Package 'fluodilution'

March 13, 2018

Title Interpretation of Fluorescence Dilution Experiments

Description Fluorescence Dilution (FD) experiments / proliferation assays are used to assess growth and migration at the single-cell level. With mathematical models parameters such as the division, death and migration rates can be estimated. This package provides generic functions to manipulate FD data and estimate these parameters. It has been used both in immunology and microbiology applications.

Version 0.2.0

biocViews CellBasedAssays, FlowCytometry, CellBiology, MathematicalBiology, Regression, Software

Date 2016

Depends R (>= 3.4.3),
nlme (>= 3.1),
ggplot2 (>= 2.2.1),
stats,
scales (>= 0.4)

Imports magrittr (>= 1.5),
memoise (>= 1.0),
stringr (>= 1.3.0),
plyr (>= 1.8.3),
dplyr (>= 0.4.3),
reshape2 (>= 1.4.1),
utils,
graphics,
Rcpp (>= 0.12.2),
RcppParallel (>= 4.3)

URL https://github.com/hchauvin/fluodilution

BugReports https://github.com/hchauvin/fluodilution/issues

License MIT, GPL-3

LazyData true

RoxygenNote 6.0.1

Suggests zoo (>= 1.7),
lmtest (>= 0.9),
sandwich (>= 2.3),
testthat (>= 1.0.2),
GenSA (>= 1.1.6),

truncnorm (>= 1.0.7),
mvtnorm (>= 1.0),
limSolve (>= 1.5.5),
flowCore (>= 1.36),
flowWorkspace (>= 3.16),
flowClust (>= 3.8),
AICcmodavg (>= 2.0),
parallel,
knitr (>= 1.12),
grid,
gridExtra (>= 2.2.1),
RColorBrewer (>= 1.1),
numDeriv,
bbmle (>= 1.0),
stats4,
microbenchmark (>= 1.4)

VignetteBuilder knitr

LinkingTo Rcpp, RcppParallel, RcppArmadillo (>= 0.8.300.1.0)

SystemRequirements GNU make

Collate 'RcppExports.R'
'create.R'
'cstr.R'
'fd_model.R'
'fd_simulate.R'
'fetch.R'
'fit.R'
'fluodilution.R'
'fmm-.R'
'fmm-af.R'
'fmm-af_bp.R'
'fmm-gaussian.R'
'gamma-distributions.R'
'graphics.R'
'modsel.R'
'peaks.R'
'population.R'
'pretty.R'
'proliferation-.R'
'proliferation-branching.R'
'proliferation-cyton.R'
'proportions.R'
'release-questions.R'

R topics documented:

`fluodilution-package` *Generic functions for analysing fluorescence dilution experiments / proliferation assays.*

Description

In Chauvin et al. (2016), we devise a framework to quantitatively analyse fluorescence dilution (Helaine et al. 2010, 2014), a technique aimed at understanding the history of bacteria in terms of division, migration and death in an *in vitro* setting, in macrophage cultures or in systemic infection of mice. This package builds upon and translates into open-source code previous efforts in immunology (Hyrien et al. 2008, Hyrien et al. 2010, Hawkins et al. 2006) and adapts those analytical techniques to the novel use case of microbiology. Therefore, this package can be used by microbiologists and immunologists alike.

Background

Proliferation assays make use of a fluorescent reporter, such as a dye like CFSE (Lyons and Parish 1994) or a fluorescence protein (e.g., GFP) expressed by a plasmid (Helaine 2010, Roostalu 2008). In both cases, the fluorophore is present in fixed quantities in progenitor cells/bacteria. Whenever a cell/bacterium divides, the fluorescence is approximately halved. The distribution of fluorescence in the population can then be recorded over time using flow cytometry. Augmented by cell counts / Colony Forming Units (Chauvin et al. 2016) or the staining of dying/dead cells (Hyrien et al. 2010), the resulting dataset can be used to infer a set of *biological constants* such as the growth/death rates, their variability, ...

Aim

We took great care to cover a number of particular cases that might arise in fluorescence dilution and extensively documented our methods, both concerning their use and more broadly their philosophy. Our aim was not only to provide a robust package that could be reused many times in the future and extended for new usages but also to transmit some important practical knowledge concerning fluorescence dilution and cell growth. We believe that a package is the best way to embody this knowledge.

A two-step hierarchical approach

An intermediary step before the estimate of the biological constants involves extracting the proportion of cells in any given *number of generations* (a cell in generation 0 has not divided since the beginning of the experiment, a cell in generation 1 or 2 has divided once or twice, ...). This is done through a Finite Mixture Model (FMM). In immunology, with the exception of Hyrien et al. (2008) and Hyrien et al. (2010), this step has usually been glossed over as fluorescence dilution in monocytes produces distinct peaks in the distribution of fluorescence, one peak for each generational cohort. In microbiology, the generational cohorts are not as clearly delineated, probably owing to more variability of the fluorescence in the bacterial inoculum than in the monocytes (Chauvin et al. 2016). Thus, in microbiology, this first step requires a more thorough treatment. In this respect, we implemented FMMs taking into account autofluorescence and the stochastic partitioning of fluorescent molecules upon division (binomial partitioning). Our system is flexible enough for other FMMs to be easily "plugged-in".

In a second step, the proportions inferred by the FMM are then fed to a *proliferation model* that links those proportions to the underlying proliferation/death mechanisms through *biological constants*. With this package, a large variety of proliferation models can be implemented. They all rest on two broad sets of assumptions. On one hand, a generalized Cyton model and its nested, simpler models, consider that division and death occur concurrently, in competition (Hawkins et al. 2007). On the other hand, a branching process (Hyrien et al. 2010) considers that birth and death have both a given probability of occurring in a cell (probability of transition) and a given probability of occurring after a certain time (time to transition). Despite these apparently distinct behaviour, Hyrien et al. (2010) have shown that a Cyton model is actually a special case of a branching process with very specific probabilities of transition (see also Chauvin et al. (2016)). However, as those probabilities are difficult to calculate, we decided to implement the Cyton model independently.

As we have seen, a variety of models in this two-step hierarchical approach can be used. Finite mixture models and proliferation models can be used in any combination imaginable. Indeed, we believe that not all models are good for all datasets. After 15 years of literature and countless models produced, the question of model selection is therefore central. Our integrated approach allows to test novel and proven models alike in a quick way so as to find an acceptable fit. Criteria researchers should look for include some information criterion such as the Akaike Information Criterion (AIC) to balance out better model fit with parcimony, whether the model is practically identifiable and whether heroskedasticity and systematic bias, across time or experiments, persist in an overly problematic way.

Overview of the implementation

Loading of experimental data from a flowJo workspace (`www.flowjo.org`) can be carried out with `fd_create`. Alternatively, an `fd_data` object, which is the form the datasets take in this package, can be generated directly. A hierarchical model can be specified through `fd_model`. The resulting model can be fed to generic nonlinear optimization algorithms such as `nls`, `nlsList`, `nlme` or `gnls`. We also implemented the wrapper `fd_nls` around a generic nonlinear optimizer. This wrapper automatically sets the boundary and starting values from an `fd_model` object and performs additional checks, especially on the maximum number of generations considered by the algorithm (even if the number of generations one can follow using fluorescence dilution is limited, setting a maximum number of generations too low can have consequences on the fitting of the cell counts and it is absolutely necessary to check for that).

Strategy of nonlinear optimization

We found out, echoing Miao et al. (2012), that traditional fitting algorithms perform poorly on fluorescence dilution problems (Chauvin et al. 2016). However, generalized simulated annealing

(a stochastic global search approach), implemented in package **GenSA** (Xiang 2013), is promising. However, GenSA is a general optimizer and cannot be directly fed a nonlinear formula of the form `y ~ f(x)` as provided by `fd_formula`. Therefore, we provide the function `fd_minuslogl` that, instead of returning a formula, returns a minus log-likelihood function that can be directly fed to GenSA. To get confidence intervals and other such niceties, the user is invited to use GenSA through the package **bbmle**, an extension of package **stats4** that allows an arbitrary function as a nonlinear optimizer. Alternatively, the user can use the nonlinear square wrapper `nlsSA`, not part of this package (the focus here is on fluorescence dilution, not nonlinear optimization) but included on the side in `contrib/nlsSA.R` (see examples in `fd_nls`). `nlsSA` borrows code from `nls` but extends it so that any nonlinear optimizer, whether specifically tailored to least-square problems (e.g., Levenberg-Marquardt) or general (e.g., BFGS), can be used. (The **stats** package, from which `nls` originates, has been authored by the R Core Team and contributors worldwide.) For generalized nonlinear square (gnls) and nonlinear mixed-effects fitting (nlme), a global optimization strategy is (as of now) still not available. Therefore, the user is invited to use the result of `nlsSA` as starting value for a round of `gnls` or `nlme`. Wrappers are given in `contrib/fitwrappers.R`.

We believe that generalized simulated annealing provides a very flexible way to deal with nonlinear regressions with overparametrization or poor choices of starting values. Other possibilities in R include self-starting models and initial grid search using brute force (package **nls2**). Although there is no straightforward way to self-start an FD model, brute force could be a viable option, but generalized simulated annealing offers in our opinion a more elegant way to go about. Moreover, we found out that **nls2** performed poorly in our benchmarking (Chauvin et al. 2016). In any case, neither self-starting nor **nls2** deal with overparametrization (in the worst case, they fail after encountering a singular gradient). Now, overparametrization is not good either for GenSA that will struggle to find an optimum (slow convergence, if any), but at least the results of GenSA can be used for further improvements to the model.

Generalized Estimating Equations

Notice that as a fluorescence dilution is not an ordinary nonlinear regression but a generalized estimating equation (GEE) with potential clustering (repeat experiments) and "space" and time autocorrelations, the degree of freedom is difficult to assess. Therefore, correcting for small-sample bias using the `AICc` instead of the `AIC` information criterion (or QIC, see Pan [2001]) should be done with caution. If the correction can be done with the number of clusters, as one usually does with logistic growth curves in the context of nonlinear mixed-effects models, the AIC will be overestimated. If the correction is made taking the number of residuals, the AIC will be underestimated.

Moreover, if a precise determination of the confidence intervals is required, the user is advised, when possible, to perform a parametric boostrap for one-time experiments (Hyrien 2008) or a clustered bootstrap or jack-knife when the data is clustered in the context of repeat experiments (Chauvin et al. 2016). Sometimes, a sandwich correction is enough (see the **sandwich** package) and we accommodate the use of such "nonstandard" variance-covariance matrices.

Representing, loading and simulating datasets

`fd_data` Represent a fluorescence dilution dataset.

`fd_create` Create a histogram-based dataset from row fluorescence data.

`fd_simulate` Simulate a dataset.

Available finite mixture models (FMM)

`fd_fmm_gaussian` Log-normal cohorts.

`fd_fmm_af` Correction for autofluorescence only.

`fd_fmm_af_bp` Correction for autofluorescence and binomial partitioning.

Available proliferation models

`fd_proliferation_cyton` A fairly general Cyton model.

`fd_proliferation_branching` A fairly general branching process.

Optimization

`fd_model` Two-tier hierarchical fluorescence dilution model.

`fd_nls` Nonlinear least square optimizer.

`fd_minuslogl` Minus log-likelihood for more general optimization.

Agent-based model

In `contrib/agent.R`, we implemented an agent-based model. It can be used to explore the validity of our various assumptions used for the proliferation models and to factor stochasticity in. However, this model should not be used for "routine" fitting and as a consequence is not formally included in the package.

