Package ‘CMLS’

March 31, 2023

Type Package

Title Constrained Multivariate Least Squares

Version 1.0-1

Date 2023-03-29

Author Nathaniel E. Helwig <helwig@umn.edu>

Maintainer Nathaniel E. Helwig <helwig@umn.edu>

Depends quadprog, parallel

Description Solves multivariate least squares (MLS) problems subject to constraints on the coefficients, e.g., non-negativity, orthogonality, equality, inequality, monotonicity, unimodality, smoothness, etc. Includes flexible functions for solving MLS problems subject to user-specified equality and/or inequality constraints, as well as a wrapper function that implements 24 common constraint options. Also does k-fold or generalized cross-validation to tune constraint options for MLS problems. See ten Berge (1993, ISBN:9789066950832) for an overview of MLS problems, and see Goldfarb and Idnani (1983) <doi:10.1007/BF02591962> for a discussion of the underlying quadratic programming algorithm.

License GPL (>= 2)

NeedsCompilation no

Repository CRAN

Date/Publication 2023-03-31 17:30:02 UTC

R topics documented:

CMLS-package 2
cmls 3
const 11
cv.cmls 12
IsplineBasis 15
mlsei 16
mlsun 19
MsplineBasis 23

Index **25**

`CMLS-package` *Constrained Multivariate Least Squares*

Description

Solves multivariate least squares (MLS) problems subject to constraints on the coefficients, e.g., non-negativity, orthogonality, equality, inequality, monotonicity, unimodality, smoothness, etc. Includes flexible functions for solving MLS problems subject to user-specified equality and/or inequality constraints, as well as a wrapper function that implements 24 common constraint options. Also does k-fold or generalized cross-validation to tune constraint options for MLS problems. See ten Berge (1993, ISBN:9789066950832) for an overview of MLS problems, and see Goldfarb and Idnani (1983) <doi:10.1007/BF02591962> for a discussion of the underlying quadratic programming algorithm.

Details

The DESCRIPTION file:

Package:	CMLS
Type:	Package
Title:	Constrained Multivariate Least Squares
Version:	1.0-1
Date:	2023-03-29
Author:	Nathaniel E. Helwig <helwig@umn.edu>
Maintainer:	Nathaniel E. Helwig <helwig@umn.edu>
Depends:	quadprog, parallel
Description:	Solves multivariate least squares (MLS) problems subject to constraints on the coefficients, e.g., non-negativity
License:	GPL (>=2)

Index of help topics:

```
CMLS-package            Constrained Multivariate Least Squares
IsplineBasis            I-Spline Basis for Monotonic Polynomial Splines
MsplineBasis            M-Spline Basis for Polynomial Splines
cmls                    Solve a Constrained Multivariate Least Squares
                        Problem
const                   Print or Return Constraint Options for cmls
cv.cmls                 Cross-Validation for cmls
mlsei                   Multivariate Least Squares with
                        Equality/Inequality Constraints
mlsun                   Multivariate Least Squares with Unimodality
                        (and E/I) Constraints
```

The `cmls` function provides a user-friendly interface for solving the MLS problem with 24 common constraint options (the `const` function prints or returns the different contraint options). The `cv.cmls` function does k-fold or generalized cross-validation to tune the constraint options of the cmls function. The mlsei function solves the MLS problem subject to user-specified equality and/or inequality (E/I) constraints on the coefficients. The mlsun function solves the MLS problem subject to unimodality constraints and user-specified E/I constraints on the coefficients.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

Maintainer: Nathaniel E. Helwig <helwig@umn.edu>

References

Goldfarb, D., & Idnani, A. (1983). A numerically stable dual method for solving strictly convex quadratic programs. Mathematical Programming, 27, 1-33. doi:10.1007/BF02591962

Helwig, N. E. (in prep). Constrained multivariate least squares in R.

Ten Berge, J. M. F. (1993). Least Squares Optimization in Multivariate Analysis. Volume 25 of M & T Series. DSWO Press, Leiden University. ISBN: 9789066950832

Turlach, B. A., & Weingessel, A. (2019). quadprog: Functions to solve Quadratic Programming Problems. R package version 1.5-8. https://CRAN.R-project.org/package=quadprog

Examples

```
# See examples for cmls, cv.cmls, mlsei, and mlsun
```

---

cmls — *Solve a Constrained Multivariate Least Squares Problem*

Description

Finds the p x m matrix B that minimizes the multivariate least squares problem

```
sum(( Y - X %*% B )^2)
```

subject to the specified constraints on the rows of B.

Usage

```
cmls(X, Y, const = "uncons", struc = NULL,
     z = NULL, df = 10, degree = 3, intercept = TRUE,
     backfit = FALSE, maxit = 1e3, eps = 1e-10,
     del = 1e-6, XtX = NULL, mode.range = NULL)
```

Arguments

X	Matrix of dimension n x p.
Y	Matrix of dimension n x m.
const	Constraint code. See const for the 24 available options.

struc	Structural constraints (defaults to unstructured). See Note.
z	Predictor values for the spline basis (for smoothness constraints). See Note.
df	Degrees of freedom for the spline basis (for smoothness constraints). See Note.
degree	Polynomial degree for the spline basis (for smoothness constraints). See Note.
intercept	Logical indicating whether the spline basis should contain an intercept (for smoothness constraints). See Note.
backfit	Estimate B via back-fitting (TRUE) or vectorization (FALSE). See Details.
maxit	Maximum number of iterations for back-fitting algorithm. Ignored if backfit = FALSE.
eps	Convergence tolerance for back-fitting algorithm. Ignored if backfit = FALSE.
del	Stability tolerance for back-fitting algorithm. Ignored if backfit = FALSE.
XtX	Crossproduct matrix: XtX = crossprod(X).
mode.range	Mode search ranges (for unimodal constraints). See Note.

