Package: GADAG (via r-universe)

October 11, 2026

Type Package

Title A Genetic Algorithm for Learning Directed Acyclic Graphs

Version 0.99.1

Date 2026-09-10

Description Learns sparse large Directed Acyclic Graphs with a combination of a convex program and a tailored genetic algorithm.

License GPL-2

Encoding UTF-8

Depends igraph, MASS, cvTools

Imports Rcpp (>= 0.12.5)

LinkingTo Rcpp, RcppArmadillo

NeedsCompilation yes

RoxygenNote 7.3.3

LazyData true

Author Magali Champion [aut, cre], Victor Picheny [aut], Matthieu Vignes [aut]

Maintainer Magali Champion <magali.champion@u-paris.fr>

Repository https://cran.r-universe.dev

Date/Publication 2026-09-10 14:20:27 UTC

RemoteUrl https://github.com/cran/GADAG

RemoteRef HEAD

RemoteSha 85b8add873a3de18007438d77ec19299efada977

Contents

GADAG-package 2
evaluation 4
fitness 6

GADAG_Analyze 7
GADAG_CV 9
GADAG_Run 12
generateToyData 15
toy_data 16

Index **18**

GADAG-package *A Genetic Algorithm for Learning Directed Acyclic Graphs*

Description

Learns sparse large Directed Acyclic Graphs with a combination of a convex program and a tailored genetic algorithm.

Details

GADAG aims at recovering the structure of an unknow DAG G, whose edges represent the interactions that exist between p nodes, using n noisy observations of these nodes (design matrix X). GADAG is more precisely based on a l1-penalized (to make the estimated graph sparse enough) maximum log-likelihood estimation procedure, with the constraint that the estimated graph is a DAG. This DAG learning problem is particularly critical in the high-dimensional setting, the exploration of the whole of set of DAGs being a NP-hard problem. GADAG proposes an original formulation for the estimated DAG, splitting the initial problem into two sub-problems: node ordering and graph topology search. The node ordering, modelled as a permutation of [1,p] or the associated pxp matrix P, represents the importance of the p nodes of the graph, from the node with the smallest number of children to the node with the largest number of children. The topological structure of the graph, which is given as a lower triangular matrix T, then sets the graph edges weights (including 0, equivalent to no edges). GADAG works as follows: it efficiently looks for the best permution in an outer loop with a genetic algorithm, while a nested loop is used to find the optimal T associated to each given P. The latter internal optimization problem is solved by a steepest gradient descent approach.

The DESCRIPTION file:

Package:	GADAG
Type:	Package
Title:	A Genetic Algorithm for Learning Directed Acyclic Graphs
Version:	0.99.1
Date:	2026-09-10
Authors@R:	c(person(given = "Magali", family = "Champion", role = c("aut", "cre"), email = "magali.champion@u-
Description:	Learns sparse large Directed Acyclic Graphs with a combination of a convex program and a tailored ger
License:	GPL-2
Encoding:	UTF-8
Depends:	igraph, MASS, cvTools
Imports:	Rcpp (>= 0.12.5)
LinkingTo:	Rcpp, RcppArmadillo
NeedsCompilation:	yes

Packaged:	2017-09-12 14:56:12 UTC; magali
RoxygenNote:	7.3.3
LazyData:	true
Author:	Magali Champion [aut, cre], Victor Picheny [aut], Matthieu Vignes [aut]
Maintainer:	Magali Champion <magali.champion@u-paris.fr>

Author(s)

Magali Champion [aut, cre], Victor Picheny [aut], Matthieu Vignes [aut]

Maintainer: Magali Champion <magali.champion@u-paris.fr>

References

M. Champion, V. Picheny, M. Vignes, Inferring large graphs using l-1 penalized likelihood, Statistics and Computing (2017).

