Package 'ifultools'

November 26, 2020

Title Insightful Research Tools

Version 2.0-19

Depends R (>= 3.0.2)

Imports MASS, methods, splus2R (>= 1.2-0)

Description Insightful Research Tools is a collection of signal processing, image processing, and time series modeling routines written in the C language for speed and efficency. The routines were originally developed at Insightful for use in S-PLUS.

License GPL-2

Collate it_mutils.R it_plot.R it_util.R it_calls.R package.R

Repository CRAN

NeedsCompilation yes

Author William Constantine [aut], Stephen Kaluzny [aut, cre]

Maintainer Stephen Kaluzny <spkaluzny@gmail.com>

Date/Publication 2020-11-26 11:00:02 UTC

R topics documented:

`ACVStoPACS` *Computes partial autocorrelations from autocovariances*

Description

Given an autocovariance sequence (ACVS) for a stationary process, computes the corresponding partial autocorrelation sequence (PACS).

Usage

```
ACVStoPACS(acvs)
```

Arguments

`acvs`	the ACVS for lags 0, 1, ..., p, where p must be a positive integer.

Details

The PACS (sometimes called the reflection coefficient sequence) is computed from the ACVS using the Levinson-Durbin recursions. Note that the autocorrelation sequence can be used as input rather than the ACVS since the PACS does not in fact depend on the variance of the process (i.e., the ACVS at lag 0).

Value

a vector with the PACS for lags 1, ..., p.

References

S-Plus 5 Guide to Statistics,Section 24.2.

D. Percival and A. Walden, *Spectral Analysis for Physical Applications*, Cambridge University Press, 1993, Section 9.4.

See Also

ar.yw.

Examples

```
ACVStoPACS(c(3,2,1))
ACVStoPACS(c(1,0.9,0.81,0.9^3,0.9^4))
```

---

aggregateData *Time series aggregation*

Description

Partitions data into blocks and and applies the specifed function to each of the bocks.

Usage

```
aggregateData(x, by, FUN, moving=FALSE, ...)
```

Arguments

x	a numeric vector.
by	.
FUN	a scalar function to compute the summary statistics which can be applied to all data subsets.
...	additional arguments to pass to FUN.
moving	either FALSE to do standard aggregation, or a positive integer N to perform a moving aggregation (normally used for a moving average) over N samples.

Value

the aggregated series.

See Also

aggregate.

Examples

```
## Group a simple series into blocks containing 8
## elements, and take the mean of each block.
## Each block is lagged by 3 elements
aggregateData(1:30, by=3, FUN=mean, moving=8)
```

---

autoKey *Automatic key placement*

Description

Automatically places a key in the sparsest location of the space spanned by the the x and y input coordinates.

Usage

```
autoKey(x, y=NULL, args.=NULL, nquadrant=5)
```

Arguments

x	if y is not NULL, this argument is a numeric vector containing the abscissa (x-axis) values for the current plot. If y=NULL, this argument is taken as a numeric vector of ordinate (y-axis) values.
args.	a list of arguments that define the legend ala the key function. Default: NULL (no matpoints).
nquadrant	an integer denoting the number of quadrants to partition the space spanned by x and y. For example, if nquadrants=3, the x-y space is partitioned into a 3x3 grid of equisized blocks. Default: 5.
y	a numeric vector containing the ordinate (y-axis) values in the current plot. If NULL, the x input argument is taken as the y-axis values while the x-axis values are extracted from x via the positions function. If there are no positions available, then the sequence 1:length(x) is used as the x-axis positions. Default: NULL.

Details

The argument `args.` is a list of arguments that define the key and `nquadrant` is the number of equi-sized quadrants to use in dividing the space in both the x- and y-direction. e.g., if nquadrant=3, the x-y plane is partitioned into a 3x3 uniform grid and the position of the sparsest quadrant (as defined by `sparsetQuadrant`) is used to place the key. This function does not take into consideration lines connecting the data points and only considers the spatial distribution of the points in the x-y plane. Futhermore, this function assumes that `par("usr")` is approximately `c(range(x),range(y))`, i.e., that the user will not explicitly make space outside span of the data to place the key. In this case, the user should explicitly place the key as usual.

Value

no output is returned.

Note

The key as described by `args.` is positioned in estimated sparsest region of the specified data set.

See Also

autoText, sparsestQuadrant, mergeList.

