Package 'Datasmith'

January 6, 2017

Version 1.0-1

Date 2016-12-31

Title Tools to Complete Euclidean Distance Matrices

Description Implements several algorithms for Euclidean distance matrix completion, Sensor Network Localization, and sparse Euclidean distance matrix completion using the minimum spanning tree.

Author Adam Rahman and Wayne Oldford <rwoldford@uwaterloo.ca>

Maintainer Adam Rahman <a45rahma@uwaterloo.ca>

Depends R (>= 3.1.0)

Imports Matrix, igraph, lbfgs, truncnorm, MASS, nloptr, vegan

KeepSource yes

License GPL-2 | GPL-3

Encoding UTF-8

RoxygenNote 5.0.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2017-01-06 14:44:20

R topics documented:

`colamdR` *Column Approximate Minimum Degree Permutation*

Description

`colamdR` returns the column approximate minimum degree permutation of a sparse matrix S. The permutation of S, S[,p], will result in LU factors sparser than S.

Usage

```
colamdR(M)
```

Arguments

`M`	A matrix to be permuted.

Details

This is an implementation of the colamd function available in SuiteSparse, and also implemented in Matlab.

Value

A vector containing the column minimum degree permutation of the matrix M.

References

The authors of the code for "colamd" are Stefan I. Larimore and Timothy A. Davis (davis@cise.ufl.edu), University of Florida.

Examples

```
M <- matrix(c(1,1,0,0,1,0,0,1,0,1,1,1,1,1,0,0,1,0,1,0),ncol=4)
p <- colamdR(M)
M[,p]
```

dpf *Dissimilarity Parameterization Formulation*

Description

dpf returns a completed Euclidean Distance Matrix D, with dimension d, from a partial Euclidean Distance Matrix using the methods of Trosset (2000)

Usage

```
dpf(D, d, toler = 1e-08, lower = NULL, upper = NULL, retainMST = FALSE)
```

Arguments

D	An nxn partial-distance matrix to be completed. D must satisfy a list of conditions (see details), with unkown entries set to NA
d	The dimension for the resulting completion.
toler	The convergence tolerance of the algorithm. Set to a default value of 1e-8
lower	An nxn matrix containing the lower bounds for the unknown entries in D. If NULL, lower is set to be a matrix of 0s.
upper	An nxn matrix containing the upper bounds of the unknown entries in D. If NULL, upper[i,j] is set to be the shortest path between node i and node j.
retainMST	D logical input indicating if the current minimum spanning tree structure in D should be retained. If TRUE, a judicious choice of Lower is calculated internally such that the MST is retained.

Details

This is an implementation of the Dissimilarity Parameterization Formulation (DPF) for Euclidean Distance Matrix Completion, as proposed in 'Distance Matrix Completion by Numerical Optimization' (Trosset, 2000).

The method seeks to minimize the following:

$$\sum_{i=1}^{d}(\lambda_i - \lambda_{max}) + \sum_{i=d+1}^{n} \lambda_i^2$$

where λ_i are the ordered eigenvalues of $\tau(\Delta)$. For details, see Trosset(2000)

The matrix D is a partial-distance matrix, meaning some of its entries are unknown. It must satisfy the following conditions in order to be completed:

- diag(D) = 0
- If a_{ij} is known, $a_{ji} = a_{ij}$
- If a_{ij} is unknown, so is a_{ji}
- The graph of D must be connected. If D can be decomposed into two (or more) subgraphs, then the completion of D can be decomposed into two (or more) independent completion problems.

Value

D	The completed distance matrix with dimensionality d
optval	The minimum function value achieved during minimization (see details)

References

Trosset, M.W. (2000). Distance Matrix Completion by Numerical Optimization.Computational Optimization and Applications, 17, 11–22, 2000.