See Also

GADAG_Run, GADAG_Analyze

Examples

```
############################################################
# Loading toy data
############################################################
data(toy_data)
# toy_data is a list of two matrices corresponding to a "star"
# DAG (node 1 activates all other nodes):
# - toy_data$X is a 100x10 design matrix
# - toy_data$G is the 10x10 adjacency matrix (ground trough)

############################################################
# Running GADAG
############################################################
# Simple run, with only the penalty term specified
GADAG_results <- GADAG_Run(X=toy_data$X, lambda=0.1)
print(GADAG_results$G.best) # optimal adjacency matrix graph

# Expensive run with many evaluations if we refine the
# termination conditions
## Not run:
n.gen <- 1e10 # we allow a very large number of iterations
tol.Shannon <- 1e-10 # the entropy of Shannon of the population
                     # has to be very small
pop.size <- 5*ncol(toy_data$G) # this is usually a good
                               # population size
max.eval <- n.gen * pop.size # maximal number of nested
                             # evaluation
```

```
  GADAG_results <- GADAG_Run(X=toy_data$X, lambda=0.1,
       GADAG.control=list(n.gen=n.gen, tol.Shannon=tol.Shannon,
                          pop.size = pop.size, max.eval=max.eval))
  print(GADAG_results$G.best) # optimal adjacency matrix graph
## End(Not run)

  # Expensive run if we also increase the population size
  ## Not run:
  pop.size <- 10*ncol(toy_data$G)
  GADAG_results <- GADAG_Run(X=toy_data$X, lambda=0.1,
      GADAG.control=list(pop.size=pop.size))

## End(Not run)

  # You can have more information about the evolution of the
  # algorithm by turning return.level on
  ## Not run:
  return.level <- 1
  GADAG_results <- GADAG_Run(X=toy_data$X, lambda=0.1, return.level = return.level)
  print(GADAG_results$f.best.evol) # this shows the evolution of the fitness
                                   # across the iterations

## End(Not run)
```

`evaluation` *Evaluate the fitness of a population*

Description

Internal function of the genetic algorithm that evaluates the fitness (penalized log-likelihood) of a set (population) of permutations. It internally computes the best triangular matrix associated to each permutation of the population.

Usage

```
evaluation(
  Pop,
  X,
  XtX = NULL,
  lambda,
  grad.control = list(tol.obj = 1e-06, max.ite = 50),
  ncores = 1
)
```

Arguments

`Pop`	Population of permutations from [1,p]: matrix with `pop.size` rows and p columns, each row corresponding to one permutation of the population.
`X`	Design matrix, with samples (n) in rows and variables (p) in columns.

XtX	(optional) Cross-product of X; computed if not provided.
lambda	Parameter of penalization (>0).
grad.control	A list containing the parameters for controlling the inner optimization, i.e. the gradient descent tol.obj.inner tolerance (>0), max.ite.inner maximum number of iterations (>0).
ncores	Number of cores (>1, depending on your computer).

Value

A list with the following elements:

Tpop	Matrix with pxp columns, each column corresponding to the best triangular matrix (in a vector form) associated to each permutation of the population.
f	Fitness of the population.

A list with the following elements:

Tpop Matrix with p rows and pop.size columns, each column corresponding to the best triangular matrix (in a vector form) associated to each permutation of the population.

f Fitness of the population.

Author(s)

Magali Champion [aut, cre], Victor Picheny [aut], Matthieu Vignes [aut]

See Also

GADAG, GADAG_Run, fitness.

Examples

```
############################################################
# Loading toy data
############################################################
data(toy_data)
# toy_data is a list of two matrices corresponding to a "star"
# DAG (node 1 activates all other nodes):
# - toy_data$X is a 100x10 design matrix
# - toy_data$G is the 10x10 adjacency matrix (ground trough)

#########################################################
# Creating a population of permutations
#########################################################
# first, define the parameters
p <- ncol(toy_data$G) # number of nodes
pop.size <- 10 # population size

# then create your population of permutations
Pop <- matrix(data = 0, nrow = pop.size, ncol = p)
```

```
for(i in 1:pop.size){
    Pop[i,] = sample(p)
}

#########################################################
# Evaluating the fitness of the population
#########################################################
# evaluation of the whole population
Evaluation <- evaluation(Pop=Pop,X=toy_data$X,lambda=0.1)
print(Evaluation$f) # here is the fitness of the whole population

# evaluation of one of the permutation
Perm <- Pop[1,]
Evaluation <- evaluation(Pop=Perm,toy_data$X,lambda=0.1)

# optimal matrix T associated to Perm:
T <- matrix(Evaluation$Tpop,p,p)
```