Examples

```
zkey <- list(x=-1,y=-1,
lines=list(type=c("o","l"), pch=c(18,0), lty=1, col=1:2),
border=1,
corner=c(1,1),
cex=1.5 )

old.mfrow <- par(mfrow=c(2,2))

## rectangular hole in cloud
set.seed(100)
N <- 10024
y <- rnorm(N)
x <- seq(N)
xhole <- x > .4*N & x < .9*N
yhole <- y > -2 & y < 2
hole  <- xhole & yhole
y <- y[!hole]
x <- x[!hole]
plot(x, y, type="p")
autoKey(x, y, args=zkey)

## linear chirp (more room on left)
y <- sin(.125*pi*100*((0:99)/100)^2)
plot(y, type="p")
autoKey(y, args=zkey)

## quadratic
```

```
x <- seq(-5,5)
y <- x^2
plot(x,y,type="p")
autoKey(x, y, args=zkey)

## circle
tt <- seq(0,2*pi,length=100)
x  <- sin(tt)
y  <- cos(tt)
plot(x, y, type="p")
autoKey(x, y, args=zkey)
par(old.mfrow)
```

---

autoText *Automatic text placement*

Description

Automatically places a given character string in the sparsest location of the space spanned by the the x and y input coordinates.

Usage

```
autoText(x, y=NULL, text.="", cex=1, col=1, nquadrant=5)
```

Arguments

x	if y is not NULL, this argument is a numeric vector containing the abscissa (x-axis) values for the current plot. If y=NULL, this argument is taken as a numeric vector of ordinate (y-axis) values.
cex	par function character expansion value. Default: 1.
col	par function color value for the text. Default: 1.
nquadrant	an integer denoting the number of quadrants to partition the space spanned by x and y. For example, if nquadrants=3, the x-y space is partitioned into a 3x3 grid of equisized blocks. Default: 5.
text.	a character string to place in the current plot. Default: ""
y	a numeric vector containing the ordinate (y-axis) values in the current plot. If NULL, the x input argument is taken as the y-axis values while the x-axis values are extracted from x via the positions function. If there are no positions available, then the sequence 1:length(x) is used as the x-axis positions. Default: NULL.

Value

no output is returned.

Note

The defined text is positioned in the estimated sparsest region of the specified data set.

See Also

text, autoKey, sparsestQuadrant, mergeList.

Examples

```
## quadratic
x <- seq(-5,5)
y <- x^2
plot(x,y,type="p")
autoText(x, y, text="Some text in a parabola", cex=1.5)
```

---

checkRange *Check the range of a numeric object*

Description

Tests the input range based on the specified interval.

Usage

```
checkRange(x, range.=0:1, inclusion=c(TRUE,TRUE))
```

Arguments

x	an object belonging to class numeric.
inclusion	a two-element logical vector defining the boundary tests. For example, if range.=0:1, the interval boundaries are tested as follows **inclusion=c(TRUE,TRUE)** range(x) must be on [0,1]. **inclusion=c(TRUE,FALSE)** range(x) must be on [0,1). **inclusion=c(FALSE,TRUE)** range(x) must be on (0,1]. **inclusion=c(FALSE,FALSE)** range(x) must be on (0,1). If only a single logical element is specified, it is replicated to form a two-element vector. If more than two elements are specified, only the first two are used as described above. Default: c(TRUE,TRUE).
range.	a two-element numeric vector containing the limits of the test interval. Default: 0:1.

Value

no output is returned. If these tests fail, a stop condition is executed.

See Also

isVectorAtomic, checkVectorType, checkScalarType.

Examples

```
## expect no output from the following calls
checkRange(pi,range=c(0,5))
checkRange(1:5,range=c(1,5), inclusion=c(TRUE,TRUE))
```

---

checkScalarType *Check for scalar type and class*

Description

Tests the input for being a scalar value and checks the class of the scalar. This function is meant to be used internally for function input argument verification and typically will not be used otherwise.

Usage

```
checkScalarType(x, isType="numeric")
```

Arguments

x	an S-PLUS object.
isType	a character string defining the class of the object to be checked ala `is(x,isType)`. Default: `"numeric"`.

Value

no output is returned. If these tests fail, a `stop` condition is executed.

See Also

isVectorAtomic, checkVectorType, checkRange.

Examples

```
## expect no output from the following calls
checkScalarType(pi,"numeric")
checkScalarType(100,"integer")
checkScalarType(letters[1],"character")
```

checkVectorType *Check for vector type and class*

Description

Tests the input for being a vector (as defined by the `isVectorAtomic` function and checks the class of the vector. This function is meant to be used internally for function input argument verification and typically will not be used otherwise.