Examples

```
#Complete the matrix D in three dimensions.
set.seed(1337)
D <- matrix(c(0,3,4,3,4,3,
              3,0,1,NA,5,NA,
              4,1,0,5,NA,5,
              3,NA,5,0,1,NA,
              4,5,NA,1,0,5,
              3,NA,5,NA,5,0),byrow=TRUE, nrow=6)
d <- 3
toler <- 1e-8

dpf(D,d,toler)
```

---

edm2gram *Linear Matrix Operator*

Description

edm2gram Linear transformation of a Euclidean Distance Matrix to a Gram Matrix

Usage

```
edm2gram(D)
```

Arguments

D	A Euclidean Distance Matrix

Details

While we specify that the input should be a Euclidean Distance Matrix (as this results in a Gram Matrix) the domain of edm2gram is the set of all real symmetric matrices. This function is particularly useful as it has the following property:

$$edm2gram(D_n^-) = B_n^+$$

where D_n^- is the space of symmetric, hollow matrices, negative definite on the space spanned by $x'e = 0$ and B_n^+ is the space of centered positive definite matrices.

We can combine these two properties with a well known result: If D is a real symmetric matrix with 0 diagonal (call this matrix pre-EDM), then D is a Euclidean Distance Matrix iff D is negative semi-definite on D_n^-.

Using this result, combined with the properties of edm2gram we therefore have that D is an EDM iff D is pre-EDM and $edm2gramD$ is positive semi-definite.

Value

G A Gram Matrix, where G = XX', and X is an nxp matrix containing the point configuration.

Examples

```
XY <- cbind(runif(100,0,1),runif(100,0,1))
D <- dist(XY)
edm2gram(as.matrix(D))
```

---

edm2psd *Linear Matrix Operator*

Description

edm2psd Convert an Euclidean Distance Matrix to a Positive Semi-definite Matrix

Usage

```
edm2psd(D, V = NULL)
```

Arguments

D	A matrix in the set D_n^-.
V	A projection matrix satisfying V'1 = 0 and VV' = I

Details

For a matrix D in D_n^-, edm2psd will be in the space of positive semi-definite matrices. Therefore, if D also has zero diagonal, we have the following property:

D is a Euclidean Distance Matrix if and only if edm2psd is positive semi-definite.

This operator gives us another method to characterize the existence of a Euclidean distance matrix.

Value

S A symmetric, positive semi-definite matrix

See Also

psd2edm edm2gram

Examples

```
XY <- cbind(runif(100,0,1),runif(100,0,1))
D <- dist(XY)
edm2psd(as.matrix(D))
```

---

getConfig *Create a Point Configuration from a Distance Matrix*

Description

getConfig - given an nxn Euclidean distance matrix, produces a d-dimensional point configuration of size n via eigendecomposition

Usage

```
getConfig(D, d)
```

Arguments

D	an nxn Euclidean distance matrix
d	the dimension for the configuration

Details

Given a distance matrix D, transform to a semi-definite matrix S using the linear transformation $\tau(D)$. Using S, compute the eigen-decomposition $S = ULV'$, where L is a diagonal matrix containing the singular-values of S, and the columns of U contain the eigen-vectors. A point configuration X is then computed as:

$$X = US^{.5}$$

To compute a configuration in d dimensions, the first d eigenvalues of S are used.

Value

Y	an nxd matrix containing the d-dimensional point configuration
Accuracy	the ratio of the sum of retained eigenvalues to the sum of all n eigenvalues obtained during decomposition

Examples

```
set.seed(1337)
D <- matrix(c(0,3,4,3,4,3,
              3,0,1,NA,5,NA,
              4,1,0,5,NA,5,
              3,NA,5,0,1,NA,
              4,5,NA,1,0,5,
              3,NA,5,NA,5,0),byrow=TRUE, nrow=6)
d <- 3
DStar <- dpf(D,d)$D

getConfig(DStar,3)
```

---

gram2edm *Linear Matrix Operator*

Description

gram2edm Inverse Operator of edm2gram

Usage

```
gram2edm(B)
```

Arguments

B	A centered, positive semi-definite matrix.

Details

The edm2gram function performs the following transformation:

$$edm2gram(D_n^-) = B_n^+$$

where D_n^- is the space of symmetric, hollow matrices, negative definite on the space spanned by $x'e = 0$ and B_n^+ is the space of centered positive definite matrices.