`fitness` *Compute the fitness of a potential solution*

Description

Internal function of the genetic algorithm that evaluates the fitness (penalized log-likelihood) of a potential solution, given as a pair of a permutation (P) and a triangular matrix (T).

Usage

```
fitness(P,X,T,lambda)
```

Arguments

`P`	A permutation from [1,p] in a matrix form.
`X`	Design matrix, with samples (n) in rows and variables (p) in columns.
`T`	A pxp lower-triangular matrix.
`lambda`	Parameter of penalization (>0).

Value

A numeric value corresponding to the fitness of the potential solution.

Author(s)

Magali Champion [aut, cre], Victor Picheny [aut], Matthieu Vignes [aut]

See Also

`GADAG`, `GADAG_Run`, `evaluation`.

Examples

```
############################################################
# Loading toy data
############################################################
data(toy_data)
# toy_data is a list of two matrices corresponding to a "star"
# DAG (node 1 activates all other nodes):
# - toy_data$X is a 100x10 design matrix
# - toy_data$G is the 10x10 adjacency matrix (ground trough)

############################################################
# Creating a candidate solution
############################################################
# define parameters
p <- ncol(toy_data$G)

# permutation matrix
Perm <- sample(p) # permutation in a vector form
P <- matrix(0,p,p)
P[p*0:(p-1) + Perm] <- 1 # Perm is tranformed into a matrix form

# lower-triangular matrix
T <- matrix(rnorm(p),p,p)
T[upper.tri(T,diag=TRUE)] <- 0

########################################################
# Computing the fitness of the potential solution
########################################################
Fitness <- fitness(P=P, X=toy_data$X, T=T, lambda=0.1)
print(Fitness) # here is the fitness of the candidate solution (P,T)
```

GADAG_Analyze *Analyze GADAG results.*

Description

Function to Analyze GADAG results.

Usage

```
GADAG_Analyze(
  Results,
  G = NULL,
  X = NULL,
  plot.control = list(plot.graph = 0, plot.evol = 0, plot.png = 0),
  Nodes = NULL
)
```

Arguments

`Results`	Outputs from `GADAG_Run()` function.
`G`	(optional) Adjacency matrix corresponding to the true DAG (pxp matrix).
`X`	(optional) Design matrix with samples (n) in rows and variables (p) in columns.
`plot.control`	A list containing parameters to control the produced graph outputs (`return.level` has to be turned to 1 in the main code beforehand): `plot.graph` If 1, generates the figures with the actual and estimated graphs, `plot.evol` If 1, generates the figures showing the evolution of the genetic algorithm (fitness value, Shannon entropy and best node ordering), `plot.png` If 1, saves the figures in .png.
`Nodes`	(optional) If plot.evol is turned on, specifies the evolution of which nodes you want to highlight.

Details

This function returns as primary outputs the performances of the estimation graph procedure in terms of TP, FP, FN, TN, precision and recall, obtained by comparing the estimated graph with the true one (if known and provided). If specified (`plot.graph`, `plot.evol`), the user can plot both the estimated graph and the true DAG (if known and provided) and the evolution of the algorithm. This generates three figures: the first one represents the evolution of the fitness value (best fitness in red, averaged population fitness and quantiles across the iterations), the second one, the evolution of the Shannon entropy of each node across the iterations, the third one, the best node ordering (permutation that minimizes the fitness) across the iterations.

Value

A vector containing the scores of precision, recall, number of false positives (FP), false negatives (FN), true positives (TP), true negatives (TN) and mean squared error (only if `G` and `X` are provided).

