Package 'QuickFunc'

August 25, 2021

Type Package

Title Set of Quick Functions I use quite often

Version 0.1.0

Author Samuel E. Jackson

Maintainer Samuel E. Jackson <s.e.jackson@soton.ac.uk>

Description Set of Quick Functions I use quite often.

Imports MASS, tgp, tmvtnorm, FNN, pdist

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Suggests testthat (>= 2.1.0)

R topics documented:

`ab` *Show first rows and columns of a matrix or dataframe.*

Description

Show first rows and columns of a matrix or dataframe.

Usage

```
ab(X, k = min(nrow(X), ncol(X), 10), l = k)
```

Arguments

`X`	matrix or dataframe.
`k`	scalar - number of rows to show.
`l`	scalar - number of columns to show.

Value

first `k` rows and `l` columns of `X`

Examples

```
X <- matrix( 1:210, ncol = 15)
ab(X)
```

`add_arrays` *Add Arrays*

Description

Add arrays of different sizes across chosen dimensions of the larger array.

Usage

```
add_arrays(A, B, d, operation = "addition")
```

Arguments

A	An array.
B	An array or vector.
d	A vector, indicating the dimensions of A over which we want to add B.
operation	The operation to be applied - by default addition, but multiplication is also available.

Value

The array resulting from adding array B to each layer of array A across specified dimensions d.

Examples

```
A <- array( 1:24, dim = c(2,3,4) )
B <- matrix( 1:12, nrow = 3 )
add_arrays( A, B, d = 1 )
add_arrays( A, B, d = 1, operation = "multiplication" )
```

---

arg_min *Arg min for monotonically increasing functions.*

Description

Function to find minimum argument for which function is at least as big as some value v, within some accepted precision.

Usage

```
arg_min(f, v, l, u, p)
```

Arguments

f	function
v	value
l	initial lower bound
u	initial upper bound
p	precision

Value

a, a scalar such that f(a) >= v and f(a-p) < v, along with fa = f(a).

Examples

```
sam <- sort( sample( 1:100, 10 ) )
fn <- function( x ){ if( x >= sam[1] ){ sam[sum( x >= sam )] }else{ 0 } }
arg_min( fn, v = 33, l = 0, u = 100, p = 10 )
arg_min( fn, v = 33, l = 0, u = 100, p = 0.1 )
# In the above example we know of course that the answer should be
# minimum of sam values that are greater than v.
```

arg_min_eval_count *Number of function evaluations.*

Description

Related to function arg_min, this function simply calculates the number of required evaluations of f to ensure the chosen precision p, where p in this case is taken to be a proportion of u-l.

Usage

```
arg_min_eval_count(p)
```

Arguments

p	precision

Value

number of required evaluations

Examples

```
arg_min_eval_count( 0.0001 )
```

ArrayListConversion *Array List Conversion*

Description

Convert lists of vectors or arrays (of dimension d-1) to an array of dimension d, with the elements of the list indicated by the value of the dth dimension.

Also, convert a matrix to a list of vectors, either by row or column.

Usage

```
ArrayListConversion(X, rows = FALSE, levels = NULL)
```

Arguments

X	A list of vectors/matrices or 2/3 dimensional array to be converted.
rows	Option only for converting a matrix to a list of vectors which specifies whether the vectors in the list are the rows (TRUE) or columns (FALSE, default) of the matrix.
levels	Option for converting to a list - the names of the items of the list to have.

Value

A 2/3 dimensional array or list, depending on X.

Examples

```
X <- matrix( 1:12, ncol = 3 )
ArrayListConversion( X, levels = c("cat", "dog", "rabbit") )
Y <- matrix( 1:6, ncol = 2)
Z <- matrix( 7:12, ncol = 2)
L <- list( "cat" = Y, "dog" = Z )
ArrayListConversion( L )
```

AVEM *Array Vector-Element Multiplication*

Description

Multiply each matrix/column in a 3D/2D array by the corresponding element of a vector.

Usage

```
AVEM(A, v)
```

Arguments

A	two or three-dimensional array.
v	vector

Value

array resulting from multiplying the matrices/columns in A by the corresponding elements in v.

