which is added to any variance given by S) is assumed to be constant across all observations. Setting var_type = 1 estimates a single variance parameter for each row; var_type = 2 estimates one

parameter for each column; and `var_type = c(1, 2)` optimizes over all rank-one matrices (that is, it assumes that the residual variance parameter s_{ij} can be written $s_{ij} = a_i b_j$, where the n-vector a and the p-vector b are to be estimated). Note that if any portion of the residual variance is to be estimated, then it is usually faster to set `S = NULL` and to let `flash` estimate all of the residual variance. Further, `var_type = c(1, 2)` is typically much slower than other options, so it should be used with care.

`S_dim` — If the argument to `S` is a vector and the data matrix is square, then `S_dim` must specify whether `S` encodes row-wise or column-wise standard errors. More precisely, if `S_dim = 1`, then `S` will be interpreted as giving standard errors that vary across rows but are constant within any particular row; if `S_dim = 2`, then it will be interpreted as giving standard errors that vary across columns but are constant within any particular column. If `S` is a matrix or scalar, or if the data matrix is not square, then `S_dim` should be left unspecified (`NULL`).

Value

An initialized `flash` object (with no factors).

`flash_nullcheck` *Nullcheck flash factors*

Description

Sets factor/loadings pairs to zero if doing so improves the variational lower bound (ELBO). See `flash` for examples of usage.

Usage

```
flash_nullcheck(flash, kset = NULL, remove = TRUE, tol = NULL, verbose = NULL)
```

Arguments

`flash` — A `flash` or `flash_fit` object.

`kset` — A vector of integers specifying which factors to nullcheck. If `kset = NULL`, then all existing factors will be checked.

`remove` — Whether to remove factors that have been set to zero from the `flash` object. Note that this might change the indices of existing factors.

`tol` — The "tolerance" parameter: if a factor does not improve the ELBO by at least `tol`, then it will be set to zero. Note that `flash_nullcheck` does not respect "global" tolerance parameters set by `flash_set_conv_crit` (which only affects the convergence tolerance for greedy fits and backfits). The default tolerance is $np\sqrt{\epsilon}$, where n is the number of rows in the dataset, p is the number of columns, and ϵ is equal to `.Machine$double.eps`.

`verbose` — When and how to display progress updates. For nullchecks, updates are only displayed when `verbose` > 0.

Value

The flash object from argument flash, with factors that do not improve the ELBO by at least tol either set to zero or removed (depending on the argument to parameter remove).

See Also

flash_factors_remove, flash_factors_set_to_zero

---

flash_set_conv_crit *Set convergence criterion and tolerance parameter*

Description

Used in a flash pipeline to set the criterion for determining whether a greedy fit or backfit has "converged."

Usage

```
flash_set_conv_crit(flash, fn = NULL, tol)
```

Arguments

flash	A flash or flash_fit object.
fn	The convergence criterion function (see Details below). If NULL, then only the tolerance parameter is updated (thus a convergence criterion can be set at the beginning of a flash pipeline, allowing the tolerance parameter to be updated at will without needing to re-specify the convergence criterion each time). The default convergence criterion, which is set when the flash object is initialized, is flash_conv_crit_elbo_diff, which calculates the difference in the variational lower bound or "ELBO" from one iteration to the next.
tol	The tolerance parameter (see Details below). The default, which is set when the flash object is initialized (see flash_init), is $np\sqrt{\epsilon}$, where n is the number of rows in the dataset, p is the number of columns, and ϵ is equal to .Machine$double.eps.

Details

Function flash_set_conv_crit can be used to customize the convergence criterion for a flash object. This criterion determines when to stop optimizing a newly added factor (see flash_greedy) and when to stop backfitting (flash_backfit). Note that, because most alternative convergence criteria do not make sense in the context of a nullcheck, it does not set the "convergence" criterion for flash_nullcheck (for example, flash_conv_crit_max_chg_L would simply return the maximum L^2-normalized loading for each set of loadings $\ell_{\cdot k}$).