Author(s)

Magali Champion [aut, cre], Victor Picheny [aut], Matthieu Vignes [aut]

See Also

`GADAG`, `GADAG_CV`, `GADAG_Run`, `GADAG_Analyze`.

Examples

```
############################################################
# Loading toy data
############################################################
data(toy_data)
# toy_data is a list of two matrices corresponding to a "star"
# DAG (node 1 activates all other nodes):
# - toy_data$X is a 100x10 design matrix
# - toy_data$G is the 10x10 adjacency matrix (ground trough)
```

```
#######################################################
# Evaluating GADAG Results
#######################################################
# simple run, where you only get the precision, recall, number
# of false positives, true positives, false negatives, true negatives
# and mean squared error of the estimated graph

# run GADAG with the predefined parameters
GADAG_results <- GADAG_Run(X=toy_data$X, lambda=0.1)

# analyze the results
GADAG_analysis <- GADAG_Analyze(GADAG_results, G=toy_data$G, X=toy_data$X)
print(GADAG_analysis) # here are the results


# more complex run, where you want to have some details about the procedure
## Not run:
# run GADAG with return.level set to 1 beforehand
GADAG_results <- GADAG_Run(X=toy_data$X, lambda=0.1,return.level=1)

# print the evolution of the algorithm and highlight node 1
plot.evol <- 1
GADAG_analysis <- GADAG_Analyze(GADAG_results, G=toy_data$G, X=toy_data$X,
         plot.control = list(plot.evol=1), Nodes=c(1))

# in addition, print the estimated and the true graph
plot.graph <- 1
GADAG_analysis <- GADAG_Analyze(GADAG_results, G=toy_data$G, X=toy_data$X,
         plot.control = list(plot.evol=plot.evol, plot.graph= plot.graph))

# now save the results in .png, but only for the graphs
plot.png <- 1
GADAG_analysis <- GADAG_Analyze(GADAG_results, G=toy_data$G, X=toy_data$X,
         plot.control = list(plot.graph= plot.graph, plot.png = plot.png))

# in case you don't know the true DAG, you can't really know how good the
# estimation is. You can't compute the precision, recall, MSE but you can
# still plot the estimated graph and see the evolution of the algorithm
plot.graph <- plot.evol <- 1
plot.png <- 0
GADAG_analysis <- GADAG_Analyze(GADAG_results, X=toy_data$X,
         plot.control = list(plot.graph= plot.graph, plot.evol = plot.evol))

## End(Not run)
```

GADAG_CV *Cross-validation for GADAG*

Description

Function to run k-fold cross-validation for GADAG to optimally tune the parameter of penalization.

Usage

```
GADAG_CV(
  X,
  Lambdas = NULL,
  n.folds = 10,
  threshold = 0.1,
 GADAG.control = list(n.gen = 250, tol.Shannon = 1e-06, max.eval = 1e+07, pop.size = 5 *
    ncol(X), p.xo = 0.25, p.mut = 0.05),
  grad.control = list(tol.obj.inner = 1e-06, max.ite.inner = 50),
  ncores = 1,
  plot.CV = 0
)
```

Arguments

X	Design matrix, with samples (n) in rows and variables (p) in columns.
Lambdas	Optional user-supplied lambda sequence. Default is null and GADAG chooses its own sequence.
n.folds	Number of folds for cross-validation (10 by default). Can be as large as the sample size (leave-one-out cross validation) but not recommended for large data sets.
threshold	Thresholding value for the estimated edges.
GADAG.control	A list containing parameters for controlling GADAG (termination conditions and inherent parameters of the Genetic Algortihm). Some parameters (n.gen, max.eval and pop.size) are particularly critical for reducing the computational time.
	n.gen maximal number of population generations (>0),
	pop.size initial population size for the genetic algorithm (>0),
	max.eval overall maximal number of calls of the evaluation function (>0, should be of the order of n.gen*pop.size),
	tol.Shannon threshold for the Shannon entropy (>0),
	p.xo crossover probability of the genetic algorithm (between 0 and 1),
	p.mut mutation probability of the genetic algorithm (between 0 and 1).
grad.control	A list containing the parameters for controlling the inner optimization, i.e. the gradient descent.
	tol.obj.inner tolerance (>0),
	max.ite.inner maximum number of iterations (>0).
ncores	Number of cores (>0, depending on your computer).
plot.CV	If 1, plots the averaged cross validation error given sequence of lambdas.