Details

If backfit = FALSE (default), a closed-form solution is used to estimate B whenever possible. Otherwise a back-fitting algorithm is used, where the rows of B are updated sequentially until convergence. The backfitting algorithm is determined to have converged when

mean((B.new - B.old)^2) < eps * (mean(B.old^2) + del),

where B.old and B.new denote the parameter estimates at iterations t and $t+1$ of the backfitting algorithm.

Value

Returns the estimated matrix B with attribute "df" (degrees of freedom), which gives the df for each row of B.

Note

Structure constraints (struc) should be specified with a p x m matrix of logicals (TRUE/FALSE), such that FALSE elements indicate a weight should be constrained to be zero. Default uses unstructured weights, i.e., a p x m matrix of all TRUE values.

Inputs z, df, degree, and intercept are only applicable when using one of the 12 constraints that involves a spline basis, i.e., "smooth", "smonon", "smoper", "smpeno", "ortsmo", "orsmpe", "monsmo", "mosmno", "unismo", "unsmno", "unsmpe", "unsmpn".

Input mode.range is only applicable when using one of the 8 constraints that enforces unimodality: "unimod", "uninon", "uniper", "unpeno", "unismo", "unsmno", "unsmpe", "unsmpn". Mode search ranges (mode.range) should be specified with a 2 x p matrix of integers such that

1 <= mode.range[1,j] <= mode.range[2,j] <= m for all j = 1:p.

Default is mode.range = matrix(c(1, m), 2, p).

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Goldfarb, D., & Idnani, A. (1983). A numerically stable dual method for solving strictly convex quadratic programs. Mathematical Programming, 27, 1-33. doi:10.1007/BF02591962

Helwig, N. E. (in prep). Constrained multivariate least squares in R.

Ten Berge, J. M. F. (1993). Least Squares Optimization in Multivariate Analysis. Volume 25 of M & T Series. DSWO Press, Leiden University. ISBN: 9789066950832

Turlach, B. A., & Weingessel, A. (2019). quadprog: Functions to solve Quadratic Programming Problems. R package version 1.5-8. https://CRAN.R-project.org/package=quadprog

See Also

`const` prints/returns the contraint options.

`cv.cmls` performs k-fold cross-validation to tune the constraint options.

`mlsei` and `mlsun` are used to implement several of the constraints.

