

# Package ‘Cyclops’

October 31, 2023

**Type** Package

**Title** Cyclic Coordinate Descent for Logistic, Poisson and Survival Analysis

**Version** 3.4.0

**Description** This model fitting tool incorporates cyclic coordinate descent and majorization-minimization approaches to fit a variety of regression models found in large-scale observational healthcare data. Implementations focus on computational optimization and fine-scale parallelization to yield efficient inference in massive datasets. Please see:  
Suchard, Simpson, Zorych, Ryan and Madigan (2013) <[doi:10.1145/2414416.2414791](https://doi.org/10.1145/2414416.2414791)>.

**License** Apache License 2.0

**LazyData** Yes

**Biarch** true

**URL** <https://github.com/ohdsi/cyclops>

**BugReports** <https://github.com/ohdsi/cyclops/issues>

**Depends** R (>= 3.5.0)

**Imports** rlang,  
Matrix,  
Rcpp (>= 0.12.12),  
Andromeda (>= 0.3.1),  
dplyr,  
methods,  
survival,  
bit64

**LinkingTo** Rcpp,  
RcppEigen (>= 0.3.2)

**Suggests** testthat,  
readr,  
MASS,  
gnm,  
ggplot2,  
microbenchmark,  
cmprsk

**NeedsCompilation** yes

**RoxygenNote** 7.2.3

**Roxygen** list(markdown = TRUE)

**Encoding** UTF-8

**R topics documented:**

coef.cyclopsFit . . . . .	3
confint.cyclopsFit . . . . .	3
convertToCyclopsData . . . . .	4
convertToTimeVaryingCoef . . . . .	7
coverage . . . . .	7
createAutoGridCrossValidationControl . . . . .	8
createControl . . . . .	9
createCyclopsData . . . . .	11
createNonSeparablePrior . . . . .	13
createParameterizedPrior . . . . .	14
createPrior . . . . .	14
createWeightBasedSearchControl . . . . .	16
cyclops . . . . .	16
fitCyclopsModel . . . . .	17
fitCyclopsSimulation . . . . .	18
getCovariateIds . . . . .	19
getCovariateTypes . . . . .	19
getCyclopsProfileLogLikelihood . . . . .	20
getFineGrayWeights . . . . .	20
getFloatingPointSize . . . . .	21
getHyperParameter . . . . .	21
getNumberOfCovariates . . . . .	22
getNumberOfRows . . . . .	22
getNumberOfStrata . . . . .	23
getUnivariableCorrelation . . . . .	23
getUnivariableSeparability . . . . .	24
isInitialized . . . . .	24
logLik.cyclopsFit . . . . .	25
meanLinearPredictor . . . . .	25
mse . . . . .	26
Multitype . . . . .	26
oxford . . . . .	27
predict.cyclopsFit . . . . .	27
print.cyclopsData . . . . .	28
print.cyclopsFit . . . . .	28
readCyclopsData . . . . .	29
simulateCyclopsData . . . . .	30
splitTime . . . . .	31
summary.cyclopsData . . . . .	32
survfit.cyclopsFit . . . . .	32
vcov.cyclopsFit . . . . .	33

---

coef.cyclopsFit	<i>Extract model coefficients</i>
-----------------	-----------------------------------

---

**Description**

coef.cyclopsFit extracts model coefficients from an Cyclops model fit object

**Usage**

```
## S3 method for class 'cyclopsFit'
coef(object, rescale = FALSE, ignoreConvergence = FALSE, ...)
```

**Arguments**

object	Cyclops model fit object
rescale	Boolean: rescale coefficients for unnormalized covariate values
ignoreConvergence	Boolean: return coefficients even if fit object did not converge
...	Other arguments

**Value**

Named numeric vector of model coefficients.

---

confint.cyclopsFit	<i>Confidence intervals for Cyclops model parameters</i>
--------------------	--

---

**Description**

confint.cyclopsFit profiles the data likelihood to construct confidence intervals of arbitrary level. Usually it only makes sense to do this for variables that have not been regularized.

**Usage**

```
## S3 method for class 'cyclopsFit'
confint(
  object,
  parm,
  level = 0.95,
  overrideNoRegularization = FALSE,
  includePenalty = TRUE,
  rescale = FALSE,
  ...
)
```

**Arguments**

object	A fitted Cyclops model object
parm	A specification of which parameters require confidence intervals, either a vector of numbers of covariateId names
level	Numeric: confidence level required
overrideNoRegularization	Logical: Enable confidence interval estimation for regularized parameters
includePenalty	Logical: Include regularized covariate penalty in profile
rescale	Boolean: rescale coefficients for unnormalized covariate values
...	Additional argument(s) for methods

**Value**

A matrix with columns reporting lower and upper confidence limits for each parameter. These columns are labelled as  $(1 - \text{level}) / 2$  and  $1 - (1 - \text{level}) / 2$  in percent (by default 2.5 percent and 97.5 percent)

**Examples**

```
#Generate some simulated data:
sim <- simulateCyclopsData(nstrata = 1, nrows = 1000, ncovars = 2, eCovarsPerRow = 0.5,
                           model = "poisson")
cyclopsData <- convertToCyclopsData(sim$outcomes, sim$covariates, modelType = "pr",
                                    addIntercept = TRUE)

#Define the prior and control objects to use cross-validation for finding the
#optimal hyperparameter:
prior <- createPrior("laplace", exclude = 0, useCrossValidation = TRUE)
control <- createControl(cvType = "auto", noiseLevel = "quiet")

#Fit the model
fit <- fitCyclopsModel(cyclopsData, prior = prior, control = control)

#Find out what the optimal hyperparameter was:
getHyperParameter(fit)

#Extract the current log-likelihood, and coefficients
logLik(fit)
coef(fit)

#We can only retrieve the confidence interval for unregularized coefficients:
confint(fit, c(0))
```

---

convertToCyclopsData *Convert data from two data frames or ffd objects into a CyclopsData object*

---

**Description**

convertToCyclopsData loads data from two data frames or ffd objects, and inserts it into a Cyclops data object.