Usage

```
checkVectorType(x, isType="numeric")
```

Arguments

`x`	an S-PLUS object.
`isType`	a character string defining the class of the object to be checked ala `is(x,isType)`. Default: `"numeric"`.

Value

no output is returned. If these tests fail, a `stop` condition is executed.

See Also

`isVectorAtomic`, `checkScalarType`, `checkRange`.

Examples

```
## expect no output from the following calls
checkVectorType(rnorm(1024),"numeric")
checkVectorType(1:3,"integer")
checkVectorType(letters,"character")
```

decibel *Conversion to decibels*

Description

Convert numeric input into decibels.

Usage

```
decibel(x, type=1, na.zero=TRUE)
```

Arguments

x	a real positive numeric value.
na.zero	logical value controlling the handing of zeros in the data. If TRUE, the zero values are replaced with NAs prior to converting to decibels, avoiding an error when attempting to take the logarithm of zero.
type	an integer signifying the decible form to use in the conversion. If type=1 $10 * log10(x)$ is used to convert the input and $20 * log10(x)$ isued for type=2.

Value

the decibel equivalent of the input x.

See Also

nextDyadic, isVectorAtomic.

Examples

```
decibel(c(1,10,100,1000), type=1)
decibel(c(1,10,100,1000), type=2)
```

---

em *Typesetting unit*

Description

Return the width and height of the letter 'm' in the current font.

Usage

```
em()
```

Value

a two-element vector containing the width and height, respectively, of the letter 'm' in user coordinates for the current font.

See Also

stringSize.

Examples

```
em()
```

gridOverlay *Add a grid over the existing plot*

Description

Overlays a grid on the current plot at the ordinate and abscissa tick markers.

Usage

```
gridOverlay(lty = "dotted", col = "gray", density = 3, ...)
```

Arguments

...	additional arguments passed directly to the par function.
col	line color ala par function. Default: "gray".
density	line density ala par function. Default: 3.
lty	line type ala par function. Default: "dotted".

See Also

par.

Examples

```
## plot the chaotic beam data and overlay a grid
old.plt <- splitplot(1,1,1)
plot(rnorm(100))
gridOverlay()
par(old.plt)
```

ilogb *Integer truncation of logb output*

Description

Calculates logb(x,base=base) + eps where eps is a user-supplied (small) value and converts the result ala as.integer. The purpose of this function is to avoid logb variations between platforms. As an example, it is conceivable that logb(32,base=2) on one platform could return a value of 5 while on another platform it could return a slightly smaller value, e.g., 4.999999999999999. The difference is due to different compiler and platform-specific implementations of the S-PLUS or R languages. If logb output is truncated (as is the case with as.integer(logb(x))) then the result may be unexpected, leading to intractable errors. By adding an appropriate and small value to logb(x), subsequent truncation results in consistent value(s) across platforms.

Usage

```
ilogb(x, base=2, eps=.Machine$double.eps * 1e9)
```

Arguments

x	a numeric or complex vector.
base	a positive or complex number: the base with respect to which logarithms are computed. Default: 2.
eps	a small numeric value that is added to the logb result prior to truncation via as.integer. Default: .Machine$double.eps * 1e9.

Value

a numeric scalar or vector containing the result.

See Also

logScale.

Examples

```
## should return 3
ilogb(8 - .Machine$double.eps, base=2)
```

---

isVectorAtomic *Tests whether an object is a vector or not*

Description

The is.vector function returns a FALSE value in some cases where intuitively one might expect a TRUE value to be returned. For example, is.vector(z) returns FALSE for each of the following:

i z <-1:3;names(z) <-1:3

ii z <-matrix(1:3,nrow=1)

iii z <-matrix(1:3,ncol=1)

These results are not necessarily incorrect, they are just one interpretion of the definition of a vector. Contrarily, the isVectorAtomic(z) function returns TRUE for each of the above examples. Thus, isVectorAtomic expands the basic definition of a vector to allow matrices containing a single row or column and named vectors. Also, unlike is.vector, isVectorAtomic returns FALSE for objects of class list.

Usage

```
isVectorAtomic(x)
```

Arguments

x	an object of arbitrary class.

Value

a vector of character strings containing the result. The length of this vector is equal to `length(x)`.

See Also

rotateVector, mergeList.