Details

The function runs GADAG n.folds times for each of the tested lambdas to compute the best solution associated to each omitted fold. The error is accumulated and the averaged error over the folds is computed. The best lambda lambda.min corresponds to the one that minimizes the error.

Value

A list with the following elements:

`lambda.min` Value of `lambda` that minimizes the averaged cross validation error `error.CV`.

`lambda.1se` Largest value of `lambda` such that `error.CV` is within 1 % of the minimum.

`nzero` Number of non-zero coefficients at each `lambda`.

`Lambdas` The values of `lambda` used in the fits.

`error.CV` The averaged cross validation error.

Author(s)

Magali Champion [aut, cre], Victor Picheny [aut], Matthieu Vignes [aut]

References

M. Champion, V. Picheny, M. Vignes, Inferring large graphs using l-1 penalized likelihood, Statistics and Computing (2017).

See Also

GADAG, GADAG_Run, GADAG_Analyze.

Examples

```
 ############################################################
 # Loading toy data
 ############################################################
 data(toy_data)
 # toy_data is a list of two matrices corresponding to a "star"
 # DAG (node 1 activates all other nodes):
 # - toy_data$X is a 100x10 design matrix
 # - toy_data$G is the 10x10 adjacency matrix (ground trough)

 ############################################################
 # Tuning the parameter of penalization
 ############################################################
 # Simple run, whithout specifying GADAG parameters
 ## Not run:
 GADAG_CV_results <- GADAG_CV(X=toy_data$X)
 print(GADAG_CV_results$lambda.1se) # best lambda

## End(Not run)
 # If desired, additional plot for the averaged cross validation
 # error
 ## Not run:
 GADAG_CV_results <- GADAG_CV(X=toy_data$X,plot.CV=1)

## End(Not run)
 # Given the best lambda, re-run GADAG
 ## Not run:
```

```
GADAG_results <- GADAG_Run(X=toy_data$X, lambda=GADAG_CV_results$lambda.1se)
print(GADAG_results$G.best) # optimal adjacency matrix graph

## End(Not run)
```

GADAG_Run *Run GADAG*

Description

Function to run GADAG, an algorithm that aims at inferring large sparse directed acyclic graphs based on an observation sample X, by minimizing the penalized negative log-likelihood with a convex program embedded in a genetic algorithm.

Usage

```
GADAG_Run(
  X,
  lambda = 0.1,
  threshold = 0.1,
 GADAG.control = list(n.gen = 250, tol.Shannon = 1e-06, max.eval = 1e+07, pop.size = 5 *
    ncol(X), p.xo = 0.25, p.mut = 0.05),
  grad.control = list(tol.obj.inner = 1e-06, max.ite.inner = 50),
  ncores = 1,
  print.level = 0,
  return.level = 0
)
```

Arguments

X	Design matrix, with samples (n) in rows and variables (p) in columns.
lambda	Parameter of penalization (>0).
threshold	Thresholding value for the estimated edges.
GADAG.control	A list containing parameters for controlling GADAG (termination conditions and inherent parameters of the Genetic Algortihm). Some parameters (n.gen, max.eval and pop.size) are particularly critical for reducing the computational time. n.gen maximal number of population generations (>0), pop.size initial population size for the genetic algorithm (>0), max.eval overall maximal number of calls of the evaluation function (>0, should be of the order of n.gen*pop.size), tol.Shannon threshold for the Shannon entropy (>0), p.xo crossover probability of the genetic algorithm (between 0 and 1), p.mut mutation probability of the genetic algorithm (between 0 and 1).
grad.control	A list containing the parameters for controlling the inner optimization, i.e. the gradient descent.