The gram2edm function performs the inverse operation, taking a matrix in B_n^+ and transforming it to a matrix in D_n^-.

$$gram2edm(B_n^+) = D_n^-$$

Therfore, gram2edm on B_n^+ is the inverse operator of edm2gram on D_n^-.

Value

D A matrix in D_n^-. If the input matrix B is a gram matrix, D is a Euclidean Distance Matrix.

See Also

edm2gram

Examples

```
X <- cbind(runif(100,0,1),runif(100,0,1))
G <- X %*% t(X)
gram2edm(G)
```

---

grs *Guided Random Search*

Description

grs performs Euclidean Distance Matrix Completion using the guided random search algorithm of Rahman & Oldford. Using this method will preserve the minimum spanning tree in the partial distance matrix.

Usage

```
grs(D, d)
```

Arguments

D	An nxn partial-distance matrix to be completed. D must satisfy a list of conditions (see details), with unkown entries set to NA
d	The dimension for the resulting completion.

Details

The matrix D is a partial-distance matrix, meaning some of its entries are unknown. It must satisfy the following conditions in order to be completed:

- diag(D) = 0
- If a_{ij} is known, $a_{ji} = a_{ij}$
- If a_{ij} is unknown, so is a_{ji}
- The graph of D must contain ONLY the minimum spanning tree distances

Value

P	The completed point configuration in dimension d
D	The completed Euclidean distance matrix

References

Rahman, D., & Oldford, R.W. (2016). Euclidean Distance Matrix Completion and Point Configurations from the Minimal Spanning Tree.

Examples

```
#D matrix containing only the minimum spanning tree
D <- matrix(c(0,3,NA,3,NA,NA,
              3,0,1,NA,NA,NA,
              NA,1,0,NA,NA,NA,
              3,NA,NA,0,1,NA,
              NA,NA,NA,1,0,1,
              NA,NA,NA,NA,1,0),byrow=TRUE, nrow=6)
d <- 3
grs(D,d)
```

`mst` *Compute Minimum Spanning Tree*

Description

`mst` Compute a minimum spanning tree using Prim's algorithm

Usage

```
mst(D)
```

Arguments

`D`	A distance matrix

Value

MST a data frame object of 3 columns containing the parent nodes, child nodes, and corresponding weight of the MST edge

Examples

```
X <- runif(10,0,1)
Y <- runif(10,0,1)
D <- dist(cbind(X,Y))

mst(as.matrix(D))
```

mstLB *Minimum Spanning Tree Preserving Lower Bound*

Description

mstLB Returns an nxn matrix containing the lower bounds for all unknown entries in the partial distance matrix D such that the minimum spanning tree of the partial matrix D is preserved upon completion.

Usage

```
mstLB(D)
```

Arguments

D	An nxn partial distance matrix to be completed

Details

The insight in constructing the lower bound is drawn from single-linkage clustering. Every edge in a spanning tree separates the vertices into two different groups, depending on which points remain connected to either one vertex or the other of that edge. Because the tree is a minimum spanning tree, if we select the largest edge, then the distance between any vertex of one group and any vertex of the other group must be at least as large as that of the the largest edge. This gives a lower bound for these distances that will preserve that edge in the minimum spanning tree. The same reasoning is applied recursively to each separate group, thus producing a lower bound on all edges.

The details of the algorithm can be found in Rahman & Oldford (2016).

Value

Returns an nxn matrix containing the lower bound for the unknown entries in D

References

Rahman, D., & Oldford R.W. (2016). Euclidean Distance Matrix Completion and Point Configurations from the Minimal Spanning Tree.

Examples

```
D <- matrix(c(0,3,4,3,4,3,
              3,0,1,NA,5,NA,
              4,1,0,5,NA,5,
              3,NA,5,0,1,NA,
              4,5,NA,1,0,5,
              3,NA,5,NA,5,0),byrow=TRUE, nrow=6)
mstLB(D)
```

mstUB *Shortest Path Upper Bound*

Description

mstUB Compute the shortest path upper bound for all unknown entries in a partial distance matrix

Usage

```
mstUB(A)
```

Arguments

A	A (connected) partial distance matrix, with unknown entries set to Inf

Details

This function uses the shortest.paths() function, available in the igraph package.

