Package 'maotai'

October 25, 2019

Type Package

Title Tools for Matrix Algebra, Optimization and Inference

Version 0.1.1

Description Matrix is an universal and sometimes primary object/unit in applied mathematics and statistics. We provide a number of algorithms for selected problems in optimization and statistical inference. For general exposition to the topic with focus on statistical context, see the book by Banerjee and Roy (2014, ISBN:9781420095388).

Encoding UTF-8

License GPL (>= 3)

Suggests igraph, rstiefel

Imports Rcpp, Rdpack, RSpectra, Matrix, mclust, shapes, stats, utils

LinkingTo Rcpp, RcppArmadillo

RdMacros Rdpack

RoxygenNote 6.1.1

URL http://github.com/kyoustat/maotai

BugReports http://github.com/kyoustat/maotai/issues

NeedsCompilation yes

Author Kisung You [aut, cre] (<https://orcid.org/0000-0002-8584-459X>)

Maintainer Kisung You <kyoustat@gmail.com>

Repository CRAN

Date/Publication 2019-10-25 20:10:02 UTC

R topics documented:

`distecdf` *Distance Measures between Multiple Empirical Cumulative Distribution Functions*

Description

We measure distance between two empirical cumulative distribution functions (ECDF). For simplicity, we only take an input of `ecdf` objects from **stats** package.

Usage

```
distecdf(elist, method = c("KS", "Lp", "Wasserstein"), p = 2,
  as.dist = FALSE)
```

Arguments

`elist`	a length N list of `ecdf` objects.
`method`	name of the distance/dissimilarity measure. Case insensitive.
`p`	exponent for `Lp` or `Wasserstein` distance.
`as.dist`	a logical; `TRUE` to return `dist` object, `FALSE` to return an $(N \times N)$ symmetric matrix of pairwise distances.

Value

either `dist` object of an $(N \times N)$ symmetric matrix of pairwise distances by `as.dist` argument.

See Also

`ecdf`

Examples

```
## toy example : 10 of random and uniform distributions
mylist = list()
for (i in 1:10){
  mylist[[i]] = stats::ecdf(stats::rnorm(50, sd=2))
}
for (i in 11:20){
  mylist[[i]] = stats::ecdf(stats::runif(50, min=-5))
}
```

```
## compute Kolmogorov-Smirnov distance
dm = distecdf(mylist, method="KS")

## visualize
opar = par(pty="s")
image(dm[,nrow(dm):1], main="KS distances of 2 Types")
par(opar)
```

`distgmm` *Distance Measures between Multiple Samples using Gaussian Mixture Models*

Description

Taking multiple observations (a sample) as a unit of analysis requires a measure of discrepancy between samples. `distgmm` fits finite Gaussian mixture models to each sample and use the fitted model as a representation of a single sample. A single model is selected via Bayesian Information Criterion (BIC).

Usage

```
distgmm(datalist, method = c("L2"), maxk = 5, as.dist = FALSE)
```

Arguments

`datalist`	a length N list of samples. All elements of the list should be of same type, either `vector` or `matrix` of same dimension (number of columns).
`method`	name of the distance/dissimilarity measure.
`maxk`	maximum number of clusters to be fitted using GMM.
`as.dist`	a logical; `TRUE` to return `dist` object, `FALSE` to return an $(N \times N)$ symmetric matrix of pairwise distances.

Value

either `dist` object of an $(N \times N)$ symmetric matrix of pairwise distances by `as.dist` argument.

Examples

```
## let's try two-dimensional data of 30 samples
## single or mixture of two and three gaussian distributions.
dlist = list()
for (i in 1:10){
  dlist[[i]] = matrix(rnorm(120),ncol=2)
}
for (i in 11:20){
  A = matrix(rnorm(60,mean=-4),ncol=2)
```

```
  B = matrix(rnorm(60,mean= 4),ncol=2)
  dlist[[i]] = rbind(A,B)
}
for (i in 21:30){
  A = matrix(rnorm(40,mean=-4),ncol=2)
  B = matrix(rnorm(40),ncol=2)
  C = matrix(rnorm(40,mean= 4),ncol=2)
  dlist[[i]] = rbind(A,B,C)
}

## compute pairwise distances, expecting (3 x 3) block structure.
mm = distgmm(dlist, maxk=5)

## visualize
opar <- par(pty="s")
image(mm[,nrow(mm):1], main="3-block pattern as expected")
par(opar)
```

---

dpmeans *DP-means Algorithm for Clustering Euclidean Data*

Description

DP-means is a nonparametric clustering method motivated by DP mixture model in that the number of clusters is determined by a parameter λ. The larger the λ value is, the smaller the number of clusters is attained. In addition to the original paper, we added an option to randomly permute an order of updating for each observation's membership as a common heuristic in the literature of cluster analysis.

