Package 'systemicrisk'

May 5, 2024

Type Package

Title Systemic Risk and Network Reconstruction

Version 0.4.3

Date 2024-05-03

Description Analysis of risk through liability matrices. Contains a Gibbs sampler for network reconstruction, where only row and column sums of the liabilities matrix as well as some other fixed entries are observed, following the methodology of Gandy&Veraart (2016) <doi:10.1287/mnsc.2016.2546>. It also incorporates models that use a power law distribution on the degree distribution.

License GPL-3

Imports lpSolve, Rcpp (>= 0.11.2), stats, utils

LinkingTo Rcpp

Suggests coda, testthat, knitr

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.3.1

NeedsCompilation yes

Author Axel Gandy [aut, cre],
Luitgard A.M. Veraart [aut]

Maintainer Axel Gandy <a.gandy@imperial.ac.uk>

Repository CRAN

Date/Publication 2024-05-05 21:10:02 UTC

R topics documented:

`calibrate_ER` *Calibrate ER model to a given density*

Description

The model is an Erdos-Renyi model for the existence of links (a link exists independently of other links with a fixed probability) and the weight of each existing link follows an exponential distribution with a fixed rate parameter. This function chooses the two parameters such that the density of the network (the average proportion of existing links) is a certain desired value. Diagonal elements are being set to 0.

Usage

```
calibrate_ER(
  l,
  a,
  targetdensity,
  L_fixed = NA,
```

```
  nsamples_calib = 100,
  thin_calib = 100
)
```

Arguments

l	row sums of matrix to be reconstructed
a	column sum of matrix to be reconstructed
targetdensity	desired proportion of reconstructed entries to be positive
L_fixed	Matrix containing known values of L, where NA signifies that an element is not known. If L_fixed equates to NA (the default) then no values are assumed to be known.
nsamples_calib	number of matrices to generate during calibration.
thin_calib	amount of thinning to use during calibration

Value

Model that can be used to generate the desired samples using sample_HierarchicalModel.

Examples

```
## first generate a true network
n <- 10 # size of network
p <- 0.45
lambda <- 0.1
L <- matrix(nrow=n,rbinom(n*n,prob=p,size=1)*rexp(n*n,rate=lambda))

# then reconstruct with a target density of 0.55
model <- calibrate_ER(l=rowSums(L),a=colSums(L),
                      targetdensity=0.55,nsamples_calib=10)
Lsamp <- sample_HierarchicalModel(l=rowSums(L),a=colSums(L),model=model,
                                  nsamples=10,thin=1e2)
# check row sums
rowSums(L)
rowSums(Lsamp$L[[10]])
# check calibration
mean(Lsamp$L[[10]]>0)

# now an example with some fixed entries
L_fixed <- L
L_fixed[1:(n/2),] <- NA
# then reconstruct with a target density of 0.9
model <- calibrate_ER(l=rowSums(L),a=colSums(L),L_fixed=L_fixed,
                            targetdensity=0.9,nsamples_calib=10)
Lsamp <- sample_HierarchicalModel(l=rowSums(L),a=colSums(L),L_fixed=L_fixed,
                                  model=model,nsamples=10,thin=1e2)
mean(Lsamp$L[[10]][-(1:(n/2)),]>0) # known entries
mean(Lsamp$L[[10]][(1:(n/2)),]>0)  #reconstructed entries
```

`calibrate_ER.nonsquare` *Calibrate ER model to a given density with a nonsquare matrix*

Description

The model is an Erdos-Renyi model for the existence of links (a link exists independently of other links with a fixed probability) and the weight of each existing link follows an exponential distribution with a fixed rate parameter. This function chooses the two parameters such that the density of the network (the average proportion of existing links) is a certain desired value. This function does not set diagonal values to 0.

Usage

```
calibrate_ER.nonsquare(
  l,
  a,
  targetdensity,
  L_fixed = NA,
  nsamples_calib = 100,
  thin_calib = 100
)
```

Arguments

`l`	row sums of matrix to be reconstructed
`a`	column sum of matrix to be reconstructed
`targetdensity`	desired proportion of reconstructed entries to be positive
`L_fixed`	Matrix containing known values of L, where NA signifies that an element is not known. If `L_fixed` equates to `NA` (the default) then no values are assumed to be known.
`nsamples_calib`	number of matrices to generate during calibration.
`thin_calib`	amount of thinning to use during calibration

Value

Model that can be used to generate the desired samples using `sample_HierarchicalModel`.