	`tol.obj.inner` tolerance (>0),
	`max.ite.inner` maximum number of iterations (>0).
`ncores`	Number of cores (>0, depending on your computer).
`print.level`	0 no print, 1 some info on the genetic algorithm behaviour are printed.
`return.level`	0 only best solution is returned, 1 evolution of the current best solution and statistics on the population fitness values are also returned.

Details

This function returns as a primary output `G.best`, the adjacency matrix of the inferred graph. This matrix is computed thanks to its decomposition (`P.best`, `T.best`).

The values of the inputs `n.gen`, `max.eval` and `pop.size` largely influence the algorithm inference capability, but also its computational cost. As a rule-of-thumb, we recommend setting `pop.size` between 1 to 10 times the number of nodes, and `n.gen` between 10 to 250 times `pop.size`. `tol.Shannon` may be decreased in case of premature stop. The other parameters should only be modified with care.

Value

A list with the following elements:

`f.best`	Best fitness value.
`P.best`	Best node order (vector of length p).
`T.best`	Corresponding best edges values (vector of length p).
`G.best`	Best graph (matrix form).
`f.best.evol`	Evolution of the best fitness value across the iterations (if return.level=1).
`P.best.evol`	Evolution of the best node order across the iterations (if return.level=1).
`T.best.evol`	Evolution of the best edges values across the iterations (if return.level=1).
`fmin.evol`	Evolution of the minimal fitness value of the population across the iterations (if return.level=1).
`fmean.evol`	Evolution of the averaged fitness value of the population across the iterations (if return.level=1).
`fp10.evol`	Evolution of the quantiles of the fitness value across the iterations (if return.level=1).
`fp90.evol`	Evolution of the quantiles of the fitness value across the iterations (if return.level=1).
`Shannon.evol`	Evolution of the Shannon entropy of the population across the iterations (if return.level=1).

Author(s)

Magali Champion [aut, cre], Victor Picheny [aut], Matthieu Vignes [aut]

References

M. Champion, V. Picheny, M. Vignes (2017), Inferring large graphs using l-1 penalized likelihood, 'https://arxiv.org/pdf/1507.02018.pdf'.

See Also

GADAG, GADAG_CV, GADAG_Run, GADAG_Analyze.

Examples

```
 ############################################################
 # Loading toy data
 ############################################################
 data(toy_data)
 # toy_data is a list of two matrices corresponding to a "star"
 # DAG (node 1 activates all other nodes):
 # - toy_data$X is a 100x10 design matrix
 # - toy_data$G is the 10x10 adjacency matrix (ground trough)

 ############################################################
 # Running GADAG
 ############################################################
 # Simple run, with only the penalty term specified
 GADAG_results <- GADAG_Run(X=toy_data$X, lambda=0.1)
 print(GADAG_results$G.best) # optimal adjacency matrix graph


 # Expensive run with many evaluations if we refine the
 # termination conditions
 ## Not run:
 n.gen <- 1e10 # we allow a very large number of iterations
 tol.Shannon <- 1e-10 # the entropy of Shannon of the population
                      # has to be very small
 pop.size <- 5*ncol(toy_data$G) # this is usually a good
                                # population size
 max.eval <- n.gen * pop.size # maximal number of nested
                              # evaluation
 GADAG_results <- GADAG_Run(X=toy_data$X, lambda=0.1,
      GADAG.control=list(n.gen=n.gen, tol.Shannon=tol.Shannon,
                         pop.size = pop.size, max.eval=max.eval))
 print(GADAG_results$G.best) # optimal adjacency matrix graph

## End(Not run)

 # Expensive run if we also increase the population size
 ## Not run:
 pop.size <- 10*ncol(toy_data$G)
 GADAG_results <- GADAG_Run(X=toy_data$X, lambda=0.1,
      GADAG.control=list(pop.size=pop.size))
 print(GADAG_results$G.best) # optimal adjacency matrix graph

## End(Not run)

 # You can have more information about the evolution of the
 # algorithm by turning return.level on
 ## Not run:
 return.level <- 1
```

```
GADAG_results <- GADAG_Run(X=toy_data$X, lambda=0.1, return.level = return.level)
print(GADAG_results$f.best.evol) # this shows the evolution of the fitness
                                 # across the iterations

## End(Not run)
```

