

# tigre

October 5, 2010

---

ExpressionTimeSeries-class

*Class to contain time series expression assays*

---

## Description

Container for time series expression assays and experimental metadata. `ExpressionTimeSeries` class is derived from `ExpressionSet`, and requires fields `experiments` and `modeltime` in `phenoData`.

## Extends

Directly extends class `ExpressionSet`.

## Objects from the Class

```
new("ExpressionTimeSeries")
```

```
new("ExpressionTimeSeries", phenoData = new("AnnotatedDataFrame"),  
featureData = new("AnnotatedDataFrame"), experimentData = new("MIAME"),  
annotation = character(0), protocolData = phenoData[,integer(0)], exprs  
= new("matrix"), var.exprs = new("matrix"))
```

This creates an `ExpressionTimeSeries` with `assayData` implicitly created to contain `exprs` and `var.exprs`.

```
new("ExpressionTimeSeries", assayData = assayDataNew(exprs=new("matrix")),  
phenoData = new("AnnotatedDataFrame"), featureData = new("AnnotatedDataFrame"),  
experimentData = new("MIAME"), annotation = character(0), protocolData  
= phenoData[,integer(0)])
```

This creates an `ExpressionTimeSeries` with `assayData` provided explicitly. In this form, the only required named argument is `assayData`.

`ExpressionTimeSeries` instances are usually created through `new("ExpressionTimeSeries", ...)`. Usually the arguments to `new` include `exprs` (a matrix of expression data, with features corresponding to rows and samples to columns), `var.exprs`, `phenoData`, `featureData`, `experimentData`, `annotation`, and `protocolData`. `phenoData`, `featureData`, `experimentData`, `annotation`, and `protocolData` can be missing, in which case they are assigned default values.

**Slots**

**assayData:** Inherited from [ExpressionSet](#). The models in `gpsim` package assume that `exprs` contains absolute (i.e. non-logarithmic) expression values. The member `var.exprs` may contain variances of the values.

**phenoData:** Inherited from [ExpressionSet](#). The following fields are required: `experiments` which contains integers from 1 to N with measurements from the same biological assay having the same number; `modeltime` which contains observation times in model units.

**featureData:** Inherited from [ExpressionSet](#).

**experimentData:** Inherited from [ExpressionSet](#).

**annotation:** Inherited from [ExpressionSet](#).

**protocolData:** Inherited from [ExpressionSet](#).

**.\_\_classVersion\_\_:** Inherited from [ExpressionSet](#).

**Methods**

See also methods for [ExpressionSet](#).

`var.exprs(object), var.exprs(object) <- value` Access and set `var.exprs`

`initialize("ExpressionTimeSeries")` Object instantiation, used by `new`; not to be called directly by the user.

**Author(s)**

Antti Honkela, Jonatan Ropponen

**See Also**

[processData](#), [processRawData](#).

**Examples**

```
showClass("ExpressionTimeSeries")
```

---

GPLearn

*Fit a GP model*

---

**Description**

Forms an optimized model of the desired genes. The function can form a model with `GPSim` or `GPdisim` and it's also possible to use initial parameters or fix parameters for future use. The genes can also be filtered based on ratios calculated from the expression values. The given data can also be searched for the data of specific genes.

**Usage**

```
model <- GPLearn(preprocData, TF = NULL, targets = NULL,
  useGpdisim = FALSE, randomize = FALSE, addPriors = FALSE,
  fixedParams = FALSE, initParams = NULL, initialZero = TRUE,
  fixComps = NULL, dontOptimise = FALSE,
  allowNegativeSensitivities = FALSE, quiet = FALSE,
  gpsimOptions = NULL, allArgs = NULL)
```

**Arguments**

<code>preprocData</code>	The preprocessed data to be used.
<code>TF</code>	The transcription factor of the model. In GPsim, there can be only one transcription factor for each model.
<code>targets</code>	The target genes of the model.
<code>useGpdisim</code>	A logical value determining whether GPDISIM is used, FALSE (default) implies GPSIM.
<code>randomize</code>	A logical value determining whether the parameters of the model are randomized before optimization.
<code>addPriors</code>	A logical value determining whether priors are added to the model.
<code>fixedParams</code>	A logical value determining whether the initial parameters are fixed.
<code>initParams</code>	The initial parameters for the model. In combination with <code>fixedParams</code> a value NA denotes parameters to learn.
<code>initialZero</code>	Assume a zero initial TF protein concentration, default = TRUE.
<code>fixComps</code>	The blocks of the kernel the parameters of which are to be fixed. To be used together with <code>fixedParams</code> and <code>initParams</code> .
<code>dontOptimise</code>	Just create the model, do not run optimisation.
<code>allowNegativeSensitivities</code>	Allow sensitivities to go negative. This is an experimental feature, and the negative values have no physical interpretation.
<code>quiet</code>	Suppress optimiser output.
<code>gpsimOptions</code>	Internal: additional options to pass to <code>gp[di]simCreate</code> .
<code>allArgs</code>	A list of arguments that can be used to override ones with the same name.

**Value**

Returns the optimized model.

**Author(s)**

Antti Honkela, Pei Gao, Jonatan Ropponen, Magnus Rattray, Neil D. Lawrence

**See Also**

[GPRankTargets](#), [GPRankTFs](#).

**Examples**

```
# Load a mmgmos preprocessed fragment of the Drosophila developmental
# time series
data(drosophila_gpsim_fragment)

# Get the target probe names
library(annotate)
aliasMapping <- getAnnMap("ALIAS2PROBE",
                          annotation(drosophila_gpsim_fragment))
twi <- get('twi', env=aliasMapping)
fbgnMapping <- getAnnMap("FLYBASE2PROBE",
                       annotation(drosophila_gpsim_fragment))
targetProbe <- get('FBgn0035257', env=fbgnMapping)
```

```

# Create the model but do not optimise (rarely needed...)
model <- GPLearn(drosophila_gpsim_fragment,
                 TF=twi, targets=targetProbe,
                 useGpdisim=TRUE, quiet=TRUE,
                 dontOptimise=TRUE)

## Not run:
# Create and learn the model
model <- GPLearn(drosophila_gpsim_fragment,
                 TF=twi, targets=targetProbe,
                 useGpdisim=TRUE, quiet=TRUE)

## End(Not run)

```

---

GPMModel-class      *A container for gpsim models*

---

## Description

The class is a container for the internal representation of models used by the `gpsim` package.

