# Package ‘Infusion’

February 20, 2021

**Type** Package

**Title** Inference Using Simulation

**Description** Implements functions for simulation-based inference. In particular, implements functions to perform likelihood inference from data summaries whose distributions are simulated (Rousset et al. 2017 <doi:10.1111/1755-0998.12627>).

**Encoding** UTF-8

**Version** 1.5.1

**Date** 2021-02-20

**Imports** spaMM (>= 3.6.0), proxy, blackbox (>= 1.0.14), mvtnorm, methods, numDeriv, viridis, pbapply, foreach

**Suggests** testthat, ranger, Rmixmod, crayon

**Depends** R (>= 3.3.0)

**Maintainer** François Rousset <francois.rousset@umontpellier.fr>

**License** CeCILL-2

**ByteCompile** true

**URL** https://www.R-project.org,
https://kimura.univ-montp2.fr/~rousset/Infusion.htm

**NeedsCompilation** no

**Author** François Rousset [aut, cre, cph] (<https://orcid.org/0000-0003-4670-0371>)

**Repository** CRAN

**Date/Publication** 2021-02-20 15:20:02 UTC

## R topics documented:

add_simulation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
confint.SLik . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
densv . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
dMixmod . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
example_raw . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

example_reftable . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
extractors . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
get_from . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
goftest . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
handling_NAs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
infer_logLs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
infer_SLik_joint . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
infer_surface . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
Infusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
init_grid . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
MSL . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
multi_binning . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
options . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
plot.SLik . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
plot1Dprof . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26
profile.SLik . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
project.character . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
refine . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
rparam . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35

**Index** **37**

---

`add_simulation` *Create or augment a list of simulated distributions of summary statistics*

---

## Description

`add_simulation` creates or augments a list of simulated distributions of summary statistics, and formats the simulation results appropriately for further use. The user does not have to think about this return format. Instead, s-he only has to think about the very simple return format of the function given as its `Simulate` argument. Alternatively, if the simulation function cannot be called directly by the R code, simulated distributions can be added easily using the `newsimuls` argument, again using a simple format (see `onedistrib` in the Examples).

`add_reftable` is a wrapper for `add_simulation`, enforcing `nRealizations=1`: see example `example_reftable`.

These functions can run simulations in a parallel environment. Special care is then needed to ensure that all required packages are loaded in the called processes, and required all variables and function are passed therein: check the `packages` and `env` arguments.

## Usage

```
add_simulation(simulations=NULL, Simulate, par.grid=NULL,
               nRealizations = NULL, newsimuls = NULL,
            verbose = interactive(), nb_cores = NULL, packages = NULL, env = NULL,
               control.Simulate=NULL, cluster_args=list(), ...)
add_reftable(...)
```

## Arguments

| | |
|---|---|
| `simulations` | A list of simulations |
| `nRealizations` | The number of simulated samples of summary statistics, for each empirical distribution (each row of `par.grid`). If the argument is NULL, the value is obtained by `Infusion.getOption`. If the argument is not NULL, `Infusion.options(nRealizations)` is set, but restored, on exit from `add_simulation`, to its initial value. |
| `Simulate` | An *R* function, or the name (as a character string) of an *R* function used to generate empirical distributions of summary statistics. When an external simulation program is called, `Simulate` must therefore be an R function wrapping the call to the external program. The function must have a single vector as argument, matching each row of `par.grid`. It must return a vector of summary statistics with named vector members; **or** a single matrix of `nRealizations` simulations, in which case its rows and row names, must represent the summary statistics, it should have `nRealizations` columns, and `nRealizations` should be named integer of the form "`c(as_one=.)`" (see Examples). |
| `par.grid` | A data frame of which each line matches the single vector argument of `Simulate`. |
| `newsimuls` | If the function used to generate empirical distributions cannot be called by R, then `newsimuls` can be used to provide these distributions. See Details for the structure of this argument. |
| `nb_cores` | Number of cores for parallel simulation; `NULL` or integer value, actin as a shortcut for `cluster_args$spec`. If `nb_cores` is unnamed or has name `"replic"` and if the simulation function does not return a single table for all replicates (thus, if `nRealizations` is **not** a named integer of the form "`c(as_one=.)`", parallelisation is over the different samples for each parameter value. For any other explicit name (e.g., `nb_cores=c(foo=7)`), or if `nRealizations` is a named integer of the form "`c(as_one=.)`", parallelisation is over the parameter values (the rows of `par.grid`). In all cases, the progress bar is over parameter values. See Details in `Infusion.options` for the subtle way these different cases are distinguished in the progress bar. |
| `cluster_args` | A list of arguments, passed to `makeCluster`. May contain a non-null `spec` element, in which case the distinct `nb_cores` argument is ignored. |
| `verbose` | Whether to print some information or not. |
| `...` | For `add_reftable`: arguments passed to `add_simulation`. Any of the `add_simulation` arguments is valid, except `nRealizations`. For `add_simulation`: additional arguments passed to `Simulate`, beyond the parameter vector; see `nsim` argument of `myrnorm_tab()` in the Examples. These arguments should be constant through all the simulation workflow. |
| `control.Simulate` | A list, used as an exclusive alternative to "`...`" to pass additional arguments to `Simulate`, beyond the parameter vector. The list must contain the same elements as would go in the "`...`". |
| `packages` | For parallel evaluation: Names of additional libraries to be loaded on the cores, necessary for `Simulate` evaluation. |
| `env` | For parallel evaluation: an environment containing additional objects to be exported on the cores, necessary for `Simulate` evaluation. |

## Details

The `newsimuls` argument should have the same structure as the return value of `add_simulation` itself, except that `newsimuls` may include only a subset of the attributes returned by `add_simulation`. **In the reference-table case**, it is thus a data frame; its required attributes are `LOWER` and `UPPER` which are named vectors giving bounds for the parameters which are variable in the whole analysis (note that the names identify these parameters in the case this information is not available otherwise from the arguments). The values in these vectors may be incorrect in the sense of failing to bound the parameters in the `newsimuls`, as the actual bounds are then corrected using parameter values in `newsimuls` and attributes from `simulations`. **Otherwise**, `newsimuls` should be list of matrices, each with a `par` attribute (see Examples). Rows of each matrix stand for simulation replicates and columns stand for the different summary statistics.

## Value

If only one realization is computed for each (vector-valued) parameter, a data.frame (with additional attributes) is returned. Otherwise, the return value is an objet of class `EDFlist`, which is a list-with-attributes of matrices-with-attribute. Each matrix contains a simulated distribution of summary statistics for given parameters, and the `"par"` attribute is a 1-row data.frame of parameters. If `Simulate` is used, this must give all the parameters to be estimated; otherwise it must at least include all variable parameters in this **or later** simulations to be appended to the simulation list.

The value has the following attributes: `LOWER` and `UPPER` which are each a vector of per-parameter minima and maxima deduced from any `newsimuls` argument, and optionally any of the arguments `Simulate,control.Simulate,packages,env,par.grid` and `simulations` (all corresponding to input arguments when provided, except that the actual `Simulate` function is returned even if it was input as a name).

## Examples

```
# example of building a list of simulations from scratch:
myrnorm <- function(mu,s2,sample.size) {
  s <- rnorm(n=sample.size,mean=mu,sd=sqrt(s2))
  return(c(mean=mean(s),var=var(s)))
}
set.seed(123)
onedistrib <- t(replicate(100,myrnorm(1,1,10))) # toy example of simulated distribution
attr(onedistrib,"par") <- c(mu=1,sigma=1,sample.size=10) ## important!
simuls <- add_simulation(NULL, Simulate="myrnorm", nRealizations=500,
                         newsimuls=list("example"=onedistrib))

# standard use: smulation over a grid of parameter values
parsp <- init_grid(lower=c(mu=2.8,s2=0.2,sample.size=40),
                   upper=c(mu=5.2,s2=3,sample.size=40))
simuls <- add_simulation(NULL, Simulate="myrnorm", nRealizations=500,
                         par.grid = parsp[1:7,])

## Not run:  # example continued: parallel versions of the same
# Slow computations, notably because cluster setup is slow.

#    ... parallel over replicates, serial over par.grid rows
simuls <- add_simulation(NULL, Simulate="myrnorm", nRealizations=500,
                         par.grid = parsp[1:7,], nb_cores=7)
#    ... parallel over 'par.grid' rows
simuls <- add_simulation(NULL, Simulate="myrnorm", nRealizations=500,
                         par.grid = parsp[1:7,], nb_cores=c(foo=7))

## End(Not run)

####### Example where a single 'Simulate' returns all replicates:

myrnorm_tab <- function(mu,s2,sample.size, nsim) {
  ## By default, Infusion.getOption('nRealizations') would fail on nodes!
  replicate(nsim,
            myrnorm(mu=mu,s2=s2,sample.size=sample.size))
}

parsp <- init_grid(lower=c(mu=2.8,s2=0.2,sample.size=40),
                   upper=c(mu=5.2,s2=3,sample.size=40))

# 'as_one' syntax for 'Simulate' function returning a simulation table:
simuls <- add_simulation(NULL, Simulate="myrnorm_tab",
              nRealizations=c(as_one=500),
              nsim=500, # myrnorm_tab() argument, part of the 'dots'
              par.grid=parsp)

## Not run:  # example continued: parallel versions of the same
# Slow cluster setup again
simuls <- add_simulation(NULL,Simulate="myrnorm_tab",par.grid=parsp,
              nb_cores=7L,
              nRealizations=c(as_one=500),
              nsim=500, # myrnorm_tab() argument again
              # need to export other variables used by *myrnorm_tab* to the nodes:
              env=list2env(list(myrnorm=myrnorm)))

## End(Not run)

## see main documentation page for the package for other typical usage
```

## confint.SLik    *Compute confidence intervals by (profile) summary likelihood*

### Description

This takes an SLik object (as produced by MSL) and deduces confidence bounds for each parameter, using a (profile, if relevant) likelihood ratio method.

