Package 'bdpopt'

October 12, 2022

Version 1.0-1

Date 2016-03-29

Title Optimisation of Bayesian Decision Problems

Author Sebastian Jobjörnsson [aut, cre]

Maintainer Sebastian Jobjörnsson <jobjorns@chalmers.se>

Depends R (>= 3.0.2)

Description Optimisation of the expected utility in single-stage and multi-
stage Bayesian decision problems. The expected utility is estimated by simulation. For single-
stage problems, JAGS is used to draw MCMC samples.

SystemRequirements JAGS (>= 3.4.0) (see
http://mcmc-jags.sourceforge.net)

Imports rjags (>= 3-15), coda (>= 0.17-1), parallel (>= 3.0.2)

License GPL-2

LazyData true

Encoding latin1

NeedsCompilation no

Repository CRAN

Date/Publication 2016-03-30 10:14:45

R topics documented:

`create.normal.model` *Create Normal Emax Model*

Description

Create a simulation model object for optimisation of the dose and sample size for a (or several, parallel) phase III clinical trial given phase II data. There is one efficacy response and one safety response obtained as sample means for each group of patients given a specific dose. These are both normal given the true populations means. The population means for efficacy and safety are in turn modeled using two independent Emax models.

Usage

```
create.normal.model(theta.mu, theta.tau, eta.mu, eta.tau,
n.II, d.II, YE.II, YS.II, sigmaE, sigmaS, k.III, path.to.package = NA)
```

Arguments

`theta.mu`	A numeric, atomic vector with four elements that contains the mean parameters for the prior distributions for the efficacy model.
`theta.tau`	A numeric, atomic vector with four elements that contains the precision parameters for the prior distributions for the efficacy model.
`eta.mu`	A numeric, atomic vector with four elements that contains the mean parameters for the prior distributions for the safety model.
`eta.tau`	A numeric, atomic vector with four elements that contains the precision parameters for the prior distributions for the safety model.
`n.II`	A numeric, atomic vector of positive integers containing the group sample sizes corresponding to the phase II efficacy and safety responses.
`d.II`	A numeric, atomic vector containing the group dose levels corresponding to the phase II efficacy and safety responses.

`YE.II`	A numeric, atomic vector containing the observed sample means for the efficacy responses in the phase II trial.
`YS.II`	A numeric, atomic vector containing the observed sample means for the safety responses in the phase II trial.
`sigmaE`	The population standard deviation for an individual efficacy response. The population standard deviation for a group response is then obtained by dividing by the square root of the group sample size.
`sigmaS`	The population standard deviation for an individual safety response. The population standard deviation for a group response is then obtained by dividing by the square root of the group sample size.
`k.III`	A positive integer specifying the number of independent phase III trials, each of which consists of a single group. The sample size and dose is the same for all phase III trials.
`path.to.package`	The search path to the installation directory of **bdpopt**. For the default value, the function will attempt to find the path using `search`.

Details

Note that `n.II`, `d.II`, `YE.II` and `YS.II` must all be of the same length. Using the notation of the JAGS manual, the prior distributions for `theta` and `eta` are given by
`theta[i]` ~ `dnorm(theta.mu[i], theta.tau[i])`,
`eta[i]` ~ `dnorm(eta.mu[i], eta.tau[i])`, for i = `1, 2`, and
`theta[i]` ~ `dlnorm(theta.mu[i], theta.tau[i])`,
`eta[i]` ~ `dlnorm(eta.mu[i], eta.tau[i])`, for i = `3, 4`.

Value

A simulation model object of class `sim.model`, created using the data supplied as the arguments and the JAGS model file '`normal_model_jags_model.R`'.

Author(s)

Sebastian Jobjörnsson <`jobjorns@chalmers.se`>

See Also

`create.normal.model.from.file`

`create.normal.model.from.file`
Create Normal Emax Model From File

Description

Create a simulation model object for optimisation of the dose and sample size for a (or several, parallel) phase III clinical trial given phase II data. There is one efficacy response and one safety response obtained as sample means for each group of patients given a specific dose. These are both normal given the true populations means. The population means for efficacy and safety are in turn modeled using two independent Emax models. This function uses the file '`normal_model_jags_data.R`' to specify the parameters of the model instead of the user supplying these as arguments (as for `create.normal.model`).

Usage

```
create.normal.model.from.file(path.to.package = NA)
```

Arguments

`path.to.package`	The search path to the installation directory of **bdpopt**. For the default value, the function will attempt to find the path using `search`.

Details

See the argument description of `create.normal.model` for the objects that have to be specified in '`normal_model_jags_data.R`'.

Value

A simulation model object of class `sim.model`, created using the JAGS data file '`normal_model_jags_data.R`' and the JAGS model file '`normal_model_jags_model.R`'.

Author(s)

Sebastian Jobjörnsson <`jobjorns@chalmers.se`>

See Also

`create.normal.model`

`create.utility.function` *Create Utility Function For The Normal Model*

Description

Create a utility function to be used together with a normal simulation model object created using `create.normal.model` or `create.normal.model.from.file`.

