Package: KernelICA (via r-universe)

November 21, 2026

Type Package

Title Kernel Independent Component Analysis

Version 2.0.0

Maintainer Klaus Nordhausen <klausnordhausenR@gmail.com>

Description The kernel independent component analysis (kernel ICA) method introduced by Bach and Jordan (2002) <doi:10.1162/153244303768966085>. A separate function for the incomplete Cholesky decomposition used in kernel ICA is also provided.

License GPL (>= 2)

Imports methods, ManifoldOptim, JADE, ICtest

Depends Rcpp, R (>= 4.2.0)

LinkingTo Rcpp, RcppArmadillo, ManifoldOptim, RcppEigen

Encoding UTF-8

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Christoph L. Koesner [aut] (ORCID: <https://orcid.org/0000-0002-2061-8022>), Juho Eagling [aut] (ORCID: <https://orcid.org/0000-0001-8333-5057>), Klaus Nordhausen [aut, cre] (ORCID: <https://orcid.org/0000-0002-3758-8501>)

Config/pak/sysreqs make libicu-dev libpng-dev libssl-dev

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

Date/Publication 2026-11-21 02:02:00 UTC

RemoteUrl https://github.com/cranhaven/cranhaven.r-universe.dev

RemoteRef package/KernelICA

RemoteSha 5a983f134b20c93ab200559f8da22b5b58d82354

RemoteSubdir KernelICA

Contents

KernelICA-package 2
incomplete_cholesky 3
kernel_ica 4
kernel_matrix 7
MD_distant_matrices 8

Index **10**

`KernelICA-package` *KernelICA-package*

Description

The kernel independent component analysis (kernel ICA) method introduced by Bach and Jordan in 2002 (see references). A separate function for the incomplete Cholesky decomposition used in kernel ICA is also provided.

Author(s)

Maintainer: Klaus Nordhausen <klausnordhausenR@gmail.com> (ORCID)

Authors:

- Klaus Nordhausen <klausnordhausenR@gmail.com> (ORCID)
- Christoph L. Koesner <christoph@koesner.at> (ORCID)
- Juho Eagling <juho.eagling@gmail.com> (ORCID)

References

Francis R. Bach, Michael I. Jordan
Kernel independent component analysis
Journal of Machine Learning Research 2002
doi:10.1162/153244303768966085

Francis R. Bach, Michael I. Jordan
Predictive low-rank decomposition for kernel methods.
Proceedings of the Twenty-second International Conference on Machine Learning (ICML) 2005
doi:10.1145/1102351.1102356.

Sean Martin, Andrew M. Raim, Wen Huang, Kofi P. Adragni
ManifoldOptim: An R Interface to the ROPTLIB Library for Riemannian Manifold Optimization
Journal of Statistical Software 2020
doi:10.18637/jss.v093.i01

`incomplete_cholesky` *Incomplete Cholesky Decomposition*

Description

The incomplete Cholesky decomposition, which computes approximative low rank decompositions for either Gauss or Hermite kernel matrices. Its implementation is inspired by Matlab and C code of F. Bach (see references) and written with the C++ library Eigen3 for speed purposes.

Usage

```
incomplete_cholesky(
  x,
  kernel = c("gauss", "hermite"),
  eps = 1e-04,
  sigma = ifelse(length(x) < 1000, 1, 0.5),
  hermite_rank = 3
)
```

Arguments

`x`	Numeric vector.
`kernel`	One of `"gauss"` or `"hermite"`.
`eps`	Numeric precision parameter for the matrix approximation.
`sigma`	Numeric value, setting the kernel variance. Default is 1 for vectors smaller than n=1000, otherwise 0.5.
`hermite_rank`	Integer value for the rank of the Hermite kernel. This parameter is ignored, when the Gaussian kernel is chosen. Default is 3.

Details

The function approximates kernel matrices of the form $\boldsymbol{K} = (K_{ij})_{(i,j)} = K(x_i, x_j)$ for a vector $\boldsymbol{x}$ and a kernel function $K(\cdot, \cdot)$. It returns a permutation matrix $\boldsymbol{P}$ given as index vector and a numeric $n \times k$ matrix $\boldsymbol{L}$ which is a "cut off" lower triangle matrix, as it contains only the first k columns that were necessary to attain a sufficient approximation. These matrices follow the inequality $\|\boldsymbol{PKP}^T - \boldsymbol{LL}^T\|_1 \leq \epsilon$ where ϵ is the given precision parameter. The function offers approximation for kernel matrices of the following two kernels:

- Gauss Kernel (fix v1.0.1: corrected sign in exponent): $K(x, y) = e^{-(x-y)^2/2\sigma^2}$
- Hermite Kernel: $K(x, y) = \sum_{k=0}^{d} e^{-x^2/2\sigma^2} e^{-y^2/2\sigma^2} \frac{h_k(x/\sigma) h_k(y/\sigma)}{2^k k!}$, where h_k is the Hermite polynomial of grade k

Value

A list containing the following entries:

L A numeric matrix which values L_{ij} are 0 for $j > i$.

perm An integer vector of indices representing the permutation matrix.