The criterion is defined by the function supplied as argument to fn, which must accept exactly three parameters, curr, prev, and k. curr refers to the flash_fit object from the current iteration; prev, to the flash_fit object from the previous iteration; and, if the iteration is a sequential backfitting iteration (that is, a flash_backfit iteration with argument extrapolate = FALSE), k identifies the

factor/loadings pair that is currently being updated (in all other cases, k is NULL). The function must output a numeric value; if the value is less than or equal to tol, then the fit is considered to have "converged." The meaning of "convergence" here varies according to the operation being performed. In the greedy algorithm, fn simply compares the fit from one iteration to the next. During a backfit, it similarly compares fits from one iteration to the next, but it only considers the fit to have converged when the value of fn over successive updates to *all* factor/loadings pairs is less than or equal to tol. If, for example, factor/loadings pairs $1, \ldots, K$ are being sequentially backfitted, then fits are compared before and after the update to factor/loadings 1, before and after the update to factor/loadings 2, and so on through factor/loadings K, and backfitting only terminates when fn returns a value less than or equal to tol for all K updates.

Package flashier provides a number of functions that may be supplied as convergence criteria: see flash_conv_crit_elbo_diff (the default criterion), flash_conv_crit_max_chg, flash_conv_crit_max_chg_L, and flash_conv_crit_max_chg_F. Custom functions may also be defined. Typically, they will compare the fit in curr (the current iteration) to the fit in prev (the previous iteration). To facilitate working with flash_fit objects, package flashier provides a number of accessors, which are enumerated in the documentation for object flash_fit. Custom functions should return a numeric value that can be compared against tol; see **Examples** below.

Value

The flash object from argument flash, with the new convergence criterion reflected in updates to the "internal" flash_fit object. These settings will persist across all subsequent calls to flash_xxx functions in the same flash pipeline (unless, of course, flash_set_conv_crit is again called within the same pipeline).

Examples

```
fl <- flash_init(gtex) %>%
  flash_set_conv_crit(flash_conv_crit_max_chg, tol = 1e-3) %>%
  flash_set_verbose(
    verbose = 3,
    fns = flash_verbose_max_chg,
    colnames = "Max Chg",
    colwidths = 20
  ) %>%
  flash_greedy(Kmax = 3)
```

---

flash_set_timeout *Set timeout*

Description

Used in a flash pipeline to set a maximum amount of fitting time. Note that timeout conditions are only checked during greedy fits and backfits, so that the total amount of fitting time can exceed the time set by flash_set_timeout (especially if, for example, there is a nullcheck involving many factor/loading pairs). Also note that timeout conditions must be cleared using function flash_clear_timeout before any re-fitting is attempted.

Usage

```
flash_set_timeout(
  flash,
  tim,
  units = c("hours", "secs", "mins", "days", "weeks")
)
```

Arguments

flash	A flash or flash_fit object.
tim	A numeric value giving the maximum amount of fitting time, with the units of time specified by parameter units.
units	The units of time according to which parameter tim is to be interpreted.

Value

The flash object from argument flash, with the timeout settings reflected in updates to the "internal" flash_fit object. These settings will persist across all subsequent calls to flash_xxx functions until they are modified either by flash_clear_timeout or by another call to flash_set_timeout.

Examples

```
fl <- flash_init(gtex) %>%
  flash_set_timeout(1, "secs") %>%
  flash_greedy(Kmax = 30) %>%
  flash_backfit() %>%
  flash_nullcheck() %>%
  flash_clear_timeout() # Always clear timeout at the end of a pipeline.
```

---

flash_set_verbose *Set verbose output*

Description

Used in a flash pipeline to set the output that will be printed after each greedy or backfitting iteration.

