Package ‘expandFunctions’

July 22, 2025

Type Package

Date 2016-09-19

Title Feature Matrix Builder

Version 0.1.0

Imports utils, stats, graphics, plyr, orthopolynom, polynom, glmnet

Description Generates feature matrix outputs from R object inputs
using a variety of expansion functions. The generated
feature matrices have applications as inputs
for a variety of machine learning algorithms.
The expansion functions are based on coercing the input
to a matrix, treating the columns as features and
converting individual columns or combinations into blocks of
columns.
Currently these include expansion of columns by
efficient sparse embedding by vectors of lags,
quadratic expansion into squares and unique products,
powers by vectors of degree,
vectors of orthogonal polynomials functions,
and block random affine projection transformations (RAPTs).
The transformations are
magrittr- and cbind-friendly, and can be used in a
building block fashion. For instance, taking the cos() of
the output of the RAPT transformation generates a
stationary kernel expansion via Bochner's theorem, and this
expansion can then be cbind-ed with other features.
Additionally, there are utilities for replacing features,
removing rows with NAs,
creating matrix samples of a given distribution,
a simple wrapper for LASSO with CV,
a Freeman-Tukey transform,
generalizations of the outer function,
matrix size-preserving discrete difference by row,
plotting, etc.

License GPL-2

NeedsCompilation no

LazyData TRUE

RoxygenNote 5.0.1

Author Scott Miller [aut, cre]

Maintainer Scott Miller <sam3CRAN@gmail.com>

Repository CRAN

Date/Publication 2016-10-01 15:05:50

Contents

coefPlot 2
distMat 4
easyLASSO 5
eDiff 6
eLag 6
eMatrixOuter 7
eOuter 8
ePow 9
eQuad 9
eReplace 10
eTrim 11
expandFunctions 12
freemanTukey 14
lagshift 15
polywrapper 16
rapt 17
raptMake 19
reset.warnings 21
Ydiagnostics 21
yyHatPlot 22

Index **24**

`coefPlot` *Plots coefficients in an impulse response format*

Description

Given a model xObj for which coef(xObj) returns a set of coefficients, plot the coefficients.

The plots make it easier to compare which features are large, which are set to zero, and how features change from run to run in a graphical manner.

If the fitting process is linear (e.g. lm, glmnet, etc.) and the original features are appropriately ordered lags, this will generate an impulse response.

Any coefficients that are *exactly* zero (for instance, set that way by LASSO) will appear as red X's; non-zero points will be black O's.

Usage

```
coefPlot(xObj, includeIntercept = FALSE, type = "h", main = NULL, ...)
```

Arguments

`xObj`	Output of a fitting model.
`includeIntercept`	Should the 1st coefficient be plotted? Default is FALSE.
`type`	Graphics type. Default is "h", which results in an impulse-like plot.
`main`	"main" title; default is the relative number of non-zero coefficients, a measure of sparsity.
`...`	Optional additional graphical parameters, for instance to set ylim to a fixed value.

Details

If includeIntercept==TRUE, the intercept of the model will be plotted as index 0.

Changing the type using `type="b"` will result in a parallel coordinate-like plot rather than an impulse-like plot. It is sometimes easier to see the differences in coefficients with type="b" rather than type="h".

Value

Invisibly returns TRUE. Used for its graphic side effects only.

Examples

```
set.seed(1)
nObs <- 100
X <- distMat(nObs,6)
A <- cbind(c(1,0,-1,rep(0,3))) # Y will only depend on X[,1] and X[,3]
Y <- X %*% A + 0.1*rnorm(nObs)
lassoObj <- easyLASSO(X,Y)
Yhat <- predict(lassoObj,newx=X)
yyHatPlot(Y,Yhat)
coef( lassoObj ) # Sparse coefficients
coefPlot( lassoObj )
coefPlot( lassoObj, includeIntercept=TRUE )
coefPlot( lassoObj, type="b" )
```

distMat *Make a matrix with coefficients distributed as dist*

Description

Generate a pXq matrix with coefficients drawn from the univariate distribution dist with options distOpt.