Usage

```
dpmeans(data, lambda = 1, maxiter = 1234, abstol = 1e-06,
  permute.order = FALSE)
```

Arguments

`data`	an $(n \times p)$ data matrix for each row being an observation.
`lambda`	a threshold to define a new cluster.
`maxiter`	maximum number of iterations.
`abstol`	stopping criterion
`permute.order`	a logical; `TRUE` if random order for permutation is used, `FALSE` otherwise.

Value

a named list containing

cluster an $(n \times ndim)$ matrix whose rows are embedded observations.

centers a list containing information for out-of-sample prediction.

References

Kulis B, Jordan MI (2012). "Revisiting K-means: New Algorithms via Bayesian Nonparametrics." In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, 1131–1138. ISBN 978-1-4503-1285-1.

Examples

```
## define data matrix of two clusters
x1  = matrix(rnorm(50*3,mean= 2), ncol=3)
x2  = matrix(rnorm(50*3,mean=-2), ncol=3)
X   = rbind(x1,x2)
lab = c(rep(1,50),rep(2,50))

## run dpmeans with several lambda values
solA <- dpmeans(X, lambda= 5)$cluster
solB <- dpmeans(X, lambda=10)$cluster
solC <- dpmeans(X, lambda=20)$cluster

## visualize the results
opar <- par(mfrow=c(1,4), pty="s")
plot(X,col=lab,  pch=19, cex=.8, main="True", xlab="x", ylab="y")
plot(X,col=solA, pch=19, cex=.8, main="dpmeans lbd=5", xlab="x", ylab="y")
plot(X,col=solB, pch=19, cex=.8, main="dpmeans lbd=10", xlab="x", ylab="y")
plot(X,col=solC, pch=19, cex=.8, main="dpmeans lbd=20", xlab="x", ylab="y")
par(opar)

## Not run:
## let's find variations by permuting orders of update
## used setting : lambda=20, we will 8 runs
sol8 <- list()
for (i in 1:8){
  sol8[[i]] = dpmeans(X, lambda=20, permute.order=TRUE)$cluster
}

## let's visualize
vpar <- par(mfrow=c(2,4), pty="s")
for (i in 1:8){
  pm = paste("permute no.",i,sep="")
  plot(X,col=sol8[[i]], pch=19, cex=.8, main=pm, xlab="x", ylab="y")
}
par(vpar)

## End(Not run)
```

---

lgpa *Large-scale Generalized Procrustes Analysis*

Description

We modify generalized Procrustes analysis for large-scale data by first setting a subset of anchor points and applying the attained transformation to the rest data. If `sub.id` is a vector `1:dim(x)[1]`, it uses all observations as anchor points, reducing to the conventional generalized Procrustes analysis.

Usage

```
lgpa(x, sub.id = 1:(dim(x)[1]), scale = TRUE, reflect = FALSE)
```

Arguments

`x`	a $(k \times m \times n)$ 3d array, where k is the number of points, m the number of dimensions, and n the number of samples.
`sub.id`	a vector of indices for defining anchor points.
`scale`	a logical; `TRUE` if scaling is applied, `FALSE` otherwise.
`reflect`	a logical; `TRUE` if reflection is required, `FALSE` otherwise.

Value

a $(k \times m \times n)$ 3d array of aligned samples.

Author(s)

Kisung You

References

Goodall C (1991). "Procrustes Methods in the Statistical Analysis of Shape." *Journal of the Royal Statistical Society. Series B (Methodological)*, **53**(2), 285–339. ISSN 00359246, http://www.jstor.org/stable/2345744.