## Objects from the Class

Objects can be created by calls of the form `new("GPMModel", model)`.

## Slots

`model`: A model object used internally by the code of the `gpsim` package  
`type`: Type of the model object

## Methods

`modelStruct(object), modelStruct(object) <- value` Access and set the internal model structure  
`modelType(object)` Access the internal type values  
`show(object)` Informatively display object contents.  
`is.GPMModel(object)` Check if object is a GPMModel.  
`initialize("GPMModel")` Object instantiation, used by `new`; not to be called directly by the user.

## Author(s)

Antti Honkela, Jonatan Ropponen

## See Also

[GPLearn](#), [GPRankTargets](#), [GPRankTFs](#), [generateModels](#), [modelExtractParam](#), [modelLogLikelihood](#)

## Examples

```
showClass("GPMModel")
```

**Description**

Plots GP(DI)SIM models.

**Usage**

```
GPPlot(data, savepath = '', nameMapping = NULL, predt = NULL, fileOutput=FALSE)
```

**Arguments**

data	The model to plot as returned by GPLearn.
savepath	The location in the file system where the images are saved.
nameMapping	The annotation used for mapping the names of the genes for the figures.
predt	The set of time points to use in plotting (default: the time interval covering the data).
fileOutput	Is the plot being saved to a file? If yes, do not open new interactive devices for each plot.

**Details**

The function plots the fitted expression level of the transcription factor (if applicable), the inferred activity of the transcription factor, and the fitted expression level of the target(s).

**Author(s)**

Antti Honkela

**See Also**

[GPLearn](#).

**Examples**

```
## Not run:
# Load a mmgmos preprocessed fragment of the Drosophila developmental
# time series
data(drosophila_gpsim_fragment)

# Get the target probe names
library(annotate)
aliasMapping <- getAnnMap("ALIAS2PROBE",
                          annotation(drosophila_gpsim_fragment))
twi <- get('twi', env=aliasMapping)
fbgnMapping <- getAnnMap("FLYBASE2PROBE",
                        annotation(drosophila_gpsim_fragment))
targetProbe <- get('FBgn0035257', env=fbgnMapping)

# Learn the model
model <- GPLearn(drosophila_gpsim_fragment,
```

```

    TF=twi, targets=targetProbe,
    useGpdisim=TRUE, quiet=TRUE)

# Plot it
GPPlot(model, nameMapping=getAnnMap("FLYBASE",
    annotation(drosophila_gpsim_fragment)))

## End(Not run)

```

---

GPRankTargets

*Ranking possible target genes or regulators*


---

### Description

GPRankTargets ranks possible target genes by forming optimized models with a fixed transcription factor, a set of known target genes and targets to be tested. The transcription factor and the known targets are always included in the models while the tested targets are tested by including them in the models one at a time. The function determines itself whether to use GPSIM or GPDISIM based on the input arguments.

### Usage

```

scores <- GPRankTargets(preprocData, TF = NULL, knownTargets = NULL,
    testTargets = NULL, filterLimit = 1.8,
    returnModels = FALSE, options = NULL,
    scoreSaveFile = NULL)
scores <- GPRankTFs(preprocData, TFs, targets,
    filterLimit = 1.8, returnModels = FALSE, options = NULL,
    scoreSaveFile = NULL)

```

### Arguments

preprocData	The preprocessed data to be used.
TF	The transcription factor present in all models.
knownTargets	The target genes present in all models.
testTargets	Target genes that are tested by including them in the models one at a time. Can be names of genes, or a set of indices in preprocData.
filterLimit	Genes with an average expression z-score above this figure are accepted after filtering. If this value is 0, all genes will be accepted.
returnModels	A logical value determining whether the function returns the calculated models.
options	A list of additional arguments to pass to GPLearn.
scoreSaveFile	Name of file to save the scores to after processing each gene.
TFs	The transcription factors that are tested by including them in the models one at a time.
targets	The target genes present in all models.

**Details**

The models are formed by calling `GPLearn`. If there is no value given to the transcription factor, GPSIM is used. Otherwise, GPDISIM is used. GPSIM needs some known targets. If known targets are given, a model is first created with only the transcription factor and the known targets. The parameters extracted from this model are used as initial parameters of the models with test targets.

GPRankTFs is very similar to GPRankTargets, except it loops over candidate regulators, not candidate targets.

**Value**

The function returns a `scoreList` containing the genes, parameters and log-likelihoods of the models. If `returnModels` is true, the function returns a list of the calculated models.

**Author(s)**

Antti Honkela, Jonatan Ropponen, Magnus Rattray, Neil D. Lawrence

**See Also**

`GPLearn`, `scoreList`, `generateModels`.

**Examples**

```
## Not run:
# Load a mmgmos preprocessed fragment of the Drosophila developmental
# time series
data(drosophila_gpsim_fragment)

# Get the target probe names
targets <- c('FBgn0003486', 'FBgn0033188', 'FBgn0035257')
library(annotate)
aliasMapping <- getAnnMap("ALIAS2PROBE",
                          annotation(drosophila_gpsim_fragment))
twi <- get('twi', env=aliasMapping)
fbgnMapping <- getAnnMap("FLYBASE2PROBE",
                        annotation(drosophila_gpsim_fragment))
targetProbes <- mget(targets, env=fbgnMapping)

scores <- GPRankTargets(drosophila_gpsim_fragment, TF=twi,
                       testTargets=targetProbes,
                       options=list(quiet=TRUE),
                       filterLimit=1.8)

## End(Not run)
```

---

SCGoptim

*Optimise the given function using (scaled) conjugate gradients.*

---

**Description**

Optimise the given function using (scaled) conjugate gradients.

**Usage**

```
options <- optimiDefaultOptions()
newParams <- SCGoptim(x, fn, grad, options, ...)
newParams <- CGoptim(x, fn, grad, options, ...)
model <- modelOptimise(model, options, ...)
```

**Arguments**

model	the model to be optimised.
x	initial parameter values.
fn	objective function to minimise
grad	gradient function of the objective
options	options structure like one returned by <code>optimiDefaultOptions</code> . The fields are interpreted as\ option[1] : number of iterations\ option[2] : interval for the line search\ option[3] : tolerance for x to terminate the loop\ option[4] : tolerance for fn to terminate the loop\ option\$display : option of showing the details of optimisation
...	extra arguments to pass to fn and grad

**Value**

options	an options structure
newParams	optimised parameter values
model	the optimised model.