Examples

```
######*******######   GENERATE DATA   ######*******######

# make X
set.seed(2)
n <- 50
m <- 20
p <- 2
Xmat <- matrix(rnorm(n*p), nrow = n, ncol = p)

# make B (which satisfies all constraints except monotonicity)
x <- seq(0, 1, length.out = m)
Bmat <- rbind(sin(2*pi*x), sin(2*pi*x+pi)) / sqrt(4.75)
struc <- rbind(rep(c(TRUE, FALSE), each = m / 2),
               rep(c(FALSE, TRUE), each = m / 2))
Bmat <- Bmat * struc

# make noisy data
set.seed(1)
Ymat <- Xmat %*% Bmat + rnorm(n*m, sd = 0.25)


######*******######   UNCONSTRAINED   ######*******######

# unconstrained
Bhat <- cmls(X = Xmat, Y = Ymat, const = "uncons")
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# unconstrained and structured
Bhat <- cmls(X = Xmat, Y = Ymat, const = "uncons", struc = struc)
mean((Bhat - Bmat)^2)
attr(Bhat, "df")
```

```
#####**#######   NON-NEGATIVITY   #####**#######

# non-negative
Bhat <- cmls(X = Xmat, Y = Ymat, const = "nonneg")
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# non-negative and structured
Bhat <- cmls(X = Xmat, Y = Ymat, const = "nonneg", struc = struc)
mean((Bhat - Bmat)^2)
attr(Bhat, "df")


#####**#######   PERIODICITY   #####**#######

# periodic
Bhat <- cmls(X = Xmat, Y = Ymat, const = "period")
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# periodic and structured
Bhat <- cmls(X = Xmat, Y = Ymat, const = "period", struc = struc)
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# periodic and non-negative
Bhat <- cmls(X = Xmat, Y = Ymat, const = "pernon")
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# periodic and non-negative and structured
Bhat <- cmls(X = Xmat, Y = Ymat, const = "pernon", struc = struc)
mean((Bhat - Bmat)^2)
attr(Bhat, "df")


#####**#######   SMOOTHNESS   #####**#######

# smooth
Bhat <- cmls(X = Xmat, Y = Ymat, const = "smooth")
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# smooth and structured
Bhat <- cmls(X = Xmat, Y = Ymat, const = "smooth", struc = struc)
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# smooth and periodic
Bhat <- cmls(X = Xmat, Y = Ymat, const = "smoper")
mean((Bhat - Bmat)^2)
```

```
attr(Bhat, "df")

# smooth and periodic and structured
Bhat <- cmls(X = Xmat, Y = Ymat, const = "smoper", struc = struc)
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# smooth and non-negative
Bhat <- cmls(X = Xmat, Y = Ymat, const = "smonon")
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# smooth and non-negative and structured
Bhat <- cmls(X = Xmat, Y = Ymat, const = "smonon", struc = struc)
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# smooth and periodic and non-negative
Bhat <- cmls(X = Xmat, Y = Ymat, const = "smpeno")
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# smooth and periodic and non-negative and structured
Bhat <- cmls(X = Xmat, Y = Ymat, const = "smpeno", struc = struc)
mean((Bhat - Bmat)^2)
attr(Bhat, "df")


######*******   ORTHOGONALITY   ######*******

# orthogonal
Bhat <- cmls(X = Xmat, Y = Ymat, const = "orthog")
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# orthogonal and structured
Bhat <- cmls(X = Xmat, Y = Ymat, const = "orthog", struc = struc)
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# orthgonal and non-negative
Bhat <- cmls(X = Xmat, Y = Ymat, const = "ortnon")
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# orthgonal and non-negative and structured
Bhat <- cmls(X = Xmat, Y = Ymat, const = "ortnon", struc = struc)
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# orthogonal and smooth
Bhat <- cmls(X = Xmat, Y = Ymat, const = "ortsmo")
mean((Bhat - Bmat)^2)
```

```
attr(Bhat, "df")

# orthogonal and smooth and structured
Bhat <- cmls(X = Xmat, Y = Ymat, const = "ortsmo", struc = struc)
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# orthogonal and smooth and periodic
Bhat <- cmls(X = Xmat, Y = Ymat, const = "orsmpe")
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# orthogonal and smooth and periodic and structured
Bhat <- cmls(X = Xmat, Y = Ymat, const = "orsmpe", struc = struc)
mean((Bhat - Bmat)^2)
attr(Bhat, "df")


######***######   UNIMODALITY   ######***######

# unimodal
Bhat <- cmls(X = Xmat, Y = Ymat, const = "unimod")
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# unimodal and structured
Bhat <- cmls(X = Xmat, Y = Ymat, const = "unimod", struc = struc)
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# unimodal and non-negative
Bhat <- cmls(X = Xmat, Y = Ymat, const = "uninon")
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# unimodal and non-negative and structured
Bhat <- cmls(X = Xmat, Y = Ymat, const = "uninon", struc = struc)
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# unimodal and periodic
Bhat <- cmls(X = Xmat, Y = Ymat, const = "uniper")
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# unimodal and periodic and structured
Bhat <- cmls(X = Xmat, Y = Ymat, const = "uniper", struc = struc)
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# unimodal and periodic and non-negative
Bhat <- cmls(X = Xmat, Y = Ymat, const = "unpeno")
mean((Bhat - Bmat)^2)
```

```
attr(Bhat, "df")

# unimodal and periodic and non-negative and structured
Bhat <- cmls(X = Xmat, Y = Ymat, const = "unpeno", struc = struc)
mean((Bhat - Bmat)^2)
attr(Bhat, "df")


######*******######   UNIMODALITY AND SMOOTHNESS   ######*******######

# unimodal and smooth
Bhat <- cmls(X = Xmat, Y = Ymat, const = "unismo")
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# unimodal and smooth and structured
Bhat <- cmls(X = Xmat, Y = Ymat, const = "unismo", struc = struc)
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# unimodal and smooth and non-negative
Bhat <- cmls(X = Xmat, Y = Ymat, const = "unsmno")
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# unimodal and smooth and non-negative and structured
Bhat <- cmls(X = Xmat, Y = Ymat, const = "unsmno", struc = struc)
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# unimodal and smooth and periodic
Bhat <- cmls(X = Xmat, Y = Ymat, const = "unsmpe")
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# unimodal and smooth and periodic and structured
Bhat <- cmls(X = Xmat, Y = Ymat, const = "unsmpe", struc = struc)
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# unimodal and smooth and periodic and non-negative
Bhat <- cmls(X = Xmat, Y = Ymat, const = "unsmpn")
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# unimodal and smooth and periodic and non-negative and structured
Bhat <- cmls(X = Xmat, Y = Ymat, const = "unsmpn", struc = struc)
mean((Bhat - Bmat)^2)
attr(Bhat, "df")


######*******######   MONOTONICITY   ######*******######
```

```
# make B
x <- 1:m
Bmat <- rbind(1 / (1 + exp(-(x - quantile(x, 0.5)))),
              1 / (1 + exp(-(x - quantile(x, 0.8)))))
struc <- rbind(rep(c(FALSE, TRUE), c(1 * m, 3 * m) / 4),
               rep(c(FALSE, TRUE), c(m, m) / 2))
Bmat <- Bmat * struc

# make noisy data
set.seed(1)
Ymat <- Xmat %*% Bmat + rnorm(m*n, sd = 0.25)

# monotonic increasing
Bhat <- cmls(X = Xmat, Y = Ymat, const = "moninc")
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# monotonic increasing and structured
Bhat <- cmls(X = Xmat, Y = Ymat, const = "moninc", struc = struc)
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# monotonic increasing and non-negative
Bhat <- cmls(X = Xmat, Y = Ymat, const = "monnon")
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# monotonic increasing and non-negative and structured
Bhat <- cmls(X = Xmat, Y = Ymat, const = "monnon", struc = struc)
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# monotonic increasing and smooth
Bhat <- cmls(X = Xmat, Y = Ymat, const = "monsmo")
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# monotonic increasing and smooth and structured
Bhat <- cmls(X = Xmat, Y = Ymat, const = "monsmo", struc = struc)
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# monotonic increasing and smooth and non-negative
Bhat <- cmls(X = Xmat, Y = Ymat, const = "mosmno")
mean((Bhat - Bmat)^2)
attr(Bhat, "df")

# monotonic increasing and smooth and non-negative and structured
Bhat <- cmls(X = Xmat, Y = Ymat, const = "mosmno", struc = struc)
mean((Bhat - Bmat)^2)
attr(Bhat, "df")
```

