Package ‘RiemStiefel’

March 25, 2020

Type Package

Title Inference, Learning, and Optimization on Stiefel Manifold

Version 0.1.1

Description Stiefel manifold is a set of orthonormal frames in Euclidean space. We provide algorithms for statistical inference, optimization, and learning over the Stiefel manifold. For general exposition to the statistics on the manifold, see the book by Chikuse (2003) <doi:10.1007/978-0-387-21540-2>.

License GPL (>= 3)

Encoding UTF-8

Imports Rcpp, Rdpack, RiemBase, RiemBaseExt, utils, stats

LinkingTo Rcpp, RcppArmadillo, RiemBase

RoxygenNote 7.0.2

RdMacros Rdpack

NeedsCompilation yes

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

Maintainer Kisung You <kyoustat@gmail.com>

Repository CRAN

Date/Publication 2020-03-25 16:10:05 UTC

R topics documented:

RiemStiefel *Tools for Inference, Learning, and Optimization on Stiefel Manifold*

Description

Stiefel manifold $St(p,r)$ is the set of all orthonormal r-frames in R^p, which is indeed a Riemannian manifold. For $X \in St(p,r)$, it is characterized as

$$X^\top X = I_{r\times r}$$

. We provide algorithms for statistical inference, optimization, and learning over the Stiefel manifold. In our package, we use a convention to represent each data point on Stiefel manifold $St(p,r)$ as $(p \times r)$ matrix.

st.hclust *Hierarchical Agglomerative Clustering on Stiefel Manifold*

Description

Given the `type` of distance measure and agglomeration scheme `method`, `gr.hclust` performs hierarchical clustering on Grassmann manifold using **fastcluster** package, which returns the same object as **stats** package's implementation while providing more efficient computation. See `hclust` for more details.

Usage

```
st.hclust(
  x,
  type = c("intrinsic", "extrinsic"),
  method = c("single", "complete", "average", "mcquitty", "ward.D", "ward.D2",
    "centroid", "median"),
  members = NULL
)
```

Arguments

`x`	either an array of size $(p \times r \times N)$ or a list of length N whose elements are $(p \times r)$ matrix on Stiefel manifold.
`type`	type of distance measure; `"intrinsic"` or `"extrinsic"`.
`method`	he agglomeration method to be used. This must be (an unambiguous abbreviation of) one of `"single"`, `"complete"`, `"average"`, `"mcquitty"`, `"ward.D"`, `"ward.D2"`, `"centroid"` or `"median"`.
`members`	`NULL` or a vector whose length equals the number of observations. See `hclust` for details.

Value

an object of class hclust. See hclust for details.

Author(s)

Kisung You

Examples

```
#-------------------------------------------------------------------
#      Generate a dataset with two types of Stiefel elements
#-------------------------------------------------------------------
#  group1 : first four columns of (8x8) identity matrix + noise
#  group2 : last  four columns of (8x8) identity matrix + noise

mydata = list()
sdval  = 0.05
diag8  = diag(8)
for (i in 1:10){
  mydata[[i]] = qr.Q(qr(diag8[,1:4] + matrix(rnorm(8*4,sd=sdval),ncol=4)))
}
for (i in 11:20){
  mydata[[i]] = qr.Q(qr(diag8[,5:8] + matrix(rnorm(8*4,sd=sdval),ncol=4)))
}

## try hierarchical clustering
#  compare 'intrinsic' and 'extrinsic' distance types
#  and use 'single' hclust option.
hint = st.hclust(mydata, type="intrinsic", method="single")
hext = st.hclust(mydata, type="extrinsic", method="single")

## visualize
opar = par(no.readonly=TRUE)
par(mfrow=c(1,2), pty="s")
plot(hint, main="intrinsic")
plot(hext, main="extrinsic")
par(opar)
```

---

st.mean *Fréchet Mean on Stiefel Manifold*

Description

For manifold-valued data, Fréchet mean is the solution of following cost function,

$$\min_x \sum_{i=1}^n \rho^2(x, x_i), \quad x \in \mathcal{M}$$

for a given data $\{x_i\}_{i=1}^n$ and $\rho(x, y)$ is the geodesic distance between two points on manifold $\mathcal{M}$. It uses a gradient descent method with a backtracking search rule for updating. In the Stiefel manifold case, analytic formula is not known so we use numerical approximation scheme.

Usage

```
st.mean(x, type = c("intrinsic", "extrinsic"), eps = 1e-06, parallel = FALSE)
```

Arguments

`x`	either an array of size $(p \times r \times n)$ or a list of length n whose elements are $(p \times r)$ matrix on Stiefel manifold.
`type`	type of distance, either `"intrinsic"` or `"extrinsic"`.
`eps`	stopping criterion for the norm of gradient.
`parallel`	a flag for enabling parallel computation with OpenMP.