Examples

```
## cases where isVectorAtomic returns TRUE
z <- 1:3;names(z) <- letters[1:3]
isVectorAtomic(z)
isVectorAtomic(matrix(1:3, nrow=1))
isVectorAtomic(matrix(1:3, ncol=1))
isVectorAtomic(1:5)
isVectorAtomic(letters)

## cases where isVectorAtomic returns FALSE
isVectorAtomic(list(1:3))
isVectorAtomic(data.frame(1:3,2:4))
isVectorAtomic(matrix(1:12, nrow=4))
```

---

itCall *Thin itCall Wrapper to IFULTOOLS Symbols*

Description

Thin itCall Wrapper to IFULTOOLS Symbols

Usage

```
itCall(symbol, ...)
```

Arguments

symbol	character scalar defining symbol to call in DLL
...	arguments to underlying C code

Details

Foreign function calls are no longer allowed in CRAN. This function serves as a thin wrapper to avoid associated R CMD check issues when building packages that depend on IFULTOOLS.

Value

output of the itCall

See Also

itCall.

Examples

```
## Not run:
itCall("RS_fractal_filter_nonlinear_local_projection",
    as.matrix(x),
    as.integer(dimension),
    as.integer(tlag),
    as.integer(n.neighbor),
    max.distance,
    mutilsDistanceMetric(metric),
    as.integer(noise.dimension),
    corr.curve)

## End(Not run)
```

---

linearFit *Linear regression of a segmented time series*

Description

Segments the time series in approximately linear regions as defined by the linearSegmentation function and subsequently regressed the specified segment using a selected regression scheme.

Usage

```
linearFit(x, y, fit=lmsreg, method="widest",
    n.fit=5, angle.tolerance=5, aspect=TRUE)
```

Arguments

x	the independent variable of the time series.
y	the dependent variable of the time series.
angle.tolerance	the maximum angle in degrees that the running average of the slopes in the current set of points can change relative to the slope of the data calculated in the most current (rightmost) window before a change-point is recorded. Default: 5.
aspect	a logical value. If TRUE, the aspect ratio of the data (defined by range(y) / range(x)) is used to adjust the angle.tolerance. Specifically, the new angle tolerance becomes angle.tolerance / aspect.ratio. Using the aspect ratio to dilate angle.tolerance allows the user to specify the degree of variablity

they wish to impose on the data vis-a-vis a standard plot of the data, i.e. what you would see if you issued `plot(xdata,ydata)`. The idea is that when looking at such plots, one might decide (for example) that a 5 degree variablity on the plot would be acceptable. But if that range of `y` is vastly different from that of `x`, then the true change in angle from one section to the other will be much different than 5 degrees. Thus, `aspect` can be used to compensate for aspect ratios far away from unity. Default: `TRUE`.

`fit` a function representing the linear regression scheme to use in fitting the resulting statistics (on a log-log scale). Supported functions are: `lm`, `lmsreg`, and `ltsreg`. See the on-line help documentation for each of these for more information. Default: `lmsreg`.

`method` a character string used to define the criterion for selecting one of the segments returned by the `linearSegmentation` function. Choices are

`"first"` The first (leftmost) segment.

`"last"` The last (rightmost) segment.

`"widest"` The segment containing the most number of points.

`"all"` A union of all segments.

Default: `"widest"`.

`n.fit` an integer denoting the window size, not to exceed the number of samples in the time series. Default: 5.

Value

the regression model.

See Also

logScale.

Examples

```
## obtain some data with approximately
## piecewise-linear trends
x <- seq(0,2*pi,length=100)
y <- sin(x)

## perform linear segmentation with aspect ratio
## dilation using a 5 degree tolerance and 5
## point windows. regress the widest of these
## segments with the lm() function.
linearFit(x, y, n.fit=5, angle.tolerance=5, aspect=TRUE,
    method="widest", fit=lm)
```

`linearSegmentation` *Piecewise linear segmentation of a time series*

Description

Locates the change-points of time series based on a piecewise linear segmentation algorithm. Given a window size (`n.fit`) and an angle tolerance (`angle.tolerance`), the segmentation algorithm starts by finding the slope of the first `n.fit` points of the series via least squares regression. The window is slid over one point to the right, the points within the new window are regressed, and the new slope is compared to the old slope. If the change in slope exceeds the specified `angle.tolerance`, a change-point is recorded as the rightmost point of the previous iteration's window. The routine then picks up again starting at the point just to the right of the change-point. If the change in slope does not exceed the specified `angle.tolerance`, then the old slope is updated (in a running average sense), and the algorithm continues as usual. The window is slid along the until its rightmost point reaches the edge of the time series.