Usage

```
create.utility.function(model, n.min, sig.level, safety.max, cE, cS, p,
fixed.cost, cost.per.sample)
```

Arguments

`model`	A model object created using `create.normal.model` or `create.normal.model.from.file`.
`n.min`	The minimum group sample size for each phase III group trial required by the regulatory authority in order to consider market approval.
`sig.level`	The significance level used by the regulatory authority when performing the one-sided hypothesis tests for acceptable efficacy and safety levels in the phase III trial.
`safety.max`	A parameter defining the maximum safety threshold in the significance test for an acceptable safety level.
`cE`	A constant defining the utility gain per unit of efficacy.
`cS`	A constant defining the utility gain per unit of safety. The absolute value of this number defines the utility loss, and hence `cS` should typically be less than or equal to zero.
`p`	A number between 0 and 1 which weighs the relative contribution of the observed responses and the true population means to the utility upon regulatory approval. A value of 1 corresponds to no contribution made by the population means.
`fixed.cost`	The fixed cost of performing the phase III trials.
`cost.per.sample`	The cost per observation in the phase III trials.

Details

The utility function has the form:

`RA.decision * gain - trial.cost`

where

`gain = p * (cE * mean(YE) + cS * mean(YS)) + (1 - p) * (cE * mean(muE) + cS * mean(muS))`

`trail.cost = fixed.cost + cost.per.sample * k.III * n.III`

Value

An R function to be used together with `model` when calling `eval.on.grid`, `fit.gpr`, `fit.loess` and `optimise.eu`.

Author(s)

Sebastian Jobjörnsson `<jobjorns@chalmers.se>`

diag.mcmc.list *MCMC List For Diagnostic Evaluation*

Description

Get the mcmc.list object obtained when running the JAGS model corresponding to model. Diagnostic functions provided by the **coda** package may then be applied in order to check that there is a sufficent amount of adaptation and burn in for the application. This is a generic function for S3 objects.

Usage

```
diag.mcmc.list(model, utility.fun, data,
n.iter, n.burn.in, n.adapt = 1000, n.chains = 1, inits = NULL)
```

Arguments

model	Simulation model object of class sim.model created using sim.model or create.normal.model.
utility.fun	An R function defining the utility for the decision problem. It must be possible to extract the argument names of the function supplied with formals, and the argument names must constitute a subset of the names used in the JAGS model specification.
data	A named list of R objects which, when combined with the named objects in model$data, leads to a complete specification of the data for the JAGS model.
n.iter	The number of iterations in the JAGS MCMC simulation.
n.burn.in	The number of burn in iterations prior to the JAGS MCMC simulation.
n.adapt	The number of adaptation iterations prior to the burn in phase and subsequent JAGS MCMC simulation.
n.chains	The number of parallel MCMC chains to run.
inits	The initial values for the chain(s) passed on to the function rjags::jags.model, unless equal to its default value NULL, in which case JAGS chooses default initial values.

Details

The purpose of this function is to make it possible for the user diagnose the JAGS output (e.g., using the **coda** package) and select appropriate values for n.iter, n.burn.in and n.adapt.

Value

An object of type mcmc.list, as produced by rjags::coda.samples.

Author(s)

Sebastian Jobjörnsson <jobjorns@chalmers.se>

diag.mcmc.list.sim.model

MCMC List For Diagnostic Evaluation

Description

`diag.mcmc.list` method for objects of class `sim.model`.

Usage

```
## S3 method for class 'sim.model'
diag.mcmc.list(model, utility.fun, data,
n.iter, n.burn.in, n.adapt = 1000, n.chains = 1, inits = NULL)
```

Arguments

`model`	Simulation model object of class `sim.model` created using `sim.model` or `create.normal.model`.
`utility.fun`	An R function defining the utility for the decision problem. It must be possible to extract the argument names of the function supplied with `formals`, and the argument names must constitute a subset of the names used in the JAGS model specification.
`data`	A named list of R objects which, when combined with the named objects in `model$data`, leads to a complete specification of the data for the JAGS model.
`n.iter`	The number of iterations in the JAGS MCMC simulation.
`n.burn.in`	The number of burn in iterations prior to the JAGS MCMC simulation.
`n.adapt`	The number of adaptation iterations prior to the burn in phase and subsequent JAGS MCMC simulation.
`n.chains`	The number of parallel MCMC chains to run.
`inits`	The initial values for the chain(s) passed on to the function `rjags::jags.model`, unless equal to its default value `NULL`, in which case JAGS chooses default initial values.

Details

See `diag.mcmc.list` for further documentation.

`eval.eu` *Evaluate Expected Utility*

Description

Simulate an estimate of the expected utility for a specific completion of the JAGS data, i.e., for a specific point in the decision space. This is a generic function for S3 objects.