Examples

```
A <- array( 1:24, dim = c( 2,3,4 ) )
b = 1:4
AVEM( A, b )
```

---

colSort *Sort Each Column of Matrix*

Description

Sort Each Column of Matrix

Usage

```
colSort(X, decreasing = FALSE, na.last = TRUE)
```

Arguments

X	a matrix.
decreasing	logical. Should the sort order be increasing or decreasing?
na.last	for controlling the treatment of NAs. If TRUE, missing values in the data are put last; if FALSE, they are put first; if NA, they are removed (see 'Note'.)

Value

a matrix with each column containing the sorted elements of the columns in X.

See Also

order

Examples

```
X <- matrix( sample( 50, 20 ), ncol = 4 )
colSort( X )
colSort( X, decreasing = TRUE )
```

CompareAndAdd *Compare and Add Variance Arrays and Vectors*

Description

Check whether the dimension (up to 3) of two objects are the same - then perform appropriate addition of the two.

If one is an array and the other a matrix, the matrix is added to each layer of the array. Otherwise, addition happens as if the "+" operator had been used between the two.

This function needs amending when we recall what the intention of said function was (adding variance arrays/matrices). Think it may be good if there are more checks involved. Also, do the first two dimensions (of an array) need to be equal? If the answer if yes, then the example needs changing and this needs to be checked for.

Usage

```
CompareAndAdd(A, B)
```

Arguments

A	array (of dimension 2 or 3) or vector.
B	array (of dimension 2 or 3) or vector.

Value

Array resulting from the appropriate addition between A and B.

Examples

```
A <- array( 1:24, dim = c( 2,3,4 ) )
B <- matrix( 1:6, nrow=2 )
CompareAndAdd( A, B )
```

deg2rad *Convert Degrees to Radians*

Description

Convert Degrees to Radians

Usage

```
deg2rad(d)
```

Arguments

d	angle in degrees

Value

angle in radians

Examples

```
deg2rad( 60 )
```

euclidean_distance *Calculate Euclidean distance between two vectors*

Description

Calculate Euclidean distance between two vectors

Usage

```
euclidean_distance(x1, x2)
```

Arguments

x1	vector
x2	vector

Value

scalar output with the distance between the two vectors.

Examples

```
a <- c( 3, 5, 6.2 )
b <- c( 1, 9, 3.1 )
euclidean_distance( a, b )
```

ExtNames *ExtNames*

Description

ExtNames

Usage

```
ExtNames(L, pos = 1)
```

Arguments

L	List of objects with names which we wish to extract.
pos	Position for setting where the objects will be extracted to, in terms of environment.

Value

Objects, to the specified environment.

Examples

```
L <- list( "rabbit" = 3, "BigMatrix" = matrix( runif( 100 ), ncol = 10 ) )
L$rabbit
ExtNames( L )
rabbit
```

FullLinearModel *Full Linear Model Generation*

Description

Generate the full first, second or third order linear model for a given dataframe, X. Whilst the actual models can be generated in this way, this function is mostly a useful way for extracting model formulae.

Usage

```
FullLinearModel(X, y = rep(1, nrow(X)), order = 1)
```

Arguments

X	dataframe (not a matrix).
y	(optional) vector of responses
order	order of the polynomial fit

Value

Full linear model for y given X.

Examples

```
data( USArrests )
FullLinearModel( USArrests )
FullLinearModel( USArrests, order = 2 )
FullLinearModel( USArrests, order = 3 )
FullLinearModel( USArrests[,c("UrbanPop", "Assault")], y = USArrests[,"Murder"], order = 2 )
```

InvertMatrices *Invert Each Matrix in an Array*

Description

Invert each of the matrices in an array and return an array of inverted matrices. Note that if some of the matrices can't be inverted then a matrix of "NA"'s is put in it's place.

Usage

```
InvertMatrices(V)
```

Arguments

V	a 3-dimensional array

Value

an array, each matrix of which is the inversion of the matrices in V.

Examples

```
V <- array( stats::rnorm( 18, 0, 0.1) + rep( c( diag( 3 ) ), 2 ), dim = c(3,3,2) )
# ensure that in this example both matrices will be invertible
InvertMatrices( V )
```

---

kNNinterp *k Nearest Neighbour interpolation.*

Description

k Nearest Neighbour interpolation.