Value

a named list containing

mu an estimated mean matrix of size $(p \times r)$.

variation Fréchet variation with the estimated mean.

References

Bhattacharya A, Bhattacharya RN (2012). *Nonparametric inference on manifolds: with applications to shape spaces*, number 2 in Institute of mathematical statistics monographs. Cambridge University Press, Cambridge, UK ; New York. ISBN 978-1-107-01958-4, OCLC: ocn757931398.

Examples

```
#-------------------------------------------------------------------
#              Average Projection with 'iris' dataset
#-------------------------------------------------------------------
#  For PCA, take half of data from 'iris' data and repeat it 10 times.
#  We will compare naive PCA, intrinsic, and extrinsic mean.

data(iris)
label = iris$Species           # label information
idata = as.matrix(iris[,1:4])  # numeric data
ndata = nrow(idata)

# define a function for extracting pca projection
pcaproj <- function(X, p){
  return(eigen(stats::cov(X))$vectors[,1:p])
}

# extract embedding for random samples
proj10 = list()
for (i in 1:10){
    # index for random subsample
    rand.now    = base::sample(1:ndata, round(ndata/2))

    # PCA via personal tool
    proj10[[i]] = pcaproj(idata[rand.now,], p=2)
}

# compute intrinsic and extrinsic mean of projection matrices
mean.int = st.mean(proj10, type='intrinsic')$mu
mean.ext = st.mean(proj10, type='extrinsic')$mu

# compute 2-dimensional embeddings
fproj = pcaproj(idata, p=2)
f2 = idata%*%fproj     # PCA with full data
i2 = idata%*%mean.int  # projection with intrinsic mean
e2 = idata%*%mean.ext  # projection with extrinsic mean

# visualize
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3))
plot(f2, cex=0.5, pch=19, col=label, main='full PCA')
plot(i2, cex=0.5, pch=19, col=label, main='intrinsic projection')
plot(e2, cex=0.5, pch=19, col=label, main='extrinsic projection')
par(opar)
```

st.pdist *Pairwise Distance for Data on Stiefel Manifold*

Description

For data on Stiefel manifold $x_1, x_2, \ldots, x_N \in St(r,p)$, compute pairwise distances $d(x_i, x_j)$ via geodesic (`"intrinsic"`) distance or embedded Euclidean `"extrinsic"` metric. Since computing geodesic has no closed-form expressions, it relies on a numerical approximation which may incur heavier computational burden.

Usage

```
st.pdist(x, type = c("intrinsic", "extrinsic"), as.dist = TRUE)
```

Arguments

`x`	either an array of size $(p \times r \times N)$ or a list of length N whose elements are $(p \times r)$ matrix on Stiefel manifold.
`type`	type of distance, either `"intrinsic"` or `"extrinsic"`.
`as.dist`	a logical; `TRUE` to return a `dist` object or `FALSE` to return an $(N \times N)$ symmetric matrix.

Value

a `dist` object or $(N \times N)$ symmetric matrix depending on `as.dist`.

Author(s)

Kisung You

Examples

```
#-------------------------------------------------------------------
#      Generate a dataset with two types of Stiefel elements
#-------------------------------------------------------------------
#  group1 : first four columns of (8x8) identity matrix + noise
#  group2 : last  four columns of (8x8) identity matrix + noise

mydata = list()
sdval  = 0.05
diag8  = diag(8)
for (i in 1:10){
  mydata[[i]] = qr.Q(qr(diag8[,1:4] + matrix(rnorm(8*4,sd=sdval),ncol=4)))
}
for (i in 11:20){
  mydata[[i]] = qr.Q(qr(diag8[,5:8] + matrix(rnorm(8*4,sd=sdval),ncol=4)))
}

## compare 'intrinsic' and 'extrinsic' distances
dint = st.pdist(mydata, type="intrinsic", as.dist=FALSE)
dext = st.pdist(mydata, type="extrinsic", as.dist=FALSE)

## visualize
opar = par(no.readonly=TRUE)
par(mfrow=c(1,2), pty="s")
image(dint[,20:1], main="intrinsic")
image(dext[,20:1], main="extrinsic")
par(opar)
```

`st.runif` *Generate Random Samples from Uniform Distribution on Stiefel Manifold*

Description

It generates n random samples from Stiefel manifold $St(p,r)$ according to the procedure described in the reference.

Usage

```
st.runif(n, p, r, rtype = c("list", "array"))
```

Arguments

`n`	number of samples to be generated.
`p`	original dimension (of the ambient space).
`r`	dimension of the frame.
`rtype`	return type; either 3d-array (`"array"`) or list (`"list"`).

Value

a length n list or 3d array of size (p, r, n).

References

Chikuse Y (2003). *Statistics on Special Manifolds*, volume 174 of *Lecture Notes in Statistics*. Springer New York, New York, NY. ISBN 978-0-387-00160-9 978-0-387-21540-2, doi: 10.1007/9780387215402, `http://link.springer.com/10.1007/978-0-387-21540-2`.