Usage

```
eval.eu(model, utility.fun, data,
n.iter, n.burn.in, n.adapt = 1000, inits = NULL, independent.SE = FALSE)
```

Arguments

`model`	Simulation model object of class `sim.model` created using `sim.model` or `create.normal.model`.
`utility.fun`	An R function defining the utility for the decision problem. It must be possible to extract the argument names of the function supplied with `formals`, and the argument names must constitute a subset of the names used in the JAGS model specification.
`data`	A named list of R objects which, when combined with the named objects in `model$data`, leads to a complete specification of the data for the JAGS model.
`n.iter`	The number of iterations in the JAGS MCMC simulation.
`n.burn.in`	The number of burn iterations prior to the JAGS MCMC simulation.
`n.adapt`	The number of adaptation iterations prior to the burn in and JAGS MCMC simulation.
`inits`	The initial values for the chain passed on to the function `rjags::jags.model`, unless equal to its default value `NULL`, in which case JAGS chooses default initial values.
`independent.SE`	If `TRUE`, then the standard errors of the sample means used to estimate the expected utility will be computed under the assumption of i.i.d. sampling. If `FALSE`, the standard errors are instead computed using the `coda::spectrum0.ar` function.

Details

This function is called by `eval.on.grid`, but also allows the user to evaluate the expected utility for a specific combination of values for the decision variables of the problem.

Value

A list with components

`mean`	Sample mean of the simulated utilities.
`SE`	Standard error of the sample mean.

Author(s)

Sebastian Jobjörnsson <jobjorns@chalmers.se>

See Also

eval.on.grid

---

eval.eu.sim.model *Evaluate Expected Utility*

Description

eval.eu method for objects of class sim.model.

Usage

```
## S3 method for class 'sim.model'
eval.eu(model, utility.fun, data,
n.iter, n.burn.in, n.adapt = 1000, inits = NULL, independent.SE = FALSE)
```

Arguments

model	Simulation model object of class sim.model created using sim.model or create.normal.model.
utility.fun	An R function defining the utility for the decision problem. It must be possible to extract the argument names of the function supplied with formals, and the argument names must constitute a subset of the names used in the JAGS model specification.
data	A named list of R objects which, when combined with the named objects in model$data, leads to a complete specification of the data for the JAGS model.
n.iter	The number of iterations in the JAGS MCMC simulation.
n.burn.in	The number of burn iterations prior to the JAGS MCMC simulation.
n.adapt	The number of adaptation iterations prior to the burn in and JAGS MCMC simulation.
inits	The initial values for the chain passed on to the function rjags::jags.model, unless equal to its default value NULL, in which case JAGS chooses default initial values.
independent.SE	If TRUE, then the standard errors of the sample means used to estimate the expected utility will be computed under the assumption of i.i.d. sampling. If FALSE, the standard errors are instead computed using the coda::spectrum0.ar function.

Details

See eval.eu for further documentation.

eval.on.grid *Evaluate Expected Utility On A Grid*

Description

Evaluate the expected utility on a grid using MCMC samples from JAGS. The grid is a subset of the space for the decision variables.

Usage

```
eval.on.grid(model, utility.fun, grid.spec.list,
n.iter, n.burn.in, n.adapt = 1000, independent.SE = FALSE, parallel = FALSE)
```

Arguments

`model`	Simulation model object of class `sim.model` created using `sim.model` or `create.normal.model`.
`utility.fun`	An R function defining the utility for the decision problem. It must be possible to extract the argument names of the function supplied with `formals`, and the argument names must constitute a subset of the names used in the JAGS model specification.
`grid.spec.list`	A nonempty list of array grid specifications. An array grid specification is a list of two componenents. The first component is a dimension vector, giving the dimensions of the array. The second component is a list of vectors of length equal to the product of the dimension vector. Each such vector has the form `c(lower, upper, step)`. These are passed to the function `seq` in order to generate a range of values for each component of the array.
`n.iter`	The number of iterations in the JAGS MCMC simulation for each grid point.
`n.burn.in`	The number of burn in iterations prior to the JAGS MCMC simulation for each grid point.
`n.adapt`	The number of adaptation iterations prior to the burn in phase and subsequent JAGS MCMC simulation for each grid point.
`independent.SE`	If `TRUE`, then the standard errors of the sample means used to estimate the expected utility will be computed under the assumption of i.i.d. sampling. If `FALSE`, the standard errors are instead computed using the `coda::spectrum0.ar` function.
`parallel`	Logical equal to `TRUE` if the simulation should be done in parallel on a multi-core processor. The default value `FALSE` leads to single-core evaluation.

Value

A new simulation model object constructed from the object given as the first argument and the simulation results. The updated components in the new object are `model$sim.points`, `model$sim.means`, `model$sim.SEs` and `model$grid.spec.list`. See `sim.model` for a description of these components.

Author(s)

Sebastian Jobjörnsson <jobjorns@chalmers.se>

See Also

eval.eu

---

eval.on.grid.sim.model

Evaluate Expected Utility On A Grid

Description

eval.on.grid method for objects of class sim.model.