Usage

```
kNNinterp(x, y, z, k = 10, algorithm = "kd_tree", p = 2)
```

Arguments

x	a vector or matrix of values for which we wish to interpolate some output given y and z.
y	a vector or matrix of values for which we have the output value of interest (given by z).
z	a vector or matrix of output values corresponding to the points in y.
k	the number of nearest neighbours we wish to average over.
algorithm	nearest neighbour searching algorithm.
p	the power to which we wish to take the distance of each point to its nearest neighbours to in the inverse distance average weighting.

Details

If y or z are vectors, it is assumed that the corresponding input or output dimension (respectively) is 1. If x is a vector, then if the dimension of y is 1, it is assumed to be a vector of points at which to predict, otherwise it is assumed to be a single point (with dimension equal to that of y) at which to predict.

Value

a vector or matrix (depending on the whether z was a vector or matrix) of the k nearest neighbour interpolated predictions.

See Also

get.knnx

Examples

```
f <- function( y ){ c( sin( y[1] ) + cos( y[2] ), cos( y[1] ) + sin( y[2] ) ) }
y <- matrix( runif( 40 ), ncol = 2 )
z <- t( apply( y, 1, f ) )
x <- matrix( runif( 30 ), ncol = 2 )
kNNinterp( x, y, z )
kNNinterp( x, y, z[,1] )
kNNinterp( x[1,,drop=FALSE], y, z )
kNNinterp( x[1,], y, z )
```

MatrixMM *Matrix MM*

Description

Take the maximum or minimum of each column or row of a matrix.

Usage

```
MatrixMM(X, maximum = TRUE, column = TRUE)
```

Arguments

X	a matrix or a dataframe.
maximum	Take the maximum (TRUE) or minimum (FALSE)...
column	...of each column (TRUE) or row (FALSE)

Value

vector of maximum or minimum values of the columns or rows of X.

Examples

```
X <- matrix( runif( 6 ), ncol = 2 )
MatrixMM( X )
Y <- data.frame( X, row.names = c( "cat", "dog", "rabbit" ) )
MatrixMM( Y )
```

MinDist *MinDist*

Description

Computes the minimum distance between the points (rows) of a matrix.

Usage

```
MinDist(X)
```

Arguments

X	matrix or dataframe

Value

minimum distance between any two of the points in (given by the rows of) X

Examples

```
X <- matrix( runif( 12 ), ncol = 3 )
MinDist( X )
Y <- data.frame( X, row.names = c( "cat", "dog", "rabbit", "mouse" ) )
MinDist( Y )
```

MLH *Maximin Latin Hypercube*

Description

Generate a Maximin Latin hypercube design. Samples many Latin hypercubes (`niter`) and selects the one with maximum minimum distance between any two of its points.

Usage

```
MLH(
  m,
  d = 2,
  niter = 1000,
  rect = "MinusoneOne",
  plot_best_iterations = TRUE,
  print_best_iterations = TRUE
)
```

Arguments

`m`	number of points in the Latin hypercube.
`d`	number of dimensions over which the Latin hypercube is required. This can also be implicitly specified as the number of rows of `rect`.
`niter`	Number of Latin hypercubes to sample and out of which the one with maximum minimum distance between any two points will be chosen.
`rect`	a matrix with `d` rows and `2` columns indicating the ranges over which the parameters should be chosen. Can also be specified by the string `"MinusoneOne"` if they are [-1,1] (default) or `"ZeroOne"` if they are all [0,1].
`plot_best_iterations`	should the current best iterations be plotted as they are found?
`print_best_iterations`	should the iteration number of the best iterations be printed as they are found?

Value

`Dx_best`	best design according to the Maximin criterion.
`c_D_best`	iteration on which the best design was generated.

Examples

```
MLH( m = 10, d = 3 )
```

ModelMatrix *Model Matrix*

Description

Obtain the model matrix for a given data frame and model

Usage

```
ModelMatrix(X, model)
```

Arguments

X	dataframe (not a matrix)
model	a model

Value

the corresponding model matrix

Examples

```
data(USArrests)
mod <- QuickFunc::FullLinearModel( USArrests[1:40,], order = 2 )
MM <- ModelMatrix( USArrests[1:10,], mod )
MM # Can use it for part of the training set.
MN <- ModelMatrix( USArrests[41:50,], mod )
MN # Can use it for some prediction points.
```

MultNormSamples *Multiple Normal Samples*

Description

Generate multiple normal samples given a collection of means and variances.