Usage

```
distMat(p, q, dist = rnorm, distOpt = NULL)
```

Arguments

`p`	Number of rows
`q`	Number of columns
`dist`	distribution of coefficients to draw from; default is rnorm.
`distOpt`	Named list of additional parameters for dist. *Always omit the first parameter,n, of the distribution sampling function*. Defaults may be omitted if desired (see examples).

Details

The user may provide their own distribution function, but note that the number of values to return, n, *must* be the first argument, just as with the built-in distributions. The first argument does not have to be named.

Value

A pXq matrix with coefficients distributed as dist with parameters defined in '...'

Examples

```
X <- distMat(10,2)
X <- distMat(10,2,distOpt=list(mean=1,sd=2))
X <- distMat(5,3,rnorm,list(mean=1,sd=2))
X <- distMat(5,3,rnorm,list(sd=2))
X <- distMat(50,3,rt,list(df=3))
```

easyLASSO *Select and fit sparse linear model with LASSO*

Description

The purpose of this function is to make the process of LASSO modelling as simple as possible.

This is a simple wrapper on two glmnet functions which, when given input matrix X and response vector y, and a criterion for model selection, will estimate the lambda parameter, and return the LASSO results as a glmnet model. This model can then be used to find coefficients and predictions.

Usage

```
easyLASSO(X, y, criterion = "lambda.1se")
```

Arguments

`X`	Predictor matrix, nXp, with n observations and p features.
`y`	Response vector, or column or row matrix. Must have length n.
`criterion`	String describing which lambda criterion to use in selecting a LASSO model. Choices currently are c("lambda.1se","lambda.min").

Value

a glmnet model

See Also

`glmnet` and `cv.glmnet`

Examples

```
set.seed(1)
nObs <- 100
X <- distMat(nObs,6)
A <- cbind(c(1,0,-1,rep(0,3)))
  # Y will only depend on X[,1] and X[,3]
Y <- X %*% A + 0.1*rnorm(nObs)
lassoObj <- easyLASSO(X=X,y=Y) # LASSO fitting
Yhat <- predict(lassoObj,newx=X)
yyHatPlot(Y,Yhat)
coef( lassoObj ) # Sparse coefficients
coefPlot( lassoObj )
```

eDiff *Matrix size-preserving diff function*

Description

Returns a matrix, the same size as the input matrix X, containing the back difference.

Usage

```
eDiff(X, pad = NA)
```

Arguments

X	R object coercible to matrix
pad	Pad the first row with this value; the default is NA. 0 would be another value often used in signal processing.

Value

Returns a matrix, the same size as the input matrix X, containing the back difference by column. The first row is filled with copies of pad.

Examples

```
eDiff( 1:8 )
eDiff( as.data.frame(1:8) )
eDiff( matrix(1:8,ncol=2) )
```

eLag *Convert vector into a matrix of lag columns*

Description

Convert vector into a matrix of lag columns

Usage

```
eLag(x, colParamVector, pad = NA)
```

Arguments

x	Data *vector*
colParamVector	Vector of lags for embedding
pad	Scalar for padding embedding

Value

A matrix whose columns are x lagged by the corresponding values in colParamVector.

See Also

`embed` and `embedd`, which are related functions.

Examples

```
eLag(1:6, 0:2)
eLag(1:6, 0:2, pad=0)
```

---

eMatrixOuter — *Extends eOuter to allow a matrix for the first argument*

Description

Extends eOuter to allow a matrix for the first argument

Usage

```
eMatrixOuter(X, colParamVector, FUN, ...)
```

Arguments

`X`	R object coercible to a matrix the columns of this will be the argument of FUN (see below).
`colParamVector`	Vector input which will be the second argument of FUN (see below).
`FUN`	Function which will be applied to FUN(X[,i],colParamVector[j],...)
`...`	Additional arguments to FUN.