Usage

```
flash_set_verbose(
  flash,
  verbose = 1L,
  fns = NULL,
  colnames = NULL,
  colwidths = NULL
)
```

Arguments

flash	A flash or flash_fit object.
verbose	When and how to display progress updates. Set to 0 for no updates; 1 for updates after a "greedy" factor is added or a backfit is completed; 2 for additional notifications about the variational lower bound (ELBO); and 3 for updates after every iteration. By default, per-iteration update information includes the change in ELBO and the maximum (absolute) change over all L2-normalized loadings $\ell_1, \ldots, \ell_K$ and factors $f_1, \ldots, f_K$. Update information is customizable via parameters fns, colnames, and colwidths. A single tab-delimited table of values may also be output using option verbose = -1. This format is especially convenient for downstream analysis of the fitting history. For example, it may be used to plot the value of the ELBO after each iteration (see the "Advanced Flashier" vignette for an illustration).
fns	A vector of functions. Used to calculate values to display after each greedy/backfit iteration when verbose is either -1 or 3 (see Details below). Ignored for other values of verbose (0, 1, or 2).
colnames	A vector of column names, one for each function in fns.
colwidths	A vector of column widths, one for each function in fns.

Details

Function flash_set_verbose can be used to customize the output that is printed to console while fitting a flash object. After each greedy or backfitting iteration (see, respectively, flash_greedy and flash_backfit), each function in argument fns is successively evaluated and the result is printed to console in a table with column names defined by argument colnames and column widths defined by argument colwidths.

Each function in fns must accept exactly three parameters, curr, prev, and k: curr refers to the flash_fit object from the current iteration; prev, to the flash_fit object from the previous iteration; and, if the iteration is a sequential backfitting iteration (that is, a flash_backfit iteration with argument extrapolate = FALSE), k identifies the factor/loadings pair that is currently being updated (in all other cases, k is NULL). Package flashier provides a number of functions that may be used to customize output: see flash_verbose_elbo, flash_verbose_elbo_diff, flash_verbose_max_chg, flash_verbose_max_chg_L, and flash_verbose_max_chg_F. Custom functions may also be defined. They might inspect the current flash_fit object via argument curr; compare the fit in curr to the fit from the previous iteration (provided by argument prev); or ignore both flash_fit objects entirely (for example, to track progress over time, one might simply call Sys.time). To facilitate working with flash_fit objects, package flashier provides a number of accessors, which are enumerated in the documentation for object flash_fit. Custom functions should return a character string that contains the output exactly as it is to displayed; see **Examples** below.

Value

The flash object from argument flash, with the new verbose settings reflected in updates to the "internal" flash_fit object. These settings will persist across all subsequent calls to flash_xxx functions until they are modified by another call to flash_set_verbose.

Examples

```
# Suppress all verbose output.
fl <- flash_init(gtex) %>%
  flash_set_verbose(0) %>%
  flash_greedy(Kmax = 5)

# Set custom verbose output.
sparsity_F <- function(curr, prev, k) {
  g_F <- flash_fit_get_g(curr, n = 2)
  g_F_pi0 <- g_F$pi[1] # Mixture weight of the "null" component.
  return(g_F_pi0)
}
verbose_fns <- c(flash_verbose_elbo, flash_verbose_max_chg_F, sparsity_F)
colnames <- c("ELBO", "Max Chg (Tiss)", "Sparsity (Tiss)")
colwidths <- c(12, 18, 18)
fl <- flash_init(gtex) %>%
  flash_set_verbose(
    verbose = 3,
    fns = verbose_fns,
    colnames = colnames,
    colwidths = colwidths
  ) %>%
  flash_greedy(Kmax = 3)

# Output can be changed as needed.
fl <- flash_init(gtex) %>%
  flash_set_verbose(verbose = 1) %>%
  flash_greedy(Kmax = 5L) %>%
  flash_backfit(verbose = 3) %>%
  flash_greedy(Kmax = 1L)
```

`flash_verbose_elbo` *Display the current ELBO*

Description

Displays the value of the variational lower bound (ELBO) at the current iteration.

Usage

```
flash_verbose_elbo(curr, prev, k)
```

Arguments

`curr`	The `flash_fit` object from the current iteration.
`prev`	The `flash_fit` object from the previous iteration.
`k`	Only used during sequential backfits (that is, calls to `flash_backfit` where `extrapolate = FALSE`). It then takes the index of the factor/loadings pair currently being optimized.