### Usage

```
## S3 method for class 'SLik'
confint(object, parm,
                         level=0.95, verbose=interactive(),
                    fixed=NULL,which=c(TRUE,TRUE),...)
```

## Arguments

| | |
|---|---|
| `object` | an `SLik` or `SLikp` object |
| `parm` | The parameter which confidence bounds are to be computed |
| `level` | The desired coverage of the interval |
| `verbose` | Whether to print some information or not |
| `fixed` | When this is `NULL` the computed interval is a profile confidence interval over all parameters excluding `parm`. `fixed` allows one to set fixed values to some of these parameters. |
| `which` | A pair of booleans, controlling whether to compute respectively the lower and the upper CI bounds. |
| `...` | further arguments passed to or from other methods (currently not used). |

## Value

A list with sublists for each parameter, each sublist containing of three vectors: the bounds of the one-dimensional confidence interval; the "full" (only parameters variable in the `SLik` object are considered) parameter point for the lower bound, and the full parameter point for the upper bound

## Examples

```
## see main documentation page for the package
```

---

`densv` *Saved computations of inferred log-likelihoods*

---

## Description

These are saved results from toy examples used in other documentation page for the package. It gives estimates by simulation of log-likelihoods of the `(mu,s2)` parameters of a Gaussian distribution for a given sample of size 20 with mean 4.1416238 and (bias-corrected) variance 0.9460778. `densv` is based on the sample mean and sample variance as summary statistics, and `densb` on more contrived summary statistics.

## Usage

```
data("densv")
data("densb")
```

## Format

Data frames (with additional attributes) with observations on the following 5 variables.

`mu` a numeric vector; mean parameter of simulated Gaussian samples

`s2` a numeric vector; variance parameter of simulated Gaussian samples

`sample.size` a numeric vector; size of simulated Gaussian samples

`logL` a numeric vector; log probability density of a given statistic vector inferred from simulated values for the given parameters

`isValid` a boolean vector. See `infer_logLs` for its meaning.

Both data frames are return objects of a call to `infer_logLs`, and as such they includes attributes providing information about the parameter names and statistics names (not detailed here).

## See Also

See step (3) of the workflow in the Example on the main `Infusion` documentation page, showing how `densv` was produced, and the Example in `project` showing how `densb` was produced.

---

`dMixmod` *Internal S4 classes.*

---

## Description

The objects or methods referenced here are not to be called by the user, or are waiting for documentation to be written.

`dMixmod` is an S4 class describing distributions that involve discrete probability masses for some variables and gaussian mixtures for other variables conditional on such discrete events. In terms of the represented probability models, and of its slots, is effectively extends the `MixmodResults` class from the `Rmixmod` package. But it does not formally extends this class in termes of OOP programming. It should not be considered as part of the programming interface, and may be subject to backward-incompatible modifications without notice.

## Usage

```
# dMixmod: Don't try to use it!
```

## Value

A `dMixmod` object has the same slots as a `MixmodResults` object, plus additional ones: `@freq` is the frequency of the conditioning event for the gaussian mixture model. In the `Infusion` code, this event is defined jointly by the "observed" summary statistics and the reference simulation table: a probability mass for specific values **v** is identified from the simulated distribution of summary statistics in the reference table, and `freq` is an estimate of the probability mass if the summary statistics match **v**, or the converse probability if they do not match.

## Note

Use `str(attributes(.))` to see the slots of a `dMixmod` object if `str(.)` does not work.

`example_raw` *Workflow for originally described method*

## Description

Example of the workflow with `add_simulation`), implementing the method described in the original publication (Rousset et al. 2017 <doi:10.1111/1755-0998.12627>).

## Examples

```
## The following example illustrates the workflow.
## However, most steps run longer than accepted by the CRAN checks,
## So by default they will not run.
##
## (1) The user must provide the function for simulation of summary statistics
myrnorm <- function(mu,s2,sample.size) {
 s <- rnorm(n=sample.size,mean=mu,sd=sqrt(s2))
 return(c(mean=mean(s),var=var(s)))
} # simulate means and variances of normal samples of size 'sample.size'
#
## pseudo-sample:
set.seed(123)
Sobs <- myrnorm(mu=4,s2=1,sample.size=40) ## stands for the actual data to be analyzed
#
## (2) Generate, and simulate distributions for,
##        an irregular grid of parameter values, with some replicates
if (Infusion.getOption("example_maxtime")>40) {
  parsp <- init_grid(lower=c(mu=2.8,s2=0.2,sample.size=40),
                     upper=c(mu=5.2,s2=3,sample.size=40))
  simuls <- add_simulation(NULL,Simulate="myrnorm",par.grid=parsp)

  ## (3) infer logL(pars,stat.obs) for each simulated 'pars'
  # Relatively slow, hence saved as data 'densv'
  densv <- infer_logLs(simuls,stat.obs=Sobs)
} else {
  data(densv)
  .Random.seed <- saved_seed
}
#
## (4) infer a log-likelihood surface and its maximum;
##       plot and extract various information.
if (Infusion.getOption("example_maxtime")>11) {
 slik <- infer_surface(densv)
 slik <- MSL(slik) ## find the maximum of the log-likelihood surface
 plot(slik)
 profile(slik,c(mu=4)) ## profile summary logL for given parameter value
 confint(slik,"mu") ## compute confidence interval for given parameter
 plot1Dprof(slik,pars="s2",gridSteps=40) ## 1D profile
}
#
```

```
## (5) ## refine iteratively
if (Infusion.getOption("example_maxtime")>39) {
 slik <- refine(slik)
}
```

## example_reftable *Workflow for method with reference table*

### Description

Example of workflow with a reference table produced by `add_reftable`, possibly faster in many applications than the originally described method.

### Examples

```
if ((Infusion.getOption("example_maxtime")>88)) {
  myrnorm <- function(mu,s2,sample.size) {
    s <- rnorm(n=sample.size,mean=mu,sd=sqrt(s2))
    return(c(mean=mean(s),var=var(s)))
  } # simulate means and variances of normal samples of size 'sample.size'
  set.seed(123)
  # pseudo-sample with stands for the actual data to be analyzed:
  ssize <- 40
  Sobs <- myrnorm(mu=4,s2=1,sample.size=ssize)
  # Uniform sampling in parameter space:
  npoints <- 600
  parsp <- data.frame(mu=runif(npoints,min=2.8,max=5.2),
                      s2=runif(npoints,min=0.4,max=2.4),sample.size=ssize)
  # Build simulation table:
  simuls <- add_reftable(Simulate="myrnorm",par.grid=parsp)

  ## trivial projections that should produce an y=x regression:
  mufit <- project("mu",stats=c("mean","var"),data=simuls)
  s2fit <- project("s2",stats=c("mean","var"),data=simuls)

  ## additional plots for some projection method
  if (inherits(mufit,"HLfit")) mapMM(mufit,map.asp=1,
        plot.title=title(main="prediction of normal mean",xlab="mean",ylab="var"))
  if (inherits(s2fit,"HLfit")) mapMM(s2fit,map.asp=1,
        plot.title=title(main="prediction of normal var",xlab="mean",ylab="var"))

  ## apply projections on simulated statistics
  corrSobs <- project(Sobs,projectors=list("MEAN"=mufit,"VAR"=s2fit))
  corrSimuls <- project(simuls,projectors=list("MEAN"=mufit,"VAR"=s2fit))


  # Infer surface:
  densv <- infer_SLik_joint(corrSimuls,stat.obs=corrSobs)
  # Usual workflow using inferred surface:
  slik_j <- MSL(densv) ## find the maximum of the log-likelihood surface
  slik_j <- refine(slik_j,maxit=5, update_projectors=TRUE)
  plot(slik_j)
  # etc:
  profile(slik_j,c(mu=4)) ## profile summary logL for given parameter value
  confint(slik_j,"mu") ## compute 1D confidence interval for given parameter
  plot1Dprof(slik_j,pars="s2",gridSteps=40) ## 1D profile
  summary(slik_j) # or print()
  logLik(slik_j)
}
```

## extractors *Summary, print and logLik methods for Infusion results.*

### Description

summary prints information about the fit. print is an alias for summary. logLik extracts the log-likelihood (exact or approximated).

### Usage

```
## S3 method for class 'SLik'
summary(object, ...)
## S3 method for class 'SLik'
print(x, ...)
## S3 method for class 'SLik'
logLik(object, ...)
# and identical usage for 'SLik_j' objects
```

### Arguments

| | |
|---|---|
| object, x | An object of class SLik or SLik_j; |
| ... | further arguments passed to or from other methods (currently without any specific effect). |

### Value

logLik returns the inferred likelihood maximum, with attribute RMSE giving its root means square error of estimation. summary and summary return the object invisibly. They print details of the fits in a convenient form.

### Note

See workflow example in example_reftable.

### See Also

See get_from for a more general interface for extracting elements from Infusion results.