`const` *Print or Return Constraint Options for cmls*

Description

Prints or returns six letter constraint codes for `cmls`, along with corresponding descriptions.

Usage

```
const(x, print = TRUE)
```

Arguments

`x`	Vector of six letter constraint codes. If missing, prints/returns all 24 options.
`print`	Should constraint information be printed (`print = TRUE`) or returned as a data frame (`print = FALSE`).

Value

Prints (or returns) constraint codes and descriptions.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Helwig, N. E. (in prep). Constrained multivariate least squares in R.

See Also

Constraints are used in the `cmls` function.

Examples

```
# print some constraints
const(c("uncons", "smpeno"))

# return some constraints
const(c("uncons", "smpeno"), print = FALSE)

# print all constraints
const()

# return all constraints
const(print = FALSE)
```

cv.cmls *Cross-Validation for cmls*

Description

Does k-fold or generalized cross-validation to tune the constraint options for `cmls`. Tunes the model with respect to any combination of the arguments `const`, `df`, `degree`, and/or `intercept`.

Usage

```
cv.cmls(X, Y, nfolds = 2, foldid = NULL, parameters = NULL,
        const = "uncons", df = 10, degree = 3, intercept = TRUE,
        mse = TRUE, parallel = FALSE, cl = NULL, verbose = TRUE, ...)
```

Arguments

`X`	Matrix of dimension n x p.
`Y`	Matrix of dimension n x m.
`nfolds`	Number of folds for k-fold cross-validation. Ignored if `foldid` argument is provided. Set `nfolds=1` for generalized cross-validation (GCV).
`foldid`	Factor or integer vector of length n giving the fold identification for each observation.
`parameters`	Parameters for tuning. Data frame with columns `const`, `df`, `degree`, and `intercept`. See Details.
`const`	Parameters for tuning. Character vector specifying constraints for tuning. See Details.
`df`	Parameters for tuning. Integer vector specifying degrees of freedom for tuning. See Details.
`degree`	Parameters for tuning. Integer vector specifying polynomial degrees for tuning. See Details.
`intercept`	Parameters for tuning. Logical vector specifying intercepts for tuning. See Details.
`mse`	If `TRUE` (default), the mean squared error is used as the CV loss function. Otherwise the mean absolute error is used.
`parallel`	Logical indicating if `parSapply` should be used. See Examples.
`cl`	Cluster created by `makeCluster`. Only used when `parallel = TRUE`. Recommended usage: `cl = makeCluster(detectCores())`
`verbose`	If `TRUE`, tuning progress is printed via `txtProgressBar`. Ignored if `parallel = TRUE`.
`...`	Additional arguments to the `cmls` function, e.g., `z`, `struc`, `backfit`, etc.

Details

The parameters for tuning can be supplied via one of two options:

(A) Using the `parameters` argument. In this case, the argument `parameters` must be a data frame with columns `const`, `df`, `degree`, and `intercept`, where each row gives a combination of parameters for the CV tuning.

(B) Using the `const`, `df`, `degree`, and `intercept` arguments. In this case, the `expand.grid` function is used to create the `parameters` data frame, which contains all combinations of the arguments `const`, `df`, `degree`, and `intercept`. Duplicates are removed before the CV tuning.

Value

`best.parameters`
: Best combination of parameters, i.e., the combination that minimizes the `cvloss`.

`top5.parameters`
: Top five combinations of parameters, i.e., the combinations that give the five smallest values of the `cvloss`.

`full.parameters`
: Full set of parameters. Data frame with `cvloss` (GCV, MSE, or MAE) for each combination of `parameters`.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Helwig, N. E. (in prep). Constrained multivariate least squares in R.

See Also

See the `cmls` and `const` functions for further details on the available constraint options.