Details

This function is a simple extension of eOuter which allows the function eOuter(X[,i],colParamVector,FUN,...) for i in the columns of X.

Value

Returns a matrix with the matrics generated by eOuter for each column column bound together. This means that each row of the returned matrix represents single observations (at least as long as no lags are used).

Examples

```
A <- matrix(1:6,ncol=2)
temp <- eMatrixOuter(A,0:2,FUN=`^`)
```

eOuter *Extend outer product.*

Description

Extends outer {base} `outer(x,y,FUN)` to include functions `FUN(x,y,...)` where the first argument of `FUN` is a vector but the second argument must be a scalar.

Usage

```
eOuter(x, y, FUN, ...)
```

Arguments

`x`	Vector, with the same function as in outer {base}. Each value will correspond to a row in the return matrix.
`y`	Vector. Each element in the vector corresponds to a column in the return matrix.
`FUN`	Function. `x` and `y` will be the first and second arguments. Unlike `outer`, however, while a vector can be the first argument, FUN might only allow *one value* as the second argument. This means eOuter can use lagshift, for instance, as FUN.
`...`	Additional parameters for FUN.

Details

outer has limitations; it only works with functions which can take vector inputs for *both* the first and second arguments, such as "^". As a result, many functions cannot be used for FUN. The function eOuter gets around this limitation by additionally allowing functions which accept a vector for the first argument, but only scalars for the second argument. It can be used everywhere that `outer` can be used, but also when FUN is limited in this way.

Value

A matrix `Z` of size `length(x)` X `length(y)` containing `Z[,i]` with values `FUN(x,y[i],...)`.

See Also

`outer` and `ePow`

Examples

```
# This implements a function similar to ePow
# FIXME: change ePow to use eOuter!!!
eOuter(1:6,0:2,FUN = `^`)
# Other functions of columns
eOuter(1:10,0:3,FUN = lagshift,lagMax=3,pad=NA)
# FIXME: Make function to allow polynomials to be used:
# eOuter(1:10,1:3,FUN = glaguerre.polynomials, alpha=0.5)
```

ePow *Convert vector x into a matrix $X\_ij = x\_i\hat{}j$*

Description

Convert vector x into a matrix $X_{ij} = {x_i}^j$

Usage

```
ePow(x, colParamVector)
```

Arguments

`x`	Data vector.
`colParamVector`	Vector of column powers.

Value

A matrix X of size length(x) X length(colParamVector)

$$X_{ij} = {x_i}^j$$

Examples

```
x <- 1:6
ePow(x,0:2)
```

eQuad *Multivariate second order polynomial expansion.*

Description

Expand matrix columns into linear, square, and unique product columns.

Usage

```
eQuad(X, FUN = `*`, ...)
```

Arguments

`X`	vector or matrix. If a vector, it will be converted to a column matrix. If it is desired that the squares and products of a *vector* are computed, pass rbind(X) instead of X, and thereby pass a row matrix.
`FUN`	Binary function which forms the products of the columns. By default, this is '*', but other *commuting* operators or kernels can be used if desired.
`...`	Options for FUN. Not needed if FUN doesn't have options.

Details

Form a matrix with columns composed of into linear, square, and product columns:

$$[X|FUN(X[,i],X[,j])]$$

where i, j are the unique combinations of i and j, including $i = j$.

By default, the function used to form the squares and products, FUN, is just conventional multiplication = '*', but any *commuting* binary operator can be used.

This particular expansion is often applied in

- General Method of Data Handling (GMDH).
- Nonlinear Slow Feature Analysis (SFA). Performing a multivariate polynomial of second degree expansion in all the features, then performing *linear* SFA on the resulting expanded feature matrix, is a very common approach, and in fact is the default method in `sfa2 {rSFA}`.