## Examples

```
# See Note
```

---

`get_from` *Backward-compatible extractor from summary-likelihood objects*

---

## Description

A generic function, whose default method works for list, and with specific methods for objects inheriting from classes `SLik_j` and `SLik`.

## Usage

```
get_from(object, which, ...)

## S3 methods with additional argument(s)
## S3 method for class 'SLik'
get_from(object, which, raw, ...)
## S3 method for class 'SLik_j'
get_from(object, which, raw, ...)
```

## Arguments

| | |
|---|---|
| `object` | Any object with a list structure. |
| `which` | Character: names of element to be extracted. |
| `raw` | Boolean: if TRUE, `object[[which]]` is returned, ignoring any more specific processing. |
| `...` | further arguments passed to or from other methods (currently not used). |

## Value

Will depend on `which`, but aims to retain a convenient format backward compatible with version 1.4.0.

## See Also

`logLik`.

## Examples

```
  # Observed summary statistics
  #  (projected, with raw ones as attribute, if relevant)
  # get_from(slik, "obs")
  #
  # On any summary-likelihood object 'slik':
  # get_from(slik, which="par_RMSEs") # matrix
  # despite <object>$par_RMSEs being an environment if
  #  'slik' was created by version > 1.4.0, as then shown by
  # get_from(slik, which="par_RMSEs", raw=TRUE)
```

## goftest *Assessing goodness of fit of inference using simulation*

### Description

A goodness-of-fit test is performed in the case projected statistics have been used for inference. Otherwise some plots of limited interest are produced.

### Usage

```
goftest(object, nsim = 99L, method = "", stats=NULL, plot. = TRUE, nb_cores = NULL,
        Simulate = attr(object$logLs, "Simulate"),
        packages = attr(object$logLs, "packages"),
        env = attr(object$logLs, "env"), verbose = interactive())
```

### Arguments

| | |
|---|---|
| `object` | an `SLik` or `SLikp` object. |
| `nsim` | Number of draws of summary statistics. |
| `method` | For development purposes, not documented. |
| `stats` | Character vector, or NULL: the set of summary statistics to be used to construct the test. If NULL, the statistics used accross all projections are used. |
| `plot.` | Control diagnostic plots. `plot.` can be of logical, character or numeric type. If `plot.` is `FALSE`, no plot is produced. If `plot.` is `TRUE` (the default), a data frame of up to 8 goodness-of-fit statistics (the statistics denoted *u* in Details) is plotted. If more than eight raw summary statistics (denoted *s* in Details) were used, then only the first eight *u* are retained (the order or *u* being deduced from the order of use of the *s* in the different projections). If `plot.` is a **numeric vector**, then *u*`[plot.]` are retained (possibly more than 8 statistics, as in the next case). If `plot.` is a **character vector**, then it is used to match the names of the *u* statistics (not of *s*) to be retained in the plot; the names of *u* are built from names of *s* by wrapping the latter within `"Res(".")"` (see axes labels of default plots for examples of valid names). |
| `nb_cores, Simulate, packages, env, verbose` | See same-named `add_simulation` arguments. |

### Details

The test is somewhat heuristic but appears to give reasonable results (the Example shows how this can be verified). It assumes that all summary statistics are reduced to projections predicting all model parameters. It is then conceived as if any projection *p* predicting a parameter were a sufficient statistic for this parameter, given the information contained in the summary statistics **s** (this is certainly the ideal objective of machine-learning regression methods). Then a statistic *u* independent (under the fitted model) from all projections should be a suitable statistic for testing goodness of fit: if the model is correctly specified, the quantile of observed *u*, in the distribution of *u* under the fitted model, should be uniformly distributed over repeated sampling under the data-generating process. The procedure constructs statistics uncorrelated to all **p** (over repeated sampling under the fitted model) and proceeds as if they were independent from *p* (rather than simply uncorrelated). Statistics *u* uncorrelated to *p* are obtained as the residuals of the regression of each summary statistic to all projections, where the regression input is a simulation table of `nsim` replicates of **s** under the fitted model, and of their projections **p** (using the "projectors" constructed from the full reference table). The latter regression involves one more, small-`nsim`, approximation (as it is the sample correlation that is zeroed) but using the residuals is crucially better than using the original summary statistics (as some ABC software may do). An additional feature of the procedure is to construct a single test statistic *t* from joint residuals **u**, by estimating their joint distribution (using Gaussian mixture modelling) and letting *t* be the density of **u** in this distribution.

## Value

A list with currently a single element

| | |
|---|---|
| `pval` | The p-value of the test (NULL if the test is not feasible). |

## Examples

```
## Not run:
## Long example, despite minimal settings!
## Showing uniform distribution under correctly-specified model

# Normal(mu,sd) model, with inefficient summary statistics:
blurred <- function(mu,s2,sample.size) {
  s <- rnorm(n=sample.size,mean=mu,sd=sqrt(s2))
  s <- exp(s/4)
  return(c(mean=mean(s),var=var(s)))
}

# Construct reference table and projections once for all replicates:
set.seed(123)
parsp_j <- data.frame(mu=runif(5000L,min=2.8,max=5.2),
                      s2=runif(5000L,min=0.4,max=2.4),sample.size=40)
dsimuls <- add_reftable(,Simulate="blurred",par.grid=parsp_j,verbose=FALSE)
mufit <- project("mu",stats=c("mean","var"),data=dsimuls,verbose=FALSE,knotnbr=5000L)
s2fit <- project("s2",stats=c("mean","var"),data=dsimuls,verbose=FALSE,knotnbr=5000L)
dprojectors <- list(MEAN=mufit,VAR=s2fit)
dprojSimuls <- project(dsimuls,projectors=dprojectors,verbose=FALSE)

# Analysis of replicate draws from data-generating process
gof_draws <- replicate(50, {
  cat(".")
  dSobs <- blurred(mu=4,s2=1,sample.size=40) ## stands for the actual data to be analyzed
  ## ----dcflow-----------------------------------------------------------
  dprojSobs <- project(dSobs,projectors=dprojectors)
  dslik <- infer_SLik_joint(dprojSimuls,stat.obs=dprojSobs,verbose=FALSE)
  dslik <- MSL(dslik, verbose=FALSE, eval_RMSEs=FALSE)
  #dslik <- refine(dslik,maxit=2L,verbose=FALSE, update_projectors=TRUE)
  gof <- goftest(dslik,nsim = 99L,nb_cores = 1L, method="", plot.=FALSE,verbose=FALSE)
  cat(unlist(gof))
  gof
})
plot(ecdf(unlist(gof_draws)))

## End(Not run)
```

`handling_NAs` *Discrete probability masses and NA/NaN/Inf in distributions of summary statistics.*

## Description

This explains the use of the `boundaries` attribute of observed statistics withto handle (1) values of the summary statistics that can occur with some probability mass; (2) special values (NA/NaN/Inf) in distributions of summary statistics. This further explains why `Infusion` handles special values by removing affected distributions unless the `boundaries` attribute is used.

## Details

Special values may be encountered in an analysis. For example, trying to estimate a regression coefficient when the predictor variable is constant may return a NaN. Since functions such as `refine` automatically add simulated distributions, this problem must be automatically handled by the user's simulation function or by the package functions, rather than by user's tinkering with the Infusion procedures.

The user must consider what s-he would do if actual data also included NA/NaN/Inf values. If (1) such data would not be used in the statistical analysis, then the simulation procedure must reflect that, otherwise the analysis will be biased. Alternatively (2) if one considers that special values are informative about parameters (in the above example of a regression coefficient, if a constant predictor variable says something about the parameters), then NA/NaN/Inf must be replaced by a numerical value which is flagged to be distinctly handled.

Thus, in case (1) it may be necessary to simulate alternative data until no NaN's are obtained and the target size of the simulated distribution is reached. One solution is for the user to write a simulation function that calls itself recursively until a valid summary statistic is produced. Care is then needed to avoid infinite recursion (which might well indicate unlikely parameter values).

In case (2), it is necessary to assign some (fixed) dummy numerical value to the summary statistics, and to flag this value using the `boundaries` attribute of the observed summary statistics. The simulation function should return statistic `foo=-1` (say) instead of `foo=NaN`, and one should then set `attr(<observed>,"boundaries") <-c(foo=-1)`.

Without such active decisions by the user, the inference method has no way to determine whether case (1) or (2) holds, and must thus ignore all empirical distributions including NA/NaN/inf. These empirical distributions are thus ignored by the inference functions.

The boundary attribute is also useful to handle all values of the summary statistics that can occur with some probability mass. For example if the estimate `est_p` of a probability takes values 0 or 1 with positive probability, one should set `attr(<observed>,"boundaries") <-c(p_est=0,p_est=1)`.

## infer_logLs — *Infer log Likelihoods using simulated distributions of summary statistics*

### Description

For each simulated distribution of summary statistics, `infer_logLs` infers a probability density function, and the density of the observed values of the summary statistics is deduced. By default, inference of each density is performed by `infer_logL_by_Rmixmod`, which fits a distribution of summary statistics using procedures from the `Rmixmod` package.