Usage

```
MultNormSamples(Means, Variances, n = 1, uncorrelated = TRUE)
```

Arguments

Means	A k by m matrix, k-vector or m-vector of mean values. See details for information about k and m.
Variances	A k by k by m array, k by k matrix, k by m matrix, k-vector, m-vector or scalar.
n	number of samples required for each mean and variance combination.
uncorrelated	A logical indicating whether each output component should be sampled as though uncorrelated with the others. This massively reduces the computational time TRUE (as is default).

Details

This function generates m k-variate normal samples for a collection of means and variances. These samples may be univariate or multivariate, depending on whether the uncorrelated is set to be TRUE or FALSE.

Value

A matrix the drawn samples (as rows) appropriately.

Examples

```
mu <- matrix(c(2,4,5,-1,-1,2), nrow=3)
Sigma <- array( c(2, 0.6, -0.4, 0.6, 3, 0.8, -0.4, 0.8,
2.2, 3, 0.6, -0.4, 0.6, 2, 0.8, -0.4, 0.8, 2.1), dim = c(3,3,2) )
MultNormSamples( mu, Sigma, 4, uncorrelated = FALSE ) # Variances can be an 3D array...
MultNormSamples( mu, Sigma[,,1], 4, uncorrelated = FALSE ) # ...or Variances can be a matrix.
MultNormSamples( mu[,1], Variances = diag(Sigma[,,1]), 5 ) # Means can be a vector.
MultNormSamples( mu[,1], Variances = Sigma[1,1,1], 5 )
```

MultTSamples *Multiple Multivariate t-distribution Samples*

Description

Generate multiple multivariate t-distribution samples given a collection of means and variances.

Usage

```
MultTSamples(Means, Variances, n, df)
```

Arguments

Means	A vector or matrix of mean values. If a vector, the means are 1-dimensional given by each element. If a matrix, the mean vectors are given by row.
Variances	Scalar or Matrix if the variances for each sample are the same. A vector or 3D array if they are required to be different.
n	the size of each sample.
df	the number of degrees of freedom.
	#' @details This function generates multiple multivariate t-distribution (with df degrees of freedom) samples, each of size n, for each mean scalar/vector in Means and variance scalar/matrix (if variance the same for each Mean) or vector/array Variances (if the variance is different for each Mean). Note that there may be ways to improve the efficiency of this function! In addition, it should only really be used if the there is a (not uncorrelated) multivariate variance structure since the multivariate sampling function is used unless Means is a vector.

Value

A matrix containing all the drawn samples (as rows).

Examples

```
mu <- matrix(c(2,4,5,-1,-1,2), nrow=2)
Sigma <- array( c(2, 0.6, -0.4, 0.6, 3, 0.8, -0.4, 0.8,
2.2, 3, 0.6, -0.4, 0.6, 2, 0.8, -0.4, 0.8, 2.1), dim = c(3,3,2) )
MultTSamples( mu, Sigma, 4, df = 10 ) # Variances can be an 3D array...
MultTSamples( mu, Sigma[,,1], 4, df = 10) # ...or Variances can be a matrix.
MultTSamples( mu[1,], Variances = diag(Sigma[,,1]), 5, df = 10 ) # Means can be a vector.
MultTSamples( mu[1,], Variances = Sigma[1,1,1], 5, df = 10 )
```

---

periodic_dist *Periodic Difference*

Description

Calculates the difference between two values under a particular period mod value.

Usage

```
periodic_dist(X, Y = X, p = 1)
```

Arguments

X	a vector
Y	a vector
p	scalar value of period

Value

difference between the values in X and Y mod p, given as a length(X) by length(Y) matrix.

Examples

```
periodic_dist( runif( 3 ), runif( 4 ) )
```

Prediction_diagnostics *Prediction Diagnostics*

Description

Obtain Mean Absolute Standardised Prediction Error (MASPE) and Root Mean Standardised Prediction Error (RMSPE) between a set of predictions Efx, Varfx and the true model or system value fx.

Usage

```
Prediction_diagnostics(fx, Efx, Varfx, var_nug = 1e-10)
```

Arguments

fx	Model/system value.
Efx	Prediction.
Varfx	Uncertainty variance.
var_nug	optional variance nugget.