Usage

```
linearSegmentation(x, y, n.fit=5, angle.tolerance=5,
    aspect=TRUE, plot=FALSE, add=FALSE, ...)
```

Arguments

`x`	the independent variable of the time series.
`y`	the dependent variable of the time series.
`...`	additional arguments sent unaltered to the `abline` function used to draw the vertical divisions of the linear segmentation. Only used if `plot=TRUE`.
`add`	a logical value. If `TRUE`, the plot is added using the current plotting scheme. Default: `FALSE`.
`angle.tolerance`	the maximum angle in degrees that the running average of the slopes in the current set of points can change relative to the slope of the data calculated in the most current (rightmost) window before a change-point is recorded. Default: 5.
`aspect`	a logical value. If `TRUE`, the aspect ratio of the data (defined by `range(y) / range(x)`) is used to adjust the `angle.tolerance`. Specifically, the new angle tolerance becomes `angle.tolerance / aspect.ratio`. Using the aspect ratio to dilate `angle.tolerance` allows the user to specify the degree of variablity they wish to impose on the data vis-a-vis a standard plot of the data, i.e. what you would see if you issued `plot(xdata,ydata)`. The idea is that when looking at such plots, one might decide (for example) that a 5 degree variablity on the plot would be acceptable. But if that range of `y` is vastly different from that of `x`, then the true change in angle from one section to the other will be much different than 5 degrees. Thus, `aspect` can be used to compensate for aspect ratios far away from unity. Default: `TRUE`.
`n.fit`	an integer denoting the window size, not to exceed the number of samples in the time series. Default: 5.
`plot`	a logical value. If `TRUE`, a plot of the segmentation is displayed. Default: `FALSE`.

Value

a vector containing the result.

See Also

lm.

Examples

```
## obtain some data with approximately
## piecewise-linear trends
x <- seq(0,2*pi,length=100)
y <- sin(x)

## perform linear segmentation with aspect ratio
## dilation using a 5 degree tolerance and 5
## point windows
z <- linearSegmentation(x, y, n.fit=5, angle.tolerance=5, aspect=TRUE)

## plot the data and the estimated change-points
plot(x, y)
abline(v=x[z], lty=2)
```

---

logScale *Logarithmically spaced scale vector generation*

Description

Creates a vector whose values are base-2 logarithmically spaced.

Usage

```
logScale(scale.min, scale.max,
    scale.ratio=2, scale.res=NULL, coerce=NULL)
```

Arguments

scale.min	an integer denoting the minimum scale.
scale.max	an integer denoting the maximum scale.
coerce	the coercion function to use on the resulting scale vector. Default: NULL (no coercion).
scale.ratio	a numeric value representing the ratio of the next scale over the current scale, i.e., the ratio of successive scales. Default: 2.
scale.res	the numeric value denoting scale resolution. This input serves as an alternative to the scale.ratio input, and is related via scale.ratio <-1 / scale.res + 1 or conversely scale.res <-1/(scale.ratio -1). Default: NULL (scale.ratio is used instead).

Value

a numeric vector containing the scales.

See Also

scale, ilogb.

Examples

```
logScale(scale.min = 1, scale.max=34, scale.ratio=4)
```

---

mergeList *Merges the information from two lists*

Description

Given lists X and Y, this functions replaces the commonly named objects in X with those of Y and appends the uncommon Y components to X. List X is returned as the merged list.

Usage

```
mergeList(x, y)
```

Arguments

x	a list of named objects.
y	a list of named objects.

See Also

prettyPrintList.

Examples

```
## develop lists
dinner <- list(Entree="Spaghetti and Meatballs",
    Starter="Caesar Salad", Dessert="Spumoni",
    Beverage="Wine and Water")

## oops, we are all out of spumoni and we just got
## some tiramisu in from the local bakery
change <- list(Dessert="Tiramisu",Note="Please tip your waiter")

## merge the lists and prett-print
prettyPrintList(mergeList(dinner, change), header="What's for dinner?")
```

mutilsDistanceMetric *L-p metric conversion*

Description

Converts a numeric value (p) representing an L-p metric to an enumerated index appropriate for Insightful's MUTILS C library, used by many functions in the FRACTAL package involving `itCall` calls.

Usage

```
mutilsDistanceMetric(metric)
```

Arguments

`metric`	an integer (p) representing an L-p Euclidean distance metric. Supported values are 1, 2 and `Inf`.

Value

an integer representing the equivalent MUTILS enumerated type for the specifed metric.

See Also

`mutilsFilterType`, `mutilsFilterTypeContinuous`, `mutilsSDF`, `mutilsTransformPeakType`, `mutilsTransformType`, `mutilsWSShrinkageFunction`, `mutilsWSThresholdFunction`.