### Usage

```
infer_logLs(object, stat.obs,
            logLname = Infusion.getOption("logLname"),
            verbose = list(most=interactive(),
                           final=FALSE),
            method = Infusion.getOption("mixturing"),
            nb_cores = NULL, packages = NULL, cluster_args,
            ...)
infer_tailp(object, refDensity, stat.obs,
                tailNames=Infusion.getOption("tailNames"),
                verbose=interactive(), method=NULL, cluster_args, ...)
infer_logL_by_GLMM(EDF,stat.obs,logLname,verbose)
infer_logL_by_Rmixmod(EDF,stat.obs,logLname,verbose)
infer_logL_by_mclust(EDF,stat.obs,logLname,verbose)
infer_logL_by_Hlscv.diag(EDF,stat.obs,logLname,verbose)
```

### Arguments

| | |
|---|---|
| `object` | A list of simulated distributions (the return object of `add_simulation`) |
| `EDF` | An empirical distribution, with a required `par` attribute (an element of the `object` list). |
| `stat.obs` | Named numeric vector of observed values of summary statistics. |
| `logLname` | The name to be given to the log Likelihood in the return object, or the root of the latter name in case of conflict with other names in this object. |
| `tailNames` | Names of “positives” and “negatives” in the binomial response for the inference of tail probabilities. |
| `refDensity` | An object representing a reference density (such as an `HLfit` fit object or other objects with a similar `predict` method) which, together with the density inferred from each empirical density, defines a likelihood ratio used to define a rejection region. |
| `verbose` | A list as shown by the default, or simply a vector of booleans, indicating respectively whether to display (1) some information about progress; (2) a final summary of the results after all elements of `simuls` have been processed. If a |

| | |
|---|---|
| | count of 'outlier'(s) is reported, this typically means that `stat.obs` is not within the envelope of a simulated distribution (or whatever other meaning the user attaches to an `FALSE` `isValid` code: see Details) |
| `method` | A function for density estimation. See Description for the default behaviour and Details for the constraints on input and output of the function. |
| `nb_cores` | Number of cores for parallel computation. The default is `spaMM.getOption("nb_cores")`, and 1 if the latter is NULL. `nb_cores=1` which prevents the use of parallelisation procedures. |
| `cluster_args` | A list of arguments, passed to `makeCluster`. May contain a non-null `spec` element, in which case the distinct `nb_cores` argument is ignored. |
| `packages` | For parallel evaluation: Names of additional libraries to be loaded on the cores, necessary for evaluation of a user-defined 'method'. |
| `...` | further arguments passed to or from other methods (currently not used). |

## Details

By default, density estimation is based on `Rmixmod` methods. Other available methods are not routinely used and not all of `Infusion` features may work with them. The function `Rmixmod::mixmodCluster` is called, with arguments `nbCluster=Infusion.getOption("nbCluster")` and `mixmodGaussianModel=Infusion.getOpt` If `Infusion.getOption("nbCluster")` specifies a sequence of values, then several clusterings are computed and AIC is used to select among them.

`infer_logL_by_GLMM`, `infer_logL_by_Rmixmod`, `infer_logL_by_mclust`, and `infer_logL_by_Hlscv.diag` are examples of the method that may be provided for density estimation. Other `methods` may be provided with the same arguments. Their return value must include the element `logL`, an estimate of the log-density of `stat.obs`, and the element `isValid` with values `FALSE`/`TRUE` (or 0/1). The standard format for the return value is `unlist(c(attr(EDF,"par"),logL,isValid=isValid))`.

`isValid` is primarily intended to indicate whether the log likelihood of `stat.obs` inferred by a given density estimation method was suitable input for inference of the likelihood surface. `isValid` has two effects: to distinguish points for which isValid is FALSE in the plot produced by `plot.SLik`; and more critically, to control the sampling of new parameter points within `refine` so that points for which isValid is FALSE are less likely to be sampled.

Invalid values may for example indicate a likelihood estimated as zero (since log(0) is not suitable input), or (for density estimation methods which may infer erroneously large values when extrapolating), whether `stat.obs` is within the convex hull of the EDF. In user-defined `methods`, invalid inferred logL should be replaced by some alternative low estimate, as all methods included in the package do.

The source code of `infer_logL_by_Hlscv.diag` illustrates how to test whether `stat.obs` is within the convex hull of the EDF, using functions `resetCHull` and `isPointInCHull` (exported from the `blackbox` package).

`infer_logL_by_Rmixmod` calls `Rmixmod::mixmodCluster` `infer_logL_by_mclust` calls `mclust::densityMclust`, `infer_logL_by_Hlscv.diag` calls `ks::kde`, and `infer_logL_by_GLMM` fits a binned distribution of summary statistics using a Poisson GLMM with autocorrelated random effects, where the binning is based on a tesselation of a volume containing the whole simulated distribution. Limited experimentations so far suggest that the mixture models methods are fast and appropriate (`Rmixmod`, being a bit faster, is the default method); that the kernel smoothing method is more erratic and moreover requires additional input from the user, hence is not really applicable, for distributions in dimension *d*= 4 or above; and that the GLMM method is a very good density estimator for *d*=2 but will challenge one's patience for *d*=3 and further challenge the computer's memory for *d*=4.

## Value

For `infer_logLs`, a data frame containing parameter values and their log likelihoods, and additional information such as attributes providing information about the parameter names and statistics names (not detailed here). These attributes are essential for further inferences.

See Details for the required value of the `methods` called by `infer_logLs`.

## See Also

See step (3) of the workflow in the Example on the main `Infusion` documentation page.

---

`infer_SLik_joint` *Infer a (summary) likelihood surface from a simulation table*

---

## Description

This infers the likelihood surface from a simulation table where each simulated data set is drawn for a distinct (vector-valued) parameter, as is usual for reference tables in ABC. A parameter density is inferred, as well as a joint density of parameters and summary statistics, and the likelihood surface is inferred from these two densities.

## Usage

```
infer_SLik_joint(data, stat.obs, logLname = Infusion.getOption("logLname"),
                 Simulate = attr(data, "Simulate"),
                 nbCluster= Infusion.getOption("nbCluster"),
                 using = Infusion.getOption("mixturing"),
                 verbose = list(most=interactive(),pedantic=FALSE,final=FALSE),
                 marginalize = TRUE)
```

## Arguments

| | |
|---|---|
| `data` | A data frame, whose each row contains a vector of parameters and one realization of the summary statistics for these parameters. |
| `stat.obs` | Named numeric vector of observed values of summary statistics. |
| `logLname` | The name to be given to the log Likelihood in the return object, or the root of the latter name in case of conflict with other names in this object. |
| `Simulate` | Either NULL or the name of the simulation function if it can be called from the R session. |
| `nbCluster` | `nbCluster` argument of `Rmixmod::mixmodCluster` |
| `using` | Either `"Rmixmod"` or `"mclust"` to select the clustering methods used. |

| | |
|---|---|
| `marginalize` | Boolean; whether to derive the clustering of fitted parameters by marginalization of the joint clustering (default, and introduced in version 1.3.5); or by a distinct call to a clustering function. |
| `verbose` | A list as shown by the default, or simply a vector of booleans, indicating respectively whether to display (1) some information about progress; (2) more information whose importance is not clear to me; (3) a final summary of the results after all elements of `simuls` have been processed. |

## Value

An object of class `SLik_j`, which is a list including an `Rmixmod::mixmodCluster` object (or equivalent objects produced by non-default methods), and additional members not documented here. If projection was used, the list includes a data.frame `raw_data` of cumulated unprojected simulations.

## Examples

```
if (Infusion.getOption("example_maxtime")>50) {
  myrnorm <- function(mu,s2,sample.size) {
    s <- rnorm(n=sample.size,mean=mu,sd=sqrt(s2))
    return(c(mean=mean(s),var=var(s)))
  } # simulate means and variances of normal samples of size 'sample.size'
  set.seed(123)
  # pseudo-sample with stands for the actual data to be analyzed:
  ssize <- 40
  Sobs <- myrnorm(mu=4,s2=1,sample.size=ssize)
  # Uniform sampling in parameter space:
  npoints <- 600
  parsp <- data.frame(mu=runif(npoints,min=2.8,max=5.2),
                      s2=runif(npoints,min=0.4,max=2.4),sample.size=ssize)
  # Build simulation table:
  simuls <- add_reftable(Simulate="myrnorm",par.grid=parsp)
  # Infer surface:
  densv <- infer_SLik_joint(simuls,stat.obs=Sobs)
  # Usual workflow using inferred surface:
  slik_j <- MSL(densv) ## find the maximum of the log-likelihood surface
  slik_j <- refine(slik_j,maxit=5)
  plot(slik_j)
  # etc:
  profile(slik_j,c(mu=4)) ## profile summary logL for given parameter value
  confint(slik_j,"mu") ## compute 1D confidence interval for given parameter
  plot1Dprof(slik_j,pars="s2",gridSteps=40) ## 1D profile
}
```

## `infer_surface` *Infer a (summary) likelihood or tail probability surface from inferred likelihoods*

## Description

The `logLs` method uses a standard smoothing method (prediction under linear mixed models, a.k.a. Kriging) to infer a likelihood surface, using as input likelihood values themselves inferred with some error for different parameter values. The `tailp` method use a similar approach for smoothing binomial response data, using the algorithms implemented in the spaMM package for fitting GLMMs with autocorrelated random effects.