Examples

```
## first generate a true network
nrow <- 10 # size of network
ncol <- 8 # size of network
p <- 0.45
lambda <- 0.1
L <- matrix(nrow=nrow,rbinom(nrow*ncol,prob=p,size=1)*rexp(nrow*ncol,rate=lambda))
```

```
# then reconstruct with a target density of 0.55
model <- calibrate_ER.nonsquare(l=rowSums(L),a=colSums(L),
                      targetdensity=0.55,nsamples_calib=10)
Lsamp <- sample_HierarchicalModel(l=rowSums(L),a=colSums(L),model=model,
                                  nsamples=10,thin=1e2)
# check row sums
rowSums(L)
rowSums(Lsamp$L[[10]])
# check calibration
mean(Lsamp$L[[10]]>0)

# now an example with some fixed entries
L_fixed <- L
L_fixed[1:(nrow/2),] <- NA
# then reconstruct with a target density of 0.9
model <- calibrate_ER.nonsquare(l=rowSums(L),a=colSums(L),L_fixed=L_fixed,
                            targetdensity=0.9,nsamples_calib=10)
Lsamp <- sample_HierarchicalModel(l=rowSums(L),a=colSums(L),L_fixed=L_fixed,
                                  model=model,nsamples=10,thin=1e2)
mean(Lsamp$L[[10]][-(1:(nrow/2)),]>0) # known entries
mean(Lsamp$L[[10]][(1:(nrow/2)),]>0)  #reconstructed entries
```

`calibrate_FitnessEmp` *Calibrate empirical fitness model to a given density*

Description

The model is an empirical fitness based model for the existence of links (more details below) which contains one fixed parameter and the weight of each existing link follows an exponential distribution with a fixed rate parameter. This function chooses the two parameters such that the density of the network (the average proportion of existing links) with these given row and column sums is a certain desired value.

Usage

```
calibrate_FitnessEmp(
  l,
  a,
  targetdensity,
  L_fixed = NA,
  nsamples_calib = 100,
  thin_calib = 100
)
```

Arguments

`l`	row sums of matrix to be reconstructed
`a`	column sum of matrix to be reconstructed

targetdensity	desired proportion of reconstructed entries to be positive
L_fixed	Matrix containing known values of L, where NA signifies that an element is not known. If L_fixed equates to NA (the default) then no values are assumed to be known.
nsamples_calib	number of matrices to generate during calibration.
thin_calib	amount of thinning to use during calibration

Details

The empirical fitness model assumes that every node $f[i] = log(l[i] + a[i])$ has a fitness given by the observered row and column sum and that the existence probability of a link between node i and j is then given by $p[i,j] = 1/(1 + exp(-(alpha + f[i] + f[j])))$, where alpha is an additional parameter. The resulting model uses observed quantities (the row and column sums of the matrix) as input to the model and is thus an empirical Bayes approach.

Value

Model that can be used to generate the desired samples using sample_HierarchicalModel.

Examples

```
## first generate a true network
n <- 10 # size of network
ftrue <- rnorm(n) # vector of underlying fitnesses
p <- outer(ftrue,ftrue,FUN=function(x,y) 1/(1+exp(-(x+y))))
lambda <- 0.1
L <- matrix(nrow=n,rbinom(n*n,prob=p,size=1)*rexp(n*n,rate=lambda))

# then reconstruct with a target density of 0.7
model <- calibrate_FitnessEmp(l=rowSums(L),a=colSums(L),
                              targetdensity=0.7,nsamples_calib=10,thin_calib=50)
Lsamp <- sample_HierarchicalModel(l=rowSums(L),a=colSums(L),model=model,
                                  nsamples=10,thin=1e2)
# check row sums
rowSums(L)
rowSums(Lsamp$L[[10]])
# check calibration
mean(Lsamp$L[[10]]>0)

# now an example with some fixed entries
L_fixed <- L
L_fixed[1:(n/2),] <- NA
# then reconstruct with a target density of 0.9
model <- calibrate_FitnessEmp(l=rowSums(L),a=colSums(L),L_fixed=L_fixed,
                              targetdensity=0.9,nsamples_calib=10,thin_calib=50)
Lsamp <- sample_HierarchicalModel(l=rowSums(L),a=colSums(L),L_fixed=L_fixed,
                                  model=model,nsamples=10,thin=1e2)
mean(Lsamp$L[[10]][-(1:(n/2)),]>0) # known entries
mean(Lsamp$L[[10]][(1:(n/2)),]>0)  #reconstructed entries
```

`choosethin` *Calibrate Thinning*

Description

Attempts to automatically choose a thinning paramter to achieve an overall relative effective sample size (defined as the effective sample size divided by the number of samples) for all parameters in the model (that do not seem to be constant). This function provides no guarantees that the desired relative effective sample size (rESS) will actually be achieved - it is best treated as a rough guide for this.