**See Also**

[modelObjective](#), [modelGradient](#)

**Examples**

```
## Not run to speed up package checks
# model <- GPLearn(..., dontOptimise=TRUE)
# options <- optimiDefaultOptions()
# model <- modelOptimise(model, options)
```

---

drosophila\_gpsim\_fragment

*Fragment of 12 time point Drosophila embryonic development microarray gene expression time series*

---

**Description**

Four genes from the 12 time point Drosophila embryonic development Affymetrix microarray gene expression data set by Tomancak et al. (2002).

The data has been processed using [mmgmos](#) and [processData](#).

**Usage**

```
data(drosophila_gpsim_fragment)
```



**Format**

An `ExpressionTimeSeries` object with 3 repeats of the 12 time points for 4 probes.

**Source**

[ftp://ftp.fruitfly.org/pub/embryo\\_tc\\_array\\_data/](ftp://ftp.fruitfly.org/pub/embryo_tc_array_data/)

**References**

Tomancak, P et al. Systematic determination of patterns of gene expression during *Drosophila* embryogenesis. *Genome Biol* 3:RESEARCH0088, 2002.

---

drosophila\_mmgmos\_fragment

*Fragment of 12 time point Drosophila embryonic development microarray gene expression time series*

---

**Description**

Four genes from the 12 time point *Drosophila* embryonic development Affymetrix microarray gene expression data set by Tomancak et al. (2002).

The data has been processed using `mmgmos`.

**Usage**

```
data(drosophila_mmgmos_fragment)
```

**Format**

An `exprResult` object with 3 repeats of the 12 time points for 4 probes.

**Source**

[ftp://ftp.fruitfly.org/pub/embryo\\_tc\\_array\\_data/](ftp://ftp.fruitfly.org/pub/embryo_tc_array_data/)

**References**

Tomancak, P et al. Systematic determination of patterns of gene expression during *Drosophila* embryogenesis. *Genome Biol* 3:RESEARCH0088, 2002.

---

expTransform            *Constrains a parameter.*

---

### Description

contains commands to constrain parameters to be positive via exponentiation or within a fixed interval via the sigmoid function.

### Usage

```
expTransform(x, transform)
sigmoidTransform(x, transform)
boundedTransform(x, transform, bounds)
```

### Arguments

x	input argument.
transform	type of transform, 'atox' maps a value into the transformed space (i.e. makes it positive). 'xtoa' maps the parameter back from transformed space to the original space. 'gradfact' gives the factor needed to correct gradients with respect to the transformed parameter.
bounds	a 2-vector of bounds of allowed values in boundedTransform

### Value

Return value as selected by transform

### See Also

[modelOptimise](#)

### Examples

```
# Transform unconstrained parameter -4 to a positive value
expTransform(-4, 'atox')

# Transform a bounded parameter in (1,3) to an unconstrained one
boundedTransform(2, 'xtoa', c(1, 3))
```

---

generateModels            *Generating models with the given data*

---

### Description

'generateModels' recreates models based on the parameters stored in a scoreList.

### Usage

```
models <- generateModels(preprocData, scores)
```

**Arguments**

preprocData    The preprocessed data to be used.  
scores         A scoreList object containing data of the models to be generated.

**Value**

'generateModels' returns a list of the generated models.

**Author(s)**

Antti Honkela, Jonatan Ropponen

**See Also**

[GPLearn](#), [GPRankTargets](#), [GPRankTFs](#), [scoreList](#).

**Examples**

```
## Not run:  
# Load a mmgmos preprocessed fragment of the Drosophila developmental  
# time series  
data(drosophila_gpsim_fragment)  
  
# Get the target probe names  
targets <- c('FBgn0003486', 'FBgn0033188', 'FBgn0035257')  
library(annotate)  
aliasMapping <- getAnnMap("ALIAS2PROBE",  
                          annotation(drosophila_gpsim_fragment))  
twi <- get('twi', env=aliasMapping)  
fbgnMapping <- getAnnMap("FLYBASE2PROBE",  
                          annotation(drosophila_gpsim_fragment))  
targetProbes <- mget(targets, env=fbgnMapping)  
  
scores <- GPRankTargets(drosophila_gpsim_fragment, TF=twi,  
                          testTargets=targetProbes,  
                          options=list(quiet=TRUE),  
                          filterLimit=1.8)  
  
models <- generateModels(drosophila_gpsim_fragment, scores)  
  
## End(Not run)
```

---

gpsimCreate

*Create a GPSIM/GPDISIM model.*

---

**Description**

creates a model for single input motifs with Gaussian processes.

**Usage**

```

model <- gpsimCreate(numGenes, numProteins, times, geneVals,
                    geneVars, options, genes)
model <- gpdisimCreate(numGenes, numProteins, times, geneVals,
                      geneVars, options, genes)

```

**Arguments**

numGenes	number of genes to be modelled in the system.
numProteins	number of proteins to be modelled in the system.
times	the time points where the data is to be modelled.
geneVals	the values of each gene at the different time points.
geneVars	the varuabces of each gene at the different time points.
options	options structure (optional).
genes	names of the probes the model is for

**Details**

These functions are meant to be used through [GPLearn](#).

**Value**

model                    model structure containing default parameterisation.

**See Also**

[modelExtractParam](#), [modelOptimise](#), [GPLearn](#).

**Examples**

```
## missing, see GPLearn
```

---

kernCompute	<i>Compute the kernel given the parameters and X.</i>
-------------	---

---

**Description**

Compute the kernel given the parameters and X.

**Usage**

```

K <- kernCompute(kern, X)
K <- kernCompute(kern, X1, X2)
Kd <- kernDiagCompute(kern, X)

```

**Arguments**

kern	kernel structure to be computed.
X	input data matrix (rows are data points) to the kernel computation.
X1	first input matrix to the kernel computation (forms the rows of the kernel).
X2	second input matrix to the kernel computation (forms the columns of the kernel).