Author(s)

Maintainer: Hadrien Chauvin <hadrienchauvin@gmail.com>

Other contributors:

- Kathryn Watson [contributor]
- Vahid Shahrezaei <v.shahrezaei@imperial.ac.uk> [thesis advisor, contributor]
- David Holden <d.holden@imperial.ac.uk> [thesis advisor, NA]
- Sophie Helaine <s.helaine@imperial.ac.uk> [thesis advisor]

References

Hawkins ED, Turner ML, Dowling MR, van Gend C, Hodgkin PD (2007). A model of immune regulation as a consequence of randomized lymphocyte division and death times. *Proc Natl Acad Sci USA* **104** (12): 5032-5037.

Hawkins ED, Markham JF, McGuinness LP, Hodgkin PD (2006). A single-cell pedigree analysis of alternative stochastic lymphocyte fates. *Proc Natl Acad Sci USA* **106** (32): 13457-13462.

Helaine S, Thompson JA, Watson KG, Liu M, Boyle C, and Holden DW (2010). Dynamics of intracellular bacterial replication at the single cell level. *Proc Natl Acad Sci USA* **107** (8): 3746-3751.

Helaine S, Chverton AM, Watson KG, Faure LM, Matthews SA, Holden DW (2014). Internalization of Salmonella by macrophages induces formation of nonreplicating persisters. *Science* **343**: 204-8.

Hyrien O, Zand MS (2008). A Mixture Model with Dependent Observations for the Analysis of CSFE-Labeling Experiments. *Journal of the American Statistical Association* **103**: 222-239.

Hyrien O, Chen R, Zand MS (2010). An age-dependent branching process model for the analysis of CFSE-labeling experiments. *Biology Direct* **5** (41).

Lyons AB, Parish CR (1994). Determination of lymphocyte division by flow cytometry. *Journal of Immunological Methods* **171** (1): 131-137.

Miao H, Jin X, Perelson AS, Wu H (2012). Evaluation of multitype mathematical models for CFSE-labeling experiment data. *Bull Math Biol* **74** (2): 300-326.

Pan W (2001). Akaike's Information Criterion in Generalized Estimating Equations. *Biometrics* **57** (1): 120-125.

Roostalu J, Joers A, Luidalepp H, Kaldalu N, Tenson T (2008). Cell division in Escherichia coli cultures monitored at single cell resolution. *BMC Microbiology* **8** (68).

Xiang Y, Gubian S, Suomela B, Hoeng J (2013). Generalized Simulated Annealing for Global Optimization: The GenSA Package. *The R Journal* **5** (1): 13-28.

See Also

Useful links:

- https://github.com/hchauvin/fluodilution
- Report bugs at https://github.com/hchauvin/fluodilution/issues

---

`constraints` *Constrain some parameters to specific values.*

Description

Fully-fledged fluorescence dilution models feature a large amount of parameters, not all of which are practically identifiable to an acceptable degree. Therefore, it is important to constrain them: for instance, by requiring that a gamma distribution should have a coefficient of variation of 1, turning it into an exponential distribution, or that different compartments share the same time to division. We devised a set of tools to help this constraining.

Usage

```
cstrlist(constraints = NULL, start = NULL, lower = NULL, upper = NULL)

fixcstr(cstr, fit, before = TRUE)

catcstr(..., drop = TRUE)
```

Arguments

`constraints`	A set of constraints (format detailed below) that should apply to the starting parameters in an optimization, the lower and upper bounds and the result of the optimization.
`start`	A set of constraints that apply only to the starting parameters.
`lower`	A set of constraints that apply only to the lower bounds.
`upper`	A set of constraints that apply only to the upper bounds.
`cstr`	A set of constraints (format detailed below) or a `cstrlist` object.
`fit`	Either an object the `coef` method can be used with (e.g., the result of `nls`) or a named vector of coefficients.
`before`	Whether the new constraining should apply before (default) or after the current constraining.
`...`	Constraints (or lists of constraints) to "concatenate" together. The leftmost is concatenated before the rightmost and thus would apply before. If lists are given, they are "combinatorially" concatenated (as with a Cartesian product).
`drop`	If the returned list of constraints have only one element, return this element instead of a one-element list.

Details

`cstrlist` is used to bring together the specification of overall constraints and the starting values and lower and upper bounds (they are also given using constraints). The result can be used with, e.g., `fd_model`. `fixcstr` is an alternative way to constructing constraints: it returns the original set of constraints augmented by the fixing of some of the parameters to the values found in `fit`. `catcstr` is used to concatenate together an arbitrary number of constraints (it does not work on a `cstrlist` object).

Value

`makecstr` returns a `cstrlist` object suitable for use with, e.g., `fd_model`. `fixcstr` returns either the new set of constraints or a `cstrlist` object, depending on the type of argument `cstr`. `catcstr` returns the resulting concatenation (a new set of constraints).

Constraint format

In EBNF,

```
   constraint ::= [ term [ '+' ] constraint ]
         term ::= [ prefix ':' ] '{' assignments '}'
                | name
                | 'NULL'
                | [ prefix ':' ] '(' constraint ')'
       prefix ::= prefix_atom [ ':' prefix ]
  prefix_atom ::= language
                | '(' language {'/' language} ')'
  assignments ::= language '<-' (language | '..free..')
                  [ ';' assignments ]
```

This format is specifically tailored to constraining the value of nested lists with structures that are somewhat "parallel". For instance, in fluorescence dilution, the parameters for the two-tier model are structured as a list with two elements, `fmm` (parameters for the finite mixture model) and `pro` (parameters for the proliferation model). `pro` is structured as a named list, with each element giving the parameters for a specific compartment, in their flow order. In turn, each compartment is parametrized by gamma distributions, which are represented as lists of three elements (a detailed account of this structure is given in `proliferation`, and for FMM in `finite-mixture`). Therefore, the constraining can be "packed" more efficiently, benefiting from this "parallel" organization.

In the end, with nested lists, constraining a set of parameters `params` amount to using such expressions:

`params$pro$One$f0$delta <- 1`.

Such a constraining can be written in our format as `pro:One:f0:{delta <- 1}` or alternatively `pro:One:{f0$delta <- 1}`: the left-hand side is simply prefixed.

To make expressions lighter, it is possible to combine

`params$pro$One$f0$delta <- 1; params$pro$One$g0$delta <- 1`

into `pro:One:(f0/g0):{delta <- 1}`.

For the proliferation model, the `all` prefix is expanded: for three compartments `One`, `Two` and `Three`, `pro:all:(f0/g0):{delta <- 1}` expands into `pro:(One/Two/Three):(f0/g0):{delta <- 1}`.

It is also possible to set more than one parameter inside the brackets:

`pro:all:f0:{delta <- 1; ss <- 0.5}`.

Previously constrained parameters can be "unconstrained" and constraints linked together using +: `pro:all:f0:{delta <- 1; ss <- 0.5} + pro:One:f0:{delta <- ..free..}`.

A parameter that has been previously freed can be constrained again:

```
pro:all:f0:{delta <- 1; ss <- 0.5} +
pro:One:f0:{delta <- ..free..} +
pro:one:f0:{delta <- 0.5}
```

and in general the rightmost value prevails. For instance, in the following `delta` is constrained to `0.5` in compartment `One` but to `1` in every other compartment:

`pro:all:f0:{delta <- 1; ss <- 0.5} + pro:One:f0:{delta <- 0.5}`.

To bind parameters together, it is possible to use the special placeholders `.L1` (stands for the list containing the left-hand parameter in the assignment), `.L2` (the list just above) and `.L3` (one additional level up). For example, to bind probabilities together one can write `pro:all:{p0 <- .L1$p}`, to bind distributions inside the same compartment

`pro:all:{f0$mm <- .L2$f$mm}` or `pro:all:f0:{mm <- .L2$f$mm}`

and to bind distributions across compartments

`pro:all:{f$mm <- .L3$One$f$mm} + pro:One:f:{mm <- ..free..}`.

When a term is a name beginning with # or in the form `listname$name`, it is expanded in the global environment: for instance, if `` `#noss` `` is defined in the global environment as

`` `#noss` <- ~ pro:all:(f0/g0/f/g):{ss <- 0.5}``,

then `` `#noss` + pro:One:(f0/g0):{delta <- 1} `` is expanded to

`pro:all:(f0/g0/f/g):{ss <- 0.5} + pro:One:(f0/g0):{delta <- 1}` (see the *examples* section for more).

Notice also that a tilde ~ is used to indicate *R* not to evaluate the expression on the right of it (it is a valid formula but not a meaningful statement).

We believe that this format offers a very powerful, flexible and terse approach to constraining complex and large nonlinear models. It has applications well beyond the realm of fluorescence dilution.

See Also

finite-mixture, proliferation, fd_model.

Examples

```
# Constrain all 'delta' to be 1
`#c1` <- ~ pro:all:(f/g/f0/g0):{delta <- 1}

# It is also possible to use a set of often-used constraints
CC <<- FdCommonConstraints

# The constrains can be combined together in two different ways
`#c2` <- ~ `#c1` + CC$`#delta_1111`
`#c3` <- catcstr(`#c2`, CC$`#noss`, ~ pro:all:{p0 <- .L1$p})

# Starting parameters and lower/upper bounds are also
# specified using constraints
cstrlist(constraints = `#c3`,
         start = ~ pro:all:{p <- 0.5},
         lower = ~ pro:all:{p <- 0.1},
```

```
        upper = ~ pro:all:{p <- 0.95})

# An alternative way to constructing constraints uses 'fixcstr'
`#c4` <- fixcstr(`#c3`, c(pro.One.f.mm = 5))

# Finally, previously constrained parameters can be freed
`#c5` <- ~ `#c3` + pro:all:{p0 <- ..free..}
```

---

dataset-simulation *Simulation of datasets.*

Description

fd_simulate simulates an fd_data dataset. fd_proportions gives the proportion of cells in a given number of generations and other related metrics.

Usage

```
fd_simulate(params, times, mgen = NULL, breaks = 50, range = NULL,
  noise = NULL, select = c("hists", "Ns"), model = NULL, mean = FALSE)

fd_proportions(params, times, mgen = NULL, model = NULL)
```

Arguments

params	For fd_simulate, A named numeric vector of coefficients or a matrix with parameter names in columns as would be returned by coef applied to an nlme, nlsList, nls or fd_nls object or the result of fd_unif. For fd_proportion, only a named numeric vector is allowed.
times	A numeric vector of times, in hours, at which to simulate sampling.
mgen	Maximum number of generations (if NULL, the default of the proliferation model is taken).
breaks	The breakpoints to use when forming histograms. If it is a single number, number of breaks to use. If it is a numeric vector, it directly gives the breakpoints in linear fluorescence units. It is not possible to specify different breakpoints for different timepoints.
range	A two-element numeric vector giving respectively the lower and upper breakpoints, in linear fluorescence units, when breaks is a single numeric value. If not explicitly specified, the range is taken to be exp(mean(m0) + mean(sd0) * 3) * c(1 / 2.0^(mgen + 4), 1).
noise	A two-element numeric vector that gives the standard deviations of a white noise to be added respectively to histograms and cell counts, after FMM transformations apply (see fd_transform).
select	What Type the resulting fd_data should feature.
model	An fd_model object. In the case of fd_norm, mandatory only if mean was not generated by fd_unif, fd_draw_unif, fd_nls or fd_comb.
mean	If params is a matrix, whether to average the columns to give a single set of parameters (TRUE) or to generate a dataset for each row (FALSE), in which case the dataset contains multiple Individual, labelled following the row names or indiv_<i> if no row name was given.