Usage

```
choosethin(
  l,
  a,
  L_fixed = NA,
  model,
  relESStarget = 0.3,
  burnin = 100,
  matrpertheta = length(l)^2,
  silent = TRUE,
  maxthin = 10000
)
```

Arguments

`l`	observed row sum
`a`	observerd column sum
`L_fixed`	Matrix containing known values of L, where NA signifies that an element is not known. If `L_fixed` equates to `NA` (the default) then no values are assumed to be known.
`model`	Underlying model for p and lambda.
`relESStarget`	Target for the relative effective sample size, must be in (0,1). Default 0.3.
`burnin`	number of iterations for the burnin. Defaults to 5 of the steps in the sampling part.
`matrpertheta`	number of matrix updates per update of theta.
`silent`	(default FALSE) suppress all output (including progress bars).
`maxthin`	Upper bound on thinning to consider. Default 10000.

Details

The approach used involves a pilot run of the sampler, followed by a computation of the acf (auto-correlation function) for each component. The acf is used only up to (and excluding) the point used where it becomes negative for the first time. This part of the acf is then used to approximate the

rESS and to determine the amount of thinning needed. The reported result is the thinning needed to achieve the rESS for all components (the matrix as well as the parameter theta). The initial pilot run may not be sufficient and further pilot runs may have to be started.

Value

An integer describing the amount of thinning required.

Examples

```
set.seed(12689)
n <- 10
m <- Model.Indep.p.lambda(Model.p.BetaPrior(n),
                          Model.lambda.GammaPrior(n,scale=1e-1))
x <- genL(m)
l <- rowSums(x$L)
a <- colSums(x$L)
choosethin(l,a,model=m)
choosethin(l,a,model=m,relESStarget=0.7)
```

---

cloneMatrix *Creates a deep copy of a matrix*

Description

Useful when calling ERE_step_cycle or GibbsSteps_kcycle to ensure that there are no side effects for the return values.

Usage

```
cloneMatrix(M)
```

Arguments

M	A matrix

Value

A deep copy of the matrix.

Examples

```
lambda <- matrix(0.5,nrow=2,ncol=2)
p <- matrix(0.7, nrow=2,ncol=2)
L <- matrix(rexp(4),nrow=2);
L
Lold <- L
Lcopy <- cloneMatrix(L)
ERE_step_cycle(r=c(0,1),c=c(0,1),L=L,lambda=lambda,p=p)
```

```
L     ## new value
Lold  ## equal to L !!!
Lcopy ## still has the original value
```

`default` *Default of Banks*

Description

Computes bank defaults based on a liabilities matrix and external assets and liabilities.

Usage

```
default(L, ea, el = 0, method = c("clearing", "cascade"), ...)
```

Arguments

`L`	liability matrix
`ea`	vector of external assets
`el`	vector of external liabilites.
`method`	the method to be used. See Details.
`...`	Additional information for the various methods. See Details.

Value

A list with at least one element "defaultind", which is a vector indicating which banks default (1=default, 0= no default). Depending on the method, other results such as the clearing vector may also be reported.

See Also

`default_cascade`, `default_clearing`,

Examples

```
ea <- c(1/2,5/8,3/4)
el <- c(3/2,1/2,1/2)
x <- 0.5
L <- matrix(c(0,x,1-x,1-x,0,x,x,1-x,0),nrow=3)
default(L,ea,el)
default(L,ea,el,"cascade")
```

default_cascade *Default Cascade*

Description

Computes bank defaults via the default cascade algorithm.

Usage

```
default_cascade(L, ea, el = 0, recoveryrate = 0)
```

Arguments

L	liability matrix
ea	vector of external assets
el	vector of external liabilites (default 0)
recoveryrate	recovery rate in [0,1] (defaults to 0)

Value

vector indicating which banks default (1=default, 0= no default)

Examples

```
ea <- c(1/2,5/8,3/4)
el <- c(3/2,1/2,1/2)
x <- 0.5
L <- matrix(c(0,x,1-x,1-x,0,x,x,1-x,0),nrow=3)
default_cascade(L,ea,el)
```

default_clearing *Clearing Vector with Bankruptcy Costs*

Description

Computes bank defaults for the clearing vector approach without and with bankruptcy costs (Eisenberg and Noe, 2001), (Rogers and Veraart, 2013).

Usage

```
default_clearing(L, ea, el = 0, alpha = 1, beta = 1)
```

Arguments

L	Liabilities matrix
ea	Vector of external assets
el	Vector of external liabilites (default 0)
alpha	1-proportional default costs on external assets in [0, 1] (default to 1).
beta	1-proportional default costs on interbank assets in [0, 1] (defaults to 1).