`generateToyData` *Generate toy data for running GADAG*

Description

This function generates toy data that can be used to run GADAG: the adjacency matrix of a DAG with p nodes and the design matrix with n observations of the distribution of the p nodes.

Usage

```
generateToyData(n, p, edgemin = 0, Cov = diag(p), type = "star", seed = 42)
```

Arguments

`n`	Number of samples in the design matrix.
`p`	Number of nodes of the DAG.
`edgemin`	Minimal value for the non-null edges of the DAG (between 0 and 1).
`Cov`	(optional) Covariance matrix for the noise variables (identity matrix by default)
`type`	Form of the DAG. It can be chosen between 7 alternatives: `"star"`, `"bistar"`, `"full"`, `"path"`, `"quadristar"`, `"sixstar"` (see details below).
`seed`	Fix the seed.

Details

One of the following seven alternatives can be chosen for the DAG form:

`"star"` star-shaped DAG (all active edges start from node 1),

`"bistar"` half of the edges start from node 1 and the other half from node 2,

`"full"` full DAG (all the edges are active),

`"path"` path-shaped DAG (all the nodes are connected by a single path),

`"quadristar"` node 1 is connected to nodes 2 to 4, each being connected to 1/3 of the rest of the nodes,

`"sixstar"` same as `"quadristar"`, with 6 nodes.

Value

A list containing the design nxp matrix X (with samples in rows and variables in columns) and the adjacency matrix G associated to the DAG with p nodes.

Author(s)

Magali Champion [aut, cre], Victor Picheny [aut], Matthieu Vignes [aut].

See Also

GADAG, GADAG_Run.

Examples

```
 ############################################################
 # Generating toy data
 ############################################################
 toy_data <- generateToyData(n=100, p=10)

 # toy_data is a list of two matrices corresponding to a "star"
 # DAG (node 1 activates all other nodes):
 # - toy_data$X is a 100x10 design matrix
 # - toy_data$G is the 10x10 adjacency matrix (ground trough)

 ## Not run:
 # generate another type of data: a DAG with 100 nodes in a path form
 toy_data <- generateToyData(n=100, p=100,type="path")

## End(Not run)

 ## Not run:
 # set the minimal edge value to 1
 toy_data <- generateToyData(n=100, p=10, edgemin=1) # all edges are set to 1

## End(Not run)

 ## Not run:
 # non-independent noises
 A <- matrix(runif(10^2)*2-1, ncol=10)
 Sigma <- t(A) %*% A
 toy_data <- generateToyData(n=100, p=10, edgemin=1, Cov = Sigma)

## End(Not run)
```

---

toy_data *Toy data for running GADAG*

Description

An example of data for running GADAG.

Usage

```
data(toy_data)
```

Format

A list of two variables containing the adjacency matrix G of a 10-nodes "star" DAG (node 1 activates all other 9 nodes) and a design matrix X with 100 observations of the distribution of the 10 nodes.

Index

∗ **Directed Acyclic Graphs - Convex program - Optimization**
GADAG-package, 2
∗ **datasets**
toy_data, 16

evaluation, 4, *6*

fitness, *5*, 6

GADAG, *5, 6*, *8*, *11*, *14*, *16*
GADAG *(*GADAG-package*)*, 2
GADAG-package, 2
GADAG_Analyze, *3*, 7, *8*, *11*, *14*
GADAG_CV, *8*, 9, *14*
GADAG_Run, *3*, *5, 6*, *8*, *11*, 12, *14*, *16*
generateToyData, 15

toy_data, 16