Usage

```
## S3 method for class 'sim.model'
eval.on.grid(model, utility.fun, grid.spec.list,
n.iter, n.burn.in, n.adapt = 1000, independent.SE = FALSE, parallel = FALSE)
```

Arguments

model	Simulation model object of class sim.model created using sim.model or create.normal.model.
utility.fun	An R function defining the utility for the decision problem. It must be possible to extract the argument names of the function supplied with formals, and the argument names must constitute a subset of the names used in the JAGS model specification.
grid.spec.list	A nonempty list of array grid specifications. An array grid specification is a list of two componenents. The first component is a dimension vector, giving the dimensions of the array. The second component is a list of vectors of length equal to the product of the dimension vector. Each such vector has the form c(lower, upper, step). These are passed to the function seq in order to generate a range of values for each component of the array.
n.iter	The number of iterations in the JAGS MCMC simulation for each grid point.
n.burn.in	The number of burn in iterations prior to the JAGS MCMC simulation for each grid point.
n.adapt	The number of adaptation iterations prior to the burn in phase and subsequent JAGS MCMC simulation for each grid point.
independent.SE	If TRUE, then the standard errors of the sample means used to estimate the expected utility will be computed under the assumption of i.i.d. sampling. If FALSE, the standard errors are instead computed using the coda::spectrum0.ar function.
parallel	Logical equal to TRUE if the simulation should be done in parallel on a multi-core processor. The default value FALSE leads to single-core evaluation.

Details

See `eval.on.grid` for further documentation.

`fit.gpr` *Fit A Gaussian Process Regression Function*

Description

Fit a GPR regression function to the estimated expected utility values obtained through simulation via JAGS by calling `eval.on.grid`. This is a generic function for S3 objects.

Usage

```
fit.gpr(model, start, gr = TRUE, method = "L-BFGS-B",
lower = 0, upper = Inf, control = list())
```

Arguments

`model`	A model object obtained as the return value from `eval.on.grid`.
`start`	Start value passed on to `optim` when performing the marginal likelihood optimisation to find appropriate values for the hyperparameters for the GPR regression function.
`gr`	Set to `TRUE` if gradient information should be passed to `optim`. If false, `optim` uses a finite difference approximation of the gradient when performing the optimisation of the hyperparameters.
`method`	The optimisation method to be used by `optim`. One of `"Nelder-Mead"`, `"BFGS"`, `"CG"`, `"L-BFGS-B"`, `"SANN"` or `"Brent"`.
`lower`	A numeric, atomic vector containing the lower limits for the hyperparameters. The first entry is for the standard deviation parameter and the remaining entries are for the length parameters. If supplied, all elements must be >= 0.
`upper`	A numeric, atomic vector containing the upper limits for the hyperparameters. The first entry is for the standard deviation parameter and the remaining entries are for the length parameters.
`control`	A list of control parameters passed on to `optim`.

Details

The fitting operation consists of maximising the marginal likelihood of the hyperparameters for a GPR model based on a squared-exponential covariance model. This is done by minimising a function proportional to the negative marginal likelihood. The number of hyperparameters for this model equals 1 + the number of decision variables of the decision model. The first hyperparameter is a standard deviation and the rest consists of a length parameter for each decision dimension.

The optimisation strategy depends on the value of `method`. If `"L-BFGS-B"` is used, then the arguments `lower` and `upper` are passed on as specified to `optim` as the lower and upper limits for the optimisation of the hyperparameters. If any other value is provided for `method`, then `optim` will be used to minimise a function defined to be equal to the objective function when the hyperparameter argument `x` satisfies `x` >= `lower`, `x` <= `upper` and equal to `Inf` otherwise. The actual lower and upper limits passed to `optim` in this latter case are `-Inf` and `Inf`, respectively.

Value

A new simulation model object constructed from the object given as the first argument and the GPR regression results. The updated components in the new object are `model$regression.fun` and `model$gpr.hyper.params`. See `sim.model` for a description of these components.

Author(s)

Sebastian Jobjörnsson `<jobjorns@chalmers.se>`

See Also

`fit.loess`

`fit.gpr.sim.model` *Fit A Gaussian Process Regression Function*

Description

`fit.gpr` method for objects of class `sim.model`.

Usage

```
## S3 method for class 'sim.model'
fit.gpr(model, start, gr = TRUE, method = "L-BFGS-B",
lower = 0, upper = Inf, control = list())
```

Arguments

`model`	A model object obtained as the return value from `eval.on.grid`.
`start`	Start value passed on to `optim` when performing the marginal likelihood optimisation to find appropriate values for the hyperparameters for the GPR regression function.
`gr`	Set to `TRUE` if gradient information should be passed to `optim`. If false, `optim` uses a finite difference approximation of the gradient when performing the optimisation of the hyperparameters.
`method`	The optimisation method to be used by `optim`. One of `"Nelder-Mead"`, `"BFGS"`, `"CG"`, `"L-BFGS-B"`, `"SANN"` or `"Brent"`.
`lower`	A numeric, atomic vector containing the lower limits for the hyperparameters. The first entry is for the standard deviation parameter and the remaining entries are for the length parameters. If supplied, all elements must be >= 0.

`upper`	A numeric, atomic vector containing the upper limits for the hyperparameters. The first entry is for the standard deviation parameter and the remaining entries are for the length parameters.
`control`	A list of control parameters passed on to `optim`.

Details

See `fit.gpr` for further documentation.

`fit.loess` *Fit A Local Polynomial Regression Function*

Description

Fit a local polynomial regression function to the estimated expected utility values obtained through simulation via JAGS by calling `eval.on.grid`. This is a generic function for S3 objects.