Details

Without bankruptcy costs the approach of Eisenberg and Noe (2001) is used using a linear programme. With bankruptcy costs, the implementation is based on the Greatest Clearing Vector Algorithm (GA), see Definition 3.6, Rogers & Veraart (2013).

Value

A list consisting of a vector indicating which banks default (1=default, 0= no default) and the greatest clearing vector.

References

Eisenberg, L. and Noe, T.H. (2001). Systemic risk in financial systems. Management Science 47, 236–249.

Rogers, L. C. G. and Veraart, L. A. M. (2013) Failure and Rescue in an Interbank Network, Management Science 59 (4), 882–898.

Examples

```
ea <- c(1/2,5/8,3/4)
el <- c(3/2,1/2,1/2)
x <- 0.5
L <- matrix(c(0,x,1-x,1-x,0,x,x,1-x,0),nrow=3)
default_clearing(L,ea,el)
default_clearing(L,ea,el, alpha=0.5, beta=0.7)
```

---

diagnose *Outputs Effective Sample Size Diagonistics for MCMC run*

Description

Computes the Effective Sample Size using the method `effectiveSize` in of the package `coda`.

Usage

```
diagnose(res)
```

Arguments

res	output from sample_HierarchicalModel.

Details

Currently only works with L where the diagonal is 0. The function ignores the diagonal and tries to determine from the row and column sums which parts of the matrix are 0.

Value

No return value. Called for printing the diagnostics.

ERE_step_cycle *Does one Gibbs Step on a cycle*

Description

Execute one Gibbs step on a cycle keeping row and column sums fixed

Usage

```
ERE_step_cycle(r, c, L, lambda, p, eps = 1e-10)
```

Arguments

r	Row indies of cycle, starting at 0 (vector of length k)
c	Column indices of cycle, starting at 0 (vector of length k)
L	nxn matrix with nonnegative values (will be modified)
lambda	nxn matrix of intensities
p	nxn matrix of probabilities (must be in [0,1] and 0 on diagonal)
eps	Threshold for values to be interpreted as equal to 0 (default = 1e-10)

Value

no return value

Examples

```
L=matrix(rexp(9),nrow=3)
lambda <- matrix(0.5,nrow=3,ncol=3)
p <- matrix(0.7, nrow=3,ncol=3)
ERE_step_cycle(r=c(0,1),c=c(1,2),L=L,lambda=lambda,p=p)
ERE_step_cycle(r=c(0,1,2),c=c(0,1,2),L=L,lambda=lambda,p=p)
ERE_step_cycle(r=c(0,1,2),c=c(2,1,0),L=L,lambda=lambda,p=p)
```

`findFeasibleMatrix` *Finds a Nonnegative Matrix Satisfying Row and Column Sums*

Description

Given row and column sums and a matrix p which indicates which elements of the matrix can be present, this function computes a nonnegative matrix that match these row and column sums. If this is not possible then the function returns an error message.

Usage

```
findFeasibleMatrix(r, c, p, eps = 1e-09)
```

Arguments

`r`	vector of row sums (nonnegative
`c`	vector of column sums (nonnegative)
`p`	matrix of probabilities (must be in [0,1]), matching the dimensions of r and c. Values of p=0 are interpreted that the corresponding matrix elements have to be 0. Note: p=1 does not force the corresponding matrix element to exist.
`eps`	row and col sums can at most be different by eps. Default 1e-9.

Details

The function transforms the problem into a Maximum Flow problem of a graph and uses the Edmonds-Karps algorithm to solve it. If the error message "Could not find feasible matrix." is produced then this could be due to p imposing disconnected components in the graph implied by row and column sums that are not compatible with the row and column sums..

Value

A feasible matrix.

Examples

```
p=matrix(c(1,0,0,1),nrow=2)
findFeasibleMatrix(c(1,1),c(1,1),p=p)

n <- 4
M <- matrix(nrow=n,ncol=n,rexp(n*n)*(runif(n*n)>0.6))
M
r <- rowSums(M)
c <- colSums(M)
Mnew <- findFeasibleMatrix(r=r,c=c,p=(M>0)*0.5)
Mnew
rowSums(M);rowSums(Mnew)
colSums(M);colSums(Mnew)
```

findFeasibleMatrix_targetmean

Creates a feasible starting matrix with a desired mean average degree

Description

This extension of findFeasibleMatrix attempts to create a feasible matrix where a certain proportion of the entries is positive. There is no guarantee that this proportion is achieved. If it is not possible then this matrix will report a warning and simply return the matrix constructed by findFeasibleMatrix.