Examples

```
mutilsDistanceMetric(Inf)
```

mutilsFilterType *Converts wavelet filter string to MUTILS enum type*

Description

MUTILS is a C library used for certain `itCall` functions. This function maps a wavelet filter character string to the corresponding enum type in MUTILS.

Usage

```
mutilsFilterType(wavelet="s8")
```

Arguments

`wavelet`	a character string representing the wavelet filter type. Default: `"s8"`.

Value

an integer corresponding to the MUTILS enum.

See Also

mutilsDistanceMetric, mutilsFilterTypeContinuous, mutilsSDF, mutilsTransformPeakType, mutilsTransformType, mutilsWSShrinkageFunction, mutilsWSThresholdFunction.

Examples

```
## map the Haar filter
mutilsFilterType("haar")
```

---

mutilsFilterTypeContinuous

Converts a continuous wavelet filter string to MUTILS enum type

---

Description

MUTILS is a C library used for certain `itCall` functions. This function maps a conntinuous wavelet filter character string to the corresponding enum type in MUTILS.

Usage

```
mutilsFilterTypeContinuous(x)
```

Arguments

x	a character string representing the continuous wavelet filter type.

Value

an integer corresponding to the MUTILS enum.

See Also

mutilsDistanceMetric, mutilsFilterType, mutilsSDF, mutilsTransformPeakType, mutilsTransformType, mutilsWSShrinkageFunction, mutilsWSThresholdFunction.

Examples

```
## map the Morlet filter
mutilsFilterTypeContinuous("morlet")
```

mutilsSDF *SDF evaluation over a discrete uniform grid of frequencies*

Description

MUTILS is a C library used for certain itCall functions. This function evaluates an input SDF function over frequencies on the interval [0, Nyquist], i.e., normalized frequencies [0, 1/2].

Usage

```
mutilsSDF(sdf=NULL, sdfargs=NULL, n.freq=1024, sampling.interval=1)
```

Arguments

n.freq	a positive integer (greater than one) defining the number of frequencies to use in evaluating the SDF. The frequencies are uniformly distributed over the interval [0, Nyquist] ala f=[0, 1/P , 2/P, 3/P, ..., (n.freq-1)/P] where P=2*(n.freq-1)/sampling.interval. Default: 1024.
sampling.interval	a positive numeric scalar representing the sampling interval of the time series associated with the input SDF. This argument is used only develop the set of frequencies over which the input SDF is evaluated (see documentation for n.freq argument for more details). Default: 1.
sdf	the input SDF. If sdf=NULL, a negative numeric scalar is returned and serves as a flag the MUTILS C code to ignore relevant SDF calculations. Otherwise, this input must be a function whose first argument is f (representing frequency). At a minimum, the SDF must be defined over frequencies [0, Nyquist] where Nyquist=1/(2*sampling.interval). Additional arguments that are needed to calculate the SDF should be passed via the sdfargs parameter. Default: NULL.
sdfargs	a list of named additional SDF arguments. For example, if the input SDF requires additional arguments (other than f): y and z, then specify this input ala sdfargs=list(y=1,z=pi). Default: NULL (no additional inputs).

Value

a numeric vector containing the values of the input SDF evaluated over frequencies uniformly distributed on the interval [0, Nyquist]. The corresponding frequencies are assigned to the output object as the named attribute "frequency". If sdf=NULL, a negative numeric scalar is returned and serves as a flag to MUTILS C code that the SDF is missing or undefined.

See Also

mutilsDistanceMetric, mutilsFilterType, mutilsFilterTypeContinuous, mutilsTransformPeakType, mutilsTransformType, mutilsWSShrinkageFunction, mutilsWSThresholdFunction.

Examples

```
## create a faux SDF
S <- function(f, phi)  1/(1 + phi^2 - 2*phi*cos(2*pi*f))

## specify additional input arguments needed to
## evaluate the SDF
sdfargs <- list(phi=0.9)

## evaluate the SDF over frequencies on the
## interval [0, 1/2]
Sx <- mutilsSDF(sdf=S, sdfargs=sdfargs)

## plot the result
f <- attr(Sx,"frequency")
plot(f, Sx, type="l")
```

`mutilsTransformPeakType` *Converts wavelet transform local peak type string to MUTILS enum type*

Description

MUTILS is a C library used for certain `itCall` functions. This function maps a wavelet transform local peak type character string to the corresponding enum type in MUTILS.

Usage

```
mutilsTransformPeakType(x)
```

Arguments

`x`	a character string representing the wavelet transform local peak type. Choices are `"extrema"`, `"maxima"`, and `"minima"`.

Value

an integer corresponding to the MUTILS enum.