**Details**

`K <- kernCompute(kern, X)` computes a kernel matrix for the given kernel type given an input data matrix.

`K <- kernCompute(kern, X1, X2)` computes a kernel matrix for the given kernel type given two input data matrices, one for the rows and one for the columns.

`K <- kernDiagCompute(kern, X)` computes the diagonal of a kernel matrix for the given kernel.

`K <- *X*kernCompute(kern1, kern2, X)` `K <- *X*kernCompute(kern1, kern2, X1, X2)` same as above, but for cross combinations of two kernels, `kern1` and `kern2`.

**Value**

`K` computed elements of the kernel structure.

`Kd` vector containing computed diagonal elements of the kernel structure.

**See Also**

[kernCreate](#)

**Examples**

```
kern <- kernCreate(1, 'rbf')
K <- kernCompute(kern, as.matrix(3:8))
```

---

kernCreate	<i>Initialise a kernel structure.</i>
------------	---------------------------------------

---

**Description**

Initialise a kernel structure.

**Usage**

```
kern <- kernCreate(X, type)
kern <- kernCreate(dim, type)
```

**Arguments**

`X` Input data values (from which kernel will later be computed).

`type` Type of kernel to be created, some standard types are 'rbf', 'white', 'sim' and 'disim'. If a list of the form `list(type='compnd', comp=c('rbf', 'white'))` is used a compound kernel based on the sum of the individual kernels will be created. Parameters can be passed to kernels using type `list(type='parametric', options=list(opt=val), realType=...)`, where `realType` is the type that would be used otherwise.

`dim` input dimension of the design matrix (i.e. number of features in the design matrix).

**Details**

`kern <- kernCreate(X, type)` input points and a kernel type.

`kern <- kernCreate(dim, type)` creates a kernel matrix structure given the dimensions of the design matrix and the kernel type.

The `*KernParamInit` functions perform initialisation specific to different types of kernels. They should not be called directly.

**Value**

`kern`                    The kernel structure.

**See Also**

[kernDisplay](#), [modelTieParam](#).

**Examples**

```
# Create a multi kernel with two rbf blocks with bounded inverse widths
invWidthBounds <- c(0.5, 2)
kernType <- list(type="multi", comp=list())
for (i in 1:2)
  kernType$comp[[i]] <- list(type="parametric", realType="rbf",
                             options=list(isNormalised=TRUE,
                                           inverseWidthBounds=invWidthBounds))
kern <- kernCreate(1, kernType)

# Tie the inverse with parameters of the component RBF kernels
kern <- modelTieParam(kern, list(tieWidth="inverseWidth"))
kernDisplay(kern)
```

---

`kernDiagGradX`

*Compute the gradient of the kernel wrt X.*

---

**Description**

computes the gradient of the (diagonal of the) kernel matrix with respect to the elements of the design matrix given in X.

**Usage**

```
gX <- kernDiagGradX(kern, X)
gX2 <- kernGradX(kern, X)
gX2 <- kernGradX(kern, X1, X2)
```

**Arguments**

`kern`                    the kernel structure for which gradients are being computed.  
`X`                        the input data in the form of a design matrix.  
`X1`                      row locations against which gradients are being computed.  
`X2`                      column locations against which gradients are being computed.

**Value**

gX	the gradients of the diagonal with respect to each element of X. The returned matrix has the same dimensions as X.
gX2	the returned gradients. The gradients are returned in a matrix which is numData x numInputs x numData. Where numData is the number of data points and numInputs is the number of input dimensions in X.

**See Also**

[kernGradient](#)

**Examples**

```
kern <- kernCreate(1, 'mlp')
g <- kernDiagGradX(kern, as.matrix(3:8))
```

---

kernGradient	<i>Compute the gradient wrt the kernel parameters.</i>
--------------	--

---

**Description**

Compute the gradient wrt the kernel parameters.

**Usage**

```
g <- kernGradient(kern, x, partial)
g <- kernGradient(kern, x1, x2, partial_)
```

**Arguments**

kern	the kernel structure for which the gradients are being computed.
x	the input locations for which the gradients are being computed.
partial	matrix of partial derivatives of the function of interest with respect to the kernel matrix. The argument takes the form of a square matrix of dimension numData, where numData is the number of rows in X.
x1	the input locations associated with the rows of the kernel matrix.
x2	the input locations associated with the columns of the kernel matrix.
partial_	matrix of partial derivatives of the function of interest with respect to the kernel matrix. The matrix should have the same number of rows as X1 and the same number of columns as X2 has rows.

**Details**

```
g <- kernGradient(kern, x, partial) g <- *kernGradient(kern, x, partial)
```

computes the gradient of functions with respect to the kernel parameters. As well as the kernel structure and the input positions, the user provides a matrix PARTIAL which gives the partial derivatives of the function with respect to the relevant elements of the kernel matrix.

```
g <- kernGradient(kern, x1, x2, partial_) g <- *kernGradient(kern, x1, x2, partial_)
```

computes the derivatives as above, but input locations are now provided in two matrices associated with rows and columns of the kernel matrix.

```
g <- *X*kernGradient(kern1, kern2, x, partial) g <- *X*kernGradient(kern1,
kern2, x1, x2, partial_) same as above, but for cross combinations of two kernels,
kern1 and kern2.
```

**Value**

`g` gradients of the function of interest with respect to the kernel parameters. The ordering of the vector should match that provided by the function `kernExtractParam`.

**See Also**

[kernCompute](#), [kernExtractParam](#).

**Examples**

```
kern <- kernCreate(1, 'rbf')
g <- kernGradient(kern, as.matrix(c(1, 4)), array(1, c(2, 2)))
```

---

lnDiffErf

*Helper function for computing the log of difference*

---

**Description**

Helper function for computing the log of difference

**Usage**

```
v <- lnDiffErf(x1, x2)
```

**Arguments**

`x1` argument of the positive erf  
`x2` argument of the negative erf

**Details**

`v <- lnDiffErf(x1, x2)` computes the log of the difference of two erfs in a numerically stable manner.

**Value**

`v` `list(c(log(abs(erf(x1) - erf(x2))), sign(erf(x1) - erf(x2))))`

**Examples**

```
lnDiffErf(100, 10)
```



---

modelDisplay	<i>Display a model.</i>
--------------	-------------------------

---

**Description**

displays the parameters of the model/kernel and the model/kernel type to the console.