Value

`fd_simulate` returns an `fd_data` dataset. `fd_proportions` returns a list with the following elements:

- For branching processes:

  `live_pop` A named list of matrices. The element names are the categories, the rows the timepoints (in the order of argument `times`) and the columns the number of generations (the first column gives generation 0). The entries give the proportions of cells in a given number of generation at a given time and the rows sum to one.

  `Ns` A matrix giving the number of currently alive cells in each category (columns) for each timepoint (rows), relative to the `initial` argument passed along to `fd_proliferation_branching`.

  `lost_pop` Same as `live_pop` but with the lost population.

  `Ns_lost` Same as `Ns` but with (cumulative) lost cells.

  `cum_influx` A named list of matrices, following the format of `live_pop`. The entries give the cumulative influxes of cells in a given number of generation at a given time, relative to the `initial` argument passed along to `fd_proliferation_branching`.

- For Cyton models: same format but `cum_influx` is not available.

See Also

Other simulation-related entries: `fd_predict`, `random-parameters`

Examples

```
un <- fd_unif("gaussian", "branching", n=5)
times <- c(2, 4, 6, 12)
sim <- fd_simulate(un, times,
                   select=c("hists", "hists_lost", "Ns", "Ns_lost"))
unique(sim$Individual)
plot(sim, main="Simulation")

fd_proportions(un, times)

un2 <- fd_unif("gaussian", "cyton", n=5)
fd_proportions(un2, times)
```

FdClearPeaks *Artificial example dataset for one-compartment fluorescence dilution.*

Description

This dataset was generated by `fd_simulate` and features clear, distinct peaks such as those found in immunological applications.

Usage

```
data(FdClearPeaks)
```

Format

This is an `fd_data` with the FMM and proliferation parameters preset using attributes.

Details

This dataset can be best fitted with a Gaussian finite mixture model (fd_fmm_gaussian) and a branching model (fd_proliferation_branching).

See Also

fd_data for data format, fd_model for an example of use.

Examples

```
data(FdClearPeaks)
plot(FdClearPeaks)
```

FdCommonConstraints *Common constraints.*

Description

Those constraints are a useful starting point to constrain a model to make it more identifiable or to capture salient features.

Usage

```
FdCommonConstraints
```

Format

The common constraints are usually loaded in the global environment using attach. Their names start by convention with a '#', as such: '#constraint_name'. See constraints for the format of constraints in general.

Details

Those various constraints can be "combinatorially" combined and their relative merits assessed with AIC (or quasi-AIC, see Pan [2001]). The package **AICcmodavg** can be used to this end. Moreover, practical identifiability can be assessed by looking at the *effect*. See fd_aictab for a wrapper around those two concepts.

The constraints can be combined together as such: `~ `#noss` + `#nodeath` + `#delta_1111`` or using catcstr: `catcstr(`#noss`, `#nodeath`, `#delta_1111`)`.

They are fed to the hierarchical model through function fd_model.

Value

The value of FdCommonConstraints is:

```
list(
  # --- Global ---

  # No 'ss' (third parameter of shifted gamma distributions)
  `#noss` = ~ pro:all:(f0/f/g0/g):{ss <- 0.5},
```

```
# No death, only division
`#nodeath` = ~ pro:all:((g0/g):{ss <- 0.5; delta <- 1; mm <- 1} +
                              {p <- 1; p0 <- 1}),

# --- On 'delta' ---
# Convention: '#delta_abcd', with abcd the constraints respectively
# on f0, f, g0 and g.

# Exponential model
`#delta_1111` = ~ pro:all:(f0/f/g0/g):{delta <- 1},

# Exponential then fixed-length cell cycles
`#delta_1100` = ~ pro:all:((f0/g0):{delta <- 1} + (f/g):{delta <- 0.01}),
`#delta_1101` = ~ pro:all:((f0/g0/g):{delta <- 1} + f:{delta <- 0.01}),

# DGH-like model (Deenick et al. 2003)
`#delta_a101` = ~ pro:all:((g0/g):{delta <- 1} + f:{delta <- 0.01}),

`#delta_aaaa` = ~ pro:all:((f0/g0/g):{delta <- .L2$f$delta}),
`#delta_aabb` = ~ pro:all:(g0:{delta <- .L2$f0$delta} +
                           g:{delta <- .L2$f$delta}),
`#delta_abab` = ~ pro:all:(f0:{delta <- .L2$f$delta} +
                           g0:{delta <- .L2$g$delta}),
`#delta_11a1` = ~ pro:all:(f0/g0/g):{delta <- 1},
`#delta_1a01` = ~ pro:all:((g0/g):{delta <- 1} + f:{delta <- 0.0}),
`#delta_11ab` = ~ pro:all:(f0/g0):{delta <- 1},
`#delta_ab00` = ~ pro:all:((f/g):{delta <- 0.0}),

# --- On 'mm' ---
# Convention: '#mm_xyzw', with xyzw the constraints respectively on
# 'f0', 'f', 'g0', 'g'.
# If two letters are the same, it means that the two corresponding
# distributions are bound together.  The more we progress down the list,
# the simpler the model.

# No constraint on 'mm' (useful for combinatorially combining constraints)
`#mm_xyzw` = NULL,

# Divisions occur at the same pace but death rate free to vary
`#mm_xyxz` = ~ pro:all:(f0:{mm <- .L2$f$mm}),

# Divisions occur at the same pace and deaths at the same pace (but
# different from divisions)
`#mm_xyxy` = ~ pro:all:(f0:{mm <- .L2$f$mm} + g0:{mm <- .L2$g$mm}),

# All the 'mm' are the same: "timescale" model, useful for a very crude
# branching process.
# In this model, 'mm' only gives a broad order of magnitude of the
# transitions (whether division or death)
`#mm_xxxx` = ~ pro:all:(f0/g0/g):{mm <- .L2$f$mm},

# --- On transition probabilities ----
```

```
  # Convention: when relevant, '#p_rst', with rst the constraints respectively
  # on 'p0', 'res0' and 'p'.  If two letters are the same, it means that the
  # two corresponding transition probabilities are bound together.

  # The model does not have transition probabilities, e.g. the Cyton model
  # (useful for "combinatorially" combining constraints)
  `#p_na` = NULL,

  # No constraint on transition probabilities
  # (useful for "combinatorially" combining constraints)
  `#p_rst` = NULL,

  # p0 is the same as p (that is, the proportion of cells that won't die is
  # always the same)
  `#p_rsr` = ~ pro:all:{p0 <- .L1$p},

  # p is p0 * (1 - res0) (that is, the proportion of cells that divide is
  # always the same)
  `#p_RsR` = ~ pro:all:{p <- .L1$p0 * (1 - .L1$res0)}
)
```

See Also

constraints, fd_model, fd_aictab

Examples

```
attach(FdCommonConstraints)
```

FdSTyphimuriumWTC57 *Example dataset for one-compartment fluorescence dilution.*

Description

In this experiment, primary BMM extracted from C57 B1/6 mice were infected by wild-type 12023s *Salmonella* Typhimurium harbouring the pFCcGi plasmid, a plasmid with a constitutive reporter helping in bacterial recognition and an inducible reporter used for fluorescence dilution. The data was acquired on a LSRFortessa cytometer (DB). Raw data is available upon request.

Usage

```
data(FdSTyphimuriumWTC57)
```

Format

This is an fd_data with the FMM and proliferation parameters preset using attributes.

Details

This dataset can be best fitted with a Gaussian finite mixture model (fd_fmm_gaussian) and a branching model (fd_proliferation_branching) with the common constraints

~ `#noss` + `#nodeath` + `#delta_1100` + `#mm_xyxz` (see FdCommonConstraints). As this was a repeat experiment (the field Individual gives an identifier for the experiment), it is also possible to explore the use of mixed-effects models *via* nlme.

Source

Sophie Helaine, Jessica A. Thompson, Kathryn G. Watson, Mei Liu, Cliona Boyle, and David W. Holden (2010). Dynamics of intracellular bacterial replication at the single cell level. *Proc Natl Acad Sci USA*. **107** (8):3746-3751.

See Also

fd_data for data format, fd_model for an example of use.

Examples

```
data(FdSTyphimuriumWTC57)
plot(FdSTyphimuriumWTC57)
```

---

fd_comb *Comb through many constraints using fd_nls.*

Description

Comb through many constraints using fd_nls. The result can then be fed to fd_aictab.

Usage

```
fd_comb(cstr, fmm = "gaussian", proliferation = "branching", data,
  continue = TRUE, mc.cores = 1, file = NULL, globalname = "mdl", ...)
```

Arguments

cstr	A list of constraints to comb through
fmm	A finite mixture model, same format as with fd_model.
proliferation	A proliferation model, same format as with fd_model.
data	The fd_data dataset on which to perform the search.
continue	If TRUE (default), an error in an optimization does not stop the whole search but simply results in the corresponding entry of the return value to be NULL. If FALSE, an error is raised instead.
mc.cores	Number of cores to use for parallel search using **parallel**. mc.cores > 1 fails on Windows machines.
file	Output file (or file name) for parallel search (during which the console output is silenced to avoid conflicts). The output file can be read in real time on POSIX machines using tail -f <file>.
globalname	Global name to use for storing the fd_model object. See fd_model-functions for more details.
...	Additional parameters to pass along to fd_nls.

Value

A named list, each entry being either the result of the optimization (an `fd_nls` object) or `NULL` if the optimization failed, using the constraints `cstr` and in the same order as the constraints `cstr`. The names are the names of `cstr`, if provided, or `constr_<i>`, with `<i>` the index of the constraint if no name was made available.

Required packages

Please install `parallel`.

See Also

`fd_nls`, `fd_aictab`

Other optimization-related entries: `fd_gaussian_fmm_solve`, `fd_minuslogl`, `fd_nls`

Examples

```
# Shared constraints
CC <<- FdCommonConstraints
`#macr` <- ~CC$`#noss` + pro:all:{res <- 0} + fmm:{c0 <- 0} + CC$`#mm_xyxz`

control <- list(maxit = 1L)
## Not run:
control <- list(maxit = 100L, simple.function = TRUE)

## End(Not run)

source(system.file("contrib", "nlsSA.R", package="fluodilution"))
data(FdSTyphimuriumWTC57)
ans <- fd_comb(lapply(list("1100" = CC$`#delta_1100`,
                           "1111" = CC$`#delta_1111`),
                      catcstr, `#macr`),
               data = cutoff(FdSTyphimuriumWTC57),
               control = control,
               nlsfun = nlsSA,
               trace = TRUE)

summary(ans[[1L]])
summary(ans[[2L]])
```

`fd_create` *Create an `fd_data` object from separate cell counts, flow cytometry raw data, ...*

Description

`fd_create` combines together the various datasets that could be used for fluorescence dilution and converts flow cytometry raw data into histograms. `fd_moments` can be used to get a "robust" estimate of the geometric mean and geometric standard deviation of flow cytometry samples.

Usage

```
fd_create(value, fmm = NULL, categories = NULL, breaks = "Sturges",
  momentControl = NULL)

fd_moments(x, channel, clean = TRUE, varnames = NULL)
```

Arguments

value	A named list. Each element represents the data for a different weight class (the names of the list give the weight classes).
fmm	An optional list of finite mixture model arguments to annotate the result. Alternatively, could be "use0" for timepoints at 0h to be used as inoculums, or "usemin" for earliest timepoints to be used as inoculums (when value contains a "flow" element). In this case, the inoculum timepoints are not present as fd_data rows. Instead, the fmm attribute is updated with inoculum information.
categories	A character vector of category names to give their precise order (see proliferation for why it is important).
breaks	For flow cytometry raw data, the breaks to use when forming histograms. If it is a list, each element of the list is used separately for the corresponding flow cytometry timepoint and has the same format as what hist allows. Otherwise, the same breaking rules are used across the board. The rules are the same as with hist, but they apply on an an asinh transformation of the linear fluorescence.
momentControl	A list of arguments that override the default arguments of fd_moments when it is called (that is, when fmm = "use0").
x	A flowSet or flowFrame
channel	The flow cytometry channel featuring the fluorescence dilution.
clean	Whether to clean data by using model-based clustering ('flowClust' package).
varnames	If cleaning is enabled, channels on which to do the model-based clustering on.