## Usage

```
## S3 method for class 'logLs'
infer_surface(object, method="REML",verbose=interactive(),allFix=NULL,...)
## S3 method for class 'tailp'
infer_surface(object, method="PQL",verbose=interactive(),allFix,...)
```

## Arguments

| | |
|---|---|
| `object` | A data frame with attributes, containing independent prediction of logL or of LR tail probabilities for different parameter points, as produced by `infer_logLs` or `infer_tailp`. |
| `method` | methods used to estimate the smoothing parameters. If `method="GCV"`, a generalized cross-validation procedure is used (for `logLs` method only). Other methods are as described in the `HLfit` documentation. |
| `verbose` | Whether to display some information about progress or not. |
| `allFix` | Fixed values in the estimation of smoothing parameters. For development purposes, not for routine use. For `infer_surface.logLs`, this should typically include values of all parameters fitted by `spaMM::corrHLfit` ($\rho, \nu, \phi, \lambda$, and `$etaFix=`$\beta$). |
| `...` | further arguments passed to or from other methods (currently not used). |

## Value

An object of class `SLik` or `SLikp`, which is a list including an `HLfit` object as returned by `corrHLfit`, and additional members not documented here.

## Examples

```
## see main documentation page for the package
```

---

`Infusion` *Inference using simulation*

---

## Description

Implements a collection of methods to perform inferences based on simulation of realizations of the model considered. In particular it implements “summary likelihood”, an approach that effectively evaluates and uses the likelihood of simulated summary statistics.

## Details

The methods implemented in `Infusion` by default assume that the summary statistics have densities. Special values of some statistic, having discrete probability mass, can be handled using the `boundaries` attribute of the observed summary statistics (see `handling_NAs` for a further use of this attribute).

## Note

See workflow examples in `example_reftable` and `example_raw`

## Examples

```
## see Note.
```

---

`init_grid` *Define starting points in parameter space.*

---

## Description

This function is exported from the blackbox package. It samples the space of estimated parameters. Also handles other fixed arguments that need to be passed to the function simulating the summary statistics (sample size is likely to be one such argument). The current sampling strategy is crude but achieves three desirable effects: It tries to sample the space uniformly, avoiding large gaps; it is not exactly a regular grid; and it includes replicates of some parameter points, required for good smoothing of the likelihood surface.

## Usage

```
init_grid(lower=c(par=0), upper=c(par=1), steps=NULL,
          nUnique=NULL, nRepl=min(10L,nUnique),
          jitterFac=0.5
         )
```

## Arguments

| | |
|---|---|
| `lower` | A vector of lower bounds for the parameters, as well as fixed arguments to be passed to the function simulating the summary statistics. Elements must be named. Fixed parameters character strings. |
| `upper` | A vector of upper bounds for the parameters, as well as fixed parameters. Elements must be named and match those of `lower`. |
| `steps` | Number of steps of the grid, in each dimension of estimated parameters. If NULL, a default value is defined from the other arguments. If a single value is given, it is applied to all dimensions. Otherwise, this must have the same length as `lower` and `upper` and named in the same way as the variable parameters in these arguments. |

| | |
|---|---|
| `nUnique` | Number of distinct values of parameter vectors in output. Default is an heuristic guess for good start from not too many points, computed as `floor(50^((v/3)^(1/3)))` where `v` is the number of variable parameters. |
| `nRepl` | Number of replicates of distinct values of parameter vectors in output. |
| `jitterFac` | Controls the amount of jitter of the points around regular grid nodes. The default value 0.5 means that a mode can move by up to half a grid step (independently in each dimension), so that two adjacent nodes moved toward each other can (almost) meet each other. |

## Value

A data frame. Each row defines a list of arguments of vector of the function simulating the summary statistics.

## Examples

```
set.seed(123)
init_grid()
init_grid(lower=c(mu=2.8,s2=0.5,sample.size=20),
          upper=c(mu=5.2,s2=4.5,sample.size=20),
          steps=c(mu=7,s2=9),nUnique=63)
```

---

MSL *Maximum likelihood from an inferred likelihood surface*

---

## Description

This computes the maximum of an object of class `SLik` representing an inferred (summary) likelihood surface

## Usage

```
MSL(object, CIs = TRUE, level = 0.95, verbose = interactive(),
    eval_RMSEs = TRUE, cluster_args=list(),...)
```

## Arguments

| | |
|---|---|
| `object` | an object of class `SLik` as produced by `infer_surface.logLs` |
| `CIs` | If `TRUE`, construct one-dimensional confidence intervals for all parameters. |
| `level` | Intended coverage probability of the confidence intervals. |
| `verbose` | Whether to display some information about progress and results. |
| `eval_RMSEs` | Logical: whether to evaluate prediction uncertainty for likelihoods/ likelihood ratios/ parameters. |
| `cluster_args` | A list of arguments, passed to `makeCluster`. May contain a non-null `spec` element, in which case the distinct `nb_cores` argument is ignored. |
| `...` | Further arguments passed from or to other methods. |

### Details

If Kriging has been used to construct the likelihood surface, RMSEs are computed using approximate formulas for prediction (co-)variances in linear mixed midels (see Details in predict). Otherwise, a more computer-intensive bootstrap method is used. par_RMSEs are computed from RMSEs and from the numerical gradient of profile log-likelihood at each CI bound. Only RMSEs, not par_RMSEs, are compared to precision.

### Value

The object is returned invisibly, with the following added members, each of which being (as from version 1.5.0) an environment:

MSL , containing variables MSLE and maxlogL that match the par and value returned by an optim call.

RMSEs containing, as variable RMSEs, the root mean square errors of the log-likelihood at its inferred maximum and of the log-likelihood ratios at the CI bounds.

par_RMSEs containing, as variable par_RMSEs, root mean square errors of the CI bounds.

To ensure backward-compatibility of code to possible future changes in the structure of the objects, the extractor function get_from should be used to extract the RMSEs and par_RMSEs variables from their respective environments, and more generally to extract any element from the objects.

### Examples

```
## see main documentation page for the package
```

## multi_binning *Multivariate histogram*

### Description

Constructs a multivariate histogram of the points. Optionally, first tests whether a given value is within the convex hull of input points and constructs the histogram only if this test is TRUE. This function is available for development purposes but is not required otherwise . It is sparsely documented and subject to changes without notice.

### Usage

```
multi_binning(m, subsize=trunc(nrow(m)^(Infusion.getOption("binningExponent"))),
                expand=5/100, focal=NULL)
```

### Arguments

| | |
|---|---|
| m | A matrix representing points in *d*-dimensional space, where *d* is the number of columns |
| subsize | A control parameter for an undocumented algorithm |
| expand | A control parameter for an undocumented algorithm |
| focal | Value to be tested for inclusion within the convex hull. Its elements must have names. |

## Details

The algorithm may be detailed later.

## Value

Either NULL (if the optional test returned FALSE), or an histogram represented as a data frame each row of which represents an histogram cell by its barycenter (a point in $d$-dimensional space), its "binFactor" (the volume of the cell times the total number of observations) and its "count" (the number of observations within the cell). The returned data frame has the following attributes: `attr(.,"stats")` are the column names of the $d$-dimensional points; `attr(.,"count")` is the column name of the count, and `attr(.,"binFactor")` is the column name of the binFactor.

---

`options` *Infusion options settings*

---

## Description

Allow the user to set and examine a variety of *options* which affect operations of the Infusion package. However, typically these should not be modified, and if they are, not more than once in a data analysis.

## Usage

```
Infusion.options(...)

Infusion.getOption(x)
```

## Arguments

| | |
|---|---|
| `x` | a character string holding an option name. |
| `...` | A named value or a list of named values. The following values, with their defaults, are used in `Infusion`:<br>`mixturing` character string: package or function to be used for mixture modelling. Recognized packages are `"Rmixmod"` (the default) and `"mclust"`;<br>`projTrainingSize`**:** Expression for `trainingsize` argument of `project.character`. This is used only for `REML` method of projection.<br>`knotNbr`**:** Expression for `knotnbr` argument of `project.character`.<br>`projKnotNbr = 1000`**:** default value of `knotnbr` argument of `project.character` for REML (as implied by default expression for `knotNbr`).<br>`logLname = "logL"`**:** default value of `logLname` argument of `infer_logLs`. The name given to the inferred log likelihoods in all analyses.<br>`LRthreshold= - qchisq(0.999,df=1)/2`**:** A value used internally by `sample_volume` to sample points in the upper region of the likelihood surface, as defined by the given likelihood ratio threshold. |

**`precision = 0.1`:** default value of `precision` argument of `refine`. Targets RMSE of log L and log LR estimates.

**`nRealizations=1000`:** default value of `nRealizations` argument of `add_simulation`. Number of realizations for each empirical distribution.

**`mixmodGaussianModel="Gaussian_pk_Lk_Dk_A_Dk"`:** default models used in clustering by `Rmixmod`. Run `Rmixmod::mixmodGaussianModel()` for a list of possible models, and see the statistical documentation (Mixmod Team 2016) for explanations about them.

**`nbCluster = quote(seq(ceiling(nrow(data)^0.3)))`:** default value of `nbCluster` used in clustering by `Rmixmod` (see Details for discussion of this default).

**`example_maxtime=2.5`:** Used in the documentation to control whether the longer examples should be run. The approximate running time of given examples (or some very rough approximation for it) on one author's laptop is compared to this value.

**`nb_cores`** Number of cores for parallel computations (see Details for implementation of these).

and possibly other undocumented values for development purposes.

## Details

The default upper value of the `nbCluster` range is the value recommended in the `mixmod` statistical documentation (Mixmod Team, 2016).