Value

$[X, X^2, uniqueproductsofcolumnsofX]$. The unique products are in row major upper right triangular order. Thus, for X with columns 1:3, the order is

$$X[,1]^2, X[,2]^2, X[,3]^2, X[,1]*X[,2], X[,1]*X[,3], X[,2]*X[,3]$$

See Also

`sfa2`

Examples

```
# # Examples
# eQuad(1:5)
# eQuad(matrix(1:12,ncol=3),FUN=`+`)
```

`eReplace` *Replace values in an R object coerible to a matrix*

Description

Replace values in an R object coerible to a matrix. It is useful for replacing NA with other values, etc., such as with padding values.

Usage

```
eReplace(X, a, b)
```

Arguments

`X`	R object coercible to a matrix
`a`	Value to be replaced
`b`	Value to replace

Value

X with all a's replaced with b's. a may be NA

See Also

`replace`, which performs the same operation on vectors, and on which this operation is based.

Examples

```
A <- matrix(1:6,ncol=2)
A <- eReplace(A,1,NA)
A <- eReplace(A,NA,-9999)
A <- eReplace(A,-9999,0)
```

`eTrim` *Remove padded rows from matrix X*

Description

Remove padded rows from matrix X

Usage

```
eTrim(X, pad = NA)
```

Arguments

`X`	R object coercible to matrix
`pad`	Value representing padded elements. By default it is NA, but could be any value.

Value

A matrix.

Examples

```
n <- 10
x <- rnorm(n)    # x vector
X <- eLag(x,0:1) # X matrix
t <- 1:n         # time vector
T <- eLag(t,0:1) # time matrix; the column corresponding
                 # to 0 is the time for each row,
                 # even after trimming
matplot(X,type="l",lty=1)
X <- eTrim(X)
T <- eTrim(T)
matplot(x=T[,1],y=X,type="l",lty=1,
  xlab="Time")
```

`expandFunctions` *expandFunctions: a feature matrix builder*

Description

A variety of functions for conversion of vectors and matrices to other matrices to use as features. This allows one to quickly build feature structures and apply various machine learning methods to those features for exploration and pedantic purposes.

Details

The **expandFunctions** package contains functions that can be used to expand feature vectors and matrices into larger feature matrices. These functions include lag embedding, special function univariate exansion, quadratic expansion, and random vector projection.

The general steps for feature generation for time domain data (which subsumes multivariate data via lags) are:

- Preprocess data - remove mean, transform, etc., to a useful vector or matrix.
- If not a matrix, functionally expand vector into a matrix. This is typically done by lag embedding, but may also include STFT, wavelet transforms, etc.
- Functionally expand matrices generated.
- Combine resulting matrices into a single feature matrix.
- Dimensional reduction, feature selection, and/or feature extraction to reduce the number of features.
- Use machine learning method(s) on the resulting feature matrix.

Most of the steps above are well supported in R on CRAN, but the expansion steps tend to be scattered in a variety of packages, poorly represented, or custom built by the user. The **expandFunction** package is intended to collect many of these functions together in one place.

Preprocessing almost always should include centering and scaling the data. However, it may also include a variety of transformations, such as Freeman-Tukey, in order to make the vector fit more closely to a given model (say, a linear model with Gaussian noise).

If the input isn't a time domain vector, but is instead already in tabular form (for instance, Boston Housing Data), the embedding step can be skipped.

Dimension reduction is outside the scope of this package, but is normally performed to reduce the variables that need handling, reducing the memory used and speeding up the analysis.

The package functions are "magrittr-friendly", that is, built so that they can be directly pipelined since X, the data, is the first argument.

Most functions are prefixed with "e" to help distinguish them from being confused with similarly named functions.