Details

If provided, the element value$flow is either a GatingSet (bioconductor package **flowWorkspace**) or a flowSet: that is, a list of flow cytometry raw timepoints. Additional attributes instruct fd_create how to interpret this raw data:

channel Which flow cytometry channel features the fluorescence dilution.

meta A data.frame, each row giving for the corresponding raw timepoint some meta information that will be cbind-ed to the histograms. Meta information include Time, Timepoint, Category, Inoculum, Type. If any but Time is missing, the corresponding columns are filled with default values.

The other elements of value are simply rbind-ed together and the column Weight filled in with the name of those elements.

fd_moments returns either a vector of two elements (x is a flowFrame) or a matrix with two rows (x is a flowSet) giving respectively the log-mean and the log-standard deviation of the samples provided. Additional arguments allow the removal of outliers.

Value

An fd_data object.

Required packages

Please install the bioconductor packages **flowWorkspace** for `fd_create` and **flowClust** for `fd_moments`. Moreover when calling `fd_moments`, because of a bug in **flowClust**, **parallel** must be explicitly attached to the search path using `library(parallel)`. Therefore, **parallel** must be attached as well when calling `fd_create` with `fmm = "use0"`.

Examples

```
# Unzip the archive with the FCS and workspace files
unzip(system.file("extdata", "Archive.zip", package="fluodilution"),
      exdir = tempdir())

library(flowWorkspace)

## Load a flowJo workspace

ws <- openWorkspace(paste0(tempdir(), "/20150325.wsp"))
print(ws)
flow <- suppressWarnings(parseWorkspace(
  ws, "All Samples", path = tempdir(), isNcdf = FALSE,
  cleanup = FALSE, keep.indices = TRUE,
  requiregates = FALSE
))

# Meta information
channel <- "Comp-488-530_30-A"
times <- sapply(
  seq_along(flow),
  function (i) {
    # Just an example of what is possible
    as.numeric(getKeywords(ws, sub("_[0-9]*$", "",
                flow[[i]]@name))$Time)
  }
)
print(getNodes(flow))

# Go for it
meta <- data.frame(Time = times)
ans <- fd_create(
  value = list(flow = structure(flow, meta = meta, channel = channel))
)

# Display result
plot(ans, main="Workspace (1)")

# One can use "use0"
library(parallel)
ans2 <- fd_create(
  value = list(flow = structure(flow, meta = meta, channel = channel)),
  fmm = "use0"
)
print(attr(ans, "fmm"))
print(attr(ans2, "fmm"))

## Alternatively, a flowSet can be provided
```

```
channel <- "FL2-H"
gate <- rectangleGate(`FL1-H` = c(18, Inf), filterId="Bacteria")
src <- read.flowSet(paste0(tempdir(), "/",
                           c("WT-t12.023", "WT-t2.016", "WT-t6.014")))
flow <- Subset(src, filter(src, gate))

meta <- data.frame(Category = "One", Time = c(12, 2, 6), Type = "hists",
                   Inoculum = "inoc_1",
                   Timepoint = c("tp_1", "tp_2", "tp_3"))
value <- list(flow = structure(flow, meta = meta, channel = channel))
ans3 <- fd_create(value = value)
plot(ans3, main="Workspace (2)")

## fd_moments can be used to get, e.g., m0 and sd0 directly
library(parallel)
print(fd_moments(flow, channel))
```

`fd_data` *Manage a fluorescence dilution dataset.*

Description

A fluorescence dilution dataset is a `data.frame` of fluorescence histograms and Colony Forming Units (CFUs)/cell counts for various timepoints. It can contain additional attributes that can be used to simplify the construction of the finite mixture model (FMM) or proliferation model when calling `fd_model` or `fd_unif`. `fd_data` ensures the conformity of such a dataset by performing various checks. Subsetting and `rbind`-ing are also available for an `fd_data`, as well as various transformation and display mechanisms.

Usage

```
fd_data(data, categories = NULL, inoculums = NULL, timepoints = NULL,
  na.action = na.pass)

fd_transform(x, hist = "identity", N = "log10")

cutoff(x, cutoff = NULL, threshold = 0.01)

fd_expand(x, length.N = 50, seq = NULL, separate = TRUE, by = NULL)

## S3 method for class 'fd_data'
plot(x, type = "overview", main = NULL, ...)
```

Arguments

`data`	The original `data.frame` (or perhaps a `groupedData`).
`categories`	The specific order for the category levels. If `NULL` and the `Category` field is not yet a factor, the levels will be in lexicographical order.
`inoculums`	Same with the `Inoculum` field.
`timepoints`	Same with the `Timepoint` field.

na.action	An na.action object to apply to data.
x	An fd_data object.
hist	A transformation object from the (graphical) package **scales** (or a character string converted to a transformation object) and to be applied to the y field when Weight == "hist".
N	A transformation object from the (graphical) package **scales** (or a character string converted to a transformation object) and to be applied to the y field when Weight == "N".
cutoff	If specified, overrides the behaviour of the attribute cutoff of data. If a named vector of multiple entries, it contains the cutoff points in linear units of fluorescence (the names are the timepoint identifiers). If it is a scalar, gives the maximum number of generations to consider: the cut is then made at exp(m0) / 2^cutoff, with m0 the log-mean of the sample.
threshold	Below this value, proportions in fluorescence histograms are discarded. Can help with egregious heteroskedasticity problems.
length.N	The new number of cell counts per group.
seq	Alternatively, the times at which to calculate the cell counts can be explicitly given.
separate	If TRUE (default), the new timepoints to consider are found independently for each group. This is useful to avoid extrapolation when data is unbalanced.
by	The additional columns with which to do the grouping (Grouping is already made by Category). If NULL, separate = TRUE and data inherits groupedData, this is taken from the grouping information of data. Otherwise, no additional grouping is used.
type	A character vector containing the types of plots to show. If "overview", a selection of plots are shown on the same page. Otherwise, the plots are shown sequentially. Plots include "hist" (fluorescence histograms), "N" (cell count), "range" (an overview of cutoff points and initial log-mean fluorescence and log-standard deviation), "balancing" (data balancing across times and categories), "coverage" (in terms of number of cells observed under flow cytometry, if available) and "cutoff" (the percentage of the population below the cutoff point). Additionally, "all" shows all these plots at once.
main	Plot title when type="overview".
...	Additional arguments to the generics summary and plot, not used here.

Details

fd_data is used to construct and check the validity of a fluorescence dilution dataset. fd_transform applies transformations to histograms and cell counts and change the attributes "fmm" accordingly. Transformations are not chained: the dataset is back transformed using the current (or default!) transformation, then the new transformation is applied. cutoff returns a subset of the dataset where the points below the cutoff have been discarded. Notice that subset cannot be used instead of cutoff as the implementation for fd_data checks that the histograms sum to one. fd_expand uses predict to fill in more predicted timepoints between the experimental timepoints (interpolation), usually for plotting purposes. plot can be used for a quick look at the dataset in order to get an overall "feel" and spot potential mistakes.

Value

An fd_data object.

Format

data should be a data.frame (or a groupedData) with at least the following columns:

Type The type of the measurement. Current allowed types are "hists" (histogram of living cells), "hists_lost" (histogram of dying/dead cells), "Ns" (cell counts/CFUs), "Ns_lost" (cell counts of dying/dead cells), "props" (proportions of living cells in any given number of generations), "props_lost" (same for dying/dead cells). In the case of histograms, one row represents a bin. However, other types can added by the user, provided that a new processing function is given to fd_model through the argument process.

Weight Either "hist" (for types "hists" and "hists_lost"), "N", "prop" or any other weight defined by the user (with the right processing function, see argument process of fd_model.

Inoculum An identifier for the inoculum/progenitors used. This allows to associate the histograms with the geometric mean and standard deviation of the initial population. Non-histogram based types can use "none" for this field.

Timepoint A unique identifier to group together all the measurements made together on the same sample. Notice that if you do technical replicates in flow cytometry, the resulting histograms must be listed as different timepoints as within the same timepoint there would be no way to know which bin pertains to which histogram.

Time The time, in hours, at which the timepoint were taken.

Category The category/compartment (a factor) for the current row.

a, b The left (excluded) and right limits (included) of the bin if Type %in% c("hists", "hists_lost"), in linear units of fluorescence. If Type %in% c("props", "props_lost") , a is the number of generations (starting from 0) and b is ignored. In the case of a different type, should NOT be NA as the whole row can be interpreted as missing data by the fitting algorithm or the fitting fails if the na.action is na.fail. We recommend in this case to set a and b to 0, as they won't be read out by the algorithms anyway.

y If Type %in% c("hists", "hists_lost") , this is the proportion of cells in bin (a, b] (right-closed interval, the default of hist). This is a probability and not a density and all the rows for a given histogram should sum to one (this is checked by this function). Alternatively, if Type %in% c("Ns", "Ns_lost") , this is the cell count/CFU count and if Type %in% c("prop", "prop_lost") , this is the proportion of cells in a given number of generations. The y can be transformed, see below the cctrans and htrans parameters passed to the FMM.

The following attributes can also be set (but setting them is not mandatory, default values are provided):

fmm List the parameters that will be passed to the finite mixture model (FMM), see finite-mixture.

proliferation List the parameters that will be passed to the proliferation model, see proliferation.

cutoff A named numeric vector containing the cutoff points in linear units of fluorescence (the names are the timepoint identifiers). See cutoff.

counts If the user wants to store the number of events per histogram, they are advised to use this field. It should then be a named numeric vector (the names would be the timepoint identifiers).

Examples

```
# FdSTyphimuriumWTC57 is already an fd_data, so the following
# does not do much
data(FdSTyphimuriumWTC57)
dat <- fd_data(FdSTyphimuriumWTC57)
```

```
# Visually assess the data
plot(dat)

# It is possible to perform standard operations on an fd_data,
# as you would do with a data.frame, but with additional checks
datsub <- subset(dat, Individual == "160408.WT.C57")
datbind <- rbind(datsub, subset(dat, Individual == "010708.WT.C57"))

# To disable checks, use e.g. subset.data.frame
invisible(subset(dat, y > 0.1))
  # Warning: some histograms do not have proportions
  # that sum to 1
invisible(subset.data.frame(dat, y > 0.1))
  # Silently returns a data.frame (not an fd_data anymore)

# Cutoff returns an fd_data for which histograms do not have
# proportions that sum to 1, without issuing any warning
invisible(cutoff(dat))

# Sometimes, it is useful to extend the time domain for, e.g.,
# plotting
length(unique(dat$Time))
length(unique(fd_expand(dat, length.N = 50))$Time)
# This expanded dataset can then be used as the 'newdata' argument
# of 'predict'.

# fd_transform can be used for arbitrary transformation
# Notice that here Ns are already log10-transformed
# (this is the default transform as defined in FMM)
dat_tr <- fd_transform(fd_simulate(fd_unif(), c(2, 12, 24)),
                       hist = "log1p")
```

`fd_gaussian_fmm_solve` *Get the proportions of cells in any given number of generations.*

Description

The result of this function can be fed to `fd_nls` (it returns a `data.frame` that can be `rbind`-ed to form an `fd_data`). Notice that this intermediary step is usually NOT performed by the algorithm that gets the biological constants in this manner: the finite mixture model layer (see `finite-mixture`) and the proliferation model are better combined in a hierarchical fashion and fitted together, as advocated by Hyrien (2008).