Value

UB A matrix containing the upper bounds for only the unknown entries. All other entries will be set to Inf.

Examples

```
A <- dist(cbind(rnorm(10,0,1),rnorm(10,0,1)))
mstUB(as.matrix(A))
```

npf *Nonparametric Position Formulation*

Description

npf returns a completed Euclidean Distance Matrix D, with dimension d, from a partial Euclidean Distance Matrix using the methods of Fang & O'Leary (2012)

Usage

```
npf(D, A = NA, d, dmax = (nrow(D) - 1), decreaseDim = 1, stretch = NULL,
  method = "Linear", toler = 1e-08)
```

Arguments

D	An nxn partial-distance matrix to be completed. D must satisfy a list of conditions (see details), with unkown entries set to NA.
A	a weight matrix, with $h_{ij} = 0$ implying a_{ij} is unknown. Generally, if a_{ij} is known, $h_{ij} = 1$, although any non-negative weight is allowed.
d	the dimension of the resulting completion
dmax	the maximum dimension to consider during dimension relaxation
decreaseDim	during dimension reduction, the number of dimensions to decrease each step
stretch	should the distance matrix be multiplied by a scalar constant? If no, stretch = NULL, otherwise stretch is a positive scalar
method	The method used for dimension reduction, one of "Linear" or "NLP".
toler	convergence tolerance for the algorithm

Details

This is an implementation of the Nonconvex Position Formulation (npf) for Euclidean Distance Matrix Completion, as proposed in 'Euclidean Distance Matrix Completion Problems' (Fang & O'Leary, 2012).

The method seeks to minimize the following:

$$||A \cdot (D - K(XX'))||_F^2$$

where the function K() is that described in gram2edm, and the norm is Frobenius. Minimization is over X, the nxp matrix of node locations.

The matrix D is a partial-distance matrix, meaning some of its entries are unknown. It must satisfy the following conditions in order to be completed:

- diag(D) = 0
- If a_{ij} is known, $a_{ji} = a_{ij}$
- If a_{ij} is unknown, so is a_{ji}
- The graph of D must be connected. If D can be decomposed into two (or more) subgraphs, then the completion of D can be decomposed into two (or more) independent completion problems.

Value

D	an nxn matrix of the completed Euclidean distances
optval	the minimum value achieved of the target function during minimization

See Also

gram2edm

Examples

```
D <- matrix(c(0,3,4,3,4,3,
              3,0,1,NA,5,NA,
              4,1,0,5,NA,5,
              3,NA,5,0,1,NA,
              4,5,NA,1,0,5,
              3,NA,5,NA,5,0),byrow=TRUE, nrow=6)

A <- matrix(c(1,1,1,1,1,1,
              1,1,1,0,1,0,
              1,1,1,1,0,1,
              1,0,1,1,1,0,
              1,1,0,1,1,1,
              1,0,1,0,1,1),byrow=TRUE, nrow=6)
d <- 3
dmax <- 5

npf(D,A,d,dmax)
```

---

primPath *Minimum Spanning Tree Path*

Description

primPath Given a starting node, creates the minimum spanning tree path through a point configuration.

Usage

```
primPath(A, start)
```

Arguments

A	the distance matrix for which the minimum spanning tree path will be created
start	the starting node for the path

Details

Given a starting node, compute Prim's algorithm, resulting in the path taken to construct the minimum spanning tree.

Value

return a 2x(n-1) matrix, where row 1 contains the parent nodes of the MST path, and row 2 contains the corresponding child nodes.