**Usage**

```
model <- modelDisplay(model)
```

**Arguments**

model                    the model/kernel structure to be displayed.

**See Also**

[modelExtractParam](#)

**Examples**

```
# Load a mmgmos preprocessed fragment of the Drosophila developmental
# time series
data(drosophila_gpsim_fragment)

# The probe identifier for TF 'twi'
twi <- "143396_at"
# The probe identifier for the target gene
targetProbe <- "152715_at"

# Create the model, but do not optimise
model <- GPLearn(drosophila_gpsim_fragment,
                 TF=twi, targets=targetProbe,
                 useGpdisim=TRUE, quiet=TRUE,
                 dontOptimise=TRUE)

# Display the initial model
modelDisplay(model)
```

---

modelExpandParam	<i>Update a model structure with new parameters or update the posterior processes.</i>
------------------	--

---

**Description**

Update a model structure or component with new parameters, or update the posterior processes.

**Usage**

```
model <- modelExpandParam(model, param)
model <- modelUpdateProcesses(model)
```

**Arguments**

model            the model structure to be updated.  
 param            vector of parameters.

**Details**

model <- modelExpandParam(model, param) returns a model structure filled with the parameters in the given vector. This is used as a helper function to enable parameters to be optimised in, for example, the optimisation functions.

model <- modelUpdateProcesses(model) updates posterior processes of the given model.

**Value**

model            updated model structure.

**See Also**

[GPLearn](#), [modelExtractParam](#)

**Examples**

```
## Not run:
# Learn the model
model <- GPLearn(...)
params <- modelExtractParam(model, only.values=TRUE)
params[1] <- 0
new_model <- modelExpandParam(model, params)
new_model <- modelUpdateProcesses(new_model)

## End(Not run)
```

---

modelExtractParam *Extract the parameters of a model.*

---

**Description**

Extract parameters from the model into a vector of parameters for optimisation.

**Usage**

```
param <- modelExtractParam(model, only.values=TRUE)
```

**Arguments**

model            the model structure containing the parameters to be extracted.  
 only.values     include parameter names in the returned vector.

**Value**

param            vector of parameters extracted from the model.

**See Also**[modelExpandParam](#)**Examples**

```

# Load a mmgmos preprocessed fragment of the Drosophila developmental
# time series
data(drosophila_gpsim_fragment)

# The probe identifier for TF 'twi'
twi <- "143396_at"
# The probe identifier for the target gene
targetProbe <- "152715_at"

# Create the model, but do not optimise
model <- GPLearn(drosophila_gpsim_fragment,
                 TF=twi, targets=targetProbe,
                 useGpdisim=TRUE, quiet=TRUE,
                 dontOptimise=TRUE)

# Get the initial parameter values
params <- modelExtractParam(model, only.values=FALSE)

```

modelGradient

*Model log-likelihood/objective error function and its gradient.***Description**

modeGradient gives the gradient of the objective function for a model. By default the objective function (modelObjective) is a negative log likelihood (modelLogLikelihood).

**Usage**

```

v <- modelObjective(model)
ll <- modelLogLikelihood(model)
g <- modelGradient(params, model, ...)

```

**Arguments**

params	parameter vector to evaluate at.
model	model structure.
...	optional additional arguments.

**Value**

g	the gradient of the error function to be minimised.
v	the objective function value (lower is better).
ll	the log-likelihood value.

**See Also**[modelOptimise](#).

**Examples**

```

# Load a mmgmos preprocessed fragment of the Drosophila developmental
# time series
data(drosophila_gpsim_fragment)

# The probe identifier for TF 'twi'
twi <- "143396_at"
# The probe identifier for the target gene
targetProbe <- "152715_at"

# Create the model but do not optimise
model <- GPLearn(drosophila_gpsim_fragment,
                 TF=twi, targets=targetProbe,
                 useGpdisim=TRUE, quiet=TRUE,
                 dontOptimise=TRUE)
params <- modelExtractParam(model, only.values=FALSE)
ll <- modelLogLikelihood(model)
paramValues <- modelExtractParam(model, only.values=TRUE)
modelGradient(paramValues, model)

```

---

modelTieParam	<i>Tie parameters of a model together.</i>
---------------	--

---

**Description**

groups of parameters of a model to be seen as one parameter during optimisation of the model.

**Usage**

```
model <- modelTieParam(model, paramsList)
```

**Arguments**

model	the model for which parameters are being tied together.
paramsList	indices of parameteres to group together. The indices are provided in a list. Each element in the list contains a vector of indices of parameters that should be considered as one parameter. Each group of parameters in each cell should obviously be mutually exclusive.  Alternatively, the specification may consist of strings, which are interpreted as regular expressions that are matched against the parameter names returned by <code>modelExtractParam</code> or <code>kernExtractParam</code> , as appropriate fot the current object.

**Value**

model	the model with the parameters grouped together.
-------	---

**See Also**

[modelExtractParam](#), [modelExpandParam](#), [modelGradient](#).

**Examples**

```
# Create a multi kernel with two rbf blocks with bounded inverse widths
invWidthBounds <- c(0.5, 2)
kernType <- list(type="multi", comp=list())
for (i in 1:2)
  kernType$comp[[i]] <- list(type="parametric", realType="rbf",
                             options=list(isNormalised=TRUE,
                                           inverseWidthBounds=invWidthBounds))
kern <- kernCreate(1, kernType)

# Tie the inverse with parameters of the component RBF kernels
kern <- modelTieParam(kern, list(tieWidth="inverseWidth"))
kernDisplay(kern)
```

---

```
optimiDefaultConstraint
```

*Returns function for parameter constraint.*

---

**Description**

returns the current default function for constraining a parameter.

**Usage**

```
val <- optimiDefaultConstraint(constraint)
```

**Arguments**

`constraint` the type of constraint you want to place on the parameter, options include 'positive' (gives an 'exp' constraint) and 'zeroone' (gives a 'sigmoid' constraint).

**Value**

`val` a list with two components: 'func' for the name of function used to apply the constraint, and 'hasArgs' for a boolean flag if the function requires additional arguments.

**See Also**

[expTransform](#), [sigmoidTransform](#).