Examples

```
# make X
set.seed(1)
n <- 50
m <- 20
p <- 2
Xmat <- matrix(rnorm(n*p), nrow = n, ncol = p)


# make B (which satisfies all constraints except monotonicity)
x <- seq(0, 1, length.out = m)
Bmat <- rbind(sin(2*pi*x), sin(2*pi*x+pi)) / sqrt(4.75)
struc <- rbind(rep(c(TRUE, FALSE), each = m / 2),
               rep(c(FALSE, TRUE), each = m / 2))
Bmat <- Bmat * struc
```

```
# make noisy data
Ymat <- Xmat %*% Bmat + rnorm(n*m, sd = 0.5)


# 5-fold CV:  tune df (5,...,15) for const = "smooth"
kcv <- cv.cmls(X = Xmat, Y = Ymat, nfolds = 5,
               const = "smooth", df = 5:15)
kcv$best.parameters
kcv$top5.parameters
plot(kcv$full.parameters$df, kcv$full.parameters$cvloss, t = "b")


## Not run:

# sample foldid for 5-fold CV
set.seed(2)
foldid <- sample(rep(1:5, length.out = n))


# 5-fold CV:  tune df (5,...,15) w/ all 20 relevant constraints (no struc)
#             using sequential computation (default)
myconst <- as.character(const(print = FALSE)$label[-c(13:16)])
system.time({
  kcv <- cv.cmls(X = Xmat, Y = Ymat, foldid = foldid,
                 const = myconst, df = 5:15)
})
kcv$best.parameters
kcv$top5.parameters


# 5-fold CV:  tune df (5,...,15) w/ all 20 relevant constraints (no struc)
#             using parallel package for parallel computations
myconst <- as.character(const(print = FALSE)$label[-c(13:16)])
system.time({
   cl <- makeCluster(2L)  # using 2 cores
   kcv <- cv.cmls(X = Xmat, Y = Ymat, foldid = foldid,
                  const = myconst, df = 5:15,
                  parallel = TRUE, cl = cl)
   stopCluster(cl)
})
kcv$best.parameters
kcv$top5.parameters


# 5-fold CV:  tune df (5,...,15) w/ all 20 relevant constraints (w/ struc)
#             using sequential computation (default)
myconst <- as.character(const(print = FALSE)$label[-c(13:16)])
system.time({
  kcv <- cv.cmls(X = Xmat, Y = Ymat, foldid = foldid,
                 const = myconst, df = 5:15, struc = struc)
})
kcv$best.parameters
kcv$top5.parameters
```

```
# 5-fold CV:  tune df (5,...,15) w/ all 20 relevant constraints (w/ struc)
#             using parallel package for parallel computations
myconst <- as.character(const(print = FALSE)$label[-c(13:16)])
system.time({
  cl <- makeCluster(2L)  # using 2 cores
  kcv <- cv.cmls(X = Xmat, Y = Ymat, foldid = foldid,
                 const = myconst, df = 5:15, struc = struc,
                 parallel = TRUE, cl = cl)
  stopCluster(cl)
})
kcv$best.parameters
kcv$top5.parameters


## End(Not run)
```

`IsplineBasis` *I-Spline Basis for Monotonic Polynomial Splines*

Description

Generate the I-spline basis matrix for a monotonic polynomial spline.

Usage

```
IsplineBasis(x, df = NULL, knots = NULL, degree = 3, intercept = FALSE,
             Boundary.knots = range(x))
```

Arguments

`x`	the predictor variable. Missing values are **not** allowed.
`df`	degrees of freedom; if specified the number of `knots` is defined as `df - degree - ifelse(intercept, 1, 0)`; the `knots` are placed at the quantiles of `x`
`knots`	the internal breakpoints that define the spline (typically the quantiles of `x`)
`degree`	degree of the M-spline basis—default is 3 for cubic splines
`intercept`	if `TRUE`, the basis includes an intercept column
`Boundary.knots`	boundary points for M-spline basis; defaults to min and max of `x`

Details

Syntax is adapted from the `bs` function in the **splines** package (R Core Team, 2021).

Used for implementing monotonic smoothness constraints in the `cmls` fucntion.

Value

A matrix of dimension `c(length(x), df)` where either `df` was supplied or `df = length(knots) + degree + ifelse(intercept, 1, 0)`

Note

I-spline basis functions are created by integrating M-spline basis functions.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

R Core Team (2023). R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/

Ramsay, J. O. (1988). Monotone regression splines in action. *Statistical Science, 3*, 425-441. doi:10.1214/ss/1177012761

See Also

`MsplineBasis`

Examples

```
x <- seq(0, 1, length.out = 101)
I <- IsplineBasis(x, df = 8, intercept = TRUE)
plot(x, I[,1], ylim = c(0, 1), t = "l")
for(j in 2:8) lines(x, I[,j], col = j)
```

---

mlsei *Multivariate Least Squares with Equality/Inequality Constraints*

Description

Finds the q x p matrix `B` that minimizes the multivariate least squares problem

`sum(( Y - X %*% t(Z %*% B) )^2)`

subject to `t(A) %*% B[,j] >= b` for all `j = 1:p`. Unique basis functions and constraints are allowed for each column of `B`.