`Infusion` can perform parallel computations if several cores are available and requested though `Infusion.options(nb_cores=.)`. If the `doSNOW` back-end is attached (by explicit request from the user), it will be used; otherwise, `pbapply` will be used. Both provide progress bars, but `doSNOW` may provide more efficient load-balancing. The character shown in the progress bar is `'P'` for parallel via `doSNOW` backend, `'p'` for parallel via `pbapply` functions, and `'s'` for serial via `pbapply` functions. I addition, `add_simulation` can parallelise at two levels: at an outer level over parameter point, or atan inner level over simulation replicates for each parameter point. The progress bas of the outer computation is shown, but the character shown in the progress bar is `'N'` if the inner computation is parallel via the `doSNOW` backend, and `'n'` if it is parallel via `pbapply` functions. So, one should see either `'P'` or `'N'` when using `doSNOW`.

## Value

For `Infusion.getOption`, the current value set for option `x`, or `NULL` if the option is unset.

For `Infusion.options()`, a list of all set options. For `Infusion.options(name)`, a list of length one containing the set value, or `NULL` if it is unset. For uses setting one or more options, a list with the previous values of the options changed (returned invisibly).

## References

Mixmod Team (2016). Mixmod Statistical Documentation. Université de Franche-Comté, Besançon, France. Version: February 10, 2016 retrieved from http://www.mixmod.org.

## Examples

```
  Infusion.options()
  Infusion.getOption("LRthreshold")
  ## Not run:
  Infusion.options(LRthreshold=- qchisq(0.99,df=1)/2)

## End(Not run)
```

---

`plot.SLik` *Plot SLik or SLikp objects*

---

## Description

Mostly conceived for exposition purposes, for the two-parameters case. The black-filled points are those for which the observed summary statistic was outside of the convex hull of the simulated empirical distribution. The crosses mark the estimated ML point and the confidence intervals points, that is, the outmost points on the contour defined by the profile likelihood threshold for the profile confidence intervals. There is a pair of CI points for each interval. The smaller black dots mark points added in the latest iteration, if `refine` was used.

## Usage

```
## S3 method for class 'SLik'
plot(x, y, filled = FALSE, decorations = NULL,
                    color.palette = NULL, plot.axes = NULL,
                    plot.title = NULL, ...)
## S3 method for class 'SLik_j'
plot(x, y, filled = nrow(x$logLs)>5000L, decorations = NULL,
                      color.palette = NULL, plot.axes = NULL,
                      plot.title = NULL, from_refine=FALSE, ...)
```

## Arguments

| | |
|---|---|
| `x` | An object of class `SLik` or `SLikp` |
| `y` | Not used, but included for consistency with the `plot` generic. |
| `filled` | whether to plot a `mapMM` or a `filled.mapMM`. |
| `decorations` | Graphic directives added to the default `decorations` value in calls of `mapMM` or a `filled.mapMM` (see the source code of `plot.SLik` for the latter default values). |
| `color.palette` | Either NULL or a function that can replace the default color function used by `plot.SLik`. Thefunction must have a single argument, giving the number of color levels. |
| `plot.title` | statements which replace the default titles to the main plot (see Details). |
| `plot.axes` | statements which replace the default axes on the main plot (see Details). |
| `from_refine` | For programming purposes, not documented. |
| `...` | further arguments passed to or from other methods (currently can be used to pass a few arguments such as `map.asp` in all cases, or `variances` to `filled.mapMM`). |

## Details

Different graphic functions are called depending on the number of estimated parameters. For two parameters, `mapMM` or `filled.mapMM` are called. For more than two parameters, `spaMM.filled.contour` is called. See the documentation of these functions for the appropriate format of the `plot.title` and `plot.axes` arguments.

## Value

Returns the plotted object invisibly.

## Examples

```
## Not run:
## Using 'slik' object from the example in help("Infusion-package")
  plot(slik, filled=TRUE,
       plot.title=quote(title("Summary-likelihood-ratio surface",
                              xlab=expression(mu),
                              ylab=expression(sigma^2))))

## End(Not run)
```

---

`plot1Dprof` *Plot likelihood profiles*

---

## Description

These functions plot 1D and 2D profiles from an SLik object

## Usage

```
plot1Dprof(object, pars=object$colTypes$fittedPars, type="logLR",
            gridSteps=21, xlabs=list(), ylab, scales=NULL,
            plotpar=list(pch=20),
            control=list(min=-7.568353, shadow_col="grey70"))
plot2Dprof(object, pars=object$colTypes$fittedPars, type="logLR",
            gridSteps=17, xylabs=list(), main, scales=NULL,
            plotpar=list(pch=20), margefrac = 0)
```

## Arguments

| | |
|---|---|
| `object` | An SLik object |
| `pars` | The parameters for which profiles will be computed. For 2D plots, all pairs of parameters in `pars` are considered |
| `type` | Character: `"logL"` to plot the log-likelihood profile; `"logLR"` (or `"LR"` for the not-log version) to plot the log-likelihood-ratio profile (the default); or `"zoom"` or `"dual"` for variants of `"logLR"` (see details). |

| | |
|---|---|
| `gridSteps` | The number of values (in each dimension for 2D plots) which likelihood should be computed. For 1D plots, `gridSteps=0` is now obsolete. |
| `xlabs` | A *list* of alternative axis labels. The names of the list elements should be elements of `pars` (see Examples) |
| `xylabs` | Same as `xlabs` but affecting both axes in 2D plots |
| `ylab` | Same as `ylab` argument of `plot`. Default depends on `type` argument. |
| `main` | Same as `main` argument of `plot`. Default depends on `type` argument. |
| `scales` | A named character vector, which controls ticks and tick labels on axes, so that these can be expressed as (say) the exponential of the parameter inferred in the SLik object. For example if the likelihood of `logPop` = log(population size) was thus inferred, `scales=c(logPop="log")` will give population size values on the axis (but will retain a log scale for this parameter). Possible values of each element of the vector are `"identity"` (default), `"log"`, and `"log10"`, |
| `plotpar` | arguments for `par()` such as font sizes, etc. |
| `control` | Control of `"zoom"` or `"dual"` plots (see Details). |
| `margefrac` | For development purposes, not documented. |

## Details

A `"zoom"` plot shows only the top part of the profile, defined as points whose y-values are above a threshold minus-log-likelihood ratio `control$min`, whose default is -7.568353, the 0.9999 p-value threshold.

A `"dual"` plot displays both the zoom, and a shadow graph showing the full profile. The dual plot is shown only when requested and if there are values above and below `control$min`. The shadow curve color is given by `control$shadow_col`.

## Value

Both functions return a list, which currently has a single element `MSL_updated` which is a boolean indicating whether the summary-likelihood maximum (but not the intervals) has been recomputed.

## Examples

```
if (Infusion.getOption("example_maxtime")>40) {
 data(densv)
 slik <- infer_surface(densv) ## infer a log-likelihood surface
 slik <- MSL(slik) ## find the maximum of the log-likelihood surface
 plot1Dprof(slik,pars="s2",gridSteps=40,xlabs=list(s2=expression(paste(sigma^2))))
}
```

## `profile.SLik` *Compute profile summary likelihood*

### Description

Predicts the profile likelihood for a given parameter value (or vector of such values) using predictions from an `SLik` object (as produced by `MSL`).

### Usage

```
## S3 method for class 'SLik'
profile(fitted, value, fixed=NULL, return.optim=FALSE, ...)
## S3 method for class 'SLik_j'
profile(fitted, ...)
```

### Arguments

| | |
|---|---|
| `fitted` | an `SLik` object. |
| `value` | The parameter value (as a vector of named values) for which the profile is to be computed |
| `fixed` | When this is `NULL` the computed interval is a profile confidence interval over all parameters excluding `value`. `fixed` allows one to set fixed values to some of these parameters. |
| `return.optim` | If this is TRUE, and if maximization of likelihood given `value` and `fixed` is indeed required, then the full result of the optimization call is returned. |
| `...` | For `SLik_j` method, arguments passed to `SLik` method. For `SLik_j` method, currently not used. |

### Value

The predicted summary profile log-likelihood; or possibly the result of an optimization call if `return.optim` is TRUE.

### Examples

```
## see main documentation page for the package
```

## `project.character` *Learn a projection method for statistics and applies it*

### Description

`project` is a generic function with two methods. If the first argument is a parameter name, `project.character` (alias: `get_projector`) defines a projection function from several statistics to an output statistic predicting this parameter. `project.default` (alias: `get_projection`) produces a vector of projected statistics using such a projection. `project` is particulary useful to reduce a large number of summary statistics to a vector of projected summary statistics, with as many elements as parameters to infer. This dimension reduction can substantially speed up subsequent computations. The concept implemented in `project` is to fit a parameter to the various statistics available, using machine-learning or mixed-model prediction methods. All such methods can be seen as nonlinear projection to a one-dimensional space. `project.character` is an interface that allows different projection methods to be used, provided they return an object of a class that has a defined `predict` method with a `newdata` argument (as expected, see `predict`).