Details

This function is an example of a function that may be passed to parameter `fns` in function `flash_set_verbose` to customize the output that is printed after each greedy or backfitting iteration. See `flash_set_verbose` for details and examples.

Value

A character string, suitable for printing progress updates.

See Also

`flash_verbose_elbo_diff`, `flash_verbose_max_chg`, `flash_verbose_max_chg_L`, `flash_verbose_max_chg_F`

`flash_verbose_elbo_diff` *Display the difference in ELBO*

Description

Displays the difference in the variational lower bound (ELBO) from one iteration to the next.

Usage

```
flash_verbose_elbo_diff(curr, prev, k)
```

Arguments

`curr`	The `flash_fit` object from the current iteration.
`prev`	The `flash_fit` object from the previous iteration.
`k`	Only used during sequential backfits (that is, calls to `flash_backfit` where `extrapolate = FALSE`). It then takes the index of the factor/loadings pair currently being optimized.

Details

This function is an example of a function that may be passed to parameter `fns` in function `flash_set_verbose` to customize the output that is printed after each greedy or backfitting iteration. See `flash_set_verbose` for details and examples.

Value

A character string, suitable for printing progress updates.

See Also

`flash_verbose_elbo`, `flash_verbose_max_chg`, `flash_verbose_max_chg_L`, `flash_verbose_max_chg_F`

flash_verbose_max_chg *Display the maximum difference in scaled loadings and factors*

Description

Displays the maximum (absolute) change over all (posterior expected values for) loadings ℓ_{ik} and factors f_{jk}. At each iteration, the loadings vectors $\ell_{\cdot 1}, \ldots, \ell_{\cdot K}$ and factors $f_{\cdot 1}, \ldots, f_{\cdot K}$ are L^2-normalized.

Usage

```
flash_verbose_max_chg(curr, prev, k)
```

Arguments

curr	The flash_fit object from the current iteration.
prev	The flash_fit object from the previous iteration.
k	Only used during sequential backfits (that is, calls to flash_backfit where extrapolate = FALSE). It then takes the index of the factor/loadings pair currently being optimized.

Details

This function is an example of a function that may be passed to parameter fns in function flash_set_verbose to customize the output that is printed after each greedy or backfitting iteration. See flash_set_verbose for details and examples.

Value

A character string, suitable for printing progress updates.

See Also

flash_verbose_elbo, flash_verbose_elbo_diff, flash_verbose_max_chg_L, flash_verbose_max_chg_F

flash_verbose_max_chg_F *Display the maximum difference in scaled factors*

Description

Displays the maximum (absolute) change over all (posterior expected values for) factors f_{jk}. At each iteration, the factors $f_{\cdot 1}, \ldots, f_{\cdot K}$ are L^2-normalized.

Usage

```
flash_verbose_max_chg_F(curr, prev, k)
```

Arguments

curr	The flash_fit object from the current iteration.
prev	The flash_fit object from the previous iteration.
k	Only used during sequential backfits (that is, calls to flash_backfit where extrapolate = FALSE). It then takes the index of the factor/loadings pair currently being optimized.

Details

This function is an example of a function that may be passed to parameter fns in function flash_set_verbose to customize the output that is printed after each greedy or backfitting iteration. See flash_set_verbose for details and examples.

Value

A character string, suitable for printing progress updates.

See Also

flash_verbose_elbo, flash_verbose_elbo_diff, flash_verbose_max_chg, flash_verbose_max_chg_L

---

flash_verbose_max_chg_L

Display the maximum difference in scaled loadings

Description

Displays the maximum (absolute) change over all (posterior expected values for) loadings ℓ_{ik}. At each iteration, the loadings vectors $\ell_{\cdot 1}, \ldots, \ell_{\cdot K}$ are L^2-normalized.

Usage

```
flash_verbose_max_chg_L(curr, prev, k)
```

Arguments

curr	The flash_fit object from the current iteration.
prev	The flash_fit object from the previous iteration.
k	Only used during sequential backfits (that is, calls to flash_backfit where extrapolate = FALSE). It then takes the index of the factor/loadings pair currently being optimized.