Examples

```
## Not run:
## This should be run if you have 'shapes' package installed.
library(shapes)
data(gorf.dat)

## apply anchor-based method and original procGPA
out.proc = shapes::procGPA(gorf.dat, scale=TRUE)$rotated # procGPA from shapes package
out.anc4 = lgpa(gorf.dat, sub.id=c(1,4,9,7), scale=TRUE) # use 4 points
out.anc7 = lgpa(gorf.dat, sub.id=1:7, scale=TRUE)       # use all but 1 point as anchors

## visualize
opar = par(mfrow=c(3,4), pty="s")
plot(out.proc[,,1], main="procGPA No.1", pch=18)
plot(out.proc[,,2], main="procGPA No.2", pch=18)
plot(out.proc[,,3], main="procGPA No.3", pch=18)
plot(out.proc[,,4], main="procGPA No.4", pch=18)
```

```
plot(out.anc4[,,1], main="4 Anchors No.1", pch=18, col="blue")
plot(out.anc4[,,2], main="4 Anchors No.2", pch=18, col="blue")
plot(out.anc4[,,3], main="4 Anchors No.3", pch=18, col="blue")
plot(out.anc4[,,4], main="4 Anchors No.4", pch=18, col="blue")
plot(out.anc7[,,1], main="7 Anchors No.1", pch=18, col="red")
plot(out.anc7[,,2], main="7 Anchors No.2", pch=18, col="red")
plot(out.anc7[,,3], main="7 Anchors No.3", pch=18, col="red")
plot(out.anc7[,,4], main="7 Anchors No.4", pch=18, col="red")
par(opar)

## End(Not run)
```

`lyapunov` *Solve Lyapunov Equation*

Description

The Lyapunov equation is of form

$$AX + XA^\top = Q$$

where A and Q are square matrices of same size. Above form is also known as *continuous* form. This is a wrapper of `armadillo`'s `sylvester` function.

Usage

```
lyapunov(A, Q)
```

Arguments

`A`	a $(p \times p)$ matrix as above.
`Q`	a $(p \times p)$ matrix as above.

Value

a solution matrix X of size $(p \times p)$.

References

Sanderson C, Curtin R (2016). "Armadillo: a template-based C++ library for linear algebra." *The Journal of Open Source Software*, **1**(2), 26.

Eddelbuettel D, Sanderson C (2014). "RcppArmadillo: Accelerating R with high-performance C++ linear algebra." *Computational Statistics and Data Analysis*, **71**, 1054–1063.

Examples

```
## simulated example
#  generate square matrices
A = matrix(rnorm(25),nrow=5)
X = matrix(rnorm(25),nrow=5)
Q = A%*%X + X%*%t(A)

#  solve using 'lyapunov' function
solX = lyapunov(A,Q)
pm1 = "* Experiment with Lyapunov Solver"
pm2 = paste("* Absolute Error  : ",norm(solX-X,"f"),sep="")
pm3 = paste("* Relative Error  : ",norm(solX-X,"f")/norm(X,"f"),sep="")
cat(paste(pm1,"\n",pm2,"\n",pm3,sep=""))
```

`maotai` *Tools for Matrix Algebra, Optimization and Inference*

Description

Matrix is an universal and sometimes primary object/unit in applied mathematics and statistics. We provide a number of algorithms for selected problems in optimization and statistical inference. This package contains following functions,

FUNCTION	DESCRIPTION
distgmm	Distance Measures between Multisets using Gaussian Mixture Models
distecdf	Distance Measures between Multiple Empirical CDFs
dpmeans	DP-means Algorithm for Clustering Euclidean Data
lgpa	Large-scale Generalized Procrustes Analysis
lyapunov	Solve Lyapunov Equation
matderiv	Numerical Approximation to Gradient of a Function with Matrix Argument
pdeterminant	Calculate the Pseudo-Determinant of a Matrix
shortestpath	Find Shortest Path using Floyd-Warshall Algorithm
sylvester	Solve Sylvester Equation
trio	Trace Ratio Optimation

`matderiv` *Numerical Approximation to Gradient of a Function with Matrix Argument*

Description

For a given function $f : \mathbf{R}^{n \times p} \to \mathbf{R}$, we use finite difference scheme that approximates a gradient at a given point x. In Riemannian optimization, this can be used as a proxy for ambient gradient. Use with care since it may accumulate numerical error.

Usage

```
matderiv(fn, x, h = 0.001)
```

Arguments

fn	a function that takes a matrix of size $(n \times p)$ and returns a scalar value.
x	an $(n \times p)$ matrix where the gradient is to be computed.
h	step size for centered difference scheme.

Value

an approximate numerical gradient matrix of size $(n \times p)$.

References

Kincaid D, Cheney EW (2009). *Numerical analysis: mathematics of scientific computing*, number 2 in Pure and applied undergraduate texts, 3. ed edition. American Mathematical Society, Providence, RI. OCLC: 729930790.