Value

MASPE and RMSPE

Examples

```
fx <- c(3,5,6)
Efx <- c(4,3,5.75)
Varfx <- rep( 0.8, 3 )
Prediction_diagnostics( fx, Efx, Varfx )
```

---

rad2deg *Convert Radians to Degrees*

Description

Convert Radians to Degrees

Usage

```
rad2deg(r)
```

Arguments

r	angle in radians

Value

angle in degrees

Examples

```
rad2deg( 1 )
```

rowSort *Sort Each Row of Matrix*

Description

Sort Each Row of Matrix

Usage

```
rowSort(X, decreasing = FALSE, na.last = TRUE)
```

Arguments

X	a matrix.
decreasing	logical (defaulting to FALSE). Should the sort order be increasing or decreasing?
na.last	for controlling the treatment of NAs. If TRUE, missing values in the data are put last; if FALSE, they are put first; if NA, they are removed (see 'Note'.)

Value

a matrix with each row containing the sorted elements of the rows in X.

See Also

order

Examples

```
X <- matrix( sample( 50, 20 ), ncol = 4 )
rowSort( X )
rowSort( X, decreasing = TRUE )
```

Scale *Scale points*

Description

Scale point from one hypercuboid domain space to another.

Usage

```
Scale(x, a = 0, b = 1, l = -1, u = 1)
```

Arguments

x	vector, matrix or dataframe of points (given by the elements or rows respectively) of points to scale.
a	vector of lower limits of the original space, one for each dimension (column of x) If all the lower limits are the same, that scalar value can be given.
b	vector of upper limits of the original space, one for each dimension (column of x) If all the upper limits are the same, that scalar value can be given.
l	vector of lower limits of the transformed space, one for each dimension (column of x) If all the lower limits are the same, that scalar value can be given.
u	vector of upper limits of the transformed space, one for each dimension (column of x) If all the upper limits are the same, that scalar value can be given.

Details

Scales the vector of matrix of points x from the hypercuboid [a,b] to [l,u].

Value

a vector or matrix of the transformed points.

Examples

```
X <- matrix( runif(15, 2, 4), ncol = 3 )
Scale( X, a = 2, b = 4 )
# Compare with:
X - 3
```

Stand_Obs *Standardised Observation*

Description

Calculate the standardised observation of a set of quantities.

Usage

```
Stand_Obs(x, EX, VarX, abs = TRUE)
```

Arguments

x	Observed value, given as a vector or array.
EX	Expectation of X, given as a vector or array which is the same dimension of x.
VarX	Variance of X, given as a vector or arraywhich is the same dimension of x.
abs	Should the absolute standardised observation be calculated?

Details

standardised observation is calculated as `( x -EX ) / sqrt( VarX )`

Value

the standardised observations (absolute if applicable) as a vector or array the same dimension as x.

Examples

```
EX = c(3, 8, 2)
VarX = rep( 0.7, 3 )
x = c(3.82, 6.34, 1.89)
Stand_Obs( x, EX, VarX )
```

---

uniform_density *Multivariate Uniform Density (over a hypercube)*

Description

Multivariate Uniform Density (over a hypercube)

Usage

```
uniform_density(x, l, u)
```

Arguments

x	vector or matrix
l	vector of lower bounds
u	vector of upper bounds

Value

density of x for the multivariate uniform distribution specified by l and u.

Examples

```
x <- c(3, 4)
l <- c(2.3, 4.5)
u <- c(6, 6)
uniform_density( x, l, u )
uniform_density( x + 1, l, u )
```

vec_to_list *Convert a vector of elements into a list of vectors of prescribed lengths.*

Description

Convert a vector of elements into a list of vectors of prescribed lengths.

Usage

```
vec_to_list(z, lop = rep(1, length(z)), names = NA)
```

Arguments

z	a vector.
lop	a vector indicating how many elements of z should go into each vector in the list.
names	optional names parameter for the elements of the constructed list. If specified, it must be the same length as lop.

Value

a list of vectors, each taking several elements of z as presribed by the elements of lop.

Examples

```
z <- 1:10
vec_to_list( z )
vec_to_list( z, lop = c(2,3,5) )
vec_to_list( z, lop = c(2,3,5), names = c("dog", "cat", "rabbit") )
```

Index