Usage

```
findFeasibleMatrix_targetmean(r, c, p, eps = 1e-09, targetmean = 0.3)
```

Arguments

r	vector of row sums (nonnegative
c	vector of column sums (nonnegative)
p	matrix of probabilities (must be in [0,1]), matching the dimensions of r and c. Values of p=0 are interpreted that the corresponding matrix elements have to be 0. Note: p=1 does not force the corresponding matrix element to exist.
eps	row and col sums can at most be different by eps. Default 1e-9.
targetmean	Average proportion of positive entries of the resulting matrix. Defaults to 0.3

Value

A feasible matrix.

genL

Generate Liabilities Matrix from Prior

Description

Generates a libabilities matrix using a the prior distribution from a given model for p and lambda.

Usage

```
genL(model)
```

Arguments

model	a model for p and lambda.

Value

A list consisting of a liabilities matrix and the parameter vector theta.

Examples

```
n <- 5
m <- Model.Indep.p.lambda(Model.p.BetaPrior(n),
                          Model.lambda.GammaPrior(n,scale=1e-1))
genL(m)
```

`getfeasibleMatr` *Creates a feasible starting matrix*

Description

Creates a matrix with nonnegative entries, given row and column sums and 0 on the diagonal. Superseeded by the more flexible findFeasibleMatrix.

Usage

```
getfeasibleMatr(L, A)
```

Arguments

`L`	Vector of row sums
`A`	Vector of column sums

Value

A matrix with nonnegative entries and given row/column sums and 0 on the diagonal.

Examples

```
getfeasibleMatr(c(0.5,1,0),c(0.5,0,1))
getfeasibleMatr(rep(1,4),rep(1,4))
getfeasibleMatr(2^(1:3),2^(3:1))
getfeasibleMatr(1:5,1:5)
getfeasibleMatr(1:5,5:1)
```

`GibbsSteps_kcycle` *Gibbs sampling step of a matrix in the ERE model*

Description

The sampling is conditional on row and column sums and uses k-cycle steps. Then dimensions of L, lambda and p must match.

Usage

```
GibbsSteps_kcycle(L, lambda, p, it = 1000L, eps = 1e-10, debug = 0L)
```

Arguments

`L`	Starting matrix - will be modified to contain the results.
`lambda`	Matrix of intensities
`p`	Matrix of probabilities (must be in [0,1])
`it`	Number of iterations (default=1000)
`eps`	Threshold for values to be interpreted as equal to 0 (default = 1e-10)
`debug`	Should addtional debug information be printed? (0 no output, 1 output debug information)

Value

no return value

Examples

```
L <- matrix(c(1,2,3,4,5,6,7,8,9),nrow=3)
diag(L) <- 0
lambda <- matrix(0.5,nrow=3,ncol=3)
p <- matrix(0.7, nrow=3,ncol=3)
diag(p) <- 0
GibbsSteps_kcycle(L=L,lambda=lambda,p=p)
L
L <- matrix(1:16,nrow=4)
diag(L) <- 0
lambda <- matrix(0.5,nrow=4,ncol=4)
p <- matrix(0.25, nrow=4,ncol=4)
diag(p) <- 0
GibbsSteps_kcycle(L=L,lambda=lambda,p=p)
L
```

`Model.additivelink.exponential.fitness` *Fitness model for liabilities matrix*

Description

Assumes a diagonal consisting of 0s.

Usage

```
Model.additivelink.exponential.fitness(
  n,
  alpha,
  beta,
  gamma = 1,
  lambdaprior,
  sdpropfitness = 1/sqrt(n)
)
```

Arguments

`n`	Number of nodes in the model.
`alpha`	Exponent of the power law of the degree distribution. Must be <0.
`beta`	Lower endpoint of the relative expected out degree (expected out degree divided by n-1). Must be >=0.
`gamma`	Upper endpoint of the relative expected out degree (expected out degree divided by n-1). Must be at least beta and at most 1.
`lambdaprior`	Prior on zeta and eta. For the type of object required see `Model.fitness.genlambdaparprior`.
`sdpropfitness`	Standard deviation for the log-normally distributed multiplicative proposals for Metropoli-Hastings updates of the fitness. Defaults to `1/sqrt{n}`.

Value

A model to be used by sample_HierarchicalModel. This is a list of functions. It includes a function accrates() that repors acceptance rates for the Metropolis-Hasting steps involved.