**Usage**

```

convertToCyclopsData(
  outcomes,
  covariates,
  modelType = "lr",
  addIntercept = TRUE,
  checkSorting = NULL,
  checkRowIds = TRUE,
  normalize = NULL,
  quiet = FALSE,
  floatingPoint = 64
)

## S3 method for class 'data.frame'
convertToCyclopsData(
  outcomes,
  covariates,
  modelType = "lr",
  addIntercept = TRUE,
  checkSorting = NULL,
  checkRowIds = TRUE,
  normalize = NULL,
  quiet = FALSE,
  floatingPoint = 64
)

## S3 method for class 'tbl_dbi'
convertToCyclopsData(
  outcomes,
  covariates,
  modelType = "lr",
  addIntercept = TRUE,
  checkSorting = NULL,
  checkRowIds = TRUE,
  normalize = NULL,
  quiet = FALSE,
  floatingPoint = 64
)

```

**Arguments**

outcomes	A data frame or ffdi object containing the outcomes with predefined columns (see below).
covariates	A data frame or ffdi object containing the covariates with predefined columns (see below).
modelType	Cyclops model type. Current supported types are "pr", "cpr", "lr", "clr", or "cox"
addIntercept	Add an intercept to the model?
checkSorting	(DEPRECATED) Check if the data are sorted appropriately, and if not, sort.
checkRowIds	Check if all rowIds in the covariates appear in the outcomes.
normalize	String: Name of normalization for all non-indicator covariates (possible values: stdev, max, median)

quiet                    If true, (warning) messages are suppressed.  
floatingPoint        Specified floating-point representation size (32 or 64)

### Details

These columns are expected in the outcome object:

stratumId	(integer)	(optional) Stratum ID for conditional regression models
rowId	(integer)	Row ID is used to link multiple covariates (x) to a single outcome (y)
y	(real)	The outcome variable
time	(real)	For models that use time (e.g. Poisson or Cox regression) this contains time (e.g. number of days)
weights	(real)	(optional) Non-negative weights to apply to outcome
weights	(real)	(optional) Non-negative censoring weights for competing risk model; will be computed if n

These columns are expected in the covariates object:

stratumId	(integer)	(optional) Stratum ID for conditional regression models
rowId	(integer)	Row ID is used to link multiple covariates (x) to a single outcome (y)
covariateId	(integer)	A numeric identifier of a covariate
covariateValue	(real)	The value of the specified covariate

### Value

An object of type `cyclopsData`

### Methods (by class)

- `convertToCyclopsData(data.frame)`: Convert data from two `data.frame`
- `convertToCyclopsData(tbl_dbi)`: Convert data from two Andromeda tables

### Examples

```
#Convert infert dataset to Cyclops format:
covariates <- data.frame(stratumId = rep(infert$stratum, 2),
                        rowId = rep(1:nrow(infert), 2),
                        covariateId = rep(1:2, each = nrow(infert)),
                        covariateValue = c(infert$spontaneous, infert$induced))
outcomes <- data.frame(stratumId = infert$stratum,
                       rowId = 1:nrow(infert),
                       y = infert$case)

#Make sparse:
covariates <- covariates[covariates$covariateValue != 0, ]

#Create Cyclops data object:
cyclopsData <- convertToCyclopsData(outcomes, covariates, modelType = "clr",
                                   addIntercept = FALSE)

#Fit model:
fit <- fitCyclopsModel(cyclopsData, prior = createPrior("none"))
```

---

`convertToTimeVaryingCoef`

*Convert short sparse covariate table to long sparse covariate table for time-varying coefficients.*

---

### Description

`convertToTimeVaryingCoef` convert short sparse covariate table to long sparse covariate table for time-varying coefficients.

### Usage

```
convertToTimeVaryingCoef(shortCov, longOut, timeVaryCoefId)
```

### Arguments

`shortCov` A data frame containing the covariate with predefined columns (see below).  
`longOut` A data frame containing the outcomes with predefined columns (see below), output of `splitTime`.  
`timeVaryCoefId` Integer: A numeric identifier of a time-varying coefficient

### Details

These columns are expected in the `shortCov` object:

<code>rowId</code>	(integer)	Row ID is used to link multiple covariates (x) to a single outcome (y)
<code>covariateId</code>	(integer)	A numeric identifier of a covariate
<code>covariateValue</code>	(real)	The value of the specified covariate

These columns are expected in the `longOut` object:

<code>stratumId</code>	(integer)	Stratum ID for time-varying models
<code>subjectId</code>	(integer)	Subject ID is used to link multiple covariates (x) at different time intervals to a single subject
<code>rowId</code>	(integer)	Row ID is used to link multiple covariates (x) to a single outcome (y)
<code>y</code>	(real)	The outcome variable
<code>time</code>	(real)	For models that use time (e.g. Poisson or Cox regression) this contains time (e.g. number of days)

### Value

A long sparse covariate table for time-varying coefficients.