Author(s)

Scott Miller <sam3CRAN@gmail.com>

Examples

```
## Not run:
# Sunspot counts can be somewhat Gaussianized by the
# Freeman-Tukey transform.
x <- freemanTukey(sunspot.month)
par(mfrow=c(1,1)) # just in case multiplots were left over.
plot(x,type="l")

# Embed x using eLag
# Since the base period of sunspots is 11*12 months,
# pick the lags to be fractions of this.
myLags <- seq(from=0,to=200,by=1)
X <- eTrim(eLag(x,myLags))
Y <- X[,+1,drop=FALSE]
X <- X[,-1,drop=FALSE]
# matplot(X,type="l",lty=1)

# OLS fitting on the lag feature set
lmObj <- lm(y ~ .,data = data.frame(x=X,y=Y))
coefPlot(lmObj,type="b")
summary(lmObj)
Yhat <- predict(lmObj, newdata = data.frame(x=X))
Ydiagnostics(Y,Yhat)

# LASSO fitting on the lag feature set
lassoObj <- easyLASSO(X,Y,criterion="lambda.min")
coefPlot(lassoObj,type="b")
Yhat <- predict(lassoObj,newx = X)
Ydiagnostics(Y,Yhat)

# Reduce the lag feature set using non-zero
# LASSO coefficients
useCoef <- coef(lassoObj)[-1]!=0
myLags <- seq(from=0,to=200,by=1)[c(TRUE,useCoef)]
X <- eTrim(eLag(x,myLags))
Y <- X[,+1,drop=FALSE]
X <- X[,-1,drop=FALSE]
```

```
# OLS fitting on the reduced lag feature set
lmObj <- lm(y ~ .,data = data.frame(x=X,y=Y))
summary(lmObj)
coefPlot(lmObj)
Yhat <- predict(lmObj, newdata = data.frame(x=X))
Ydiagnostics(Y,Yhat)

# 1st nonlinear feature set
# Apply a few Chebyshev functions to the columns of the
# lag matrix. Note these exclude the constant values,
# but include the linear.
chebyFUN <- polywrapper(basePoly=orthopolynom::chebyshev.t.polynomials,
                        kMax=5)
Z <- eMatrixOuter(X,1:5,chebyFUN)

# OLS fitting on the 1st nonlinear feature set
lmObj <- lm(y ~ .,data = data.frame(z=Z,y=Y))
summary(lmObj)
Yhat <- predict(lmObj, newdata = data.frame(z=Z))
Ydiagnostics(Y,Yhat)

# LASSO fitting on the 1st nonlinear feature set
lassoObj <- easyLASSO(Z,Y)
coefPlot(lassoObj)
Yhat <- predict(lassoObj,newx = Z)
Ydiagnostics(Y,Yhat)

# 2nd nonlinear feature set
# Use eQuad as an alternative expansion of the lags
Z <- eQuad(X)

# OLS fitting on the 2nd nonlinear feature set
lmObj <- lm(y ~ .,data = data.frame(z=Z,y=Y))
summary(lmObj)
Yhat <- predict(lmObj, newdata = data.frame(z=Z))
Ydiagnostics(Y,Yhat)

# LASSO fitting on the 2nd nonlinear feature set
lassoObj <- easyLASSO(Z,Y)
coefPlot(lassoObj)
Yhat <- predict(lassoObj,newx = Z)
Ydiagnostics(Y,Yhat)

## End(Not run)
```

freemanTukey *Freeman-Tukey transform*

Description

This transform takes Poisson (count) information and makes it more Gaussian, then z-scales (standardizes by centering and scaling to var = 1) the results.

Usage

```
freemanTukey(x)
```

Arguments

`x`	Data vector

Value

The transformed vector

References

Taken from `https://en.wikipedia.org/wiki/Anscombe_transform`

Examples

```
x <- freemanTukey(sunspot.month)
```

`lagshift` *Helper function for eLag.*

Description

Generates shifted columns.

Usage

```
lagshift(x, i, lagMax, pad)
```

Arguments

`x`	Input vector
`i`	Shift (integer)
`lagMax`	Maximum lag that will be needed
`pad`	Scalar used for padding.

Value

vector padded front and back with padding appropriate for generating lag.

Examples

```
lagshift(1:3,0,1,NA)
```

`polywrapper` *Generate special functions using orthonormal functions*

Description

orthopolynom can be used to generate special functions, but for expansion they should be modified. As of this writing, orthopolynom generates polynomials for Chebyshev, Hermite, Legendre and many other functions, their integrals and derivatives, and more.