Usage

```
fd_gaussian_fmm_solve(data, fmm = "gaussian", mgen = 10L, rm = TRUE)
```

Arguments

`data`	An `fd_data` object that represents the population to fit.
`fmm`	The finite mixture model to use for the fitting (see `finite-mixture`).
`mgen`	The maximum number of generations to fit. Notice that by convention the first generation has number 0, so there will be `mgen+1` generations fitted.
`rm`	Whether to remove the `"hists"` and `"hists_lost"` types in the return value.

Details

This function gives the proportion of cells in any given number of generations from fluorescence histograms. This function is similar to implementations in the bioconductor package **flowFit** but is integrated with `fd_data` dataset thinking and uses a constrained least square approach (quadratic programming) instead of a Levenberg-Marquardt least square (this makes more sense from a mathematical point of view).

Value

The `fd_data` object passed as argument augmented by two new types: `"props"` and `"props_lost"` (if any `"hists_lost"` data).

Required packages

Please install **limSolve**.

References

Hyrien, Ollivier and Zand, Martin S (2008). A Mixture Model With Dependent Observations for the Analysis of CSFE-Labeling Experiments. *Journal of the American Statistical Association* **103** (481): 222-239.

See Also

`fd_fmm_gaussian`

Other optimization-related entries: `fd_comb`, `fd_minuslogl`, `fd_nls`

Examples

```
# Load an artificial data set with clear peaks
data(FdClearPeaks)
plot(FdClearPeaks)

# Solve for proportions
fd_gaussian_fmm_solve(FdClearPeaks,
                      model(attr(FdClearPeaks, "params"))$fmm)
```

`fd_minuslogl` *Minus log-likelihood of a fluorescence dilution model.*

Description

This function returns a minus log-likelihood function to be used with general maximum likelihood estimation packages such as **stats4** or **bbmle**.

Usage

```
fd_minuslogl(model, data, start = NULL, mgen = "guess", verbose = TRUE,
  weights = NULL, stop_boundary = FALSE)
```

Arguments

model	An fd_model object.
data	An fd_data dataset.
start	Optional starting parameters. The default behaviour (NULL) is to use the starting parameter provided by start(model).
mgen	Maximum number of generations to fit. If NULL, what the proliferation model provides by default is used.
verbose	If TRUE (default), print additional diagnostic messages.
weights	A numeric vector of length equal to the number of rows of data. Gives the relative weights to put on the residuals. Default (NULL) to equal weight.
stop_boundary	Whether fd_predict should raise an error if the parameters cross the boundary. By default (FALSE), just returns a very large number when it happens.

Value

Minus log-likelihood (same as logLik applied to an nls object).

See Also

Other optimization-related entries: fd_comb, fd_gaussian_fmm_solve, fd_nls

Examples

```
library(stats4)
library(bbmle)

# Create minuslogl function
data(FdSTyphimuriumWTC57)
dat <- subset(cutoff(FdSTyphimuriumWTC57), Individual == "140508.WT.C57")
mdl <- fd_model(dat, fmm="gaussian", proliferation="branching",
                constraints = attr(dat, "bestcstr"))
fun <- fd_minuslogl(mdl, dat, verbose=FALSE)
print(fun)

# This this function on lower parameters
do.call(fun, as.list(start(mdl)))

# Optimize this minuslogl.  Notice that the 'start' parameter is not
# explicitly given as the arguments to 'fun' have default values
fun <- fd_minuslogl(mdl, dat, verbose=TRUE)
control <- list(maxit=1)
## Not run:
control <- NULL

## End(Not run)
stats4::mle(fun, method="Nelder-Mead", control=control)

# The user is advised to use GenSA for global search:
bbmle::mle2(fun, optimfun=GenSA::GenSA, control=control)
```

fd_model *Create and maintain a fluorescence dilution (FD) model.*

Description

These functions create and maintain a fluorescence dilution (FD) model, linking together a Finite Mixture Model (FMM), a proliferation model and a set of constraints.

Usage

```
fd_model(fmm = "gaussian", proliferation = "branching", data = NULL,
  constraints = NULL, partial = NULL, boxed = TRUE,
  process = "fd_process_default", memoise = TRUE)

fd_clean(object)

fd_clone(object, clean = TRUE)

## S3 method for class 'fd_model'
update(object, data = NULL, fmm, proliferation, addcstr,
  start, partial, boxed, process, ...)
```

Arguments

fmm	A Finite Mixture Model (FMM), see finite-mixture.
proliferation	A proliferation model, see proliferation.
data	An optional fd_data used to construct the FMM and the proliferation model. If not specified, default values are used.
constraints	A cstr object, as produced by cstrlist or an expression if only the constraint component is set (see constraints).
partial	A numeric vector giving the index (not the name) of the categories in the dataset to consider for fitting. By default, all the categories are considered. Currently, the categories can only be contiguous and include 1 (e.g., 1:3 but not 2L:3L). partial helps to shave off some computation time.
boxed	Whether the parameters of the model should be boxed by upper and lower bounds (TRUE) or "unboxed" by an atanh transformation so as to be used with optimization methods that do not allow the specification of a boundary (such as the default algorithm of nls). However, if the likely optimal parameters are far from the boundary, boxed=TRUE can in general be used (for example with nlme).
process	A function that is used internally by fd_predict. For advanced use, the default fd_process_default can be overridden to accommodate for, e.g., "weights" other than fluorescence histograms and cell counts (see source code of fd_process_default, exported but not documented, for format).
memoise	Whether the Finite Mixture Model stage should be memoised with package **memoise**. Memoisation drastically improves performance but should probably be turned off, out of memory concerns, when optimizing autofluorescence level and specific number of molecules with FMMs fd_fmm_af and fd_fmm_af_bp.
object	An fd_model object.

clean	Whether fd_clean should be called as well.
addcstr	Additional constraints to be added (using catcstr) to the current set of constraints. Either a cstrlist or an expression if only the constraint component is to be updated.
start	Numeric vector. Overrides the starting parameters available through start(object).
...	Not used.

Details

fd_model creates a new fluorescence dilution, two-step hierarchical model. This model can then be optimized with fd_nls. Predicting experimental values necessitate the calculation of internal quantities that can be bulky in memory: fd_clean provides for their removal. Because models are environments, copying them can only be done through fd_clone. update updates an existing model by changing some of its arguments.

Value

fd_model returns an fd_model object (an environment). fd_clean modifies the model passed as a parameter and does not have a meaningful return value. fd_clone, on the other hand, returns a model that is not identical to the model passed along as argument. update also returns a new environment.

From full parameter list to vector of free parameters

The parametrization of an FD model can be conveniently made with nested lists (see constraints). More specifically, the top-level list has two elements named fmm and pro (for the format of those two elements, see respectively finite-mixture and proliferation). However, multidimensional optimization functions only take vectors as arguments (the "free" parameters). Therefore, for simple problems, the starting parameters and lower/upper bounds are unlisted and passed as arguments and the model is evaluated using a relisting. Here, we needed to introduce further steps along the way.

To anchor the discussion, we call the parametrization with which the model is eventually evaluated the "natural parametrization". A first step away from this parametrization and towards the vector of free parameters involves reparametrizing some parameters to allow the specification of bounds independent from each other and rescaling to speed up convergence. In the case of the branching processes, because the proportions must sum to 1, i.e. `p + res + d = 1`, with p the proportion of cells that divide, res the proportion of non-growers and d the proportion of cells that die or leave the compartment, we reparametrize p and res to `` `p'` = p + res `` and `` `res'` = res / (p + res) ``, allowing us to give the lower bounds `` c(`p'` = 0, `res'` = 0) `` and upper bounds `` c(`p'` = 1, `res'` = 1) ``. Alternatively, Lagrange multipliers could have been used, but this leads to instability in such large nonlinear problems.

From this reparametrized version, we go on to constraining with constraints. Constraining allows to reduce the dimension of a problem that would be too complicated to solve, even using global search (the "curse" of dimensionality), and in any case likely to be overparametrized in practice (henceforth not identifiable in practice).

The free parameters that are not constrained are then unlisted, and it is this named vector that is given to optimizers and returned by start, lower and upper.

In short, the steps from the natural parametrization to free parameters are:

```
Action                Object
--------------        --------------
                      (1) natural
transformation   ->   (2) reparametrized
constraining     ->   (3) constrained
unlisting        ->   (4) unlisted
```

and naturally when evaluating the model the reverse procedure is applied:

```
Action                        Object
----------------------        --------------
                              (4) unlisted
relisting                ->   (3) constrained
expansion                ->   (2) reparametrized
inverse transformation   ->   (1) natural
```

Warning

The user should check with `lower` and `upper` that the boundary suits their needs. If they do not, the `constraints` argument should be a `cstrlist` object.

See Also

`fd_model-functions` for functions to be used with an `fd_model` object, `fd_simulate` for simulating a population, `fd_nls` for fitting an FD model.

Examples

```
# Create a new model with default values
mdl <- fd_model()
print(mdl)
print(summary(mdl))

# Original size
sum(sapply(ls(mdl), function (nm) object.size(mdl[[nm]])))
sim <- fd_simulate(fd_draw_unif(mdl), c(2, 6, 12))

# Do calculations
fd_minuslogl(mdl, sim, verbose=FALSE)()
sum(sapply(ls(mdl), function (nm) object.size(mdl[[nm]])))

# Clean up a bit
fd_clean(mdl)
# Reduced size
sum(sapply(ls(mdl), function (nm) object.size(mdl[[nm]])))

# Clone
mdl2 <- fd_clone(mdl)
identical(mdl, mdl2)  # FALSE

# Update model (updating clones as well)
mdl3 <- update(mdl2, proliferation="cyton")
print(mdl2$pro)
print(mdl3$pro)
identical(mdl2, mdl3)  # FALSE
```

`fd_model-functions` *Miscellaneous functions for the optimization of an FD model.*

Description

These functions can be used when calling nonlinear optimization functions such as `nls`, `nlsList`, `nlme` or `gnls` (alternatively, the wrapper `fd_nls` can be used).

Usage

```
fd_formula(globalname, mgen = NULL)

## S3 method for class 'fd_model'
start(x, free = TRUE, ...)

lower(x, ...)

## S3 method for class 'fd_model'
lower(x, free = TRUE, ...)

upper(x, ...)

## S3 method for class 'fd_model'
upper(x, free = TRUE, ...)

## S3 method for class 'fd_model'
relist(flesh, skeleton)

fd_freepar(structured, x)
```

Arguments

`globalname`	The name of an `fd_model` object sitting in the global environment (that is, a `character` object). See below for the rationale.
`mgen`	Maximum number of generations (if `NULL`, the default of the proliferation model is taken).
`x`	An `fd_model` object.
`free`	Should the starting, lower or upper values returned by `start`, `lower` and `upper` be given only for the free parameters (in this case a named vector is returned) or for all the parameters, including the constrained ones (in this case a list is returned)?
`...`	Not used.
`flesh`	A named vector of free parameters, such as returned by `start(skeleton)`.
`skeleton`	An `fd_model` object.
`structured`	An unconstrained, transformed and structured list of parameters (e.g., as returned by `relist`).