**Examples**

```
optimiDefaultConstraint('positive')
optimiDefaultConstraint('bounded')
```

---

processData                      *Processing expression time series*

---

### Description

processData further processes time series data preprocessed by mmgmos.

processData further processes time series data preprocessed by mmgmos.

Both functions return [ExpressionTimeSeries](#) objects that can be used as input for the functions [GPLearn](#) and [GPRankTargets](#).

### Usage

```
preprocData <- processData(data, times = NULL, experiments = NULL,
  do.normalisation = TRUE)
preprocData <- processRawData(rawData, times, experiments = NULL,
  is.logged = TRUE, do.normalisation = ifelse(is.logged, TRUE, FALSE))
```

### Arguments

data	The preprocessed data ( <a href="#">exprResult</a> ) from mmgMOS to be used.
rawData	Raw data matrix to be used. Each row corresponds to a gene and each column to a data point.
times	Observation times of each data point. If unspecified or NULL, processData attempts to infer this from phenoData(data) field containing 'time' in the name.
experiments	The replicate structure of the data indicating which expression data points arise from which experiments. This should be an array in integers from 1 to N with length equal to the number of data points. By default all the data points are assumed to be from same replicate.
is.logged	Indicates whether the expression values are on log scale or not. Normalisation of non-logged data is unsupported.
do.normalisation	Indicates whether to perform the normalisation.

### Details

The expression data (and percentiles, if available) are normalized by equalising the mean. In processData, a normal distribution is then fitted into the data with distfit.

### Value

An [ExpressionTimeSeries](#) object containing all provided information.

### Author(s)

Antti Honkela, Jonatan Ropponen

### See Also

[GPLearn](#), [GPRankTargets](#).

**Examples**

```
## Load a mmgmos preprocessed fragment of the Drosophila developmental
## time series
data(drosophila_mmgmos_fragment)

## Process the data (3 experiments containing 12 time points each)
drosophila_gpsim_fragment <- processData(drosophila_mmgmos_fragment,
  experiments=rep(1:3, each=12))
```

---

```
scoreList-class      Class "scoreList"
```

---

**Description**

'scoreList' is an object which contain the genes, parameters, log-likelihoods and arguments of models. With the data in a scoreList item and the original data used for creating the models, the models can be reconstructed with the function 'generateModels'.

**Objects from the Class**

Objects can be created by calls of the form `scoreList(params, loglikelihoods, genes, modelArgs, knownTargets, TF, sharedModel)`.

**Slots**

**params:** The parameters of the models.  
**loglikelihoods:** The log-likelihoods of the models.  
**baseloglikelihoods:** The log-likelihoods of corresponding null models.  
**genes:** The genes used in the models.  
**modelArgs:** A list of arguments used to generate the models.  
**knownTargets:** The list of known targets used in the ranking.  
**TF:** The TF used in the ranking.  
**sharedModel:** Shared model for known targets.

**Methods**

Class-specific methods:

```
write.scores(object, ...) Writes the log-likelihoods and null log-likelihoods. Accepts
  any options write.table does.
genes(object), genes(object) <- value Access and set genes
knownTargets(object), knownTargets(object) <- value Access and set knownTargets
loglikelihoods(object), loglikelihoods(object) <- value Access and set loglikelihoods
baseloglikelihoods(object), baseloglikelihoods(object) <- value Access
  and set baseloglikelihoods
modelArgs(object), modelArgs(object) <- value Access and set modelArgs
params(object), params(object) <- value Access and set params
sharedModel(object), sharedModel(object) <- value Access and set sharedModel
```

`TF(object), TF(object) <- value` Access and set TF

Standard generic methods:

`object[(index)` Conducts subsetting of the `scoreList`.

`c(object, ...)` Concatenates `scoreLists`.

`length(object)` Returns the length of the list.

`show(object)` Informatively display object contents.

`sort(object, decreasing=FALSE)` Sort the list according to log-likelihood

### Author(s)

Antti Honkela, Jonatan Ropponen

### See Also

[GPRankTargets](#), [GPRankTFs](#), [generateModels](#), [write.table](#).

### Examples

```
showClass("scoreList")
```

---

tigre-package

*tigre - Transcription factor Inference through Gaussian process Reconstruction of Expression*

---

### Description

This package implements the method of Gao et al. (2008) and Honkela et al. (2010) for Gaussian process modelling single input motif regulatory systems with time-series expression data. The method can be used to rank potential targets of transcription factors based on such data.

### Details

Package: tigre  
Type: Package  
Version: 1.0.0  
Date: 2010-04-22  
License: A-GPL Version 3

For details of using the package please refer to the Vignette.

### Author(s)

Antti Honkela, Pei Gao, Jonatan Ropponen, Magnus Rattray, Neil D. Lawrence

Maintainer: Antti Honkela <antti.honkela@tkk.fi>



## References

P.-Gao, A.-Honkela, M.-Ratray, and N.-D.-Lawrence. Gaussian process modelling of latent chemical species: applications to inferring transcription factor activities. *Bioinformatics* 24(16):i70–i75, 2008. DOI: 10.1093/bioinformatics/btn278.

A.-Honkela, C.-Girardot, E.-H. Gustafson, Y.-H. Liu, E.-E.-M. Furlong, N.-D. Lawrence, and M.-Ratray. Model-based method for transcription factor target identification with limited data. *Proc Natl Acad Sci U S A*, Apr 2010. DOI: 10.1073/pnas.0914285107.