---

`coverage`

*Coverage*

---

**Description**

coverage computes the coverage on confidence intervals

**Usage**

```
coverage(goldStandard, lowerBounds, upperBounds)
```

**Arguments**

goldStandard	Numeric vector
lowerBounds	Numeric vector. Lower bound of the confidence intervals
upperBounds	Numeric vector. Upper bound of the confidence intervals

**Value**

The proportion of times goldStandard falls between lowerBound and upperBound

---

```
createAutoGridCrossValidationControl
```

*Create a Cyclops control object that supports multiple hyperparameters*

---

**Description**

createCrossValidationControl creates a Cyclops control object for use with [fitCyclopsModel](#) that supports multiple hyperparameters through an auto-search in one dimension and a grid-search over the remaining dimensions

**Usage**

```
createAutoGridCrossValidationControl(
  outerGrid,
  autoPosition = 1,
  refitAtMaximum = TRUE,
  cvType = "auto",
  initialValue = 1,
  ...
)
```

**Arguments**

outerGrid	List or data.frame of grid parameters to explore
autoPosition	Vector position for auto-search parameter (concatenated into outerGrid)
refitAtMaximum	Logical: re-fit Cyclops object at maximal cross-validation parameters
cvType	Must equal "auto"
initialValue	Initial value for auto-search parameter
...	Additional parameters passed through to <a href="#">createControl</a>



**Value**

A Cyclops prior object of class inheriting from "cyclopsPrior" and "cyclopsFunctionalPrior" for use with fitCyclopsModel.

---

createControl	<i>Create a Cyclops control object</i>
---------------	--

---

**Description**

createControl creates a Cyclops control object for use with [fitCyclopsModel](#).

**Usage**

```
createControl(
  maxIterations = 1000,
  tolerance = 1e-06,
  convergenceType = "gradient",
  cvType = "auto",
  fold = 10,
  lowerLimit = 0.01,
  upperLimit = 20,
  gridSteps = 10,
  cvRepetitions = 1,
  minCVData = 100,
  noiseLevel = "silent",
  threads = 1,
  seed = NULL,
  resetCoefficients = FALSE,
  startingVariance = -1,
  useKKTswindle = FALSE,
  tuneSwindle = 10,
  selectorType = "auto",
  initialBound = 2,
  maxBoundCount = 5,
  algorithm = "ccd"
)
```

**Arguments**

maxIterations	Integer: maximum iterations of Cyclops to attempt before returning a failed-to-converge error
tolerance	Numeric: maximum relative change in convergence criterion from successive iterations to achieve convergence
convergenceType	String: name of convergence criterion to employ (described in more detail below)
cvType	String: name of cross validation search. Option "auto" selects an auto-search following BBR. Option "grid" selects a grid-search cross validation
fold	Numeric: Number of random folds to employ in cross validation

lowerLimit	Numeric: Lower prior variance limit for grid-search
upperLimit	Numeric: Upper prior variance limit for grid-search
gridSteps	Numeric: Number of steps in grid-search
cvRepetitions	Numeric: Number of repetitions of X-fold cross validation
minCVData	Numeric: Minimum number of data for cross validation
noiseLevel	String: level of Cyclops screen output ("silent", "quiet", "noisy")
threads	Numeric: Specify number of CPU threads to employ in cross-validation; default = 1 (auto = -1)
seed	Numeric: Specify random number generator seed. A null value sets seed via <a href="#">Sys.time</a> .
resetCoefficients	Logical: Reset all coefficients to 0 between model fits under cross-validation
startingVariance	Numeric: Starting variance for auto-search cross-validation; default = -1 (use estimate based on data)
useKKTswindle	Logical: Use the Karush-Kuhn-Tucker conditions to limit search
tuneSwindle	Numeric: Size multiplier for active set
selectorType	String: name of exchangeable sampling unit. Option "byPid" selects entire strata. Option "byRow" selects single rows. If set to "auto", "byRow" will be used for all models except conditional models where the average number of rows per stratum is smaller than the number of strata.
initialBound	Numeric: Starting trust-region size
maxBoundCount	Numeric: Maximum number of tries to decrease initial trust-region size
algorithm	String: name of fitting algorithm to employ; default is ccd Todo: Describe convergence types

## Value

A Cyclops control object of class inheriting from "cyclopsControl" for use with [fitCyclopsModel](#).

## Examples

```
#Generate some simulated data:
sim <- simulateCyclopsData(nstrata = 1, nrows = 1000, ncovars = 2, eCovarsPerRow = 0.5,
                          model = "poisson")
cyclopsData <- convertToCyclopsData(sim$outcomes, sim$covariates, modelType = "pr",
                                   addIntercept = TRUE)

#Define the prior and control objects to use cross-validation for finding the
#optimal hyperparameter:
prior <- createPrior("laplace", exclude = 0, useCrossValidation = TRUE)
control <- createControl(cvType = "auto", noiseLevel = "quiet")

#Fit the model
fit <- fitCyclopsModel(cyclopsData, prior = prior, control = control)

#Find out what the optimal hyperparameter was:
getHyperParameter(fit)

#Extract the current log-likelihood, and coefficients
```

```

logLik(fit)
coef(fit)

#We can only retrieve the confidence interval for unregularized coefficients:
confint(fit, c(0))

```

---

createCyclopsData      *Create a Cyclops data object*

---

## Description

createCyclopsData creates a Cyclops data object from an R formula or data matrices.

## Usage

```

createCyclopsData(
  formula,
  sparseFormula,
  indicatorFormula,
  modelType,
  data,
  subset = NULL,
  weights = NULL,
  censorWeights = NULL,
  offset = NULL,
  time = NULL,
  pid = NULL,
  y = NULL,
  type = NULL,
  dx = NULL,
  sx = NULL,
  ix = NULL,
  model = FALSE,
  normalize = NULL,
  floatingPoint = 64,
  method = "cyclops.fit"
)

```

## Arguments

formula	An object of class " <a href="#">formula</a> " that provides a symbolic description of the numerically dense model response and terms.
sparseFormula	An object of class " <a href="#">formula</a> " that provides a symbolic description of numerically sparse model terms.
indicatorFormula	An object of class " <a href="#">formula</a> " that provides a symbolic description of {0,1} model terms.
modelType	character string: Valid types are listed below.
data	An optional data frame, list or environment containing the variables in the model.
subset	Currently unused

weights	Currently unused
sensorWeights	Vector of subject-specific censoring weights (between 0 and 1). Currently only supported in modelType = "fgr".
offset	Currently unused
time	Currently undocumented
pid	Optional vector of integer stratum identifiers. If supplied, all rows must be sorted by increasing identifiers
y	Currently undocumented
type	Currently undocumented
dx	Optional dense "Matrix" of covariates
sx	Optional sparse "Matrix" of covariates
ix	Optional {0,1} "Matrix" of covariates
model	Currently undocumented
normalize	String: Name of normalization for all non-indicator covariates (possible values: stdev, max, median)
floatingPoint	Integer: Floating-point representation size (32 or 64)
method	Currently undocumented