Examples

```
mod <- Model.additivelink.exponential.fitness(10,alpha=-2.5,beta=0.1,
               lambdaprior=Model.fitness.genlambdaparprior(ratescale=1e3))
theta <- mod$rtheta()
L <- genL(mod)
l <- rowSums(L$L)
a <- colSums(L$L)
## increase number of samples and thinning in real examples
res <- sample_HierarchicalModel(l=l,a=a,model=mod,nsamples=4,thin=50)
mod$accrates()
```

Model.fitness.conditionalmeandegree

Mean out-degree of a node with given fitness in the fitness model

Description

Computes the mean out-degree of a node with given fitness `x` in the fitness model implemented in `Model.additivelink.exponential.fitness`. The function returns the mean out-degree divided by n-1.

Usage

```
Model.fitness.conditionalmeandegree(x, alpha, beta, gamma = 1)
```

Arguments

`x`	Fitness of node. A nonegative number.
`alpha`	Exponent of the power law of the degree distribution. Must be <0.
`beta`	Lower endpoint of the relative expected out degree (expected out degree divided by n-1). Must be >=0.
`gamma`	Upper endpoint of the relative expected out degree (expected out degree divided by n-1). Must be at least beta and at most 1.

Value

Mean out-degree divided by n-1.

Model.fitness.genlambdaparprior

Prior distribution for eta and zeta in the fitness model

Description

Assumes a uniform distribution on the shape parameter `zeta` and an exponential distribution on the scale parameter `eta`. To be used as prior for `Model.additivelink.exponential.fitness`.

Usage

```
Model.fitness.genlambdaparprior(
  shapemin = 0.75,
  shapemax = 1.5,
  ratescale,
  sdshapeprob = 0.1,
  sdpropscale = 0.1
)
```

Arguments

shapemin	Minimal Value of the shape parameter. Default: 0.75.
shapemax	Maximal Value of the shape parameter. Default: 1.5.
ratescale	Rate parameter for the prior distribution of the scale parameter. In the model this is on the same scale as the entries of L
sdshapeprob	Standard deviation for the additivel normally distributed random walk proposal for the shape parameter. Defaults to 0.1.
sdpropscale	Standard deviation for the multiplicative lognormal proposals for the scale parameter.

Value

list of functions necessary for constructing Metropolis-Hastings updates.

Model.fitness.meandegree

Mean out-degree of a random node the fitness model

Description

Computes the relative mean out-degree of a randomly chosen node given fitness x in the fitness model implemented in Model.additivelink.exponential.fitness. The function returns the mean out-degree divided by n-1.

Usage

```
Model.fitness.meandegree(alpha, beta, gamma = 1)
```

Arguments

alpha	Exponent of the power law of the degree distribution. Must be <0.
beta	Lower endpoint of the relative expected out degree (expected out degree divided by n-1). Must be >=0.
gamma	Upper endpoint of the relative expected out degree (expected out degree divided by n-1). Must be at least beta and at most 1.

Value

Mean out-degree divided by n-1.

Model.Indep.p.lambda *Combination of Independent Models for p and lambda*

Description

Combination of Independent Models for p and lambda

Usage

```
Model.Indep.p.lambda(model.p, model.lambda)
```

Arguments

model.p	model for p.
model.lambda	model for lambda.

Value

the resulting model.

Examples

```
n <- 5
m <- Model.Indep.p.lambda(Model.p.BetaPrior(n),
                          Model.lambda.GammaPrior(n,scale=1e-1))
genL(m)
```

Model.lambda.constant *Model for a Constant lambda*

Description

This model assumes that the parameter lambda is known.

Usage

```
Model.lambda.constant(lambda, n)
```

Arguments

lambda	paramer for the size of the liabilities. Either a matrix of dimension n or a single numeric value.
n	dimension of matrix.

Value

the resulting model.

Examples

```
m <- Model.lambda.constant(n=5,lambda=0.25)
m$matr(m$rtheta())
lambda<-matrix(c(NA,1,1,1e-4,NA,1e-4,1e4,1e4,NA),nrow=3)
m <- Model.lambda.constant(n=3,lambda=lambda)
m$matr(m$rtheta())
```

Model.lambda.constant.nonsquare

Model for a Constant lambda and Non-Square Matrices

Description

This model assumes that the parameter lambda is known.

Usage

```
Model.lambda.constant.nonsquare(lambda, nrow, ncol)
```

Arguments

lambda	paramer for the size of the liabilities. A single numeric value.
nrow	number of rows of the matrix.
ncol	number of columns of the matrix.

Value

the resulting model.

Examples

```
m <- Model.lambda.constant.nonsquare(nrow=5,ncol=3,lambda=0.25)
m$matr(m$rtheta())
```

`Model.lambda.GammaPrior`

Model with Gamma Prior on Lambda

Description

Assumes that all elements of lambda are equal to a parameter θ, which has a Gamma prior.