## See Also

[puma](#)

## Examples

```
## Not run:
# Load a mmgmos preprocessed fragment of the Drosophila developmental
# time series
data(drosophila_gpsim_fragment)

# Get the target probe names
library(annotate)
aliasMapping <- getAnnMap("ALIAS2PROBE",
                          annotation(drosophila_gpsim_fragment))
twi <- get('twi', env=aliasMapping)
fbgnMapping <- getAnnMap("FLYBASE2PROBE",
                        annotation(drosophila_gpsim_fragment))
targetProbe <- get('FBgn0035257', env=fbgnMapping)

# Learn the model
model <- GPLearn(drosophila_gpsim_fragment,
                 TF=twi, targets=targetProbe,
                 useGpdisim=TRUE, quiet=TRUE)

# Plot it
GPPlot(model, nameMapping=getAnnMap("FLYBASE",
                                    annotation(drosophila_gpsim_fragment)))

## End(Not run)
```

# Index

## \*Topic classes

ExpressionTimeSeries-class, 1  
 GPModel-class, 4  
 scoreList-class, 23

## \*Topic datasets

drosophila\_gpsim\_fragment, 8  
 drosophila\_mmgmos\_fragment, 9

## \*Topic model

expTransform, 10  
 generateModels, 10  
 GPLearn, 2  
 GPPlot, 5  
 GPRankTargets, 6  
 gpsimCreate, 11  
 kernCompute, 12  
 kernCreate, 13  
 kernDiagGradX, 14  
 kernGradient, 15  
 lnDiffErfcs, 16  
 modelDisplay, 17  
 modelExpandParam, 17  
 modelExtractParam, 18  
 modelGradient, 19  
 modelTieParam, 20  
 optimiDefaultConstraint, 21  
 processData, 22  
 SCGoptim, 7

## \*Topic package

tigre-package, 24

[, scoreList-method  
 (scoreList-class), 23

baseloglikelihoods  
 (scoreList-class), 23

baseloglikelihoods, scoreList-method  
 (scoreList-class), 23

baseloglikelihoods<-  
 (scoreList-class), 23

baseloglikelihoods<-, scoreList, numeric-method  
 (scoreList-class), 23

boundedTransform (expTransform),  
 10

c, scoreList-method

(scoreList-class), 23

CGoptim (SCGoptim), 7

cgpdisimExpandParam  
 (modelExpandParam), 17

cgpdisimExtractParam  
 (modelExtractParam), 18

cgpdisimGradient (modelGradient),  
 19

cgpdisimLogLikeGradients  
 (modelGradient), 19

cgpdisimLogLikelihood  
 (modelGradient), 19

cgpdisimObjective  
 (modelGradient), 19

cgpdisimUpdateProcesses  
 (modelExpandParam), 17

cgpsimExpandParam  
 (modelExpandParam), 17

cgpsimExtractParam  
 (modelExtractParam), 18

cgpsimGradient (modelGradient), 19

cgpsimLogLikeGradients  
 (modelGradient), 19

cgpsimLogLikelihood  
 (modelGradient), 19

cgpsimObjective (modelGradient),  
 19

cgpsimOptimise (SCGoptim), 7

cgpsimUpdateProcesses  
 (modelExpandParam), 17

cmpndKernCompute (kernCompute), 12

cmpndKernDiagCompute  
 (kernCompute), 12

cmpndKernDiagGradX  
 (kernDiagGradX), 14

cmpndKernDisplay (modelDisplay),  
 17

cmpndKernExpandParam  
 (modelExpandParam), 17

cmpndKernExtractParam  
 (modelExtractParam), 18

cmpndKernGradient (kernGradient),  
 15

- cmpndKernGradX (*kernDiagGradX*), 14
- cmpndKernParamInit (*kernCreate*), 13
- disimKernCompute (*kernCompute*), 12
- disimKernDiagCompute (*kernCompute*), 12
- disimKernDisplay (*modelDisplay*), 17
- disimKernExpandParam (*modelExpandParam*), 17
- disimKernExtractParam (*modelExtractParam*), 18
- disimKernGradient (*kernGradient*), 15
- disimKernParamInit (*kernCreate*), 13
- disimXdisimKernCompute (*kernCompute*), 12
- disimXdisimKernGradient (*kernGradient*), 15
- disimXrbfKernCompute (*kernCompute*), 12
- disimXrbfKernGradient (*kernGradient*), 15
- disimXsimKernCompute (*kernCompute*), 12
- disimXsimKernGradient (*kernGradient*), 15
- drosophila\_gpsim\_fragment, 8
- drosophila\_mmgnos\_fragment, 9
- ExpressionSet, 1, 2
- ExpressionTimeSeries, 9, 22
- ExpressionTimeSeries-class, 1
- exprReslt, 9, 22
- expTransform, 10, 21
- generateModels, 4, 7, 10, 24
- genes (*scoreList-class*), 23
- genes, *scoreList*-method (*scoreList-class*), 23
- genes<- (*scoreList-class*), 23
- genes<- , *scoreList*, *list*-method (*scoreList-class*), 23
- gpdisimCreate (*gpsimCreate*), 11
- gpdisimDisplay (*modelDisplay*), 17
- gpdisimExpandParam (*modelExpandParam*), 17
- gpdisimExtractParam (*modelExtractParam*), 18
- gpdisimGradient (*modelGradient*), 19
- gpdisimLogLikeGradients (*modelGradient*), 19
- gpdisimLogLikelihood (*modelGradient*), 19
- gpdisimObjective (*modelGradient*), 19
- gpdisimUpdateProcesses (*modelExpandParam*), 17
- GPLearn, 2, 4, 5, 7, 11, 12, 18, 22
- GPMModel-class, 4
- GPPlot, 5
- GPRankTargets, 3, 4, 6, 11, 22, 24
- GPRankTFs, 3, 4, 11, 24
- GPRankTFs (*GPRankTargets*), 6
- gpsimCreate, 11
- gpsimDisplay (*modelDisplay*), 17
- gpsimExpandParam (*modelExpandParam*), 17
- gpsimExtractParam (*modelExtractParam*), 18
- gpsimGradient (*modelGradient*), 19
- gpsimLogLikeGradients (*modelGradient*), 19
- gpsimLogLikelihood (*modelGradient*), 19
- gpsimObjective (*modelGradient*), 19
- gpsimUpdateProcesses (*modelExpandParam*), 17
- initialize, *ExpressionTimeSeries*-method (*ExpressionTimeSeries-class*), 1
- initialize, *GPMModel*-method (*GPMModel-class*), 4
- is.GPMModel (*GPMModel-class*), 4
- is.GPMModel, *GPMModel*-method (*GPMModel-class*), 4
- kernCompute, 12, 16
- kernCreate, 13, 13
- kernDiagCompute (*kernCompute*), 12
- kernDiagGradX, 14
- kernDisplay, 14
- kernDisplay (*modelDisplay*), 17
- kernExpandParam (*modelExpandParam*), 17
- kernExtractParam, 16
- kernExtractParam (*modelExtractParam*), 18
- kernGradient, 15, 15
- kernGradX (*kernDiagGradX*), 14
- kernParamInit (*kernCreate*), 13
- knownTargets (*scoreList-class*), 23