Usage

```
fit.loess(model, span = 0.75, degree = 2)
```

Arguments

`model`	A model object obtained as the return value from `eval.on.grid`.
`span`	A parameter which controls the degree of smoothing.
`degree`	The degree of the polynomials to be used, normally 1 or 2.

Details

This function calls `loess` in package **stats** to perform a regression. Note that the number of decision variables must be between 1 and 4, since this is the range supported by `loess`.

The formula passed as `formula` to `loess` has the form `"y ~ x1 + x2"` (for two decision variables, and correspondingly for any other number between 1 and 4). The `span` and `degree` arguments are passed on to `loess` as given. Further, `surface = "direct"` is used as a `loess` control value in order to allow for extrapolation for the fitted function. For the remaining arguments of `loess`, the default values are used.

Value

A new simulation model object constructed from the object given as the first argument and the local polynomial regression results. The updated components in the new object are `model$regression.fun` and `model$gpr.hyper.params` (set to `NA`). See `sim.model` for a description of these components.

Author(s)

Sebastian Jobjörnsson `<jobjorns@chalmers.se>`

See Also

`fit.gpr`

`fit.loess.sim.model` *Fit A Local Polynomial Regression Function*

Description

`fit.loess` method for objects of class `sim.model`.

Usage

```
## S3 method for class 'sim.model'
fit.loess(model, span = 0.75, degree = 2)
```

Arguments

`model`	A model object obtained as the return value from `eval.on.grid`.
`span`	A parameter which controls the degree of smoothing.
`degree`	The degree of the polynomials to be used, normally 1 or 2.

Details

See `fit.loess` for further documentation.

`n.opt` *Optimise A Simple Normal Model*

Description

Find an approximation of the optimal sample size and corresponding expected utility for a simple phase III clinical trial model with a single, normally distributed response and a utility function of a fixed form.

Usage

```
n.opt(nu = 0, tau = 1, sigma = 1, alpha = 0.025,
gain.constant = 1, gain.function = function(X, mu) 0,
fixed.cost = 0, sample.cost = 0.005,
k = 1, n.min = 1, n.max = 50, n.step = 1,
n.iter = 10000, n.burn.in = 1000, n.adapt = 1000,
regression.type = "loess",
plot.results = TRUE, independent.SE = FALSE,
parallel = FALSE, path.to.package = NA)
```

Arguments

`nu`	The mean of the conjugate normal prior distribution for the unknown population mean.
`tau`	The standard deviation of the conjugate normal prior distribution for the unknown population mean.
`sigma`	The known population standard deviation for each individual response in the trial.
`alpha`	The significance level in the one-sided test used by the regulatory authority to decide upon marketing approval for the new treatment.
`gain.constant`	A constant utility gain received upon treatment approval. The total gain consists of the sum of `gain.constant` and `gain.function`.
`gain.function`	A variable utility gain obtained in addition to the constant utility gain upon treatment approval.
`fixed.cost`	The fixed cost of performing the trial.
`sample.cost`	The marginal cost per observation for the trial.
`k`	The number independent, parallel trials. Must be an integer greater than zero.
`n.min`	Lower limit for the one-dimensional grid for the sample size.
`n.max`	Upper limit for the one-dimensional grid for the sample size.
`n.step`	The step size of the grid for the sample size.
`n.iter`	The number of iterations in the JAGS MCMC simulation.
`n.burn.in`	The number of burn iterations prior to the JAGS MCMC simulation.
`n.adapt`	The number of adaptation iterations prior to the burn in and JAGS MCMC simulation.
`regression.type`	If set to `"loess"`, the default value, then local polynomial regression will be used (via a call to `fit.loess`) to fit the grid simulation results. If set to `"gpr"`, GPR regression will be used instead. For any other value, no regression is performed and the optimisation done will consist of a maximisation over the values corresponding to the grid points.
`plot.results`	Set to `TRUE` if a plot of the results of the simulation over the grid is to be constructed.
`independent.SE`	If `TRUE`, then the standard errors of the sample means used to estimate the expected utility will be computed under the assumption of i.i.d. sampling. If `FALSE`, the standard errors are instead computed using the `coda::spectrum0.ar` function.
`parallel`	Set to `TRUE` if the simulations over the grid should be done in parallel on a multi-core processor. The default value `FALSE` leads to single-core computations.
`path.to.package`	The search path to the installation directory of **bdpopt**. For the default value, the function will attempt to find the path using `search`.

Value

A list with components

`ns`	A numeric, atomic vector containing the sample size grid points.
`eus`	A numeric, atomic vector containing the sample means of the simulated expected utilities corresponding to the sample size grid points.
`opt.arg`	The optimal sample size found by maximising the estimated expected utility.
`opt.eu`	The estimated optimal utility corresponding to the optimal sample size found.

Author(s)

Sebastian Jobjörnsson <`jobjorns@chalmers.se`>

See Also

`optimise.eu`

`optimise.eu` *Optimise Expected Utility*

Description

Optimisation of expected utility, either directly over the results of a grid evaluation performed by `eval.on.grid` or by optimisation of the regression function constructed by `fit.gpr` or `fit.loess`. This is a generic function for an S3 object.