Examples

```
## function f(X) = <a,Xb> for two vectors 'a' and 'b'
#  derivative w.r.t X is ab'
#  take an example of (5x5) symmetric positive definite matrix

#  problem settings
a   <- rnorm(5)
b   <- rnorm(5)
ftn <- function(X){
  return(sum(as.vector(X%*%b)*a))
}       # function to be taken derivative
myX <- matrix(rnorm(25),nrow=5)  # point where derivative is evaluated
myX <- myX%*%t(myX)

# main computation
sol.true <- base::outer(a,b)
sol.num1 <- matderiv(ftn, myX, h=1e-1) # step size : 1e-1
sol.num2 <- matderiv(ftn, myX, h=1e-5) #             1e-3
sol.num3 <- matderiv(ftn, myX, h=1e-9) #             1e-5

## visualize/print the results
expar = par(mfrow=c(2,2),pty="s")
image(sol.true, main="true solution")
image(sol.num1, main="h=1e-1")
image(sol.num2, main="h=1e-5")
image(sol.num3, main="h=1e-9")
par(expar)

ntrue = norm(sol.true,"f")
cat('* Relative Errors in Frobenius Norm ')
cat(paste("*  h=1e-1   : ",norm(sol.true-sol.num1,"f")/ntrue,sep=""))
```

```
cat(paste("*  h=1e-5   : ",norm(sol.true-sol.num2,"f")/ntrue,sep=""))
cat(paste("*  h=1e-9   : ",norm(sol.true-sol.num3,"f")/ntrue,sep=""))
```

---

pdeterminant *Calculate the Pseudo-Determinant of a Matrix*

Description

When a given square matrix A is rank deficient, determinant is zero. Still, we can compute the pseudo-determinant by multiplying all non-zero eigenvalues. Since thresholding to determine near-zero eigenvalues is subjective, we implemented the function as of original limit problem. When matrix is non-singular, it coincides with traditional determinant.

Usage

```
pdeterminant(A)
```

Arguments

A	a square matrix whose pseudo-determinant be computed.

Value

a scalar value for computed pseudo-determinant.

References

Holbrook A (2018). "Differentiating the pseudo determinant." *Linear Algebra and its Applications*, **548**, 293–304.

Examples

```
## show the convergence of pseudo-determinant
#  settings
n = 10
A = cov(matrix(rnorm(5*n),ncol=n))   # (n x n) matrix
k = as.double(Matrix::rankMatrix(A)) # rank of A

# iterative computation
ntry = 11
del.vec = exp(-(1:ntry))
det.vec = rep(0,ntry)
for (i in 1:ntry){
  del = del.vec[i]
  det.vec[i] = det(A+del*diag(n))/(del^(n-k))
}

# visualize the results
```

```
plot(1:ntry, det.vec, main=paste("true rank is ",k," out of ",n,sep=""),"b", xlab="iterations")
abline(h=pdeterminant(A),col="red",lwd=1.2)
```

---

shortestpath *Find Shortest Path using Floyd-Warshall Algorithm*

Description

This is a fast implementation of Floyd-Warshall algorithm to find the shortest path in a pairwise sense using RcppArmadillo. A logical input is also accepted. The given matrix should contain pairwise distance values $d_{i,j}$ where 0 means there exists no path for node i and j.

Usage

```
shortestpath(dist)
```

Arguments

dist	either an $(n \times n)$ matrix or a dist class object.

Value

an $(n \times n)$ matrix containing pairwise shortest path length.

References

Floyd RW (1962). "Algorithm 97: Shortest path." *Communications of the ACM*, **5**(6), 345.

Warshall S (1962). "A Theorem on Boolean Matrices." *Journal of the ACM*, **9**(1), 11–12.

Examples

```
## simple example : a ring graph
#  edges exist for pairs
A = array(0,c(10,10))
for (i in 1:9){
  A[i,i+1] = 1
  A[i+1,i] = 1
}
A[10,1] <- A[1,10] <- 1

# compute shortest-path and show the matrix
sdA <- shortestpath(A)
image(sdA, main="shortest path length for ring graph")
```

`sylvester` *Solve Sylvester Equation*

Description

The Sylvester equation is of form

$$AX + XB = C$$

where X is the unknown and others are given. Though it's possible to have non-square A and B matrices, we currently support square matrices only. This is a wrapper of `armadillo`'s `sylvester` function.

Usage

```
sylvester(A, B, C)
```

Arguments

`A`	a $(p \times p)$ matrix as above.
`B`	a $(p \times p)$ matrix as above.
`C`	a $(p \times p)$ matrix as above.