## Details

This function creates a Cyclops model data object from R "formula" or directly from numeric vectors and matrices to define the model response and covariates. If specifying a model using a "formula", then the left-hand side define the model response and the right-hand side defines dense covariate terms. Objects provided with "sparseFormula" and "indicatorFormula" must be include left-hand side responses and terms are coerced into sparse and indicator representations for computational efficiency.

Items to discuss:

- Only use formula or (y,dx,...)
- stratum() in formula
- offset() in formula
- when "stratum" (renamed from pid) are necessary
- when "time" are necessary

## Value

A list that contains a Cyclops model data object pointer and an operation duration

## Models

Currently supported model types are:

"ls"	Least squares
"pr"	Poisson regression
"lr"	Logistic regression
"clr"	Conditional logistic regression
"cpr"	Conditional Poisson regression
"sccs"	Self-controlled case series
"cox"	Cox proportional hazards regression
"fgr"	Fine-Gray proportional subdistribution hazards regression

## Examples

```
## Dobson (1990) Page 93: Randomized Controlled Trial :
counts <- c(18, 17, 15, 20, 10, 20, 25, 13, 12)
outcome <- gl(3, 1, 9)
treatment <- gl(3, 3)
cyclopsData <- createCyclopsData(
  counts ~ outcome + treatment,
  modelType = "pr")
cyclopsFit <- fitCyclopsModel(cyclopsData)

cyclopsData2 <- createCyclopsData(
  counts ~ outcome,
  indicatorFormula = ~ treatment,
  modelType = "pr")
summary(cyclopsData2)
cyclopsFit2 <- fitCyclopsModel(cyclopsData2)
```

---

createNonSeparablePrior

*Create a Cyclops prior object that returns the MLE of non-separable coefficients*

---

## Description

createNonSeparablePrior creates a Cyclops prior object for use with [fitCyclopsModel](#).

## Usage

```
createNonSeparablePrior(maxIterations = 10, ...)
```

## Arguments

maxIterations Numeric: maximum iterations to achieve convergence  
... Additional argument(s) for [fitCyclopsModel](#)

## Value

A Cyclops prior object of class inheriting from "cyclopsPrior" for use with [fitCyclopsModel](#).

## Examples

```
prior <- createNonSeparablePrior()
```

---

```
createParameterizedPrior
```

*Create a Cyclops parameterized prior object*

---

### Description

`createParameterizedPrior` creates a Cyclops prior object for use with `fitCyclopsModel` in which arbitrary R functions parameterize the prior location and variance.

### Usage

```
createParameterizedPrior(
  priorType,
  parameterize,
  values,
  useCrossValidation = FALSE,
  forceIntercept = FALSE
)
```

### Arguments

<code>priorType</code>	Character vector: specifies prior distribution. See below for options
<code>parameterize</code>	Function list: parameterizes location and variance
<code>values</code>	Numeric vector: initial parameter values
<code>useCrossValidation</code>	Logical: Perform cross-validation to determine parameters.
<code>forceIntercept</code>	Logical: Force intercept coefficient into prior

### Value

A Cyclops prior object of class inheriting from "cyclopsPrior" and "cyclopsFunctionalPrior" for use with `fitCyclopsModel`.

---

```
createPrior
```

*Create a Cyclops prior object*

---

### Description

`createPrior` creates a Cyclops prior object for use with `fitCyclopsModel`.

### Usage

```
createPrior(
  priorType,
  variance = 1,
  exclude = c(),
  graph = NULL,
  neighborhood = NULL,
  useCrossValidation = FALSE,
  forceIntercept = FALSE
)
```

**Arguments**

priorType	Character: specifies prior distribution. See below for options
variance	Numeric: prior distribution variance
exclude	A vector of numbers or covariateId names to exclude from prior
graph	Child-to-parent mapping for a hierarchical prior
neighborhood	A list of first-order neighborhoods for a partially fused prior
useCrossValidation	Logical: Perform cross-validation to determine prior variance.
forceIntercept	Logical: Force intercept coefficient into prior

**Value**

A Cyclops prior object of class inheriting from "cyclopsPrior" for use with fitCyclopsModel.

**Prior types**

We specify all priors in terms of their variance parameters. Similar fitting tools for regularized regression often parameterize the Laplace distribution in terms of a rate "lambda" per observation. See "glmnet", for example.

$$\text{variance} = 2 * (\text{nobs} * \text{lambda})^2 \text{ or } \text{lambda} = \text{sqrt}(2 / \text{variance}) / \text{nobs}$$

**Examples**

```
#Generate some simulated data:
sim <- simulateCyclopsData(nstrata = 1, nrows = 1000, ncovars = 2, eCovarsPerRow = 0.5,
                           model = "poisson")
cyclopsData <- convertToCyclopsData(sim$outcomes, sim$covariates, modelType = "pr",
                                   addIntercept = TRUE)

#Define the prior and control objects to use cross-validation for finding the
#optimal hyperparameter:
prior <- createPrior("laplace", exclude = 0, useCrossValidation = TRUE)
control <- createControl(cvType = "auto", noiseLevel = "quiet")

#Fit the model
fit <- fitCyclopsModel(cyclopsData, prior = prior, control = control)

#Find out what the optimal hyperparameter was:
getHyperParameter(fit)

#Extract the current log-likelihood, and coefficients
logLik(fit)
coef(fit)

#We can only retrieve the confidence interval for unregularized coefficients:
confint(fit, c(0))
```

---

```
createWeightBasedSearchControl
```

*Create a Cyclops control object that supports in- / out-of-sample hyperparameter search using weights*

---

### Description

`createWeightBasedSearchControl` creates a Cyclops control object for use with `fitCyclopsModel` that supports hyperparameter optimization through an auto-search where `weight = 1` identifies in-sample observations and `weight = 0` identifies out-of-sample observations.