Usage

```
optimise.eu(model, start, method = "L-BFGS-B",
lower = -Inf, upper = Inf, control = list())
```

Arguments

`model`	A simulation model object returned by `eval.on.grid`, `fit.gpr` or `fit.loess`. Specifying any method other than `"Grid"` requires that the object has been obtained from `fit.gpr` or `fit.loess`.
`start`	The start value when performing the search for a maximum. Passed on to `optim`.
`method`	The optimisation method to be used. Must be one of `"Nelder-Mead"`, `"BFGS"`, `"CG"`, `"L-BFGS-B"`, `"SANN"`, `"Brent"` or `"Grid"`.
`lower`	A numeric, atomic vector giving the lower limits for the decision variables when performing the maximisation.
`upper`	A numeric, atomic vector giving the upper limits for the decision variables when performing the maximisation.
`control`	A list of control parameters passed on to `optim`.

Details

The optimisation strategy depends on the value of `method`. All arguments except `model` are ignored if the method `"Grid"` is used. If `"L-BFGS-B"` is used, then the arguments `lower` and `upper` are passed on as specified to `optim` as the lower and upper limits for the optimisation of the decision variables. If any other value is provided for `method`, then `optim` will be used to maximise a function defined to be equal to the objective function when the decision variable argument `x` satisfies `x >= lower`, `x <= upper` and equal to `-Inf` otherwise. The actual lower and upper limits passed to `optim` in this last case are `-Inf` and `Inf`, respectively.

Value

A list with components

`opt.arg`	A named vector containing the optimal values for the decision variables.
`opt.eu`	An estimate of the optimal expected utility.

Author(s)

Sebastian Jobjörnsson <jobjorns@chalmers.se>

`optimise.eu.sim.model` *Optimise Expected Utility*

Description

`optimise.eu` method for objects of class `sim.model`.

Usage

```
## S3 method for class 'sim.model'
optimise.eu(model, start, method = "L-BFGS-B",
lower = -Inf, upper = Inf, control = list())
```

Arguments

`model`	A simulation model object returned by `eval.on.grid`, `fit.gpr` or `fit.loess`. Specifying any method other than `"Grid"` requires that the object has been obtained from `fit.gpr` or `fit.loess`.
`start`	The start value when performing the search for a maximum. Passed on to `optim`.
`method`	The optimisation method to be used. Must be one of `"Nelder-Mead"`, `"BFGS"`, `"CG"`, `"L-BFGS-B"`, `"SANN"`, `"Brent"` or `"Grid"`.
`lower`	A numeric, atomic vector giving the lower limits for the decision variables when performing the maximisation.
`upper`	A numeric, atomic vector giving the upper limits for the decision variables when performing the maximisation.
`control`	A list of control parameters passed on to `optim`.

Details

See `optimise.eu` for further documentation.

`optimise.sequential.eu` *Optimise Sequential Expected Utility*

Description

Optimise the expected utility for a sequential decision problem. The optimisation proceeds by backward induction, computing the optimal decision and corresponding expected utility at each stage over a grid for the current state summarising the posterior distribution of the unknown parameter.

Usage

```
optimise.sequential.eu(dp, mins, maxs, steps, n.sims, state.start = NA)
```

Arguments

`dp`	A decision problem object constructed by calling `sequential.dp`.
`mins`	A numeric, atomic vector specifying the lower boundary of the grid points for each dimension of the state vector.
`maxs`	A numeric, atomic vector specifying the upper boundary of the grid points for each dimension of the state vector.
`steps`	A numeric, atomic vector specifying the step size between grid points for each dimension of the state vector.
`n.sims`	A numeric, atomic vector of length equal to `dp$n.stages` specifying the number of simulation draws to perform at each stage.
`state.start`	An optional start value for the state. If provided, this value effectively determines a fixed prior for the parameter of the decision problem (before the first stage). The first stage computations will then only be performed for this particular value. If the default value `NA` is used, computations for the first stage will be performed for all grid points, as for the subsequent stages.

Value

If `state.start` equals `NA`, then the value returned is a list with components

`opt.decision`	Function taking a stage and a state into the optimal decision corresponding to the closest grid point at that stage.
`opt.utility`	Function taking a stage and a state into the optimal expected utility corresponding to the closest grid point at that stage.

If an explicit value for `state.start` is provided, the value returned is a list with components

`opt.stage1.decision`	Optimal stage 1 decision.
`opt.stage1.utility`	Optimal stage 1 utility.
`opt.decision`	Function taking a stage (greater than 1) and a state into the optimal decision corresponding to the closest grid point at that stage.
`opt.utility`	Function taking a stage (greater than 1) and a state into the optimal expected utility corresponding to the closest grid point at that stage.

Author(s)

Sebastian Jobjörnsson `<jobjorns@chalmers.se>`

See Also

`optimise.sequential.normal.eu`

`optimise.sequential.normal.eu` *Optimise A Sequential Normal Decision Problem*

Description

Optimise a sequential normal decision problem constructed by a call to the function `sequential.normal.dp`.