Usage

```
mlsei(X, Y, Z, A, b, meq,
      backfit = FALSE, maxit = 1000,
      eps = 1e-10, del = 1e-6,
```

```
XtX = NULL, ZtZ = NULL,
simplify = TRUE, catchError = FALSE)
```

Arguments

`X`	Matrix of dimension n x p.
`Y`	Matrix of dimension n x m.
`Z`	Matrix of dimension m x q. Can also input a list (see Note). If missing, then `Z = diag(m)` so that $q = m$.
`A`	Constraint matrix of dimension q x r. Can also input a list (see Note). If missing, no constraints are imposed.
`b`	Consraint vector of dimension r x 1. Can also input a list (see Note). If missing, then `b = rep(0, r)`.
`meq`	The first `meq` columns of `A` are equality constraints, and the remaining `r - meq` are inequality constraints. Can also input a vector (see Note). If missing, then `meq = 0`.
`backfit`	Estimate `B` via back-fitting (`TRUE`) or vectorization (`FALSE`). See Details.
`maxit`	Maximum number of iterations for back-fitting algorithm. Ignored if `backfit = FALSE`.
`eps`	Convergence tolerance for back-fitting algorithm. Ignored if `backfit = FALSE`.
`del`	Stability tolerance for back-fitting algorithm. Ignored if `backfit = FALSE`.
`XtX`	Crossproduct matrix: `XtX = crossprod(X)`.
`ZtZ`	Crossproduct matrix: `ZtZ = crossprod(Z)`.
`simplify`	If `Z` is a list, should `B` be returned as a matrix (if possible)? See Note.
`catchError`	If `catchError = FASLE`, an error induced by `solve.QP` will be returned. Otherwise `tryCatch` will be used in attempt to catch the error.

Details

If `backfit = FALSE` (default), a closed-form solution is used to estimate `B` whenever possible. Otherwise a back-fitting algorithm is used, where the columns of `B` are updated sequentially until convergence. The backfitting algorithm is determined to have converged when

`mean((B.new - B.old)^2) < eps * (mean(B.old^2) + del)`,

where `B.old` and `B.new` denote the parameter estimates at iterations t and $t + 1$ of the backfitting algorithm.

Value

If `Z` is a list with $q_j = q$ for all $j = 1, \ldots, p$, then...

`B`	is returned as a q x p matrix when `simplify = TRUE`
`B`	is returned as a list of length p when `simplify = FALSE`

If `Z` is a list with $q_j \neq q$ for some j, then `B` is returned as a list of length p.

Otherwise `B` is returned as a q x p matrix.

Note

The `Z` input can also be a list of length p where `Z[[j]]` contains a m x q_j matrix. If $q_j = q$ for all $j = 1, \ldots, p$ and `simplify = TRUE`, the output `B` will be a matrix. Otherwise `B` will be a list of length p where `B[[j]]` contains a q_j x 1 vector.

The `A` and `b` inputs can also be lists of length p where `t(A[[j]]) %*% B[,j] >= b[[j]]` for all $j = 1, \ldots, p$. If `A` and `b` are lists of length p, the `meq` input should be a vector of length p indicating the number of equality constraints for each element of `A`.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Goldfarb, D., & Idnani, A. (1983). A numerically stable dual method for solving strictly convex quadratic programs. Mathematical Programming, 27, 1-33. doi:10.1007/BF02591962

Helwig, N. E. (in prep). Constrained multivariate least squares in R.

Ten Berge, J. M. F. (1993). Least Squares Optimization in Multivariate Analysis. Volume 25 of M & T Series. DSWO Press, Leiden University. ISBN: 9789066950832

Turlach, B. A., & Weingessel, A. (2019). quadprog: Functions to solve Quadratic Programming Problems. R package version 1.5-8. https://CRAN.R-project.org/package=quadprog

See Also

`cmls` calls this function for several of the constraints.

Examples

```
######*********   GENERATE DATA   ######*********

# make X
set.seed(2)
n <- 50
m <- 20
p <- 2
Xmat <- matrix(rnorm(n*p), nrow = n, ncol = p)

# make B (which satisfies all constraints except monotonicity)
x <- seq(0, 1, length.out = m)
Bmat <- rbind(sin(2*pi*x), sin(2*pi*x+pi)) / sqrt(4.75)
struc <- rbind(rep(c(TRUE, FALSE), each = m / 2),
               rep(c(FALSE, TRUE), each = m / 2))
Bmat <- Bmat * struc

# make noisy data
set.seed(1)
Ymat <- Xmat %*% Bmat + rnorm(n*m, sd = 0.25)
```

```
#####*******#####   UNCONSTRAINED   #####*******#####

# unconstrained
Bhat.cmls <- cmls(X = Xmat, Y = Ymat, const = "uncons")
Bhat.mlsei <- t(mlsei(X = Xmat, Y = Ymat))
mean((Bhat.cmls - Bhat.mlsei)^2)

# unconstrained and structured (note: cmls is more efficient)
Bhat.cmls <- cmls(X = Xmat, Y = Ymat, const = "uncons", struc = struc)
Amat <- vector("list", p)
meq <- rep(0, p)
for(j in 1:p){
   meq[j] <- sum(!struc[j,])
   if(meq[j] > 0){
      A <- matrix(0, nrow = m, ncol = meq[j])
      A[!struc[j,],] <- diag(meq[j])
      Amat[[j]] <- A
   } else {
      Amat[[j]] <- matrix(0, nrow = m, ncol = 1)
   }
}
Bhat.mlsei <- t(mlsei(X = Xmat, Y = Ymat, A = Amat, meq = meq))
mean((Bhat.cmls - Bhat.mlsei)^2)


#####*******#####   NON-NEGATIVITY   #####*******#####

# non-negative
Bhat.cmls <- cmls(X = Xmat, Y = Ymat, const = "nonneg")
Bhat.mlsei <- t(mlsei(X = Xmat, Y = Ymat, A = diag(m)))
mean((Bhat.cmls - Bhat.mlsei)^2)

# non-negative and structured (note: cmls is more efficient)
Bhat.cmls <- cmls(X = Xmat, Y = Ymat, const = "nonneg", struc = struc)
eye <- diag(m)
meq <- rep(0, p)
for(j in 1:p){
   meq[j] <- sum(!struc[j,])
   Amat[[j]] <- eye[,sort(struc[j,], index.return = TRUE)$ix]
}
Bhat.mlsei <- t(mlsei(X = Xmat, Y = Ymat, A = Amat, meq = meq))
mean((Bhat.cmls - Bhat.mlsei)^2)


# see internals of cmls.R for further examples
```