### Usage

```
createWeightBasedSearchControl(cvType = "auto", initialValue = 1, ...)
```

### Arguments

<code>cvType</code>	Must equal "auto"
<code>initialValue</code>	Initial value for auto-search parameter
<code>...</code>	Additional parameters passed through to <code>createControl</code>

### Value

A Cyclops prior object of class inheriting from "cyclopsControl" for use with `fitCyclopsModel`.

---

```
cyclops
```

*Cyclops: Cyclic coordinate descent for logistic, Poisson and survival analysis*

---

### Description

The Cyclops package incorporates cyclic coordinate descent and majorization-minimization approaches to fit a variety of regression models found in large-scale observational healthcare data. Implementations focus on computational optimization and fine-scale parallelization to yield efficient inference in massive datasets.



---

fitCyclopsModel	<i>Fit a Cyclops model</i>
-----------------	----------------------------

---

**Description**

fitCyclopsModel fits a Cyclops model data object

**Usage**

```
fitCyclopsModel(
  cyclopsData,
  prior = createPrior("none"),
  control = createControl(),
  weights = NULL,
  forceNewObject = FALSE,
  returnEstimates = TRUE,
  startingCoefficients = NULL,
  fixedCoefficients = NULL,
  warnings = TRUE,
  computeDevice = "native"
)
```

**Arguments**

cyclopsData	A Cyclops data object
prior	A prior object. More details are given below.
control	A "cyclopsControl" object constructed by <a href="#">createControl</a>
weights	Vector of 0/1 weights for each data row
forceNewObject	Logical, forces the construction of a new Cyclops model fit object
returnEstimates	Logical, return regression coefficient estimates in Cyclops model fit object
startingCoefficients	Vector of starting values for optimization
fixedCoefficients	Vector of booleans indicating if coefficient should be fix
warnings	Logical, report regularization warnings
computeDevice	String: Name of compute device to employ; defaults to "native" C++ on CPU

**Details**

This function performs numerical optimization to fit a Cyclops model data object.

**Value**

A list that contains a Cyclops model fit object pointer and an operation duration

**Prior**

Currently supported prior types are:

"none"	Useful for finding MLE
"laplace"	L_1 regularization
"normal"	L_2 regularization

## References

Suchard MA, Simpson SE, Zorych I, Ryan P, Madigan D. Massive parallelization of serial inference algorithms for complex generalized linear models. *ACM Transactions on Modeling and Computer Simulation*, 23, 10, 2013.

Simpson SE, Madigan D, Zorych I, Schuemie M, Ryan PB, Suchard MA. Multiple self-controlled case series for large-scale longitudinal observational databases. *Biometrics*, 69, 893-902, 2013.

Mittal S, Madigan D, Burd RS, Suchard MA. High-dimensional, massive sample-size Cox proportional hazards regression for survival analysis. *Biostatistics*, 15, 207-221, 2014.

## Examples

```
## Dobson (1990) Page 93: Randomized Controlled Trial :
counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- gl(3,1,9)
treatment <- gl(3,3)
cyclopsData <- createCyclopsData(counts ~ outcome + treatment, modelType = "pr")
cyclopsFit <- fitCyclopsModel(cyclopsData, prior = createPrior("none"))
coef(cyclopsFit)
confint(cyclopsFit, c("outcome2", "treatment3"))
predict(cyclopsFit)
```

---

fitCyclopsSimulation *Fit simulated data*

---

## Description

fitCyclopsSimulation fits simulated Cyclops data using Cyclops or a standard routine. This function is useful for simulation studies comparing the performance of Cyclops when considering large, sparse datasets.

## Usage

```
fitCyclopsSimulation(
  sim,
  useCyclops = TRUE,
  model = "logistic",
  coverage = TRUE,
  includePenalty = FALSE
)
```

**Arguments**

sim	A simulated Cyclops dataset generated via simulateCyclopsData
useCyclops	Logical: use Cyclops or a standard routine
model	String: Fitted regression model type
coverage	Logical: report coverage statistics
includePenalty	Logical: include regularized regression penalty in computing profile likelihood based confidence intervals

---

getCovariateIds	<i>Get covariate identifiers</i>
-----------------	----------------------------------

---

**Description**

getCovariateIds returns a vector of integer64 covariate identifiers in a Cyclops data object

**Usage**

```
getCovariateIds(object)
```

**Arguments**

object	A Cyclops data object
--------	-----------------------

---

getCovariateTypes	<i>Get covariate types</i>
-------------------	----------------------------

---

**Description**

getCovariateTypes returns a vector covariate types in a Cyclops data object

**Usage**

```
getCovariateTypes(object, covariateLabel)
```

**Arguments**

object	A Cyclops data object
covariateLabel	Integer vector: covariate identifiers to return

---

```
getCyclopsProfileLogLikelihood
```

*Profile likelihood for Cyclops model parameters*

---

### Description

getCyclopsProfileLogLikelihood evaluates the profile likelihood at a grid of parameter values.

### Usage

```
getCyclopsProfileLogLikelihood(
  object,
  parm,
  x = NULL,
  bounds = NULL,
  tolerance = 0.001,
  initialGridSize = 10,
  includePenalty = TRUE
)
```

### Arguments

object	Fitted Cyclops model object
parm	Specification of which parameter requires profiling, either a vector of numbers of covariateId names
x	Vector of values of the parameter
bounds	Pair of values to bound adaptive profiling
tolerance	Absolute tolerance allowed for adaptive profiling
initialGridSize	Initial grid size for adaptive profiling
includePenalty	Logical: Include regularized covariate penalty in profile

### Value

A data frame containing the profile log likelihood. Returns NULL when the adaptive profiling fails to converge.