Examples

```
A <- dist(cbind(rnorm(100,0,1),rnorm(100,0,1)))
primPath(as.matrix(A),1)
primPath(as.matrix(A),2)
```

---

psd2edm *Linear Matrix Operator*

Description

psd2edm Transform a positive semi-definite matrix to a Euclidean Distance Matrix

Usage

```
psd2edm(S, V = NULL)
```

Arguments

S	A symmetric, positive semi-definite matrix
V	A projection matrix satisfying V'1 = 0 and VV' = I

Details

The psd2edm function performs the inverse operation of the edm2psd function, taking a matrix in S_{n-1}^{+} and transforming it to a matrix in D_n^{-}.

$$psd2edm(S_{n-1}^{+}) = D_n^{-}$$

Therefore, psd2edm on S_{n-1}^{+} is the inverse operator of edm2psd on D_n^{-}.

For a symmetric positive semi-definite matrix S, psd2edm(S) will be in D_n^{-}.

Value

D A Euclidean Distance Matrix.

See Also

gram2edm edm2psd

Examples

```
XY <- cbind(runif(100,0,1),runif(100,0,1))
S <- edm2psd(as.matrix(dist(XY)))
D <- psd2edm(S)
```

rgrs *Relaxed Guided Random Search*

Description

rgrs Produce a point configuration given the edge lengths of the desired minimum spanning tree

Usage

```
rgrs(edges = NULL, d, n = NULL, theta = NULL, outlying = "N",
  skew = "N", stringy = "N")
```

Arguments

edges	A numeric vector containing the desired edge lengths of the minimum spanning tree. If n is specified, must be NULL.
d	the dimension of the resulting configuration.
n	the desired number of edge lengths to simulate. If edges is specified, must be set to NULL.
theta	Angle restriction during point proposal of the form (theta1,theta2,p), where p represents the probability of confining the proposal to [theta1,theta2]. Only used for d=2, otherwise NULL. See details for more in depth explanation.
outlying	One of "L", "M", or "H", specifying if the simulated edge lengths should have a Low, Medium, or High outlying scagnostic value.
skew	One of "L", "M", or "H", specifying if the simulated edge lengths should have a Low, Medium, or High skew scagnostic value.
stringy	One of "L", "M", or "H", specifying if the simulated edge lengths should have a Low, Medium, or High stringy scagnostic value. A numeric scalar specifying a value of stringy is also accepted.

Details

In 2-dimensions, when a new point is proposed, the position for the new point is determined by:

x <- x0 + r*sin(theta) y <- y0 + r*cos(theta)

where (x0,y0) is the base point, and r is the minimum spanning tree distance. theta is generated from a uniform distribution on (-pi,pi). By specifying the theta argument, the proposed theta is restricted, and is then generated from Uniform(theta1,theta2) or Uniform(-theta2,-theta1) with equal probability. This restriction allows the user to introduce striation into their point configuration.

Value

An nxd matrix containing the d-dimensional locations of the points.

Examples

```
#An example where edge lengths are supplied
EL <- runif(100,0,1)
rgrs(edges = EL, d = 2)
rgrs(edges = EL, d = 3)

#An Example where edge lengths are simulated internally
rgrs(d=2, n=100)
rgrs(d=3, n=100)
rgrs(d=2, n=100, outlying="H")
rgrs(d=2, n=100, skew = "M")
rgrs(d=2, n=100, stringy = "H")

#An Example making use of theta
rgrs(d=2, n=100, theta=c(pi/4,pi/3,.5))
```

---

sdp *Semi-Definite Programming Algorithm*

Description

sdp returns a completed Euclidean Distance Matrix D, with dimension d, from a partial Euclidean Distance Matrix using the methods of Alfakih et. al. (1999)

Usage

```
sdp(D, A, toler = 1e-08)
```

Arguments

D	An nxn partial-distance matrix to be completed. D must satisfy a list of conditions (see details), with unkown entries set to NA.
A	a weight matrix, with $h_{ij} = 0$ implying a_{ij} is unknown. Generally, if a_{ij} is known, $h_{ij} = 1$, although any non-negative weight is allowed.
toler	convergence tolerance for the algorithm

Details

This is an implementation of the Semi-Definite Programming Algorithm (sdp) for Euclidean Distance Matrix Completion, as proposed in 'Solving Euclidean Distance Matrix Completion Problems via Semidefinite Programming' (Alfakih et. al., 1999).