Usage

```
optimise.sequential.normal.eu(dp, range, step.size,
prior.mean = 0, n.sims = 1000, plot.results = TRUE)
```

Arguments

`dp`	A sequential normal decision problem created by a call to `sequential.normal.dp`.
`range`	The range of the one-dimensional grid for the state. All grid points will lie in the interval `c(prior.mean - range / 2, prior.mean + range / 2)`.
`step.size`	The step size between the grid points for the state.
`prior.mean`	Defines the mid-point of the grid for the state.
`n.sims`	The number of random draws for simulations used to estimate the expected utility for a decision at each stage.
`plot.results`	If `TRUE`, construct a plot of the optimal policy.

Details

The plot has the stage number on the x-axis. The y-axis levels of the points (one for each stage) shows the cutoff levels for when it is optimal to continue (for all but the last stage) or finalise (for the last stage). It will be optimal to continue or finalise if the posterior mean of the parameter at a given stage is above the level of the point at that stage.

If no cutoff point can be established in the interior of the region defined by the interval
I = `c(prior.mean - range / 2, prior.mean + range / 2)`, then a cross will be used instead of a circle to indicate the level of the stage. If the cross is located at `prior.mean + range / 2`, it is optimal to continue for no state in I. If the cross is located at `prior.mean - range / 2`, it is optimal to continue for all states in I.

Value

A list with components

`opt.decision`	Function taking a stage and a state into the optimal decision corresponding to the closest grid point at that stage.
`opt.utility`	Function taking a stage and a state into the optimal expected utility corresponding to the closest grid point at that stage.

Author(s)

Sebastian Jobjörnsson `<jobjorns@chalmers.se>`

See Also

`optimise.sequential.eu`

`plot.sim.model` *Plot The Results Contained In Simulation Model Object*

Description

Plot method for an object of class `sim.model`.

Usage

```
## S3 method for class 'sim.model'
plot(x, main.var.name = NULL, main.var.min = -Inf,
  main.var.max = Inf, fixed = list(), no.legends = FALSE,
no.reg = FALSE, reg.steps = 100, ...)
```

Arguments

`x`	A simulation model object obtained as output from `eval.on.grid`, `fit.gpr` or `fit.loess`.
`main.var.name`	A name for the main variable to be used when plotting. This defines the variable that varies along the x-axis of the plot. The full name of the variable must be given as a string, i.e., the name must include array brackets and array indices.
`main.var.min`	The minimum value for the main variable.
`main.var.max`	The maximum value for the main variable.
`fixed`	A list of vectors with named entries. Each such vector defines a set of fixed values for the remaining decision variables. The number of curves in the plot will be equal to the length of `fixed`. Note that the names of the entries must be written in full array notation, including explicit brackets and indices.
`no.legends`	By default, legends are included with numbers corresponding to the entries of `fixed`. Set `no.legends` to `TRUE` to remove it.
`no.reg`	Set to `TRUE` in order so suppress plotting of the regression curves.
`reg.steps`	The number of steps to use when plotting the regression curves.
`...`	Not used.

Details

The simulation grid points included in the plot are selected as follows:

1. All points for which the value of the decision variable defined by `main.var.name` is not within the region defined by `main.var.min` and `main.var.max` are excluded from the total set of grid points.
2. For the points remaining, any point which does not correspond to a value listed in `fixed` is excluded.

In the special case when there is only one decision variable, only the model `x` needs to be specified. In the special case when there are only two decision variables, `fixed` may also be given as a vector. It then specifies the values of the secondary variable and one curve will be drawn for each value.

The default behaviour is to also plot the fitted regression function if it is available, with one curve corresponding to each of the point sets defined by the entries of `fixed`.

Value

Returns `NULL`.

Author(s)

Sebastian Jobjörnsson `<jobjorns@chalmers.se>`

`print.sim.model` *Print Status Of Simulation Model Object*

Description

Print method for an object of class `sim.model`. Outputs some very concise information about the given model object.

Usage

```
## S3 method for class 'sim.model'
print(x, ...)
```

Arguments

`x`	The simulation model object.
`...`	Not used.

Details

Prints the JAGS model used. If a simulation over a grid has been performed, prints the number of simulation points. Prints a message stating whether or not regression has been performed. Prints the hyperparameters of a GPR regression if available.

Value

Returns `NULL`.

Author(s)

Sebastian Jobjörnsson `<jobjorns@chalmers.se>`

`sequential.dp` *Construct A Sequential Decision Problem*

Description

Construct a sequential decision problem. The object constructed is just a list with components named as the arguments to the function.