---

```
getFineGrayWeights
```

*Creates a Surv object that forces in competing risks and the IPCW needed for Fine-Gray estimation.*

---

### Description

getFineGrayWeights creates a list Surv object and vector of weights required for estimation.

### Usage

```
getFineGrayWeights(ftime, fstatus, cencode = 0, failcode = 1)
```

**Arguments**

ftime	Numeric: Observed event (failure) times
fstatus	Numeric: Observed event (failure) types
cencode	Numeric: Code to denote censored observations (Default is 0)
failcode	Numeric: Code to denote event of interest (Default is 1)

**Value**

A list that returns both an object of class Surv that forces in the competing risks indicators and a vector of weights needed for parameter estimation.

**Examples**

```
ftime <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
fstatus <- c(1, 2, 0, 1, 2, 0, 1, 2, 0, 1)
getFineGrayWeights(ftime, fstatus, cencode = 0, failcode = 1)
```

---

getFloatingPointSize *Get floating point size*

---

**Description**

getFloatingPointSize returns the floating-point representation size in a Cyclops data object

**Usage**

```
getFloatingPointSize(object)
```

**Arguments**

object	A Cyclops data object
--------	-----------------------

---

getHyperParameter *Get hyperparameter*

---

**Description**

getHyperParameter returns the current hyper parameter in a Cyclops model fit object

**Usage**

```
getHyperParameter(object)
```

**Arguments**

object	A Cyclops model fit object
--------	----------------------------

**Examples**

```

#Generate some simulated data:
sim <- simulateCyclopsData(nstrata = 1, nrows = 1000, ncovars = 2, eCovarsPerRow = 0.5,
                           model = "poisson")
cyclopsData <- convertToCyclopsData(sim$outcomes, sim$covariates, modelType = "pr",
                                    addIntercept = TRUE)

#Define the prior and control objects to use cross-validation for finding the
#optimal hyperparameter:
prior <- createPrior("laplace", exclude = 0, useCrossValidation = TRUE)
control <- createControl(cvType = "auto", noiseLevel = "quiet")

#Fit the model
fit <- fitCyclopsModel(cyclopsData, prior = prior, control = control)

#Find out what the optimal hyperparameter was:
getHyperParameter(fit)

#Extract the current log-likelihood, and coefficients
logLik(fit)
coef(fit)

#We can only retrieve the confidence interval for unregularized coefficients:
confint(fit, c(0))

```

---

getNumberOfCovariates *Get total number of covariates*

---

**Description**

getNumberOfCovariates returns the total number of covariates in a Cyclops data object

**Usage**

```
getNumberOfCovariates(object)
```

**Arguments**

object            A Cyclops data object

---

getNumberOfRows            *Get total number of rows*

---

**Description**

getNumberOfRows returns the total number of outcome rows in a Cyclops data object

**Usage**

```
getNumberOfRows(object)
```

**Arguments**

object            A Cyclops data object

---

getNumberOfStrata        *Get number of strata*

---

**Description**

getNumberOfStrata return the number of unique strata in a Cyclops data object

**Usage**

getNumberOfStrata(object)

**Arguments**

object            A Cyclops data object

---

getUnivariableCorrelation  
                          *Get univariable correlation*

---

**Description**

getUnivariableCorrelation reports covariates that have high correlation with the outcome

**Usage**

getUnivariableCorrelation(cyclopsData, covariates = NULL, threshold = 0)

**Arguments**

cyclopsData        A Cyclops data object  
 covariates        Integer or string vector: list of covariates to report; default (NULL) implies all covariates  
 threshold         Correlation threshold for reporting

**Value**

A list of covariates whose absolute correlation with the outcome is greater than or equal to the threshold

---

```
getUnivariableSeparability
      Get univariable linear separability
```

---

**Description**

getUnivariableSeparability reports covariates that are univariably separable with the outcome

**Usage**

```
getUnivariableSeparability(cyclopsData, covariates = NULL)
```

**Arguments**

cyclopsData	A Cyclops data object
covariates	Integer or string vector: list of covariates to report; default (NULL) implies all covariates

**Value**

A list of covariates that are univariably separable with the outcome

---

```
isInitialized      Check if a Cyclops data object is initialized
```

---

**Description**

isInitialized determines if an Cyclops data object is properly initialized and remains in memory. Cyclops data objects do not serialized/deserialize their back-end memory across R sessions.

**Usage**

```
isInitialized(object)
```

**Arguments**

object	Cyclops data object to test
--------	-----------------------------



---

logLik.cyclopsFit	<i>Extract log-likelihood</i>
-------------------	-------------------------------

---

**Description**

logLik returns the current log-likelihood of the fit in a Cyclops model fit object

**Usage**

```
## S3 method for class 'cyclopsFit'
logLik(object, ...)
```

**Arguments**

object	A Cyclops model fit object
...	Additional arguments

**Examples**

```
#Generate some simulated data:
sim <- simulateCyclopsData(nstrata = 1, nrows = 1000, ncovars = 2, eCovarsPerRow = 0.5,
                           model = "poisson")
cyclopsData <- convertToCyclopsData(sim$outcomes, sim$covariates, modelType = "pr",
                                   addIntercept = TRUE)

#Define the prior and control objects to use cross-validation for finding the
#optimal hyperparameter:
prior <- createPrior("laplace", exclude = 0, useCrossValidation = TRUE)
control <- createControl(cvType = "auto", noiseLevel = "quiet")

#Fit the model
fit <- fitCyclopsModel(cyclopsData, prior = prior, control = control)

#Find out what the optimal hyperparameter was:
getHyperParameter(fit)

#Extract the current log-likelihood, and coefficients
logLik(fit)
coef(fit)

#We can only retrieve the confidence interval for unregularized coefficients:
confint(fit, c(0))
```