`mlsun` *Multivariate Least Squares with Unimodality (and E/I) Constraints*

Description

Finds the q x p matrix B that minimizes the multivariate least squares problem

sum((Y - X %*% t(Z %*% B))^2)

subject to Z %*% B[,j] is unimodal and t(A) %*% B[,j] >= b for all j = 1:p. Unique basis functions and constraints are allowed for each column of B.

Usage

```
mlsun(X, Y, Z, A, b, meq,
      mode.range = NULL, maxit = 1000,
      eps = 1e-10, del = 1e-6,
      XtX = NULL, ZtZ = NULL,
      simplify = TRUE, catchError = FALSE)
```

Arguments

X	Matrix of dimension n x p.
Y	Matrix of dimension n x m.
Z	Matrix of dimension m x q. Can also input a list (see Note). If missing, then Z = diag(m) so that $q = m$.
A	Constraint matrix of dimension q x r. Can also input a list (see Note). If missing, no equality/inequality (E/I) constraints are imposed.
b	Consraint vector of dimension r x 1. Can also input a list (see Note). If missing, then b = rep(0, r).
meq	The first meq columns of A are equality constraints, and the remaining r - meq are inequality constraints. Can also input a vector (see Note). If missing, then meq = 0.
mode.range	Mode search ranges, which should be a 2 x p matrix of integers such that 1 <= mode.range[1,j] <= mode.range[2,j] <= m for all j = 1:p. Default is mode.range = matrix(c(1, m), 2, p).
maxit	Maximum number of iterations for back-fitting algorithm. Ignored if backfit = FALSE.
eps	Convergence tolerance for back-fitting algorithm. Ignored if backfit = FALSE.
del	Stability tolerance for back-fitting algorithm. Ignored if backfit = FALSE.
XtX	Crossproduct matrix: XtX = crossprod(X).
ZtZ	Crossproduct matrix: ZtZ = crossprod(Z).
simplify	If Z is a list, should B be returned as a matrix (if possible)? See Note.
catchError	If catchError = FASLE, an error induced by solve.QP will be returned. Otherwise tryCatch will be used in attempt to catch the error.

Details

A back-fitting algorithm is used to estimate `B`, where the columns of `B` are updated sequentially until convergence (outer loop). For each column of `B`, (the inner loop of) the algorithm searches for the j-th mode across the search range specified by the j-th column of `mode.range`. The backfitting algorithm is determined to have converged when

`mean((B.new - B.old)^2) < eps * (mean(B.old^2) + del)`,

where `B.old` and `B.new` denote the parameter estimates at outer iterations t and $t + 1$ of the backfitting algorithm.

Value

If `Z` is a list with $q_j = q$ for all $j = 1, \ldots, p$, then...

- `B` — is returned as a q x p matrix when `simplify = TRUE`
- `B` — is returned as a list of length p when `simplify = FALSE`

If `Z` is a list with $q_j \neq q$ for some j, then `B` is returned as a list of length p.

Otherwise `B` is returned as a q x p matrix.

Note

The `Z` input can also be a list of length p where `Z[[j]]` contains a m x q_j matrix. If $q_j = q$ for all $j = 1, \ldots, p$ and `simplify = TRUE`, the output `B` will be a matrix. Otherwise `B` will be a list of length p where `B[[j]]` contains a q_j x 1 vector.

The `A` and `b` inputs can also be lists of length p where `t(A[[j]]) %*% B[,j] >= b[[j]]` for all $j = 1, \ldots, p$. If `A` and `b` are lists of length p, the `meq` input should be a vector of length p indicating the number of equality constraints for each element of `A`.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Goldfarb, D., & Idnani, A. (1983). A numerically stable dual method for solving strictly convex quadratic programs. Mathematical Programming, 27, 1-33. doi:10.1007/BF02591962

Helwig, N. E. (in prep). Constrained multivariate least squares in R.

Ten Berge, J. M. F. (1993). Least Squares Optimization in Multivariate Analysis. Volume 25 of M & T Series. DSWO Press, Leiden University. ISBN: 9789066950832

Turlach, B. A., & Weingessel, A. (2019). quadprog: Functions to solve Quadratic Programming Problems. R package version 1.5-8. https://CRAN.R-project.org/package=quadprog

See Also

`cmls` calls this function for the unimodality constraints.