Details

This function is an example of a function that may be passed to parameter `fns` in function `flash_set_verbose` to customize the output that is printed after each greedy or backfitting iteration. See `flash_set_verbose` for details and examples.

Value

A character string, suitable for printing progress updates.

See Also

`flash_verbose_elbo`, `flash_verbose_elbo_diff`, `flash_verbose_max_chg`, `flash_verbose_max_chg_F`

---

gtex *GTEx data*

Description

Derived from data made available by the Genotype Tissue Expression (GTEx) project (Lonsdale et al. 2013), which provides z-scores for assessing the significance of effects of genetic variants (single nucleotide polymorphisms, or SNPs) on gene expression across 44 human tissues. To reduce the data to a more manageable size, Urbut et al. (2019) chose the "top" SNP for each gene — that is, the SNP associated with the largest (absolute) z-score over all 44 tissues. This yields a $16,069 \times 44$ matrix of z-scores, with rows corresponding to SNP-gene pairs and columns corresponding to tissues. The dataset included here is further subsampled down to 1000 rows.

Format

`gtex` is a matrix with 1000 rows and 44 columns, with rows corresponding to SNP-gene pairs and columns corresponding to tissues.

Source

<https://github.com/stephenslab/gtexresults/blob/master/data/MatrixEQTLSumStats.Portable.Z.rds>

References

Lonsdale et al. (2013). "The Genotype-Tissue Expression (GTEx) project." *Nature Genetics* 45(6), 580–585.

Urbut, Wang, Carbonetto, and Stephens (2019). "Flexible statistical methods for estimating and testing effects in genomic studies with multiple conditions." *Nature Genetics* 51(1), 187–195.

Examples

```
data(gtex)
summary(gtex)
```

gtex_colors *Colors for plotting GTEx data*

Description

A custom palette used by Wang and Stephens (2021) to plot an empirical Bayes matrix factorization of data from the GTEx project (of which the `gtex` data in package **flashier** is a subsample). The palette is designed to link similar tissues together visually. For example, brain tissues all have the same color (yellow); arterial tissues are shades of pink or red; etc.

Format

`gtex_colors` is a named vector of length 44, with names corresponding to tissues (columns) in the `gtex` dataset and values giving hexadecimal color codes.

Source

<https://github.com/stephenslab/gtexresults/blob/master/data/GTExColors.txt>

References

Wei Wang and Matthew Stephens (2021). "Empirical Bayes matrix factorization." *Journal of Machine Learning Research* 22, 1–40.

Examples

```
fl <- flash(gtex, greedy_Kmax = 4)
plot(fl, incl_scree = FALSE, pm_colors = gtex_colors)
```

ldf *LDF method for flash and flash fit objects*

Description

Given a `flash` or `flash_fit` object, returns the LDF decomposition $Y \approx LDF'$.

Usage

```
ldf(object, type)

## S3 method for class 'flash'
ldf(object, type = "f")

## S3 method for class 'flash_fit'
ldf(object, type = "f")
```

Arguments

`object`	An object inheriting from class `flash` or `flash_fit`.
`type`	Takes identical arguments to function `norm`. Use `"f"` or `"2"` for the 2-norm (Euclidean norm); `"o"` or `"1"` for the 1-norm (taxicab norm); and `"i"` or `"m"` for the infinity norm (maximum norm).

Details

When the prior families $G_\ell^{(k)}$ and $G_f^{(k)}$ are closed under scaling (as is typically the case), then the EBMF model (as described in the documention to function `flash`) is only identifiable up to scaling by a diagonal matrix D:

$$Y = LDF' + E.$$

Method `ldf` scales columns ℓ_k and f_k so that, depending on the argument to parameter `type`, their 1-norms, 2-norms, or infinity norms are equal to 1.

Value

A list with fields `L`, `D`, and `F`, each of which corresponds to one of the matrices in the decomposition $Y \approx LDF'$ (with the columns of L and F scaled according to argument `type`). Note that `D` is returned as a vector rather than a matrix (the vector of diagonal entries in D). Thus, "fitted values" LDF' can be recovered as `L %*% diag(D) %*% t(F)`.