Usage

```
Model.lambda.GammaPrior(n, shape = 1, scale = 1)
```

Arguments

`n`	dimension of matrix
`shape`	shape paramer for prior on θ. Default 1.
`scale`	scale paramer for prior on θ. Default 1.

Value

the resulting model.

`Model.lambda.Gammaprior_mult`

Model Using Multiple Independent Components

Description

Assumes a multivariate hyperparameter θ with each component following an independent Beta distribution. A matrix indicates which component θ is used for what component of lambda.

Usage

```
Model.lambda.Gammaprior_mult(Ilambda, shape = 1, scale = 1)
```

Arguments

`Ilambda`	matrix consisting of integers that describe which component of $theta$ is used for a given position in the matrix. Must consist of nonnegative integers using all integers in the range.
`shape`	shape paramer for prior on θ. Default 1.
`scale`	scale paramer for prior on θ. Default 1.

Value

the resulting model.

Model.p.BetaPrior *Model for a Random One-dimensional p*

Description

Assumes a Beta prior on the one-dimensional link existence probabilities p. This model has a one-dimensional parameter.

Usage

```
Model.p.BetaPrior(n, shape1 = 1, shape2 = 1)
```

Arguments

n	dimension of matrix.
shape1	first parameter of Beta prior. Default 1.
shape2	second parameter of Beta prior. Default 1.

Value

the resulting model.

Examples

```
m <- Model.p.BetaPrior(5)
m$matr(m$rtheta())
```

Model.p.Betaprior_mult *Model Using Multiple Independent Components*

Description

Assumes a multivariate hyperparameter θ with each component following an independent Beta distribution. A matrix indicates which component θ is used for what component of p.

Usage

```
Model.p.Betaprior_mult(Ip, shape1 = 1, shape2 = 1)
```

Arguments

Ip	matrix consisting of integers that describe which component of $theta$ is used for a given position in the matrix. Must consist of nonnegative integers (0 encoding forced 0s in the matrix), using all integers in the range.
shape1	first parameter of Beta prior on $theta$. Default 1.
shape2	second parameter of Beta prior $theta$. Default 1.

Value

the resulting model.

`Model.p.constant` *Model for a Constant p*

Description

This model assumes that the link existence probabilities of the matrix are known.

Usage

```
Model.p.constant(n, p)
```

Arguments

`n`	dimension of matrix.
`p`	existence probability of a link. Either a matrix of dimension n or a single numeric value. A single numeric value leads to a matrix of existence probabilities that has 0 on the diagonal.

Value

the resulting model.

Examples

```
m <- Model.p.constant(5,0.25)
m$matr(m$rtheta())

p <- matrix(c(0,0.99,0.99,0.5,0.5,0,0.01,0.01,0),nrow=3)
m <- Model.p.constant(5,p)
m$matr(m$rtheta())
```

`Model.p.constant.nonsquare`
Model for a constant p and Non-Square Matrices

Description

This model assumes that the link existence probabilities of the matrix are known.

Usage

```
Model.p.constant.nonsquare(nrow, ncol, p)
```

Arguments

nrow	number of rows of the matrix.
ncol	number of columns of the matrix.
p	existence probability of a link. A single numeric value.

Value

the resulting model.

Examples

```
m <- Model.p.constant.nonsquare(5,3,0.25)
m$matr(m$rtheta())
```

Model.p.Fitness.Servedio

Multiplicative Fitness Model for Power Law

Description

This model has a power law of the degree distribution with a parameter α and is tuned to a desired link existence probability. It is based on a fitness model.

Usage

```
Model.p.Fitness.Servedio(n, alpha, meandegree, sdprop = 0.1)
```

Arguments

n	dimension of matrix.
alpha	exponent for power law. Must be <=-1.
meandegree	overall mean degree (expected degree divided by number of nodes). Must be in (0,1).
sdprop	standard deviation of updated steps.

Details

Every node i has a fitness θ_i being an independent realisation of a U[0,1] distribution. The probability of a link between a node with fitness x and a node with fitness y is g(x)g(y) where g is as follows. If $\alpha = -1$ then

$$g(x) = g0 * \exp(-\log(g0) * x)$$

Otherwise,

$$g(x) = (g0^(\alpha + 1) + (1 - g0^(\alpha + 1)) * x)^(1/(\alpha + 1))$$

where $g0$ is tuned numerically to achieve the desired overall mean degree.

Updating of the model parameters in the MCMC setup is done via a Metropolis-Hastings step, adding independent centered normal random variables to each node fitness in θ.

Value

the resulting model.

References

Servedio V. D. P. and Caldarelli G. and Butta P. (2004) Vertex intrinsic fitness: How to produce arbitrary scale-free networks. *Physical Review E* 70, 056126.