### Usage

```
project(x,...)

## S3 method for building the projection
## S3 method for class 'character'
project(x, stats, data,
             trainingsize= eval(Infusion.getOption("projTrainingSize")),
             knotnbr=  eval(Infusion.getOption("knotnbr")),
             method,methodArgs=list(),verbose=TRUE,...)
get_projector(...) # alias for project.character

## S3 method for applying the projection
## Default S3 method:
project(x, projectors,...)
get_projection(...) # alias for project.default
```

### Arguments

| | |
|---|---|
| `x` | The name of the parameter to be predicted, or a vector/matrix/list of matrices of summary statistics. |
| `stats` | Statistics from which the predictor is to be predicted |
| `data` | A list of simulated empirical distributions, as produced by `add_simulation`, or a data frame with all required variables. |
| `trainingsize` | For REML only: size of random sample of realizations from the `data` from which the smoothing parameters are estimated. |
| `knotnbr` | Size of random sample of realizations from the `data` from which the predictor is built given the smoothing parameters. |

| | |
|---|---|
| `method` | character string: `"REML"`, `"GCV"`, or the name of a suitable projection function. The latter may be defined in another package, e.g. `"ranger"` or `"randomForest"`, or predefined by Infusion (function `"nnetwrap"`), or defined by the user. See Details for predefined functions and for defining new ones. The default method is `"randomForest"` if this package is installed, and `"REML"` otherwise. Defaults may change in later versions, so it is advised to provide an explicit `method` to improve reproducibility. |
| `methodArgs` | A list of arguments for the projection method. One may not need to provide arguments in the following cases, where `project` kindly (tries to) assign values to the required arguments if they are absent from `methodArgs`:<br>If `"REML"` or `"GCV"` methods are used (in which case `methodArgs` is completely ignored); or<br>if the projection method uses `formula` and `data` arguments (in particular if the formula is of the form `response ~ var1 + var2 + ...`; otherwise the formula should be provided through `methodArgs`). This works for example for methods based on `nnet`; or<br>if the projection method uses `x` and `y` arguments. This works for example for `randomForest` (though not with the generic function `method="randomForest"`, but only with the internal function `method="randomForest:::randomForest.default"`). |
| `projectors` | A list with elements of the form `<name>=<project result>`, where the `<name>` must differ from any name of `x`. `<project result>` may indeed be the return object of a `project` call. |
| `verbose` | Whether to print some information or not. In particular, `TRUE`, true-vs.-predicted diagnostic plots will be drawn for projection methods “known” by Infusion (notably `"ranger"`, `"keras::keras.engine.training.Model"`, `"randomForest"`, `"GCV"`, `caret::train`). |
| `...` | Further arguments passed to or from other functions. |

## Details

Prediction can be based on a linear mixed model (LMM) with autocorrelated random effects, internally calling the `corrHLfit` function with formula `<parameter> ~ 1+ Matern(1|<stat1>+...+<statn>)`. This approach allows in principle to produce arbitrarily complex predictors (given sufficient input) and avoids overfitting in the same way as restricted likelihood methods avoids overfitting in LMM. REML methods are then used by default to estimate the smoothing parameters. However, faster methods may be required, and random forests by `ranger` is the default method, if that package is installed (`randomForest` can also be used). Some other methods are possible (see below).

Machine learning methods such as random forests overfit, *except if* out-of-bag predictions are used. When they are not, the bias is manifest in the fact that using the same simulation table for learning the projectors and for other steps of the analyses tend to lead to too narrow confidence regions. This bias disappears over iterations of `refine` when the projectors are kept constant. `Infusion` avoid this bias by using out-of-bag predictions when relevant, when `ranger` and `randomForest` are used. But it provides no code handing that problem for other machine-learning methods. Then, users should cope with that problems, and at a minimum should not update projectors in every iteration (the “Gentle Introduction to Infusion may contain further information about this problem”).

To keep REML computation reasonably fast, the `trainingsize` and `knotnbr` arguments determine respectively the size of the subset used to estimate the smoothing parameters and the size of the subset defining the predictor given the smoothing parameters. REML fitting is already slow for data sets of this size (particularly as the number of predictor variables increase).

If `method="GCV"`, a generalized cross-validation procedure (Golub et al. 1979) is used to estimate the smoothing parameters. This is faster but still slow, so a random subset of size `knotnbr` is still used to estimate the smoothing parameters and generate the predictor.

Alternatively, various machine-learning methods can be used (see e.g. Hastie et al., 2009, for an introduction). A random subset of size `knotnbr` is again used, with a larger default value bearing the assumption that these methods are faster. Predefined methods include `"ranger"`, `"randomForest"`, and `method="neuralNet"` which interfaces a neural network method, using the `train` function from the `caret` package.

In principle, any object suitable for prediction could be used as one of the `projectors`, and `Infusion` implements their usage so that in principle unforeseen projectors could be used. That is, if predictions of a parameter can be performed using an object `MyProjector` of class `MyProjectorClass`, `MyProjector` could be used in place of a `project` result if `predict.MyProjectorClass(object,newdata,...)` is defined. However, there is no guarantee that this will work on unforeseen projetion methods, as each method tends to have some syntactic idiosyncrasies. For example, if the learning method that generated the projector used a formula-data syntax, then its `predict` method is likely to request names for its `newdata`, that need to be provided through `attr(MyProjector,"stats")` (these names cannot be assumed to be in the `newdata` when `predict` is called through `optim`).

## Value

`project.character` returns an object of class returned by the `method` (methods `"REML"` and `"GCV"` will call `corrHLfit` which returns an object of class spaMM) `project.default` returns an object of the same class and structure as the input `x`, containing the projected statistics inferred from the input summary statistics.

## References

Golub, G. H., Heath, M. and Wahba, G. (1979) Generalized Cross-Validation as a method for choosing a good ridge parameter. Technometrics 21: 215-223.

T. Hastie, R. Tibshirani, and J. Friedman. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, New York, 2nd edition, 2009.

## Examples

```
##########
if (Infusion.getOption("example_maxtime")>250) {
## Transform normal random deviates rnorm(,mu,sd)
## so that the mean of transformed sample is not sufficient for mu,
## and that variance of transformed sample is not sufficient for sd,
blurred <- function(mu,s2,sample.size) {
  s <- rnorm(n=sample.size,mean=mu,sd=sqrt(s2))
  s <- exp(s/4)
  return(c(mean=mean(s),var=var(s)))
}

set.seed(123)
dSobs <- blurred(mu=4,s2=1,sample.size=20) ## stands for the actual data to be analyzed
## Sampling design as in canonical example
parsp <- init_grid(lower=c(mu=2.8,s2=0.4,sample.size=20),
                   upper=c(mu=5.2,s2=2.4,sample.size=20))
# simulate distributions
dsimuls <- add_simulation(,Simulate="blurred", par.grid=parsp)

## Use projection to construct better summary statistics for each each parameter
mufit <- project("mu",stats=c("mean","var"),data=dsimuls)
s2fit <- project("s2",stats=c("mean","var"),data=dsimuls)

## additional plots for some projection method
if (inherits(mufit,"HLfit")) mapMM(mufit,map.asp=1,
  plot.title=title(main="prediction of normal mean",xlab="exp mean",ylab="exp var"))
if (inherits(s2fit,"HLfit")) mapMM(s2fit,map.asp=1,
  plot.title=title(main="prediction of normal var",xlab="exp mean",ylab="exp var"))

## apply projections on simulated statistics
corrSobs <- project(dSobs,projectors=list("MEAN"=mufit,"VAR"=s2fit))
corrSimuls <- project(dsimuls,projectors=list("MEAN"=mufit,"VAR"=s2fit))

## Analyze 'projected' data as any data (cf canonical example)
densb <- infer_logLs(corrSimuls,stat.obs=corrSobs)
} else data(densb)
#########
if (Infusion.getOption("example_maxtime")>10) {
slik <- infer_surface(densb) ## infer a log-likelihood surface
slik <- MSL(slik) ## find the maximum of the log-likelihood surface
}
if (Infusion.getOption("example_maxtime")>500) {
slik <- refine(slik,10, update_projectors=TRUE) ## refine iteratively
}
```

## refine *Refine estimates iteratively.*

### Description

This is a generic function with currently methods for `SLik`, `SLik_j` and `SLikp` objects (as produced by `MSL`). Depending on the value of its `newsimuls` argument, and on whether the function used to generate empirical distributions can be called by R, it (1) defines new parameters points and/or (2) infers their summary likelihood or tail probabilities for each parameter point independently, adds the inferred values results as input for refined inference of likelihood or P-value response surface, and provides new point estimates and confidence intervals.

### Usage

```
## S3 method for class 'SLik'
refine(object, method=NULL, ...)
```

```
## Default S3 method:
refine(object, surfaceData, Simulate =
            attr(surfaceData,"Simulate"), maxit = 1, n = NULL,
            useEI = list(max=TRUE,profileCI=TRUE,rawCI=FALSE),
            newsimuls = NULL, useCI = TRUE, level = 0.95,
          verbose = list(most=interactive(),final=NULL,movie=FALSE,proj=FALSE),
            precision = Infusion.getOption("precision"),
            nb_cores = NULL, packages=attr(object$logLs,"packages"),
            env=attr(object$logLs,"env"), method,
            eval_RMSEs=TRUE, update_projectors = FALSE,
            cluster_args=list(),
            ...)
```