Usage

```
polywrapper(basePoly = orthopolynom::chebyshev.t.polynomials, kMax = 0)
```

Arguments

`basePoly`	A polynomial list from orthopoly
`kMax`	Integer. The maximum order of the function generated.

Details

The function polywrapper does 2 things:

- Generate functions from polynomial coefficients.
- Uses x as the 1st argument, and the order as the second; this means the generated functions can be used in eOuter and eMatrixOuter.

The functions so generated can be used as simple special functions, as well as being useful in feature building.

Since the coefficients from orthopolynom are generated by recursion, an upper limit of the function order needs to be set when calling polywrapper. This is the main limitation of polywrapper. Fortunately, since the functions are compactly stored, kMax can be set quite high if desired. Note that usually the kMax is known, and is relatively small.

NB: The input x may need to be normalized. orthopolynom has the function scaleX for just such a purpose.

Value

Function which is compatible with eOuter and eMatrixOuter

Examples

```
# Generate a Chebyshev function of the form
# chebyFUN(x,k), where x is the input and k is the order.
# In this case, k must be no more than 5 (since that
# is the value passed to kMax), although it is
# easy to set this to a higher order if desired.
chebyFUN <- polywrapper(basePoly=orthopolynom::chebyshev.t.polynomials,
  kMax=5)
# Now the function chebyFUN
# can be used as any other function:
x <- seq(-1,+1,0.01)
plot(x,chebyFUN(x,5),type="l")
eOuter(seq(-1,+1,0.01),0:3,chebyFUN)
```

`rapt` *Expand an input matrix X using raptObj.*

Description

Expand an input matrix X using a Random Affine Projection Transformation (RAPT) object. Such objects use random affine projection transformation to the resulting matrix. This allows RAPT objects serve as a basis for a large number of kinds of expansions. If p are the number of features of X, and q are number of expanded features, the applications fall into two broad categories:

- p > q using the Johnson-Lindenstrauss theorem:
  - Compressed sensing.
  - Manifold learning.
  - Dimension reduction.
  - Graph embedding.
  - ...
- p < q using Bochner's theorem:
  - Approximate kernel projection.
  - Fast approximate SVD.
  - Estimation of dependence.
  - ...

Usage

```
rapt(X, raptObj)
```

Arguments

`X`	Input data matrix
`raptObj`	raptObj generated by raptMake

Details

Computes

$$XW + b$$

where

W = raptObj$W

b = raptObj$b

Value

A matrix of randomly (but repeatable) features.

References

https://en.wikipedia.org/wiki/Johnson%E2%80%93Lindenstrauss_lemma, https://en.wikipedia.org/wiki/Bochner%27s_theorem

See Also

Details of how the rapt object is built are in raptMake.