Usage

```
sequential.dp(n.stages, post.sample, pred.sample, update.state,
term.decisions, term.obs.decisions, cont.decisions,
term.utility.fun, term.obs.utility.fun, cont.utility.fun)
```

Arguments

`n.stages`	The number of stages in the sequential decision problem.
`post.sample`	A function taking a stage, a state and the simulation iteration count into a random sample from the posterior distribution of the model parameter given the value of the state when being at the given stage.
`pred.sample`	A function taking a stage, a list of parameter values and a decision into a list of random samples from the conditional distributions of the observable variable at the given stage. The i:th value in the list returned should be a sample from the predictive distribution conditioned on the i:th entry in the list of parameter values.
`update.state`	A function that takes a stage, a state, a decision `d` and a list of observations into a list of updated states. The updated values should be the ones obtained when combining the observations in order with the original state, given d.
`term.decisions`	A list the length of which must be equal to `n.stages`. The i:th element of the list specifies the terminal decisions available at stage i.
`term.obs.decisions`	A list the length of which must be equal to `n.stages`. The i:th element of the list specifies the terminal observation decisions available at stage i.
`cont.decisions`	A list the length of which must be equal to `n.stages`. The i:th element of the list specifies the continuation decisions available at stage i.
`term.utility.fun`	A list of terminal utility functions of length equal to `n.stages`. Each element of the list should be a function mapping a pair `(d, theta)` into to a numeric value, where `d` is a decision and `theta` is a parameter value.
`term.obs.utility.fun`	A list of terminal utility functions of length equal to `n.stages`. Each element of the list should be a function mapping a triple `(d, X, theta)` into to a numeric value, where `d` is a decision, `X` is an observation and `theta` is a parameter value.
`cont.utility.fun`	A list of terminal utility functions of length equal to `n.stages`. Each element of the list should be a function mapping a pair `(d, X)` into to a numeric value, where `d` is a decision and `X` is an observation.

Details

For any stage i, at least one of the elements of the decision lists must be nonempty, i.e., the sum of `length(term.decisions[[i]])`, `length(term.obs.decisions[[i]])` and `length(cont.decisions[[i]]` must be greater than or equal to 1.

For the last stage, all decisions must be terminal decisions, i.e., `length(cont.decisions[[n.stages]])` must equal 0 and the sum of `length(term.decisions[[n.stages]])` and `length(term.obs.decisions[[n.stages]])` must be greater than or equal to 1.

Value

A list for which the components have the same names and are in the same order as the arguments to the function.

Author(s)

Sebastian Jobjörnsson <`jobjorns@chalmers.se`>

See Also

`sequential.normal.dp`

`sequential.normal.dp` *Create A Sequential Normal Decision Problem*

Description

Create an object representing a sequential normal decision problem. A single observation with a normal distribution is made at each stage. The parameter is a true population mean with a conjugate normal prior. Under the assumption of a known population standard deviation, the variance of the posterior distribution for the parameter does not depend on the observations as is known at each stage. This implies that the state is one-dimensional and equals the mean of the posterior distribution for the parameter at each stage.

Usage

```
sequential.normal.dp(n.stages, group.size, tau, sigma,
stage.cost, final.cost, final.gain)
```

Arguments

`n.stages`	The number of stages of the sequential decision problem.
`group.size`	The sample size at each stage. The individal samples are combined into a group mean, which is the single observation at each stage.
`tau`	The standard deviation of the prior for the unknown population mean before the first stage.
`sigma`	The population standard deviation for a single individual. The standard deviation for the group response is this value divided by the square root of `group.size`.
`stage.cost`	The cost of proceeding to the next stage.
`final.cost`	The cost payed at the final stage if a finalisation decision is taken (if a stopping decision is taken, this cost is not payed).
`final.gain`	A constant which is multiplied with the true population mean in order to obtain the utility gain at the final stage, if a finalisation decision is taken (if a stopping decision is taken, this gain is not included in the total utility).

Details

In all stages but the last, the two decisions available are to either continue and pay the stage cost or to stop and abort (which costs nothing). At the final stage, the two decisions available are to either finalise the process and obtain the final gain and pay the final cost or to stop and abort (whith no gain and no cost).

Value

A list representing a sequential decision problem object. See `sequential.dp` for further description of the components.

Author(s)

Sebastian Jobjörnsson <jobjorns@chalmers.se>

See Also

`sequential.dp`

`sim.model` *Construct A Simulation Model Object*

Description

Construct a simulation model object from a JAGS model file and a JAGS data file or a named list of data objects.

Usage

```
sim.model(model.file, data)
```

Arguments

`model.file`	Name of the file defining the JAGS model. This must be an atomic character vector of a single element.
`data`	Name of the file defining the JAGS data or a named list of R objects. If a file name, it must be an atomic character vector of a single element.

Value

An S3 object of class `sim.model` with components

`model.file`	The name of the JAGS model file used to create the object.
`data`	A named list of data objects extracted from the JAGS data file or directly supplied as an argument.
`grid.spec.list`	The grid specification list after grid evaluation. Initially set to `NA` when the object is created.
`sim.points`	A matrix with with columns holding the positions of the grid points after grid evaluation. Initially set to `NA` when the object is created.
`sim.means`	A vector holding the sample means after grid evaluation. Initially set to `NA` when the object is created.
`sim.SEs`	A vector holding the standard errors corresponding to the sample means after grid evaluation. Initially set to `NA` when the object is created.

`regression.fun`	The approximate, smoothed function after regression has been performed. Initially set to `NA` when the object is created.
`gpr.hyper.params`	The hyperparameters selected in the GPR regression after it has been performed. Initially set to `NA` when the object is created.

Author(s)

Sebastian Jobjörnsson `<jobjorns@chalmers.se>`

See Also

`create.normal.model`

Index