---

meanLinearPredictor	<i>Calculates <math>\bar{x}\beta</math></i>
---------------------	---

---

**Description**

meanLinearPredictor computes  $\bar{x}\beta$  for model fit

**Usage**

```
meanLinearPredictor(cyclopsFit)
```

**Arguments**

cyclopsFit      A Cyclops model fit object

---

mse                      *Mean squared error*

---

**Description**

mse computes the mean squared error between two numeric vectors

**Usage**

```
mse(goldStandard, estimates)
```

**Arguments**

goldStandard      Numeric vector  
estimates          Numeric vector

**Value**

```
MSE(goldStandard, estimates)
```

---

Multitype              *Create a multitype outcome object*

---

**Description**

Multitype creates a multitype outcome object, usually used as a response variable in a hierarchical Cyclops model fit.

**Usage**

```
Multitype(y, type)
```

**Arguments**

y                      Numeric: Response count(s)  
type                    Numeric or factor: Response type

**Value**

An object of class Multitype with length equal to the length of y and type.

**Examples**

```
Multitype(c(0,1,0), as.factor(c("A","A","B")))
```

---

 oxford

*Oxford self-controlled case series data*


---

**Description**

A dataset containing the MMR vaccination / meningitis in Oxford example from Farrington and Whitaker. There are 10 patients comprising 38 unique exposure intervals.

**Usage**

```
data(oxford)
```

**Format**

A data frame with 38 rows and 6 variables:

**indiv** patient identifier  
**event** number of events in interval  
**interval** interval length in days  
**agegr** age group  
**exgr** exposure group  
**loginterval** log interval length ...

---

 predict.cyclopsFit

*Model predictions*


---

**Description**

predict.cyclopsFit computes model response-scale predictive values for all data rows

**Usage**

```
## S3 method for class 'cyclopsFit'
predict(object, newOutcomes, newCovariates, ...)
```

**Arguments**

object	A Cyclops model fit object
newOutcomes	An optional data frame or Andromeda table object, similar to the object used in <a href="#">convertToCyclopsData</a> .
newCovariates	An optional data frame or Andromeda table object, similar to the object used in <a href="#">convertToCyclopsData</a> .
...	Additional arguments

---

print.cyclopsData      *Print a Cyclops data object*

---

**Description**

print.cyclopsData displays information about a Cyclops data model object.

**Usage**

```
## S3 method for class 'cyclopsData'  
print(x, show.call = TRUE, ...)
```

**Arguments**

x	A Cyclops data model object
show.call	Logical: display last call to construct the Cyclops data model object
...	Additional arguments

---

print.cyclopsFit      *Print a Cyclops model fit object*

---

**Description**

print.cyclopsFit displays information about a Cyclops model fit object

**Usage**

```
## S3 method for class 'cyclopsFit'  
print(x, show.call = TRUE, ...)
```

**Arguments**

x	A Cyclops model fit object
show.call	Logical: display last call to update the Cyclops model fit object
...	Additional arguments

---

readCyclopsData	<i>Read Cyclops data from file</i>
-----------------	------------------------------------

---

### Description

readCyclopsData reads a Cyclops-formatted text file.

### Usage

```
readCyclopsData(fileName, modelType)
```

### Arguments

fileName	Name of text file to be read. If fileName does not contain an absolute path,
modelType	character string: Valid types are listed below.

### Details

This function reads a Cyclops-formatted text file and returns a Cyclops data object. The first line of the file may start with '#', indicating that it contains header options. Valid header options are:

row_label	(assume file contains a numeric column of unique row identifiers)
stratum_label	(assume file contains a numeric column of stratum identifiers)
weight	(assume file contains a column of row-specific model weights, currently unused)
offset	(assume file contains a dense column of linear predictor offsets)
bbr_outcome	(assume logistic outcomes are encoded -1/+1 following BBR)
log_offset	(assume file contains a dense column of values $x_i$ for which $\log(x_i)$ is the offset)
add_intercept	(automatically include an intercept column of all 1s for each entry)
indicator_only	(assume all covariates 0/1-valued and only covariate name is given)
sparse	(force all BBR formatted covariates to be represented as sparse, instead of sparse-indicator, columns .. really only for debugging)
dense	(force all BBR formatted covariates to be represented as dense columns.. really only for debugging)

Successive lines of the file are white-space delimited and follow the format:

```
[Row ID] {Stratum ID} [Weight] <Outcome> {Censored} {Offset} <BBR covariates>
```

- [optional]
- <required>
- {required or optional depending on model}

Bayesian binary regression (BBR) covariates are white-space delimited and generally in a sparse '<name>:<value>' format, where 'name' must (currently) be numeric and 'value' is non-zero. If option 'indicator\_only' is specified, then format is simply '<name>'. 'Row ID' and 'Stratum ID' must be numeric, and rows must be sorted such that equal 'Stratum ID' are consecutive. 'Stratum ID' is required for 'clr' and 'sccs' models. 'Censored' is required for a 'cox' model. 'Offset' is (currently) required for a 'sccs' model.

**Value**

A list that contains a Cyclops model data object pointer and an operation duration

**Models**

Currently supported model types are:

"ls"	Least squares
"pr"	Poisson regression
"lr"	Logistic regression
"clr"	Conditional logistic regression
"cpr"	Conditional Poisson regression
"sccs"	Self-controlled case series
"cox"	Cox proportional hazards regression
"fgr"	Fine-Gray proportional subdistribution hazards regression

**Examples**

```
## Not run:
dataPtr = readCyclopsData(system.file("extdata/infert_ccd.txt", package="Cyclops"), "clr")

## End(Not run)
```

---

simulateCyclopsData     *Simulation Cyclops dataset*

---

**Description**

simulateCyclopsData generates a simulated large, sparse data set for use by fitCyclopsSimulation.