Examples

```
n <- 5
mf <- Model.p.Fitness.Servedio(n=n,alpha=-2.5,meandegree=0.5)
m <- Model.Indep.p.lambda(model.p=mf,
                          model.lambda=Model.lambda.GammaPrior(n,scale=1e-1))
x <- genL(m)
l <- rowSums(x$L)
a <- colSums(x$L)
res <- sample_HierarchicalModel(l,a,model=m,nsamples=10,thin=10)
```

---

sample_ERE *Sample from the ERE model with given row and column sums*

Description

Samples from the Erdos Reny model with Exponential weights and known marginals. Runs a Gibbs sampler to do this. A starting liabilities is generated via `getfeasibleMatr` before `steps_ERE` is called.

Usage

```
sample_ERE(l, a, p, lambda, nsamples = 10000, thin = 1000, burnin = 10000)
```

Arguments

`l`	vector of interbank libabilities
`a`	vector of interbank assets
`p`	probability of existence of a link (either a numerical value or a matrix). A single numerical value is converted into a matrix with 0s on the diagonal.
`lambda`	instensity parameters - either a numerical value or a matrix with positive entries)
`nsamples`	Number of samples to return.
`thin`	Frequency at which samples should be generated (default=1, every step)
`burnin`	Number of initial steps to discard.

Value

List of simulation results

Examples

```
l <- c(1,2.5,3)
a <- c(0.7,2.7,3.1)
L <- sample_ERE(l,a,p=0.5,lambda=0.25,nsamples=5,thin=20,burnin=10)
L
```

---

sample_HierarchicalModel

Sample from Hierarchical Model with given Row and Column Sums

---

Description

Sample from Hierarchical Model with given Row and Column Sums

Usage

```
sample_HierarchicalModel(
  l,
  a,
  L_fixed = NA,
  model,
  nsamples = 10000,
  thin = choosethin(l = l, a = a, L_fixed = L_fixed, model = model, matrpertheta =
    matrpertheta, silent = silent),
  burnin = NA,
  matrpertheta = length(l)^2,
  silent = FALSE,
  tol = .Machine$double.eps^0.25
)
```

Arguments

l	observed row sum
a	observerd column sum
L_fixed	Matrix containing known values of L, where NA signifies that an element is not known. If L_fixed equates to NA (the default) then no values are assumed to be known.
model	Underlying model for p and lambda.
nsamples	number of samples to return.
thin	how many updates of theta to perform before outputting a sample.

burnin	number of iterations for the burnin. Defaults to 5 of the steps in the sampling part.
matrpertheta	number of matrix updates per update of theta.
silent	(default FALSE) suppress all output (including progress bars).
tol	tolerance used in checks for equality. Defaults to .Machine$double.eps^0.25.

Value

The resulting samples. A list with the first element, L, giving the samples of matrices, and the second element, theta, giving the samples of the hyperparameter (if hyperparameters are present).

Examples

```
n <- 10
m <- Model.Indep.p.lambda(Model.p.BetaPrior(n),
                          Model.lambda.GammaPrior(n,scale=1e-1))
x <- genL(m)
l <- rowSums(x$L)
a <- colSums(x$L)

res <- sample_HierarchicalModel(l,a,model=m)

# fixing one values
L_fixed <- matrix(NA,ncol=n,nrow=n)
L_fixed[1,2:5] <- x$L[1,2:5]

res <- sample_HierarchicalModel(l,a,model=m,L_fixed=L_fixed,
                                nsamples=1e2)
sapply(res$L,function(x)x[1,2:5])
```

---

steps_ERE *Perform Steps of the Gibbs Sampler of the ERE model*

Description

Runs a Gibbs sampler in the Erdos Reny model with Exponential weights (ERE model) and fixed marginals. The algorithm starts from a given matrix.

Usage

```
steps_ERE(L, p, lambda, nsamples = 10000, thin = 1000, burnin = 10000)
```

Arguments

L	Starting matrix for the Gibbs sampler. Implicitly defines the fixed marginals.
p	A matrix with entries in [0,1]
lambda	A matrix with nonnegative entries
nsamples	Number of samples to return.
thin	Frequency at which samples should be generated (default=1, every step)
burnin	Number of initial steps to discard.

Value

List of simulation results

See Also

sample_ERE

Examples

```
L <- matrix(rexp(4*4),nrow=4,ncol=4); diag(L)=0;
p <- matrix(0.5,nrow=4,ncol=4); diag(p) <-0;
lambda <- matrix(1,nrow=4,ncol=4); diag(lambda)<-0;

L <- steps_ERE(L=L,p=p,lambda=lambda,nsamples=5,thin=50,burnin=20)
L
```

Index