Examples

```
## let's simply draw 3 times from St(10,5)
dat3 = st.runif(3, 10, 5, rtype="array")

## visualize
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3))
image(dat3[,,1], main="sample 1")
image(dat3[,,2], main="sample 2")
image(dat3[,,3], main="sample 3")
par(opar)
```

`st.utestR` *Test of Uniformity via Rayleigh Statistic on Stiefel Manifold*

Description

This function is for hypothesis testing on Stiefel manifold $St(p, r)$ whether the given data is uniformly distributed or not. We provide two options (original and modified) for Rayleigh-type statistics, which both follow Chi-squared distribution of degrees of freedom pr.

Usage

```
st.utestR(x, method = c("Original", "Modified"))
```

Arguments

`x`	either an array of size $(p \times r \times n)$ or a list of length n whose elements are $(p \times r)$ matrix on Stiefel manifold.
`method`	`"original"` for conventional Rayleigh statistic or `"modified"` for better order of error.

Value

a (list) object of S3 class htest containing:

statistic a test statistic.

p.value p-value under H_0.

alternative alternative hypothesis.

method name of the test.

data.name name(s) of provided sample data.

References

Mardia KV, Jupp PE (eds.) (1999). *Directional Statistics*, Wiley Series in Probability and Statistics. John Wiley \& Sons, Inc., Hoboken, NJ, USA. ISBN 978-0-470-31697-9 978-0-471-95333-3, doi: 10.1002/9780470316979, http://doi.wiley.com/10.1002/9780470316979.

Examples

```
## Test of Uniformity for 100 samples from St(10,5)
#  Data Generation
mydat = st.runif(n=100, p=10, r=5, rtype='list')

#  Run Tests using two methods
st.utestR(mydat, method='original')
st.utestR(mydat, method='modified')


## empirical Type 1 error using the same setting as above.
niter   = 10000
counter = rep(0,niter)  # record p-values
for (i in 1:niter){
  X = st.runif(n=100, p=10, r=5, rtype='list')
  counter[i] = ifelse(st.utestR(X)$p.value < 0.05, 1, 0)
  print(paste0("iteration ",i,"/10000 complete..."))
}

## print the result
print(paste0("* empirical Type 1 error for 'st.utestR': ",round(sum(counter/niter),5)))
```

stopt.SA   *Optimization over Stiefel Manifold with Simulated Annealing*

Description

Simulated Annealing is a black-box, derivative-free optimization algorithm that iterates via stochastic search in the neighborhood of current position.

Usage

```
stopt.SA(
  func,
  size,
  n.start = 10,
  stepsize = 0.1,
  maxiter = 100,
  cooling = c("exponential", 10, 0.9),
  init.val = NULL,
  print.progress = FALSE
)
```

Arguments

`func`	a function to be *minimized*.
`size`	a length-2 vector containing dimension information (p, r).
`n.start`	number of runs, i.e., algorithm is executed `n.start` times.
`stepsize`	size of random walk on each component.
`maxiter`	maximum number of iterations for each run.
`cooling`	triplet for cooling schedule. See the section for the usage.
`init.val`	if `NULL`, starts from a random point. Otherwise, a Stiefel matrix of size (p, r) should be provided for fixed starting point.
`print.progress`	a logical; `TRUE` to show

Value

a named list containing:

cost minimized function value.

solution a $(p \times r)$ matrix that attains the `cost`.

accfreq frequency of acceptance moves.

Examples

```
## Optimization for eigen-decomposition
#  Let's find top-3 eigenvalues
set.seed(121)                        # set seed
A = cov(matrix(rnorm(100*5), ncol=5)) # define covariance
myfunc <- function(p){               # cost function
  return(sum(-diag(t(p)%*%A%*%p)))
}

#  Solve the optimization problem
Aout = stopt.SA(myfunc, size=c(5,3), n.start=40, maxiter=500)

#  Compute 3 Eigenvalues
#  1. use computed basis
```

```
abase   = Aout$solution
eig3sol = sort(diag(t(abase)%*%A%*%abase), decreasing=TRUE)

#  2. use 'eigen' function
eig3dec = sort(eigen(A)$values, decreasing=TRUE)[1:3]


#   Visualize
opar <- par(no.readonly=TRUE)
yran = c(min(min(eig3sol),min(eig3dec))*0.95,
         max(max(eig3sol),max(eig3dec))*1.05)
plot(1:3, eig3sol, type="b", col="red",  pch=19, ylim=yran,
     xlab="index", ylab="eigenvalue", main="compare top 3 eigenvalues")
lines(1:3, eig3dec, type="b", col="blue", pch=19)
legend(1, 1, legend=c("optimization","decomposition"), col=c("red","blue"),
       lty=rep(1,2), pch=19)
par(opar)
```

Index