Examples

```
# Toy problem
set.seed(1)
nObs <- 100 # Number of observations
X <- matrix(seq(-4,+4,length.out = nObs),ncol=1)
Ytrue <- sin(5*X) + 2*cos(2*X) # True value Ytrue = g(X)
Y <- Ytrue + rnorm(nObs) # Noisy measurement Y

# Standardize X
Xstd <- scale(X)
attributes(Xstd) <- attributes(X)

# Bochner (random Fourier) projection object
nDim <- NCOL(X)
h <- 10 # Estimated by goodness of fit Adj R^2.
  # Normally this would be fit by cross validation.
raptObj <- raptMake(nDim,nDim*200,WdistOpt = list(sd=h),
                    bDistOpt=list(min=-pi,max=+pi))

# Apply raptObj to Xstd to generate features,
# keeping unaltered features Xstd as well.
Xrapt <- cbind( Xstd, cos( rapt(Xstd,raptObj) ) )

# Standardize results
XraptStd <- scale(Xrapt)
attributes(XraptStd) <- attributes(Xrapt)

# A linear fitting of Y to the features XraptStd
lmObj <- lm(Y ~ XraptStd)
summary(lmObj)

# Plot measurements (Y), predictions (Yhat),
# Kernel smoothing with Gaussian kernel and same bandwidth,
# true Y without noise.
Yhat <- predict(lmObj)
plot (X,Y   ,main="Linear Fitting", ylim=c(-6,10))
lines(X,Yhat,col="red",lty=1,lwd=2)
grid(col="darkgray")
kFit <- ksmooth(X,Y,kernel="normal",bandwidth=1/h)
```

```
lines(kFit$x,kFit$y,lty=1,col="green",lwd=2)
lines(X,Ytrue,lty=1,col="blue",lwd=2)
legend("topleft",
        legend=c("Noisy measurements",
                 "Estimated Y from RAPT",
                 "Estimated Y from Kernel Smooth",
                 "True Y"),
        col=1:4,
        pch=c( 1,NA,NA,NA),
        lty=c(NA, 1, 1, 1),
        lwd=2,
        bty="n")

# Fit sparse model w/LASSO and
# lambda criteria = 1 standard deviation.
# This avoids overgeneralization errors usually
# associated with fitting large numbers of features
# to relatively few data points.  It also improves
# the end effects, which are of paramount importance
# in high dimensional problems (since by the curse
# of dimensionality, almost all points are close an edge
# in high dimensional problems).
lassoObj <- easyLASSO(XraptStd,Y)
Yhat <- predict(lassoObj, newx = XraptStd)
# Use linear fit of prediction Yhat as goodness of fit.
summary(lm(Y ~ Yhat))

# Plot results of LASSO fitting
# These show LASSO does a better job fitting edges.
plot(X,Y,main="LASSO Fitting",ylim=c(-6,10))
lines(X,Yhat,col="red",lty=1,lwd=2)
grid(col="darkgray")
kFit <- ksmooth(X,Y,kernel="normal",bandwidth=1/h)
lines(kFit$x,kFit$y,lty=1,col="green",lwd=2)
lines(X,Ytrue,lty=1,col="blue",lwd=2)
legend("topleft",
        legend=c("Noisy measurements",
                 "Estimated Y from RAPT",
                 "Estimated Y from Kernel Smooth",
                 "True Y"),
        col=1:4,
        pch=c( 1,NA,NA,NA),
        lty=c(NA, 1, 1, 1),
        lwd=2,
        bty="n")
```

`raptMake` *Define a Random Affine Projection Transformation (RAPT) object*

Description

Create a Random Affine Projection Transformation (RAPT) object. Such objects use random affine projection transformation to the resulting matrix. This allows RAPT objects serve as a basis for a large number of kinds of expansions.

Usage

```
raptMake(p, q, Wdist = rnorm, WdistOpt = NULL, bDist = runif,
  bDistOpt = list(min = 0, max = 0))
```

Arguments

`p`	Number of input features (columns of `X`).
`q`	Number of output features,
`Wdist`	W distribution function. Coefficients for the random projection matrix W are drawn from this distribution. The default is rnorm.
`WdistOpt`	List of optional parameters for Wdist. If this is NULL (default), then only defaults of the distribution are used.
`bDist`	b distribution function. Coefficients for the offset vector b are drawn from this distribution. The default is runif.
`bDistOpt`	List of optional parameters for bDist. If this is NULL then only defaults of the distribution are used. The default is `bDistOpt=list(min=0,max=0)`, which results in b = 0, with no offset.

Details

This initializes a eRAPTobj, which holds all the parameters needed to perform a random projection transformation expansion (RAPT).

An RAPT of X is defined as

$$XW + b$$

where

X is the input matrix

W is a matrix of coefficients drawn from Wdist with options WdistOpt

b is a column matrix of coefficients drawn from bDist with options bDistOpt

If there is a need for multiple W or b distributions, then make multiple raptObj. This makes each raptObj fairly simple, while allowing arbitrary complexity through multiple expansion and composition.