See Also

`mutilsDistanceMetric`, `mutilsFilterType`, `mutilsFilterTypeContinuous`, `mutilsSDF`, `mutilsTransformType`, `mutilsWSShrinkageFunction`, `mutilsWSThresholdFunction`.

Examples

```
## map the peak type
mutilsTransformPeakType("maxima")
```

mutilsTransformType *Converts wavelet transform string to MUTILS enum type*

Description

MUTILS is a C library used for certain `itCall` functions. This function maps a wavelet transform character string to the corresponding enum type in MUTILS.

Usage

```
mutilsTransformType(x)
```

Arguments

x	a character string representing the wavelet transform type. Choices are `"modwt"`, `"modwpt"`, `"dwt"`, and `"dwpt"`.

Value

an integer corresponding to the MUTILS enum.

See Also

`mutilsDistanceMetric`, `mutilsFilterType`, `mutilsFilterTypeContinuous`, `mutilsSDF`, `mutilsTransformPeakType`, `mutilsWSShrinkageFunction`, `mutilsWSThresholdFunction`.

Examples

```
## map the MODWPT
mutilsTransformType("modwpt")
```

mutilsWSShrinkageFunction *Converts wavelet shrinkage function to MUTILS enum type*

Description

MUTILS is a C library used for certain `itCall` functions. This function maps a wavelet transform character string to the corresponding enum type in MUTILS.

Usage

```
mutilsWSShrinkageFunction(x)
```

Arguments

x	a character string representing the waveshrink function. Choices are "hard", "soft", and "mid".

Value

a list containg objects index and shrinkfun, containing an integer corresponding to the MUTILS enum and name of the shrinkage function, respectively.

See Also

mutilsDistanceMetric, mutilsFilterType, mutilsFilterTypeContinuous, mutilsSDF, mutilsTransformPeakType, mutilsTransformType, mutilsWSThresholdFunction.

Examples

```
## obtain MUTILS enum map for soft thresholding
mutilsWSShrinkageFunction("soft")
```

---

mutilsWSThresholdFunction

Converts wavelet shrinkage theshold function to MUTILS enum type

---

Description

MUTILS is a C library used for certain itCall functions. This function maps a wavelet transform character string to the corresponding enum type in MUTILS.

Usage

```
mutilsWSThresholdFunction(x)
```

Arguments

x	a character string representing the waveshrink function. Choices are "universal", "minimax", and "adaptive".

Value

a list containg objects index and threshfun, containing an integer corresponding to the MUTILS enum and name of the waveshrink threshold function, respectively.

See Also

mutilsDistanceMetric, mutilsFilterType, mutilsFilterTypeContinuous, mutilsSDF, mutilsTransformPeakType, mutilsTransformType, mutilsWSShrinkageFunction.

Examples

```
## obtain MUTILS enum map for the adaptive
## threshold function
mutilsWSThresholdFunction("adaptive")
```

---

nextDyadic *Next integer power of 2*

Description

Find smallest power of two that is greater than or equal to an input x.

Usage

```
nextDyadic(x)
```

Arguments

x	a real number. May be positive, zero, or negative.

Value

returns the next power of two that is greater than, or equal to, the input x. If x is negative, 1 is returned.

See Also

isVectorAtomic, decibel.

Examples

```
nextDyadic(15)
```

---

ordinal *Ordinal value conversion*

Description

Converts a given number to the nearest ordinal value.

Usage

```
ordinal(x)
```

Arguments

x	a numeric scalar.

Value

a character string denoting the nearest ordinal value.

See Also

properCase.

Examples

```
ordinal(1)
ordinal(2)
ordinal(3)
ordinal(pi)
ordinal(123)
ordinal(124)
ordinal(1000)
```

---

prettyPrintList *Prints a list of named objects*

Description

Prints a list of named objects in a two column format, with the names of objects in the list in the first column and a summary of the object in the second. The user can control the justification and separation character between the columns.

Usage

```
prettyPrintList(x, header=NULL, justify="left", sep=":")
```

Arguments

x	a list of named objects.
header	a character string defining the header. Default: NULL (no header).
justify	a character string defining the aligment format for the objects in the list. Choices are "none", "left", "right" and "decimal".
sep	a siingle character used to divide the list object names from their corresponding values.

See Also

isVectorAtomic.

Examples

```
## develop menu
dinner <- list(Entree="Spaghetti and Meatballs",
    Starter="Caesar Salad", Dessert="Spumoni",
    Beverage="Wine and Water")

## pretty-print the menu
prettyPrintList(dinner, header="What's for dinner?")
```

`properCase` *Case conversion ala a proper noun*

Description

Converts a character string with arbitrary case to a character string whose first character is capitalized and remaining characters are set to lower case.