References

Kernel ICA implementation in Matlab and C by F. Bach containing the Incomplete Cholesky Decomposition:
https://www.di.ens.fr/~fbach/kernel-ica/index.htm

Francis R. Bach, Michael I. Jordan
Predictive low-rank decomposition for kernel methods.
Proceedings of the Twenty-second International Conference on Machine Learning (ICML) 2005
doi:10.1145/1102351.1102356.

Francis R. Bach, Michael I. Jordan
Kernel independent component analysis
Journal of Machine Learning Research 2002
doi:10.1162/153244303768966085

Examples

```
# approximation of a Gauss kernel matrix
x <- rnorm(500)
x_kernel_mat <- kernel_matrix(x, sigma = 1)
x_chol <- incomplete_cholesky(x, sigma = 1)
L_perm <- x_chol$L[x_chol$perm, ]
x_kernel_approx <- L_perm %*% t(L_perm)
## largest differing value:
max(abs(x_kernel_approx - x_kernel_mat))


# approximation of a Hermite kernel matrix
x_kernel_mat <- kernel_matrix(x, kernel = "hermite", sigma = 0.5)
x_chol <- incomplete_cholesky(x, kernel = "hermite", sigma = 0.5)
L_perm <- x_chol$L[x_chol$perm, ]
x_kernel_approx <- L_perm %*% t(L_perm)
## largest differing value:
max(abs(x_kernel_approx - x_kernel_mat))
```

`kernel_ica` *Kernel Independent Component Analysis*

Description

The kernel ICA method by Bach and Jordan (see references). The contrast function was written in C++ under use of the Eigen3 library for computational speed. The package ManifoldOptim is utilized for minimization of the contrast function on the Stiefel manifold.

Usage

```
kernel_ica(
  x,
```

```
  variant = c("kgv", "kcca"),
  kernel = c("gauss", "hermite"),
  nstarts = 1,
  eps = 1e-04,
  sigma = ifelse(ncol(x) < 1000, 1, 0.5),
  kappa = ifelse(ncol(x) < 1000, 0.02, 0.002),
  hermite_rank = 3,
  init = MD_distant_matrices(p = ncol(x), n = nstarts),
  solver_params = ManifoldOptim::get.solver.params(),
  optim_method = "RSD"
)
```

Arguments

x	A numeric matrix, where each column contains the measurements of a mixed data source.
variant	Either "kcca" or "kgv".
kernel	Either "gauss" or "hermite".
nstarts	The number of restarts of the kernel ICA method with a default value of one. Ignored, if the starting values in parameter init are set manually.
eps	Numeric precision parameter for the approximation of the kernel matrices.
sigma	Numeric value of the kernel variance. Default value is 1 for a given x with less than 1000 rows, otherwise 0.5.
kappa	Numeric dimming parameter. Default value is 2e^-2 for a given x with less than 1000 rows, otherwise 2e^-3.
hermite_rank	Integer. Rank of the hermite polynomial with a default value of 3. Ignored, when kernel was set to "gauss".
init	A list of $p \times p$ orthogonal matrices, which are the starting points for the optimization in the Stiefel manifold. By default a number of orthogonal matrices specified in parameter nstarts is generated.
solver_params	An object returned from the method ManifoldOptim::get.solver.params which can be given several parameters for the optimization.
optim_method	The optimization method used in the Stiefel manifold. Default value is "RSD". This value is directly passed to ManifoldOptim::manifold.optim.

Details

Several points need to be considered when using kernel_ica:

- To comply with the notions of the JADE package, model $\boldsymbol{X} = \boldsymbol{S}\boldsymbol{A}'$ with a $n \times p$ source matrix $\boldsymbol{S}$ and a $p \times p$ mixing matrix $\boldsymbol{A}$ is assumed.
- The returned unmixing matrix $\boldsymbol{W}$ is found such that $\boldsymbol{X}\boldsymbol{W}' = \boldsymbol{S}\boldsymbol{A}'\boldsymbol{W}'$ results in the desired independent data.
- It is not possible to reconstruct the original order of the sources nor their sign.

- The contrast function which is to be minimized can have several local optimal. Therefore setting the `nstart` parameter to a larger value than one or instead providing more matrices in `init` for several starts should be considered.
- Kernel ICA is started for each element given in the list `init` separately and returns the best result by the lowest resulting value of the contrast function.