A simple way to get a linear feature, in addition to the RAPT features, is to simply cbind the original matrix X in with the raptObj matrix.

Value

An expand object, which defines the following fields: W Input weighting matrix b Input offset matrix

Examples

```
raptObj <- raptMake(21,210,bDistOpt=list(min=-pi,max=+pi))
```

`reset.warnings` *Reset annoyingly persistent warning messages.*

Description

Reset annoyingly persistent warning messages.

Usage

```
reset.warnings()
```

Value

Returns TRUE invisibly. Used for side effects only.

References

This function is built around the snippet found here: http://stackoverflow.com/questions/5725106/r-how-to-clear-all-warnings

Examples

```
## reset.warnings()
```

`Ydiagnostics` *Informative plots for Y and Yhat*

Description

This function presents diagnostic plots of estimate Yhat and response Y.

Usage

```
Ydiagnostics(Y, Yhat, ...)
```

Arguments

`Y`	R object representing response, coercible to a vector.
`Yhat`	R object representing estimate, coercible to a vector. The length of Y and Yhat must be equal.
`...`	Options for `cor` function. The defaults are use = "everything" and method = "pearson".

Details

The plots shown are:

- Y vs Yhat. Under a perfect noise-free fitting, this would be a straight line with the points lined up on the red line, and the correlation wpuld be 1.0000.
- Y, Yhat and Y-Yhat (residual) time domain plots. The time steps are in samples.
- These show the ACF for the original Y, the residual, and |residual|. The latter helps identify nonlinearity in the residual.

Value

Invisibly returns TRUE; this routine is only used for its graphical side effects described in Details.

See Also

`cor`

Examples

```
# The order here looks backwards, but is chosen to
# simulate a typical pair - Yhat will normally have
# a smaller range than Y.
set.seed(2)
nObs <- 100 # Number of observations
x <- stats::filter(rnorm(nObs),c(-0.99),
     method="recursive")
x <- x + (x^2) # Nonlinear component
myLags <- 0:2
X <- eTrim(eLag(x,myLags))
Y <- X[,+1,drop=FALSE]
X <- X[,-1,drop=FALSE]
lmObj <- lm(Y ~ X)
Yhat <- predict(lmObj)
Ydiagnostics(Y,Yhat)
```

`yyHatPlot` *Plot y and yHat on the same scale w/reference line*

Description

Plots y and yHat on the same scale as a scatterplot with a 1:1 reference line in red. This is useful for visually comparing actual data y with estimates yHat, determining outliers, etc.

Usage

```
yyHatPlot(y, yHat, ...)
```

Arguments

`y`	Vector or matrix coercible to vector. Typically will be the quantity to be predicted.
`yHat`	Vector or matrix coercible to vector, same length as y. Typically will be the prediction.
`...`	Optional additional graph parameters.

Details

Normally only makes sense with vectors, column matrices, or row matrices.

Value

Returns invisibly - only used for graphic side effects.

Examples

```
set.seed(1)
nObs <- 80
X <- distMat(nObs,2)
A <- cbind(c(1,-1))
Y <- X %*% A + rnorm(nObs) # Response data
lmObj <- lm(Y ~ X)
Yhat <- predict(lmObj) # Estimated response
yyHatPlot(Y,Yhat)
```

Index

coefPlot, 2
cor, *21, 22*
cv.glmnet, *5*

distMat, 4

easyLASSO, 5
eDiff, 6
eLag, 6
eMatrixOuter, 7
embed, *7*
embedd, *7*
eOuter, 8
ePow, *8*, 9
eQuad, 9
eReplace, 10
eTrim, 11
expandFunctions, 12
expandFunctions-package
        (expandFunctions), 12

freemanTukey, 14

glmnet, *5*

lagshift, 15

outer, *8*

polywrapper, 16

rapt, 17
raptMake, *18*, 19
replace, *11*
reset.warnings, 21

sfa2, *10*

Ydiagnostics, 21
yyHatPlot, 22