Examples

```
######*********   GENERATE DATA   ######*********

# make X
set.seed(2)
n <- 50
m <- 20
p <- 2
Xmat <- matrix(rnorm(n*p), nrow = n, ncol = p)

# make B (which satisfies all constraints except monotonicity)
x <- seq(0, 1, length.out = m)
Bmat <- rbind(sin(2*pi*x), sin(2*pi*x+pi)) / sqrt(4.75)
struc <- rbind(rep(c(TRUE, FALSE), each = m / 2),
               rep(c(FALSE, TRUE), each = m / 2))
Bmat <- Bmat * struc

# make noisy data
set.seed(1)
Ymat <- Xmat %*% Bmat + rnorm(n*m, sd = 0.25)


######*********   UNIMODALITY   ######*********

# unimodal
Bhat.cmls <- cmls(X = Xmat, Y = Ymat, const = "unimod")
Bhat.mlsun <- t(mlsun(X = Xmat, Y = Ymat))
mean((Bhat.cmls - Bhat.mlsun)^2)

# unimodal and structured
Bhat.cmls <- cmls(X = Xmat, Y = Ymat, const = "unimod", struc = struc)
Amat <- vector("list", p)
meq <- rep(0, p)
for(j in 1:p){
   meq[j] <- sum(!struc[j,])
   if(meq[j] > 0){
      A <- matrix(0, nrow = m, ncol = meq[j])
      A[!struc[j,],] <- diag(meq[j])
      Amat[[j]] <- A
   } else {
      Amat[[j]] <- matrix(0, nrow = m, ncol = 1)
   }
}
Bhat.mlsun <- t(mlsun(X = Xmat, Y = Ymat, A = Amat, meq = meq))
mean((Bhat.cmls - Bhat.mlsun)^2)

# unimodal and non-negative
Bhat.cmls <- cmls(X = Xmat, Y = Ymat, const = "uninon")
Bhat.mlsun <- t(mlsun(X = Xmat, Y = Ymat, A = diag(m)))
mean((Bhat.cmls - Bhat.mlsun)^2)

# unimodal and non-negative and structured
```

```
Bhat.cmls <- cmls(X = Xmat, Y = Ymat, const = "uninon", struc = struc)
eye <- diag(m)
meq <- rep(0, p)
for(j in 1:p){
   meq[j] <- sum(!struc[j,])
   Amat[[j]] <- eye[,sort(struc[j,], index.return = TRUE)$ix]
}
Bhat.mlsun <- t(mlsun(X = Xmat, Y = Ymat, A = Amat, meq = meq))
mean((Bhat.cmls - Bhat.mlsun)^2)


# see internals of cmls.R for further examples
```

MsplineBasis *M-Spline Basis for Polynomial Splines*

Description

Generate the M-spline basis matrix for a polynomial spline.

Usage

```
MsplineBasis(x, df = NULL, knots = NULL, degree = 3, intercept = FALSE,
             Boundary.knots = range(x), periodic = FALSE)
```

Arguments

x	the predictor variable. Missing values are **not** allowed.
df	degrees of freedom; if specified the number of `knots` is defined as `df - degree - ifelse(intercept, 1, 0)`; the `knots` are placed at the quantiles of `x`
knots	the internal breakpoints that define the spline (typically the quantiles of `x`)
degree	degree of the piecewise polynomial—default is 3 for cubic splines
intercept	if `TRUE`, the basis includes an intercept column
Boundary.knots	boundary points for M-spline basis; defaults to min and max of `x`
periodic	if `TRUE`, the M-spline basis is constrained to be periodic

Details

Syntax is adapted from the `bs` function in the **splines** package (R Core Team, 2021).

Used for implementing various types of smoothness constraints in the `cmls` fucntion.

Value

A matrix of dimension `c(length(x), df)` where either `df` was supplied or `df = length(knots) + degree + ifelse(intercept, 1, 0)`

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

R Core Team (2023). R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/

Ramsay, J. O. (1988). Monotone regression splines in action. *Statistical Science, 3*, 425-441. doi:10.1214/ss/1177012761

See Also

IsplineBasis

Examples

```
x <- seq(0, 1, length.out = 101)
M <- MsplineBasis(x, df = 8, intercept = TRUE)
M <- scale(M, center = FALSE)
plot(x, M[,1], ylim = range(M), t = "l")
for(j in 2:8) lines(x, M[,j], col = j)
```

Index

∗ **models**
cmls, 3
cv.cmls, 12
mlsei, 16
mlsun, 19
∗ **multivariate**
cmls, 3
cv.cmls, 12
mlsei, 16
mlsun, 19
∗ **optimize**
cmls, 3
cv.cmls, 12
mlsei, 16
mlsun, 19
∗ **package**
CMLS-package, 2
∗ **regression**
cmls, 3
cv.cmls, 12
IsplineBasis, 15
mlsei, 16
mlsun, 19
MsplineBasis, 23
∗ **smooth**
cmls, 3
cv.cmls, 12
IsplineBasis, 15
mlsei, 16
mlsun, 19
MsplineBasis, 23

CMLS *(*CMLS-package*)*, 2
cmls, *2, 3*, 3, *11–13*, *15*, *18*, *21*, *23*
CMLS-package, 2
const, *2, 3*, *5*, 11, *13*
cv.cmls, *2*, *5*, 12

expand.grid, *13*

IsplineBasis, 15, *24*

makeCluster, *12*
mlsei, *3*, *5*, 16
mlsun, *3*, *5*, 19
MsplineBasis, *16*, 23

parSapply, *12*

solve.QP, *17*, *20*

tryCatch, *17*, *20*
txtProgressBar, *12*