Methods (by class)

- `ldf(flash)`: LDF decomposition for `flash` objects
- `ldf(flash_fit)`: LDF decomposition for `flash_fit` objects

`plot.flash` *Plot method for flash objects*

Description

Given a `flash` object, produces up to two figures: one showing the proportion of variance explained per factor/loadings pair, and one that plots posterior means for either factors or loadings (depending on the argument to parameter `pm_which`).

Usage

```
## S3 method for class 'flash'
plot(
  x,
  include_scree = TRUE,
  include_pm = TRUE,
  order_by_pve = TRUE,
  kset = NULL,
  pm_which = c("factors", "loadings"),
```

```
  pm_subset = NULL,
  pm_groups = NULL,
  pm_colors = NULL,
  ...
)
```

Arguments

x	An object inheriting from class flash.
include_scree	Whether to include a figure ("scree plot") showing the proportion of variance explained by each factor/loadings pair.
include_pm	Whether to include a figure showing the posterior means for either loadings L or factors F (depending on the argument to parameter pm_which). One plot panel is produced for each factor/loadings pair k. If argument pm_groups is left unspecified, then bar plots will be produced, with each bar corresponding to a single value ℓ_{ik} or f_{jk}. Otherwise, overlapping histograms will be produced, with each histogram corresponding to one of the groups specified by pm_groups.
order_by_pve	If TRUE, then the factor/loadings pairs will be re-ordered according to proportion of variance explained (from highest to lowest).
kset	A vector of integers specifying the factor/loadings pairs to be plotted. If kset = NULL, then all will be plotted.
pm_which	Whether to plot loadings L or factors F in the plots of posterior means. This parameter is ignored when include_pm = FALSE.
pm_subset	A vector of row indices i or column indices j (depending on the argument to pm_which) specifying which values $\ell_{i\cdot}$ or $f_{j\cdot}$ are to be shown in the plots of posterior means. If the dataset has row or column names, then names rather than indices may be specified. If pm_subset = NULL, then all values will be plotted. This parameter is ignored when include_pm = FALSE.
pm_groups	A vector specifying the group to which each row of the data $y_{i\cdot}$ or column $y_{\cdot j}$ belongs (groups may be numeric indices or strings). If pm_groups = NULL, then a bar plot of the ungrouped data is produced (see include_pm above). Otherwise, a group must be provided for each plotted row i or column j, so that the length of pm_groups is exactly equal to the number of rows or columns in the full dataset or, if pm_subset is specified, in the subsetted dataset. When pm_groups is not NULL, a set of overlapping histograms is produced for each factor/loadings pair, with one histogram per group (again see include_pm). This parameter is ignored when include_pm = FALSE.
pm_colors	A vector specifying a color for each bar (if pm_groups = NULL) or histogram (if pm_groups is not NULL). Passed directly to parameter values in **ggplot2** function scale_color_manual. This parameter is ignored when include_pm = FALSE.
...	Additional parameters are ignored.

Value

If arguments include_scree and include_pm specify that only one figure be produced, then plot.flash() returns a ggplot2 object. If both figures are to be produced, then plot.flash()

prints both plots but does not return a value.

`residuals.flash` *Residuals method for flash objects*

Description

Given a `flash` object, returns the expected residuals $Y - E(LF') = Y - E(L)E(F)'$.

Usage

```
## S3 method for class 'flash'
residuals(object, ...)
```

Arguments

`object`	An object inheriting from class `flash`.
`...`	Additional parameters are ignored.

Value

The matrix of expected residuals.

`residuals.flash_fit` *Residuals method for flash fit objects*

Description

Given a `flash_fit` object, returns the expected residuals $Y - E(LF') = Y - E(L)E(F)'$.

Usage

```
## S3 method for class 'flash_fit'
residuals(object, ...)
```

Arguments

`object`	An object inheriting from class `flash_fit`.
`...`	Additional parameters are ignored.

Value

The matrix of expected residuals.

Index