Value

fd_formula builds a formula suitable for using with nls, nlsList, nlme or gnls. start returns the default starting parameters for the FD model, either as a named vector of free parameters (the default) or an unconstrained, transformed and structured list of parameters. lower and upper are two new generic functions, aligned with the way start is defined in the stats package (originally with time-series in mind), and for an fd_model return the lower and upper bounds along the same modalities. relist takes a named vector of free parameters flesh such as the ones returned by start, lower and upper (when free=TRUE) and turns them into an unconstrained, transformed and structured list of parameters. fd_freepar performs the inverse operations and from a the result of relist returns a named vector of free parameters.

Why a "global name" instead of a regular object

Because nls-like functions and nlme use different scoping rules, great care must be exercised concerning the scoping of the functions called in a formula. Indeed, for nls, the functions are first looked at in the environment of the formula, then in the data parameter if it is a list or an environment, and finally in the global environment. For nlme, however, the scope is *always* the global environment (Lumley 2003) and data can only be a data.frame for obvious reasons (it is cut down in many pieces according to the various levels of grouping). In the end, to stress this point, we decided that not an fd_model object should be passed along to formula but the *name* of an fd_model (that is, a character object) sitting in the global environment. This implementation, of course, has many drawbacks, including the mandatory use of global variables, but it seems to be the only sensible one when using such complex nonlinear models within the formula paradigm. To avoid such a contortion, general likelihood optimizers such as mle or mle2 could well be used instead of nls (see fd_minuslogl). Unfortunately, no such alternative exists for either nlme or gnls.

References

Lumley T (2003). *Standard nonstandard evaluation rules.* http://developer.r-project.org/nonstandard-eval.pdf

Examples

```
data(FdSTyphimuriumWTC57)
dat <- cutoff(FdSTyphimuriumWTC57)
mdl <<- fd_model(data=dat, constraints=attr(dat, "bestcstr"))

rbind(start = unlist(relist(start(mdl), mdl)),
      lower = unlist(relist(lower(mdl), mdl)),
      upper = unlist(relist(upper(mdl), mdl)))

# fd_formula can be fed directly to, e.g., nls
# (alternatively, fd_nls can be used: it is essentially a wrapper
# around nls with additional checks and an automatic, albeit far from
# perfect, determination of mgen)
fd_formula("mdl")

control <- list(maxit = 1L)
## Not run:
control <- NULL

## End(Not run)

# Let's use the "port" algorithm of 'nls'
```

```
## Not run:
fit <- nls(fd_formula("mdl"), data = FdSTyphimuriumWTC57,
           algorithm="port", start=start(mdl),
           lower=lower(mdl), upper=upper(mdl),
           control=control)
# Error in nls(fd_formula("mdl"), data = FdSTyphimuriumWTC57,
# algorithm = "port",  :
#     Convergence failure: iteration limit reached without convergence (10)

## End(Not run)

# As it is likely to fail, we can use the global search
# algorithm 'GenSA' as well, again in an 'nls'-like framework
source(system.file("contrib", "nlsSA.R", package="fluodilution"))
fit <- nlsSA(fd_formula("mdl"), data = FdSTyphimuriumWTC57,
             start=start(mdl),
             lower=lower(mdl), upper=upper(mdl),
             control=control)

# A third option is to use fd_minuslogl, see the relevant
# page for an example.
```

`fd_nls` *Ordinary Least Square (OLS) optimization of a fluorescence dilution model.*

Description

`fd_nls` acts as a wrapper around an `nls`-like optimizer with additional checks and capabilities. Other functions support the activity of `fd_nls`.

Usage

```
fd_nls(globalname, data, mgen = "guess", start = NULL, loop = 3,
  trace = TRUE, control = NULL, ..., nlsfun = nls)

model(object)

fd_residuals(data, ..., .list = NULL)

## S3 method for class 'fd_nls'
predict(object, newdata = stop("'newdata' is missing"),
  model = NULL, ...)

relisted_coef(object, drop = TRUE)

relisted_vcov(object, vcov. = vcov, n = 1000)

relisted_fit(object, ...)

## S3 method for class 'relisted_fit'
vcov(object, ...)
```

```
guess_mGen(data)
```

Arguments

globalname	The name of an fd_model object sitting in the global environment (that is, a character object). See fd_model-functions for the rationale.
data	The (experimental) fd_data dataset.
mgen	The maximum number of generations (integer). If NULL, the default of the proliferation model is considered. If "guess", guessed through guess_mGen.
start	Alternative starting coefficients to start.
loop	If mgen is not enough (there is a lot of cells in the maximum number of generations), number of additional optimization rounds that fd_nls can undertake with ever larger mgen. This parameter is important for automatic combing, e.g. with fd_comb.
trace	Whether to output additional trace information.
control	Control parameters to pass along to the optimizer.
...	For fd_nls, additional arguments to pass along to the optimizer. For fd_residuals, the fd_nls, nls, nlsList or nlme objects to find the residuals of (the arguments have to be named). For relisted_fit, additional parameters to pass along to relisted_vcov. Not used by predict or vcov.
nlsfun	A user-specified optimization function. Must (at least) accept the same arguments as nls (default). The "contributed" nlsSA function can also be used (see details).
object	An fd_nls, nls, nlsList or nlme object.
.list	Named list of additional fd_nls, nls, nlsList or nlme objects.
newdata	The new fd_data dataset for which to make predictions. Notice that, contrary to predict.nls, here newdata cannot be missing.
model	An fd_model object. Must be specified if object is not an fd_nls object (i.e. model(object) == NULL).
drop	Whether to return the relisted parameters directly if there is only one set of coefficients or a one-element list containing the relisted parameters.
vcov.	Variance-covariance function to use on the free parameters. For example, the **sandwich** package offers an alternative to vcov and could be used for a Huber-White estimation. Alternatively, variance-covariance matrix to use.
n	Number of Monte-Carlo simulations to run.

Details

fd_nls returns an fd_nls object, inheriting nls (the return class of nls), augmented by model information. Therefore, the functions that can be used on an nls object can be used on an fd_nls object, such as coef, predict or summary.

model returns the fd_model object that was used to perform the optimization. fd_residuals returns data with the content of the y column replaced by the predicted values. predict returns a numeric vector of the predicted values for the fd_data dataset newdata. relisted_coef is a wrapper around relist.fd_model and return the relisted optimal coefficients. In some cases, e.g. nlme and nlsList, instead of a vector, coef returns a matrix with the coefficients in column and the group levels in rows. In this case, relisted_coef returns a list of relisted coefficients. If drop=FALSE, in

the case only one set of coefficients is returned, a one-element list of relisted coefficients is returned instead of the relisted coefficients directly.

`relisted_vcov` uses a Monte Carlo simulation, dependent on package **mvtnorm**, to get to the variance-covariance matrix of the relisted parameters from the variance-covariance matrix of the free parameters returned by `vcov` (using singular value decomposition). The rows and columns are ordered as per

`names(unlist(relisted_coef(object)))`.

`relisted_fit` calls both `relisted_coef` and `relisted_vcov` and returns an object with which `coef` and `vcov` can be used.

Value

`fd_nls` returns an `fd_nls` object, `model` an `fd_model` object, `fd_residuals` returns a `data.frame`, `relisted_coef` returns a list of structured parameter values, `relisted_vcov` returns a variance-covariance matrix, `relisted_fit` an object with which `coef` and `vcov` can be used (see *details*).

Remark

Instead of `nls`, `GenSA` can be used for optimization through the "contributed" wrapper `nlsSA`. `nlsSA` can be sourced with

`source(system.file("contrib", "nlsSA.R", package="fluodilution"))`.

Required packages

Please install **mvtnorm** for `relisted_vcov` and `relisted_fit`.

See Also

Other optimization-related entries: `fd_comb`, `fd_gaussian_fmm_solve`, `fd_minuslogl`

Examples

```
data(FdSTyphimuriumWTC57)
dat <- cutoff(FdSTyphimuriumWTC57)
mdl <<- fd_model(data = dat, constraints = attr(dat, "bestcstr"))
guess_mGen(dat)
control <- list(maxit = 1L)
## Not run:
control <- NULL

## End(Not run)

## Not run:
fit <- fd_nls("mdl", dat,
              algorithm="port", control=control)
# Error in match.fun(nlsfun)(fd_formula(globalname, mgen = mgen), data,  :
#     Convergence failure: iteration limit reached without convergence (10)

## End(Not run)

# Alternatively, GenSA can be used through nlsSA:
source(system.file("contrib", "nlsSA.R", package="fluodilution"))
fit <- fd_nls("mdl", dat, nlsfun=nlsSA, control=control)
```

```
## Not run:
# This gives a summary for free parameters
print(summary(fit))

## End(Not run)

# To go back to the unconstrained parameters, one can use
# relisted_fit or relisted_coef/relisted_vcov separately
relfit <- relisted_fit(fit)
invisible(relisted_coef(fit))
invisible(relisted_vcov(fit))

# It is also possible to use a sandwich estimator instead
relfit <- relisted_fit(fit, vcov. = sandwich::sandwich)
print(lmtest::coeftest(relfit))
```

`fd_peaks` *Locate the distinct peaks in fluorescence dilution histograms.*

Description

Find the peaks, using a smoothing method, of fluorescence histograms. This could be useful in order to eschew the *a priori* assumptions concerning the evolution over time and generations of the space between those peaks that finite mixture models make (Hyrien et al. (2008) have showed that the "naive", Gaussian FMM is sometimes inappropriate). However, clear peaks are not discernible in microbiology applications so far, restricting this technique to immunological applications.

Usage

```
fd_findpeaks(data, plot = TRUE, w = 5, span = 0.1)

fd_peaks(data, peaks, rm = TRUE)
```

Arguments

`data`	An `fd_data` dataset.
`plot`	Whether a diagnostic plot of the peaks should be displayed.
`w`	Window parameter: how large (in number of histogram bins) should the window used to calculate the maximum be?
`span`	The parameter that controls the degree of smoothing in `loess`.
`peaks`	The return value of an `fd_findpeaks` call, with an additional `Generation` column mapping the peaks to numbers of generations.
`rm`	Whether the rows with `Weight=="hist"` should be removed in the returned `fd_data` object.

Details

`fd_findpeaks` returns a `data.frame` giving the `"Timepoint"`, position of the peak `"x"`, proportion of cells in the peak `"prop"` and position of the peaks from right to left `"i"`, starting from 1. Notice that the number of generation is not directly given as sometimes very low proportions could mean that a generation is skipped and not recorded as a peak. Since this should not pose any problem, apart from labelling, the labelling should be done by the user (see the examples section for a straightforward simple mapping).

`fd_peaks` takes such a mapping and appends to the dataset the proportions, with weight `"prop"` and types `"props"` (corresponding to `"hists"` histograms) and `"props_lost"` (corresponding to `"hists_lost"` histograms).

Value

`fd_findpeaks` and `fd_peaks` both return a `data.frame` (see *details*).

Required packages

Please install **zoo**.

See Also

`finite-mixture`, `fd_gaussian_fmm_solve`.

Examples

```
data(FdClearPeaks)
dat <- FdClearPeaks

# The above dataset has a low noise level, therefore
# no transformation is really needed to map peaks to
# generation numbers
peaks <- transform(fd_findpeaks(dat, span=0.05, w=6),
                   Generation = i - 1)
fd_peaks(dat, peaks, rm=TRUE)
```

`fd_predict` *Predict the histograms and cell counts of an* `fd_data` *dataset from a set of model parameters.*

Description

Predict the histograms and cell counts of an `fd_data` dataset from a set of model parameters. `fd_predict` is used internally when `fd_formula`, `fd_simulate` or `fd_minuslogl` are called, but it can also be called directly.