Value

A class of type `bss` containing the following values:

Xmu the restored mean values

S The unmixed data

W The unmixing matrix

cmin The smallest resulting contrast function value of all kernel ICA runs

References

Kernel ICA implementation in Matlab and C by F. Bach:
https://www.di.ens.fr/~fbach/kernel-ica/index.htm

Francis R. Bach, Michael I. Jordan
Kernel independent component analysis
Journal of Machine Learning Research 2002
doi:10.1162/153244303768966085

Sean Martin, Andrew M. Raim, Wen Huang, Kofi P. Adragni
ManifoldOptim: An R Interface to the ROPTLIB Library for Riemannian Manifold Optimization
Journal of Statistical Software 2020
doi:10.18637/jss.v093.i01

See Also

`ManifoldOptim::manifold.optim`
`ManifoldOptim::get.solver.params`

Examples

```
require(JADE)
require(ICtest)

n <- 2000
p <- 3
S <- matrix(0, n, p)

# the three data sources used in this example
S[, 1] <- rexp(n, rate = 0.4)
S[, 2] <- runif(n, 2, 4)
S[, 3] <- rt(n, 5)

W <- ICtest::rorth(p) # creates an orthogonal matrix
y <- S %*% t(W) # mixes the data
```

```
# applying kernel ICA method
res <- KernelICA::kernel_ica(y, variant = "kgv", kernel = "hermite")

res$W # unmixing matrix
apply(S, 2, mean) # original means
res$Xmu # restored means (unordered and possibly with different sign each)

# restored data
z <- scale(res$S, center = -res$Xmu, scale = FALSE)
# MD distance of the returned matrix to the original mixing matrix.
JADE::MD(res$W, W)

## Not run:
# Runs kernel ICA with the slower Gauss kernel method and
# a the starting matrix returned from the first method call.
# The maximal iteration number in the optimization is reduced to a tenth.
res2 <- KernelICA::kernel_ica(
  y,
  variant = "kgv",
  kernel = "gauss",
  init = list(res$W),
  solver_params = ManifoldOptim::get.solver.params(Max_Iteration = 100)
)
JADE::MD(res2$W, W)

## End(**Not run**)
```

`kernel_matrix` *Kernel Matrix Computation*

Description

Computes kernel matrices for Gaussian and Hermitian kernels.

Usage

```
kernel_matrix(
  x,
  y = x,
  kernel = c("gauss", "hermite"),
  sigma = 1,
  hermite_rank = 3
)
```

Arguments

`x`	Numeric vector.
`y`	Numeric vector, default is `x`.

kernel	Either "gauss" or "hermite".
sigma	Numeric value of the kernel variance. Default is 1.
hermite_rank	Rank of the Hermite kernel. Default is 3. Ignored, when the Gaussian kernel is chosen.

Details

The function computes a matrix in the form of $(K_{ij})_{(i,j)} = K(x_i, x_j)$ or $(K_{ij})_{(i,j)} = K(x_i, y_j)$ for a kernel function K depending if a second vector was given. The following two kernels are offered:

- Gauss Kernel: $K(x, y) = e^{-(x-y)^2/2\sigma^2}$
- Hermite Kernel: $K(x, y) = \sum_{k=0}^{d} e^{-x^2/2\sigma^2} e^{-y^2/2\sigma^2} \frac{h_k(x/\sigma) h_k(y/\sigma)}{2^k k!}$ where h_k is the Hermite polynomial of grade k

Value

A numeric kernel matrix.

Examples

```
x <- rnorm(10)
kernel_matrix(x, kernel = "gauss", sigma = 4)
kernel_matrix(x, kernel = "hermite", sigma = 4, hermite_rank = 3)
```

MD_distant_matrices *MD Distant Matrices*

Description

Creates orthogonal matrices in the Stiefel manifold, which are distant to each other by the MD index and optionally also distant to a given set of matrices.

Usage

```
MD_distant_matrices(p, n = 1, mat_list = list(), bestof = 10)
```

Arguments

p	The dimension of the orthogonal matrices.
n	The length of the returned matrix list.
mat_list	A list of already existing orthogonal matrices.
bestof	The number of candidates evaluated for each new matrix.

Details

If a matrix list should be created from scratch, i.e. the parameter `mat_list` was not provided, then the first orthogonal matrix of the returned list is randomly generated by `ICtest::rorth`. If `n` is larger than one or if a matrix list was provided, then for each additional matrix M_{k+1} we consider the distance $\min(\text{MD}(M_1, M_{k+1}), \text{MD}(M_2, M_{k+1}), \ldots, \text{MD}(M_k, M_{k+1}))$ to all previous list entries. This distance is evaluated for `bestof` randomly generated orthogonal candidate matrices from which the furthest is selected.

Value

A list which contains the already given and the additionally created matrices.

See Also

ICtest::rorth

Examples

```
# creates one orthogonal 3x3 matrix (result of ICtest::rorth(3)), wrapped in a list
MD_distant_matrices(3, 1)

# creates a 4x4 matrix, distant to the unit matrix and returns both
MD_distant_matrices(4, 2, mat_list = list(diag(4)))

# creates two orthogonal 3x3 matrices with more candidates to get better distances.
m <- MD_distant_matrices(3, 2, bestof = 20)
JADE::MD(m[[1]], m[[2]])
```

Index

incomplete_cholesky, 3

kernel_ica, 4
kernel_matrix, 7
KernelICA *(*KernelICA-package*)*, 2
KernelICA-package, 2

MD_distant_matrices, 8