## Arguments

| | |
|---|---|
| `object` | an `SLik` object |
| `surfaceData` | A data.frame with attributes, usually taken from the `object` and thus **not** specified by user, usable as input for `infer_surface`. |
| `Simulate` | Character string: name of the function used to simulate sample. The only meaningful non-default value is `NULL`, in which case `refine` may return (if `newsimuls` is also `NULL`) a data frame of parameter points on which to run a simulation function. |
| `maxit` | Maximum number of iterative refinements (see also `precision` argument) |
| `n` | A number of parameter points (excluding replicates and confidence interval points), whose likelihood should be computed (see `n` argument of `sample_volume`) |
| `useEI` | Cf this argument in `rparam` |
| `newsimuls` | For the `SLik_j` method, a matrix or data frame, with the same parameters and summary statistics as the `data` of the original `infer_SLik_joint` call. For other methods, a `list` of simulation of distributions of summary statistics, in the same format as for `link{add_simulation}`. If no such list is provided (i.e., if `newsimuls` remains `NULL`), the `attr(object$logLs,"Simulate")` function is used (it is inherited from the `Simulate` argument of `add_simulation` through the initial sequence of calls of functions `add_simulation`, `infer_logLs` or `infer_tailp`, and `infer_surface`). If no such function is available, then this function returns parameters for which new distribution should be provided by the user. |
| `useCI` | whether to include parameter points near the inferred confidence interval points in the set of points which likelihood should be computed |
| `level` | Intended coverage of confidence intervals |
| `verbose` | A list as shown by the default, or simply a vector of booleans. `verbose$most` controls whether to display information about progress and results, except plots; `$final` controls whether to `plot()` the final `object` to show the final likelihood surface. Default is to plot it only in an interactive session and if fewer than three parameters are estimated; `$movie` controls whether to `plot()` the updated `object` in each iteration; `verbose$proj` controls the `verbose` argument |

| | |
|---|---|
| | of `project.character`. If `verbose` is a vector of booleans, they are matched to as many elements from `"most","movie","final","proj"`, in that order. |
| `precision` | Requested local precision of surface estimation, in terms of prediction standard errors (RMSEs) of both the maximum summary log-likelihood and the likelihood ratio at any CI bound available. Iterations will stop when either `maxit` is reached, or if the RMSEs have been computed for the object (see `eval_RMSEs` argument) and this precision is reached for the RMSEs. A given precision on the CI bounds themselves might seem more interesting, but is not well specified by a single precision parameter if the parameters are on widely different scales. |
| `cluster_args` | A list of arguments, passed to `makeCluster`. If thus affects parallel computations in the functions to which it may be passed: `add_simulation`, `MSL` (for bootstrap computations) or `infer_logLs`. It may contain a non-null `spec` element, in which case the distinct `nb_cores` argument of `refine` is ignored. If `spec` is a **named** integer, it has an additional effect on `add_simulation`. |
| `nb_cores` | Shortcut for `cluster_args$spec`. |
| `packages` | NULL or a list with possible elements `add_simulation` and `logL_method`, passed respectively as the `packages` arguments of `add_simulation` and `infer_logLs`, wherein they are the additional packages to be loaded on child processes. The default value keeps pre-`refine` values over iterations. |
| `env` | An environment, passed as the `env` argument to `add_simulation`. The default value keeps the pre-`refine` value over iterations. |
| `method` | (A vector of) suggested method(s) for estimation of smoothing parameters (see `method` argument of `infer_surface`). The ith element of the vector is used in the ith iteration, if available; otherwise the last element is used. This argument is not always heeded, in that REML may be used if the suggested method is GCV but it appears to perform poorly. The default for `SLikp`, `SLik_j`, and `SLikp` objects are `"REML"`, `"mixmodCluster"`, and `"PQL"`, respectively. |
| `eval_RMSEs` | passed to `MSL` |
| `update_projectors` | Boolean; whether to update the projectors at each iteration. |
| `...` | further arguments passed to or from other methods. `refine` passes these arguments to the `plot` method suitable for the `object`. |

## Details

New parameter points are sampled as follows: the algorithm aims to sample uniformly the space of parameters contained in the confidence regions defined by the `level` argument, and to surround it by a region sampled proportionally to likelihood. In each iteration the algorithm aims to add as many points (say *n*) as computed in the first iteration, so that after *k* iterations of `refine`, there are $n * (k + 1)$ points in the simulation table. However, when not enough points satisfy certain criteria, only *n/5* points may be added in an iteration, this being compensated in further iterations. For example, if $n = 600$, the table may include only 720 points after the first refine, but 1800 after the second.

## Value

An updated `SLik` or `SLik_j` object.

### Note

See workflow examples in `example_reftable` and `example_raw`

### Examples

```
## see Note.
```

---

## rparam — *Sample the parameter space*

---

### Description

These functions take an `SLik` object (as produced by `MSL`) and samples its parameter space in (hopefully) clever ways, not yet well documented. `rparam` calls `sample_volume` to define points targeting the likelihood maximum and the bounds of confidence intervals, with `n` for these different targets dependent on the mean square error of prediction of likelihood at the maximum and at CI bounds.

### Usage

```
rparam(object, n= 1, useEI = list(max=TRUE,profileCI=TRUE,rawCI=FALSE),
       useCI = TRUE, verbose = interactive(), tryn=30*n,
       level = 0.95, CIweight=Infusion.getOption("CIweight"))

sample_volume(object, n = 6, useEI, vertices=NULL,
              dlr = NULL, verbose = interactive(),
              fixed = NULL, tryn= 30*n)
```

### Arguments

| | |
|---|---|
| `object` | an `SLik` object |
| `n` | The number of parameter points to be produced |
| `useEI` | List of booleans, each determining whether to use an “expected improvement” (EI) criterion (e.g. Bingham et al., 2014) to select candidate parameter points to better ascertain a particular focal point. The elements `max`, `profileCI` and `rawCI` determine this for three types of focal points, respectively the MSL estimate, profile CI bounds, and full-dimensional bounds. When EI is used, `n` points with best EI are selected among `tryn` points randomlygenerated in some neighborhood of the focal point. |
| `vertices` | Points are sampled within a convex hull defined by `vertices`. By default, these vertices are taken from `object$fit$data`. |
| `useCI` | Whether to define points targeting the bounds of confidence intervals for the parameters. An expected improvement criterion is also used here. |
| `level` | If `useCI` is `TRUE` but confidence intervals are not available from the `object`, such intervals are computed with coverage `level`. |

| | |
|---|---|
| `dlr` | A (log)likelihood ratio threshold used to select points in the upper region of the likelihood surface. Default value is given by `Infusion.getOption("LRthreshold")` |
| `verbose` | Whether to display some information about selection of points, or not |
| `fixed` | A list or named vector, of which each element is of the form `<parameter name>=<value>`, defining a one-dimensional constraint in parameter space. Points will be sampled in the intersection of the volume defined by the `object` and of such constraint(s). |
| `tryn` | See `useEI` argument. |
| `CIweight` | For development purposes, not documented. |

## Value

a data frame of parameter points. Only parameters variable in the `SLik` object are considered.

## References

D. Bingham, P. Ranjan, and W.J. Welch (2014) Design of Computer Experiments for Optimization, Estimation of Function Contours, and Related Objectives, pp. 109-124 in Statistics in Action: A Canadian Outlook (J.F. Lawless, ed.). Chapman and Hall/CRC.

## Examples

```
if (Infusion.getOption("example_maxtime")>10) {
 data(densv)
 summliksurf <- infer_surface(densv) ## infer a log-likelihood surface
 sample_volume(summliksurf)
}
```

# Index

∗ **datasets**
densv, 6
∗ **package**
Infusion, 19

add_reftable (add_simulation), 2
add_simulation, 2, *12*, *15*, *24*, *29*, *33*, *34*

boundaries-attribute (handling_NAs), 14

class:dMixmod (dMixmod), 7
class:NULLorChar (dMixmod), 7
class:NULLorNum (dMixmod), 7
confint (confint.SLik), 5
confint.SLik, 5
corrHLfit, *19*, *30*, *31*

densb (densv), 6
densv, 6
dMixmod, 7
dMixmod-class (dMixmod), 7

example_raw, 8, *20*, *35*
example_reftable, *2*, 9, *10*, *20*, *35*
extractors, 10

filled.mapMM, *25*, *26*

get_from, *10*, 11, *22*
get_projection (project.character), 29
get_projector (project.character), 29
goftest, 12

handling_NAs, 14, *20*
HLfit, *15*, *19*

infer_logL_by_GLMM (infer_logLs), 15
infer_logL_by_Hlscv.diag (infer_logLs), 15
infer_logL_by_mclust (infer_logLs), 15
infer_logL_by_Rmixmod (infer_logLs), 15
infer_logLs, *7*, 15, *19*, *23*
infer_SLik_joint, 17, *33*
infer_surface, 18, *33*, *34*
infer_surface.logLs, *21*
infer_tailp, *19*
infer_tailp (infer_logLs), 15
Infusion, *7*, *17*, 19
Infusion-package (Infusion), 19
Infusion.getOption (options), 23
Infusion.options, *3*
Infusion.options (options), 23
init_grid, 20

logLik, *11*
logLik (extractors), 10

makeCluster, *3*, *16*, *21*, *34*
mapMM, *25*, *26*
MSL, *5*, 21, *28*, *32*, *34*, *35*
multi_binning, 22

neuralNet (project.character), 29
NULLorChar (dMixmod), 7
NULLorChar-class (dMixmod), 7
NULLorNum (dMixmod), 7
NULLorNum-class (dMixmod), 7

options, 23

parallel (options), 23
plot.dMixmod (dMixmod), 7
plot.SLik, *16*, 25
plot.SLik_j (plot.SLik), 25
plot.SLikp (plot.SLik), 25
plot1Dprof, 26
plot2Dprof (plot1Dprof), 26
predict, *22*, *29*
print (extractors), 10
profile (profile.SLik), 28
profile.SLik, 28
project, *7*

project *(*project.character*)*, 29
project.character, *23*, 29, *34*

refine, *16*, *24*, *30*, 32
rparam, *33*, 35

sample_volume, *23*, *33*
sample_volume *(*rparam*)*, 35
saved_seed *(*densv*)*, 6
spaMM.filled.contour, *26*
summary *(*extractors*)*, 10