Usage

```
fd_predict(model, param, data, mgen = NULL, stop_boundary = FALSE)
```

Arguments

model	An fd_model object.
param	A named vector of free parameters, such as returned by start.
data	An fd_data dataset.
mgen	Maximum number of generations. If NULL, taken from the default found in the proliferation model.
stop_boundary	Whether the function should stop when param is out of bounds or simply return very large predicted values.

Value

The original fd_data, not necessarily in the same order, with the "y" column filled with the predicted values.

See Also

Other simulation-related entries: dataset-simulation, random-parameters

Examples

```
library(dplyr)

data(FdSTyphimuriumWTC57)
dat <- cutoff(FdSTyphimuriumWTC57)
dat <- dat[order(dat$Type, dat$Inoculum), ]
mdl <- fd_model(data=dat, constraints=attr(dat, "bestcstr"))
pred <- fd_predict(mdl, start(mdl), dat)
if (any(all.equal(pred %>% dplyr::select(-y),
                  as.data.frame(dat) %>% dplyr::select(-y)) != TRUE))
  stop("should be equal")
dat_residuals <- pred$y - dat$y
```

---

finite-mixture *Finite mixture models.*

Description

Finite mixture models relate the fluorescence to the numbers of generations. They are based on the decomposition, using Bayes' law, of the fluorescence histograms into various cohorts whose individual shape is derived from properties of the inoculum and hypotheses concerning the impact of autofluorescence (AF) and binomial partitioning (BP). By extension, our implementations also contain variables concerning the behaviour of cell counts and act more generally as an interface between the dataset and the proliferation model.

Usage

```
fd_fmm_af(...)

fd_fmm_af_bp(...)

fd_fmm_gaussian(...)
```

Arguments

`...` Not used by `summary`. For the other functions:

`m0, sd0` Log-normal mean and standard deviation of fluorescence in the inoculum/of progenitors. Either a scalar or a named numeric vector, with the names being the levels of the `"Inoculum"` column of the `fd_data` dataset, in the exact same order.

`cctrans` The transformation that has been applied to the cell counts of the dataset, if any. `"log10"` by default. See package **scales** for other possible values.

`htrans` The transformation that has been applied to the proportions found in the fluorescence histograms, if any. `"identity"` by default. See package **scales** for other possible values.

Details

The simplest hypothesis to make is that the successive cohorts are no more than translations, in logarithmic units, of the inoculum. If the inoculum is furthermore considered log-normally distributed, the resulting model is the Gaussian FMM (`fd_fmm_gaussian`). `fd_fmm_af` also takes autofluorescence into account and `fd_fmm_af_bp` takes both autofluorescence and binomial partitioning into account (Chauvin *et al.* 2016). Autofluorescence impacts the shape of the cohorts: as the "real" fluorescence dilutes, autofluorescence becomes more important relative to "real" fluorescence. Binomial partitioning, or how fluorescent molecules are partitioned at division between daughter cells, creates an asymmetry whose magnitude depends on the number of molecules in the mother cell. `fd_fmm_af` and `fd_fmm_af_bp` are significantly slower than `fd_fmm_gaussian`. An alternative to mitigate the effect of AF and BP is to gate away the low fluorescence, where it is the most distorted, using `cutoff`.

Value

An `fd_fmm` object, suitable for use with `fd_model`, `fd_unif`, ...

Parallel computing

`fd_fmm_af_bp` makes use of `mclapply`. The number of clusters to use can be set using, e.g., `options(mc.cores = 4L)`.

Required packages

For `fd_fmm_af_bp`, please install **parallel**.

Parametrization

`c0` Make the link between the relative cell counts `Nrel` given by the model and the experimental cell counts `Nexp`: `Nexp = Nrel * 10^c0`.

`sdaf` Standard deviation of fluorescence, in linear units, for `fd_fmm_af` and `fd_fmm_af_bp`.

`ftor` Log-specific fluorescence: that is, the number of molecules for a given fluorescence level `x` (in linear units) is `x / exp(ftor)`.

Reparametrization

The natural parametrization (see `fd_model`) does not differ from the reparametrized form.

Word of caution

In all those models, the mean autofluorescence is assumed to be 0. This supposes that the flow cytometer was correctly calibrated. If this was not the case, autofluorescence needs to be measured independently and the original fluorescence histograms shifted.

References

McLachlan JG, Peel D (2000). *Finite Mixture Models.* John Wiley & Sons. ISBN 978-0-471-00626-8.

See Also

fd_gaussian_fmm_solve.

Examples

```
fd_model(fmm="gaussian")
fd_model(fmm=fd_fmm_gaussian(m0 = 10, sd0 = 0.3,
                             cctrans="log1p", htrans="identity"))
fd_fmm_af(m0 = 10, sd0 = 0.3)
fd_fmm_af_bp(m0 = 10, sd0 = 0.3)
```

---

gamma-distributions *Reparametrization and evaluation of shifted gamma distributions.*

Description

Our proliferation models use gamma distributions for times to division/death. They are parametrized by their mean, coefficient of variation and a parametrization of the skewness. These functions help in the manipulation of such distributions.

Usage

```
fd_gamma_dist(a, b, loc = 0, mul = 1)

fd_reparam_gamma_dist(mm, delta = 1, ss = 0.5)

fd_pack_dist(dst, mul = 1)

fd_pdist(t, g)

fd_ddist(t, g, dt = NULL)
```

Arguments

a	The shape α of the gamma distribution.
b	The rate β of the gamma distribution.
loc	The location parameter λ.
mul	A multiplier factor. Used internally for representing mixtures.

`mm`	The mean of the gamma distribution, in hours.
`delta`	The coefficient of variation (i.e., standard deviation over mean) of the gamma distribution. It is dimensionless and should lie between 0 and 1.
`ss`	The reparametrization of the location parameter. It is dimensionless and should lie between 0, excluded (high skewness as compared to a non-shifted gamma distribution), and 0.5, included (same skewness as compared to a shifted gamma distribution).
`dst`	A list of the parameters of a gamma distribution in their reparametrized form: `dst$mm`, `dst$delta`, `dst$ss`.
`t`	A numeric vector of times in hours at which to evaluate the gamma distribution.
`g`	A gamma distribution in the natural form (output of `fd_gamma_dist`).
`dt`	A list of small time increments for evaluating the gamma distribution (see details).

Details

Gamma distributions are usually parametrized using scale and shape parameters. Additionally, a shifted gamma distribution makes use of a location parameter. It is possible to use a more "natural" parametrization from the mean, coefficient of variation and an alternative reparametrization of the location parameter (we could think of it as a *second-order* shape).

`fd_gamma_dist` takes the parameters of a shifted gamma distribution in their non-reparametrized (natural) form and returns a one-row matrix with those parameters. `fd_reparam_gamma_dist` just returns a list made with its parameters, in the reparametrized form. `fd_pack_dist` takes a repametrized form and returns a natural form in the format of `fd_gamma_dist`.

`fd_pdist` evaluates the cumulative distribution function of the gamma distribution `g` (natural form) at times `t`. `fd_ddist` evaluates the density function $g(t)$ when `dt == NULL`. When `dt` is a numeric vector, it evaluates the small increments $g(t)\ dt$ (so `dt` must have the size of `t`).

Value

See *details*.

(Shifted) gamma distributions

Times to division/death can be parametrized by any continuous distribution. However, as observed by Hawkins et al. (2007) and Miao et al. (2012), a gamma distribution is often flexible enough. A gamma distribution, contrary to a normal distribution, has a nonnegative support and is skewed, two properties that are expected of distributions of times to division/death (Hyrien et al. (2008)). To ascertain whether finer parametrization can lead to better results, we allow the user to set independently not only the mean and variance of the distribution, but also its skewness (third moment) through a distribution *shift*.

More specifically, is possible to extend the gamma distribution by shifting it by a location parameter $\lambda \geq 0$ (type III distribution in the Pearson Distribution System): $\Gamma_{shifted}(t; \delta, \bar{\tau}, \lambda) = \Gamma(t-\lambda; \delta, \bar{\tau})$ (with the convention that $\Gamma(t) = 0$ for $t < 0$). (The gamma distribution is scale-invariant, so it would not make sense to add a fourth, scale parameter as is usually done to get location-scale families.) To get relevant bounds and help convergence, we can reparametrize the distribution as follows. Noting $\gamma > 0$ the skewness, $\Delta > 0$ the coefficient of variation and $\mu > 0$ the mean, we have

$$\alpha = \frac{4}{\gamma^2} \qquad \beta = \frac{2}{\mu\gamma\Delta} \qquad \lambda = \mu(1 - 2\Delta/\gamma)$$

We clearly see with this last equality that we must have $\Delta/\gamma \leq 1/2$: instead of parametrizing with γ, we parametrize with the ratio $0 < u = \Delta/\gamma \leq 1/2$ of the coefficient of variation over the skewness.

Special values

All the proliferation models in this package allow the special, "degenerate" value `delta = 0`. In this case, the distribution of times to division/death is a Dirac function with the mean still given by `mm` and `ss` is irrelevant. Moreover, because the distribution does not have a density anymore, `fd_ddist` must be called with `dt` non NULL (it gives a numeric vector filled with `0` except for one element). Also, `fd_pdist` reduces to a step function.

References

Hawkins ED, Turner ML, Dowling MR, van Gend C, Hodgkin PD (2007). A model of immune regulation as a consequence of randomized lymphocyte division and death times. *Proc Natl Acad Sci USA* **104** (12): 5032-5037.

Hyrien O, Zand MS (2008). A Mixture Model with Dependent Observations for the Analysis of CSFE-Labeling Experiments. *Journal of the American Statistical Association* **103** (481): 222-239.

Miao H, Jin X, Perelson AS, Wu H (2012). Evaluation of multitype mathematical models for CFSE-labeling experiment data. *Bull Math Biol* **74** (2): 300-326.

Examples

```
mdl <- fd_model()
dst <- fd_pack_dist(relist(start(mdl), mdl)$pro$One$f)
curve(fd_pdist(x, dst), from=0, to=20)
curve(fd_ddist(x, dst), from=0, to=20, add=TRUE, col="red")
```

`model-selection` *Model selection based on an* `fd_comb` *object.*

Description

Wrappers around the functions of package **AICcmodavg** and extensions inspired by it specifically tailored to nonlinear optimization.

Usage

```
fd_aictab(second.ord = FALSE, data, ..., .list = NULL)

fd_modavg(tab, vcov. = vcov)

## S3 method for class 'fd_modavg'
vcov(object, ...)

## S3 method for class 'fd_modavg'
weights(object, ...)

best(tab, subset = TRUE, pos = 1, verbose = TRUE)
```

Arguments

second.ord	Whether to use AIC (FALSE) or AICc (TRUE). We discourage the use of AICc as this has been insufficiently dealt with within the statistical literature on generalized estimating equations. If TRUE, the number of observations used for the small-sample adjustment is the number of clusters. This clearly makes the adjustment bigger than it should be.
data	The fd_data dataset on which optimization was performed.
...	(Potentially named) fd_nls/link[stats]{nls} objects.
.list	Alternatively or additionally, a (potentially named) list of fd_nls/link[stats]{nls} objects can be provided.
tab	An fd_aictab object.
vcov.	Variance-covariance function to use on the free parameters. For example, the **sandwich** package offers an alternative to vcov and could be used for a Huber-White estimation. Alternatively, variance-covariance matrix.
object	An fd_modavg object.
subset	Used to subset tab the same way the subset argument of subset works.
pos	If 1, returns the best model. If 2, returns the second-best model, etc.
verbose	whether the name of the chosen model should be explicitly printed.