**Usage**

```
simulateCyclopsData(
  nstrata = 200,
  nrows = 10000,
  ncovars = 20,
  effectSizeSd = 1,
  zeroEffectSizeProp = 0.9,
  eCovarsPerRow = ncovars/100,
  model = "survival"
)
```

**Arguments**

nstrata	Numeric: Number of strata
nrows	Numeric: Number of observation rows
ncovars	Numeric: Number of covariates
effectSizeSd	Numeric: Standard derivation of the non-zero simulated regression coefficients
zeroEffectSizeProp	Numeric: Expected proportion of zero effect size

eCovarsPerRow    Number: Effective number of non-zero covariates per data row  
 model            String: Simulation model. Choices are: logistic, poisson or survival

### Value

A simulated data set

### Examples

```
#Generate some simulated data:
sim <- simulateCyclopsData(nstrata = 1, nrows = 1000, ncovars = 2, eCovarsPerRow = 0.5,
                           model = "poisson")
cyclopsData <- convertToCyclopsData(sim$outcomes, sim$covariates, modelType = "pr",
                                    addIntercept = TRUE)

#Define the prior and control objects to use cross-validation for finding the
#optimal hyperparameter:
prior <- createPrior("laplace", exclude = 0, useCrossValidation = TRUE)
control <- createControl(cvType = "auto", noiseLevel = "quiet")

#Fit the model
fit <- fitCyclopsModel(cyclopsData, prior = prior, control = control)

#Find out what the optimal hyperparameter was:
getHyperParameter(fit)

#Extract the current log-likelihood, and coefficients
logLik(fit)
coef(fit)

#We can only retrieve the confidence interval for unregularized coefficients:
confint(fit, c(0))
```

---

splitTime	<i>Split the analysis time into several intervals for time-varying coefficients.</i>
-----------	--

---

### Description

splitTime split the analysis time into several intervals for time-varying coefficients

### Usage

```
splitTime(shortOut, cut)
```

### Arguments

shortOut            A data frame containing the outcomes with predefined columns (see below).  
 cut                 Numeric: Time points to cut at  
                     These columns are expected in the shortOut object:

rowId	(integer)	Row ID is used to link multiple covariates (x) to a single outcome (y)
y	(real)	Observed event status
time	(real)	Observed event time

**Value**

A long outcome table for time-varying coefficients.

---

summary.cyclopsData    *Cyclops data object summary*

---

**Description**

summary.cyclopsData summarizes the data held in an Cyclops data object.

**Usage**

```
## S3 method for class 'cyclopsData'
summary(object, ...)
```

**Arguments**

object	A Cyclops data object
...	Additional arguments

**Value**

Returns a data.frame that reports simply summarize statistics for each covariate in a Cyclops data object.

---

survfit.cyclopsFit    *Calculate baseline hazard function*

---

**Description**

survfit.cyclopsFit computes baseline hazard function

**Usage**

```
## S3 method for class 'cyclopsFit'
survfit(formula, type = "aalen", ...)
```

**Arguments**

formula	A Cyclops survival model fit object
type	type of baseline survival, choices are: "aalen" (Breslow)
...	for future methods

**Value**

Baseline survival function for mean covariates



---

vcov.cyclopsFit	<i>Calculate variance-covariance matrix for a fitted Cyclops model object</i>
-----------------	---

---

**Description**

vcov.cyclopsFit returns the variance-covariance matrix for all covariates of a Cyclops model object

**Usage**

```
## S3 method for class 'cyclopsFit'  
vcov(object, control, overrideNoRegularization = FALSE, ...)
```

**Arguments**

object	A fitted Cyclops model object
control	A "cyclopsControl" object constructed by <a href="#">createControl</a>
overrideNoRegularization	Logical: Enable variance-covariance estimation for regularized parameters
...	Additional argument(s) for methods

**Value**

A matrix of the estimates covariances between all covariate estimates.

# Index

`coef.cyclopsFit`, 3  
`confint.cyclopsFit`, 3  
`convertToCyclopsData`, 4, 27  
`convertToTimeVaryingCoef`, 7  
`coverage`, 7  
`createAutoGridCrossValidationControl`, 8  
`createControl`, 8, 9, 16, 17, 33  
`createCyclopsData`, 11  
`createNonSeparablePrior`, 13  
`createParameterizedPrior`, 14  
`createPrior`, 14  
`createWeightBasedSearchControl`, 16  
`cyclops`, 16

`fitCyclopsModel`, 8–10, 13, 14, 16, 17  
`fitCyclopsSimulation`, 18  
`formula`, 11, 12

`getCovariateIds`, 19  
`getCovariateTypes`, 19  
`getCyclopsProfileLogLikelihood`, 20  
`getFineGrayWeights`, 20  
`getFloatingPointSize`, 21  
`getHyperParameter`, 21  
`getNumberOfCovariates`, 22  
`getNumberOfRows`, 22  
`getNumberOfStrata`, 23  
`getUnivariableCorrelation`, 23  
`getUnivariableSeparability`, 24

`isInitialized`, 24

`logLik.cyclopsFit`, 25

`Matrix`, 12  
`meanLinearPredictor`, 25  
`mse`, 26  
`Multitype`, 26

`oxford`, 27

`predict.cyclopsFit`, 27  
`print.cyclopsData`, 28  
`print.cyclopsFit`, 28

`readCyclopsData`, 29

`simulateCyclopsData`, 30  
`splitTime`, 31  
`summary.cyclopsData`, 32  
`survfit.cyclopsFit`, 32  
`Sys.time`, 10

`vcov.cyclopsFit`, 33