- knownTargets, scoreList-method  
     (scoreList-class), 23  
 knownTargets<- (scoreList-class),  
     23  
 knownTargets<- , scoreList, character-method  
     (scoreList-class), 23  
  
 length, scoreList-method  
     (scoreList-class), 23  
 lnDiffErf, 16  
 loglikelihoods (scoreList-class),  
     23  
 loglikelihoods, scoreList-method  
     (scoreList-class), 23  
 loglikelihoods<-  
     (scoreList-class), 23  
 loglikelihoods<- , scoreList, numeric-method  
     (scoreList-class), 23  
  
 mlpKernCompute (kernCompute), 12  
 mlpKernDiagGradX (kernDiagGradX),  
     14  
 mlpKernExpandParam  
     (modelExpandParam), 17  
 mlpKernExtractParam  
     (modelExtractParam), 18  
 mlpKernGradient (kernGradient), 15  
 mlpKernGradX (kernDiagGradX), 14  
 mlpKernParamInit (kernCreate), 13  
 mmgmos, 8, 9  
 modelArgs (scoreList-class), 23  
 modelArgs, scoreList-method  
     (scoreList-class), 23  
 modelArgs<- (scoreList-class), 23  
 modelArgs<- , scoreList, list-method  
     (scoreList-class), 23  
 modelDisplay, 17  
 modelExpandParam, 17, 19, 20  
 modelExtractParam, 4, 12, 17, 18, 18, 20  
 modelGradient, 8, 19, 20  
 modelLogLikelihood, 4  
 modelLogLikelihood  
     (modelGradient), 19  
 modelObjective, 8  
 modelObjective (modelGradient), 19  
 modelOptimise, 10, 12, 19  
 modelOptimise (SCGoptim), 7  
 modelStruct (GPModel-class), 4  
 modelStruct, GPModel-method  
     (GPModel-class), 4  
 modelStruct<- (GPModel-class), 4  
 modelStruct<- , GPModel, list-method  
     (GPModel-class), 4  
  
 modelTieParam, 14, 20  
 modelType (GPModel-class), 4  
 modelType, GPModel-method  
     (GPModel-class), 4  
 modelUpdateProcesses  
     (modelExpandParam), 17  
 multiKernCompute (kernCompute), 12  
 multiKernDiagCompute  
     (kernCompute), 12  
 multiKernDisplay (modelDisplay),  
     17  
 multiKernExpandParam  
     (modelExpandParam), 17  
 multiKernExtractParam  
     (modelExtractParam), 18  
 multiKernGradient (kernGradient),  
     15  
 multiKernParamInit (kernCreate),  
     13  
  
 optimiDefaultConstraint, 21  
 optimiDefaultOptions (SCGoptim), 7  
  
 params (scoreList-class), 23  
 params, scoreList-method  
     (scoreList-class), 23  
 params<- (scoreList-class), 23  
 params<- , scoreList, list-method  
     (scoreList-class), 23  
 processData, 2, 8, 22  
 processRawData, 2  
 processRawData (processData), 22  
 puma, 25  
  
 rbfKernCompute (kernCompute), 12  
 rbfKernDiagCompute (kernCompute),  
     12  
 rbfKernDisplay (modelDisplay), 17  
 rbfKernExpandParam  
     (modelExpandParam), 17  
 rbfKernExtractParam  
     (modelExtractParam), 18  
 rbfKernGradient (kernGradient), 15  
 rbfKernParamInit (kernCreate), 13  
  
 SCGoptim, 7  
 scoreList, 7, 11  
 scoreList-class, 23  
 sharedModel (scoreList-class), 23  
 sharedModel, scoreList-method  
     (scoreList-class), 23  
 sharedModel<- (scoreList-class),  
     23

- sharedModel<-, scoreList, list-method  
(scoreList-class), 23
- show, GPModel-method  
(GPModel-class), 4
- show, scoreList-method  
(scoreList-class), 23
- sigmoidTransform, 21
- sigmoidTransform(expTransform),  
10
- simKernCompute(kernCompute), 12
- simKernDiagCompute(kernCompute),  
12
- simKernDisplay(modelDisplay), 17
- simKernExpandParam  
(modelExpandParam), 17
- simKernExtractParam  
(modelExtractParam), 18
- simKernGradient(kernGradient), 15
- simKernParamInit(kernCreate), 13
- simXrbfKernCompute(kernCompute),  
12
- simXrbfKernGradient  
(kernGradient), 15
- simXsimKernCompute(kernCompute),  
12
- simXsimKernGradient  
(kernGradient), 15
- sort, scoreList-method  
(scoreList-class), 23
  
- TF(scoreList-class), 23
- TF, scoreList-method  
(scoreList-class), 23
- TF<-(scoreList-class), 23
- TF<-, scoreList, character-method  
(scoreList-class), 23
- tigre(tigre-package), 24
- tigre-package, 24
- translateKernCompute  
(kernCompute), 12
- translateKernDiagCompute  
(kernCompute), 12
- translateKernExpandParam  
(modelExpandParam), 17
- translateKernExtractParam  
(modelExtractParam), 18
- translateKernGradient  
(kernGradient), 15
- translateKernParamInit  
(kernCreate), 13
  
- var.exprs  
(ExpressionTimeSeries-class),  
1
- var.exprs, ExpressionTimeSeries-method  
(ExpressionTimeSeries-class),  
1
- var.exprs<-  
(ExpressionTimeSeries-class),  
1
- var.exprs<-, ExpressionTimeSeries-method  
(ExpressionTimeSeries-class),  
1
  
- whiteKernCompute(kernCompute), 12
- whiteKernDiagCompute  
(kernCompute), 12
- whiteKernDisplay(modelDisplay),  
17
- whiteKernExpandParam  
(modelExpandParam), 17
- whiteKernExtractParam  
(modelExtractParam), 18
- whiteKernGradient(kernGradient),  
15
- whiteKernParamInit(kernCreate),  
13
- whiteXwhiteKernCompute  
(kernCompute), 12
- whiteXwhiteKernGradient  
(kernGradient), 15
- write.scores(scoreList-class), 23
- write.scores, scoreList-method  
(scoreList-class), 23
- write.table, 24