Details

fd_aictab returns a data.frame of various summary statistics concerning the models, ranked by increasing AIC or AICc. The output builds on the output of aictab, with two additional columns: Effect and Q (see the section on "effect size" below for interpretation).

The fd_modavg function works similarly to modavg function. It is based on a conceptualization of AIC values as weights (Burnham and Anderson 2004). The weighting is done both on the estimate and on the variance-covariance matrix. coef, vcov and weight can be used to extract, respectively, the estimated model-averaged coefficients, the variance-covariance matrix and the weights used for the averaging. Using QIC for correcting for variance structure misspecification (Pan 2001) would have been better, but implementation in R is sketchy and this is a fundamental matter beyond the scope of this work.

best returns the best (or second-best, ...) model of an fd_aictab object.

Value

fd_aictab returns a data.frame (see *details*), fd_modavg an fd_modavg object, best a "fit" object (e.g., an fd_nls object).

Effect size

The additional columns Effect and Q are to be interpreted as follows.

We think of a parameter as practically identifiable when its 95% confidence interval does not include its lower or upper bounds. This translates into a normalized effect size of roughly 2 (p-value < 0.05, *). Effect sizes greater than 2.6 ($p < 0.01$, **) or even 3.3 ($p < 0.001$, ***) are even more desirable. We think of a model as practically identifiable when the effect size of each of its parameters that have not reached the lower bounds is at least 2, and in general we quantify the practical identifiability of a model with the minimum of the effect sizes (we simply call it the Effect of the model). A qualitative appraisal of this effect is given by Q. Its value is either "" (Effect < 2), "*", "**" or "***" depending on the minimum effect size.

Required packages

Please install **AICcmodavg** for `fd_aictab`.

References

Burnham KP, Anderson DR (2004). Multimodel inference: understanding AIC and BIC in model selection. *Sociological Methods Research* **33**: 261-304.

Pan W (2001). Akaike's Information Criterion in Generalized Estimating Equations. *Biometrics* **57** (1): 120-125.

Examples

```
control <- list(maxit=1)

## Not run:
control <- NULL

## End(Not run)

data(FdSTyphimuriumWTC57)
dat <- cutoff(FdSTyphimuriumWTC57)
source(system.file("contrib", "nlsSA.R", package="fluodilution"))
CC <<- FdCommonConstraints
comb <- fd_comb(catcstr(attr(dat, "bestcstr"),
                        list(CC$`#delta_1100`, CC$`#delta_1111`)),
                data=dat,
                nlsfun="nlsSA",
                control=control)

tab <- fd_aictab(data = dat, .list = comb)
avg <- fd_modavg(tab)
invisible(vcov(avg))
weights(avg)
best(tab)
```

`proliferation` *Implementation of the Cyton and branching models.*

Description

With this package, a large variety of proliferation models can be implemented. They all rest on two broad sets of assumptions. On one hand, a generalized Cyton model and its nested, simpler models, consider that division and death occur concurrently, in competition (Hawkins et al. 2007). On the other hand, a branching process (Hyrien et al. 2010) considers that birth and death have both a given probability of occurring in a cell (probability of transition) and a given probability of occurring after a certain time (time to transition).

Usage

```
fd_proliferation_branching(...)

fd_proliferation_cyton(...)
```

Arguments

`...`

`categories` Name of categories for the proliferation model (branching processes only).

`mgen` Maximum number of generations to model.

`length.out` Number of time points at which division/death rates are evaluated. The more the better, but also the slower (Cyton models only).

`log10_scale` Whether to reparametrize, when fitting, proportions (p, p0, res/res0, mrates/mrates0) using a `log10` scale in order to potentially speed up convergence (branching processes only).

`initial` A matrix giving the initial relative number of seeds in each category (rows) and for each number of generations (columns). Categories are given in the order of `levels(data$Category)`, with `data` an `fd_data` object and generations from 0 to `mgen` (included). If a different `mgen` is chosen for `fd_formula`, the matrix is automatically resized and padded by zeroes if necessary.

Value

An `fd_proliferation` object.

Parametrization

A proliferation model is parametrized as a named list with as many elements as there are compartments/categories. The order in which those elements are arranged has its importance: because no feedback is allowed, migration can only happen from a compartment with a lesser index to a compartment with a greater index. Moreover, although with Cyton models migration is not implemented, for symmetry the parameters are bundled in a list with one element named `"One"`.

The shared parameters within those elements, for both models, are as follows (be careful, they however DO NOT share the exact meaning, see Chauvin et al. (2016)):

`f0$mm, f0$delta, f0$ss` Parametrization of the times to first division. See `fd_gamma_dist` for details.

`f$mm, f$delta, f$ss` Parametrization of the times to subsequent divisions.

`g0$mm, g0$delta, g0$ss` Parametrization of the times to death before the first division.

`g$mm, g$delta, g$ss` Parametrization of the times to death after the first division.

`res0, res0` Proportion of cells that never divide nor die (`res0`) and that do not divide further nor die after the first generation (`res`).

Additionally,

- In the natural parametrization (see `fd_model`), `fd_proliferation_branching` accepts `p0` and `p`, the proportion of cells that undergo first division and undergo division in any number of generations greater than 0, and `mrates`, which is a named list of migration "rates" between 0 and 1L for instance, `Two$mrates$One` gives the migration "rate" from compartment "One" to "Two".
- For branching processes only, additional parameters `h$mm, h$delta, h$ss` can specify a decay distribution, giving times to disintegration for dying/dead cells, and `"hist_lost"`, `"props_lost"` and `"Ns_lost"` types (see `fd_data`) take this into account. The default constraining does not specify `h` and as a consequence no decay is used. This decaying is as of now not available for Cyton models. Notice moreover that, as of now, decaying is not implemented properly for multicompartmental models as decay would also impact emigrating cells.

Reparametrization

For `fd_proliferation_branching`, if `log10_trans` is set to `TRUE`, proportions are reparametrized (see `fd_model`) by their logarithm in base 10. Moreover, `p0` and `p` are the proportions of cells that do not die at the current generation, either because they divide or are nongrowers, and `res0` and `res` are given relative to `p0` and `p`: therefore, for branching processes the proportion of nongrowers, among the total number of cells in a given number of generations, is `p0 * res0` or `p * res`.

For `fd_proliferation_cyton`, no reparametrization is undertaken.

Special cases

Notice that proliferation models allow the `delta` to be 0, in which case the respective distributions reduce to Dirac distributions, and the `g$mm` and `g0$mm` to be infinite (`Inf`), in which case no death occurs (see `gamma-distributions`).

Examples

```
fd_model(proliferation="cyton")
fd_model(proliferation=fd_proliferation_branching(c("One", "Two"), mgen=10))
fd_proliferation_branching(c("One", "Two"), mgen = 10L,
                           initial = matrix(c(1, 0.5, 0, 0), nrow=2))
```

---

random-parameters *Generate random parameters.*

Description

Generate random parameters from an `fd_model` with either a uniform draw (`fd_draw_unif` or the shortcut `fd_unif`) or distributed normally (`fd_norm`).

Usage

```
fd_draw_unif(model, n = 1, coverage = 0.8)

fd_unif(fmm = "gaussian", proliferation = "branching", constraints = NULL,
  n = 1, coverage = 0.8)

fd_norm(n, mean, cv = 0.1, model = NULL)
```

Arguments

`model`	An `fd_model` object. In the case of `fd_norm`, mandatory only if `mean` was not generated by `fd_unif`/`fd_draw_unif`.
`n`	Number of sets of parameters to draw.
`coverage`	A number between `0` and `1` indicating how much of the lower bounds-upper bounds interval should be covered by the random draw: for example, a coverage of `0.8` means that the interval will be shrank symmetrically to 80% of its size. This avoids drawing too extreme values.
`fmm`	A Finite Mixture Model (FMM), see `finite-mixture`.
`proliferation`	A proliferation model, see `proliferation`.

constraints	A cstr object, as produced by cstrlist or an expression if only the constraint component is set (see constraints).
mean	A named numeric vector of coefficients or a matrix with parameter names in columns as would be returned by coef applied to an nlme, nlsList, nls or fd_nls object or the result of fd_unif. If it is a matrix, the columns are simply averaged.
cv	Coefficient of variation (equal across all coefficients) of the normal distribution from which to draw.

Value

A named numeric vector of free parameters (n=1) or a matrix (n>1), augmented by information about the model that can be queried with model. Therefore, it is possible to use relisted_coef on the return value.

Required packages

Please install **truncnorm** for fd_norm

See Also

Other simulation-related entries: dataset-simulation, fd_predict

Examples

```
(ans <- fd_unif("gaussian", "branching"))
fd_draw_unif(model(ans))
fd_norm(3, ans)
```

---

stat_unitarea *Display a histogram with unit area in* **ggplot2**.

Description

This function requires the **numDeriv** package (this package is not automatically installed when the fluodilution package is installed).

Usage

```
stat_unitarea(mapping = NULL, data = NULL, geom = "line",
  position = "identity", ..., na.rm = FALSE, show.legend = NA,
  inherit.aes = TRUE)

transform_hist(y, a, b, trans)
```

Arguments

mapping	Set of aesthetic mappings created by aes or aes_. If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot. A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame., and will be used as the layer data.
geom	Use to override the default connection between geom_density and stat_density.
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
...	other arguments passed on to layer. These are often aesthetics, used to set an aesthetic to a fixed value, like color = "red" or size = 3. They may also be parameters to the paired geom/stat.
na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders.
y	A numeric vector of histogram proportions to be transformed conserving unit area.
a	The left breakpoints of the cells of the histogram.
b	The right breakpoints.
trans	Either a character string or a transformation object directly, see package **scales** for possible transformers.

Details

In **ggplot2**, densities displayed with stat_density automatically have unit area when the x axis is transformed. If one wants to use geom_line to display histograms from an fd_data object this transformation is not made. For instance, as an fd_data object stores its breakpoints in linear units of fluorescence, when one wants to apply a log-transformation on the x axis then the area under the curve does not sum to one anymore. stat_unitarea takes those transformations automatically into account. For the arguments, see stat_density. Additionally, transform_hist can be used to apply a unit area transformation and return a numeric vector of transformed proportions.

Value

An object that can be chained to a ggplot call.

Aesthetics

stat_unitarea understands the following required aesthetics: x and yu (the y component of the histograms for which we want to ensure a unit area).

Computed variables

`unitarea` transformation of yu to ensure unit area (the default y).

`scaled` transformation of yu to ensure unit area, scaled to a maximum of 1.

Examples

```
data(FdClearPeaks)
ggplot(subset(FdClearPeaks, Type == "hists"),
       aes(x = (a + b) / 2.0, yu = y))+
  stat_unitarea()+
  facet_wrap(~ Timepoint, scales="free")+
  labs(y = "Unit area")+
  scale_x_log10()

ggplot(subset(FdClearPeaks, Type == "hists"),
       aes(x = (a + b) / 2.0, yu = y, y=..scaled..))+
  stat_unitarea()+
  labs(y = "Unit area, scaled to mode")+
  facet_wrap(~ Timepoint, scales="free")+
  scale_x_log10()

ggplot(transform(subset(FdClearPeaks, Type == "hists"),
                 y = transform_hist(y, a, b, "log10")),
       aes(x = (a + b) / 2.0, y = y))+
  geom_line()+
  labs(y = "Unit area")+
  facet_wrap(~ Timepoint, scales="free")+
  scale_x_log10()
```

Index