The method seeks to minimize the following:

$$||A \cdot (D - psd2edm(S))||_F^2$$

where the function psd2edm() is that described in psd2edm(), and the norm is Frobenius. Minimization is over S, a positive semidefinite matrix.

The matrix D is a partial-distance matrix, meaning some of its entries are unknown. It must satisfy the following conditions in order to be completed:

- diag(D) = 0
- If a_{ij} is known, $a_{ji} = a_{ij}$
- If a_{ij} is unknown, so is a_{ji}
- The graph of D must be connected. If D can be decomposed into two (or more) subgraphs, then the completion of D can be decomposed into two (or more) independent completion problems.

Value

D	an nxn matrix of the completed Euclidean distances
optval	the minimum value achieved of the target function during minimization
noiter	the number of iterations performed during minimization

See Also

psd2edm

Examples

```
D <- matrix(c(0,3,4,3,4,3,
              3,0,1,NA,5,NA,
              4,1,0,5,NA,5,
              3,NA,5,0,1,NA,
              4,5,NA,1,0,5,
              3,NA,5,NA,5,0),byrow=TRUE, nrow=6)
A <- matrix(c(1,1,1,1,1,1,
              1,1,1,0,1,0,
              1,1,1,1,0,1,
              1,0,1,1,1,0,
              1,1,0,1,1,1,
              1,0,1,0,1,1), byrow=TRUE, nrow=6)
toler <- 1e-4
sdp(D,A,toler)
```

snl *Sensor Network Localization*

Description

snl solves the sensor network problem with partial distance (squared) matrix D, and anchor positions anchors, in dimension d.

Usage

```
snl(D, d, anchors = NULL)
```

Arguments

D	The partial distance matrix specifying the known distances between nodes. If anchors is specified (and is a pxr matrix), the p final columns and p final rows specify the distances between the anchors specified in anchors.
d	the dimension for the resulting completion
anchors	a pxr matrix specifying the d dimensional locations of the p anchors. If the anchorless problem is to be solved, anchors = NULL

Details

Set anchors=NULL to solve the anchorless (Euclidean distance matrix completion) problem in dimension d.

NOTE: When anchors is specified, the distances between the anchors must be in the bottom right corner of the matrix D, and anchors must have d columns.

Value

X the d-dimensional positions of the localized sensors. Note that it may be the case that not all sensors could be localized, in which case X contains the positions of only the localized sensors.

References

Nathan Krislock and Henry Wolkowicz. Explicit sensor network localization using semidefinite representations and facial reductions. SIAM Journal on Optimization, 20(5):2679-2708, 2010.

Examples

```
D <- matrix(c(0,NA,.1987,NA,.0595,NA,.0159,.2251,.0036,.0875,
              NA,0,.0481,NA,NA,.0515,NA,.2079,.2230,NA,
              .1987,.0481,0,NA,NA,.1158,NA,NA,.1553,NA,
              NA,NA,NA,0,NA,NA,NA,.2319,NA,NA,
              .0595,NA,NA,NA,0,NA,.1087,.0894,.0589,.0159,
              NA,.0515,.1158,NA,NA,0,NA,NA,NA,NA,
              .0159,NA,NA,NA,.1087,NA,0,.3497,.0311,.1139,
              .2251,.2079,NA,.2319,.0894,NA,.3497,0,.1918,.1607,
```

```
                .0036,.2230,.1553,NA,.0589,NA,.0311,.1918,0,.1012,
                .0875,NA,NA,NA,.0159,NA,.1139,.1607,.1012,0),nrow=10, byrow=TRUE)

anchors <- matrix(c(.5131,.9326,
                    .3183,.3742,
                    .5392,.7524,
                    .2213,.7631), nrow=4,byrow=TRUE)
d <- 2

#Anchorless Problem
snl(D,d,anchors=NULL)

#Anchored Problem
snl(D,d,anchors)
```

Index