Value

a solution matrix X of size $(p \times p)$.

References

Sanderson C, Curtin R (2016). "Armadillo: a template-based C++ library for linear algebra." *The Journal of Open Source Software*, **1**(2), 26.

Eddelbuettel D, Sanderson C (2014). "RcppArmadillo: Accelerating R with high-performance C++ linear algebra." *Computational Statistics and Data Analysis*, **71**, 1054–1063.

Examples

```
## simulated example
#  generate square matrices
A = matrix(rnorm(25),nrow=5)
X = matrix(rnorm(25),nrow=5)
B = matrix(rnorm(25),nrow=5)
C = A%*%X + X%*%B

#  solve using 'sylvester' function
solX = sylvester(A,B,C)
pm1 = "* Experiment with Sylvester Solver"
pm2 = paste("* Absolute Error  : ",norm(solX-X,"f"),sep="")
pm3 = paste("* Relative Error  : ",norm(solX-X,"f")/norm(X,"f"),sep="")
cat(paste(pm1,"\n",pm2,"\n",pm3,sep=""))
```

`trio` *Trace Ratio Optimation*

Description

This function provides several algorithms to solve the following problem

$$\max \frac{tr(V^\top AV)}{tr(V^\top BV)} \text{ such that } V^\top CV = I$$

where V is a projection matrix, i.e., $V^\top V = I$. Trace ratio optimization is pertained to various linear dimension reduction methods. It should be noted that when $C = I$, the above problem is often reformulated as a generalized eigenvalue problem since it's an easier proxy with faster computation.

Usage

```
trio(A, B, C, dim = 2, method = c("2003Guo", "2007Wang", "2009Jia",
  "2012Ngo"), maxiter = 1000, eps = 1e-10)
```

Arguments

`A`	a $(p \times p)$ symmetric matrix in the numerator term.
`B`	a $(p \times p)$ symmetric matrix in the denomiator term.
`C`	a $(p \times p)$ symmetric constraint matrix. If not provided, it is set as identical matrix automatically.
`dim`	an integer for target dimension. It can be considered as the number of loadings.
`method`	the name of algorithm to be used. Default is `2003Guo`.
`maxiter`	maximum number of iterations to be performed.
`eps`	stopping criterion for iterative algorithms.

Value

a named list containing

V a $(p \times dim)$ projection matrix.

tr.val an attained maximum scalar value.

References

Guo Y, Li S, Yang J, Shu T, Wu L (2003). "A generalized Foley–Sammon transform based on generalized fisher discriminant criterion and its application to face recognition." *Pattern Recognition Letters*, **24**(1-3), 147–158.

Wang H, Yan S, Xu D, Tang X, Huang T (2007). "Trace Ratio vs. Ratio Trace for Dimensionality Reduction." In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, 1–8.

Yangqing Jia, Feiping Nie, Changshui Zhang (2009). "Trace Ratio Problem Revisited." *IEEE Transactions on Neural Networks*, **20**(4), 729–735.

Ngo TT, Bellalij M, Saad Y (2012). "The Trace Ratio Optimization Problem." *SIAM Review*, **54**(3), 545–569.

Examples

```
## simple test
#  problem setting
p = 5
mydim = 2
A = matrix(rnorm(p^2),nrow=p); A=A%*%t(A)
B = matrix(runif(p^2),nrow=p); B=B%*%t(B)
C = diag(p)

#  approximate solution via determinant ratio problem formulation
eigAB  = eigen(solve(B,A))
V      = eigAB$vectors[,1:mydim]
eigval = sum(diag(t(V)%*%A%*%V))/sum(diag(t(V)%*%B%*%V))

#  solve using 4 algorithms
m12 = trio(A,B,dim=mydim, method="2012Ngo")
m09 = trio(A,B,dim=mydim, method="2009Jia")
m07 = trio(A,B,dim=mydim, method="2007Wang")
m03 = trio(A,B,dim=mydim, method="2003Guo")

#  print the results
cat('* Evaluation of the cost function ')
cat(paste("* approx. via determinant : ",eigval,sep=""))
cat(paste("* trio by 2012Ngo         : ",m12$tr.val, sep=""))
cat(paste("* trio by 2009Jia         : ",m09$tr.val, sep=""))
cat(paste("* trio by 2007Wang        : ",m07$tr.val, sep=""))
cat(paste("* trio by 2003Guo         : ",m03$tr.val, sep=""))
```

Index