Usage

```
properCase(x, pre="", post="", sep="")
```

Arguments

`x`	a character string or vector of character strings.
`post`	a character string to append to each character string in `x` after conversion. Default: `""`.
`pre`	a character string to prepend to each character string in `x` after conversion. Default: `""`.
`sep`	a character string that is pasted between the other portions of the combined string ala pre-sep-formatted x-sep-post. Default: `""`.

Value

a vector of character strings containing the result. The length of this vector is equal to `length(x)`.

See Also

`ordinal`.

Examples

```
## produces "myBasketBall" "mySoccerBall"
## "myFootBall"
properCase(c("baSkeT","SoCcEr","foot"),post="Ball",pre="my")
```

rotateVector *Circularly vector rotation*

Description

Circularly rotates elememnts in a vector.

Usage

```
rotateVector(x, shift=1)
```

Arguments

x	a vector.
shift	an integer denoting the shift (number of elements to circularly rotate). A negative value implies a negative (leftward) shift of the data.

Value

a numeric vector containing the result.

See Also

isVectorAtomic.

Examples

```
rotateVector(1:5, 2)
rotateVector(1:5, -2)
```

scaleData *Scale numeric data*

Description

Scales the input data via a selected logarithmic function.

Usage

```
scaleData(x, scale.="linear")
```

Arguments

x	an object that inherits from the numeric class, typically a scalar, vector, or matrix.
scale.	a character string defining the type of scaling to perform. Choices are "linear","log2","log10","log" or "db". Default: "linear" (no scaling).

Value

the scaled data is returned with an attribute "scalestr" attached, which defines the scaling treatment used on the input data.

See Also

logb, decibel.

Examples

```
scaleData(c(1,10,100,1000),scale="db")
```

---

scaleZoom *Zoom in on a specified region of a data matrix*

Description

Returns the portion of a matrix based on the specified ranges of abscissa (x) and ordinate (y) values.

Usage

```
scaleZoom(x, y, z=NULL, zoom=NULL, logxy="y",
    xy.linked=FALSE, xlab="x", ylab="y", log.base=2)
```

Arguments

x	a numeric vector of abscissa values corresponding to the provided z matrix.
y	a numeric vector of ordinate values corresponding to the provided z matrix.
log.base	the base of the logarithm used to scale the output zoomed coordinates. See the help for the logxy input for more details. Default: 2.
logxy	a character string indicating containing either or both of the characters "x" and "y". If the character "x" is present in the string, the logarithm of the resulting zoomed x coordinate is returned use base log.base. A similar logic follows for the presence of character "y" in the string. For example, to take the log10 of both the zoomed x- and y-coordinates, one would set logxy="xy",log.base=10. If logarithms are used, the xlab and ylab strings are altered accordingly. Default: "y".
xlab	character string defining x-label. Default: "x".
xy.linked	a logical value. If TRUE, the algorithm used to define the output (zoomed) x- and y-coordinates as follows: **1** Let ix be the indices of x s.t. x[ix] is on the interval [zoom[1],zoom[2]]. **2** Redefine x and y as x=x[ix] and y=y[ix]. **3** Let iy be the indices of y s.t. y[iy] is on the interval [zoom[3],zoom[4]]. **4** Redefine x and y as x=x[iy] and y=y[iy].

Thus, both x- and y-zoom conditions must be met for both coordinates. Default: FALSE.

ylab — character string defining x-label. Default: "y".

z — a matrix of numeric values whose x- and y-axis values are defined by the x and y vectors, respectively. This argument may be NULL signifying that no z matrix is present and (in this case) only the zoomed x and y coordinates will be returned in the output. Default: NULL.

zoom — a four-element numeric vector containing the ranges to zoom into the data in c(xmin,xmax,ymin,ymax) format. Default: NULL (no zoom).

Value

a named list containing the following components:

x — The zoomed portion of the input x vector.

y — The zoomed portion of the input y vector.

z — The zoomed portion of the input z matrix, if z is not NULL.

ix — The indices of the original x vector which span the range specified by zoom[1:2].

iy — The indices of the original y vector which span the range specified by zoom[3:4].

xlab — A character string defining an appropriate x-label for the zoomed x-coordinate output.

ylab — A character string defining an appropriate y-label for the zoomed y-coordinate output.

See Also

image, stackPlot.

Examples

```
## create an image
x <- y <- seq(-4*pi, 4*pi, len=50)
r <- sqrt(outer(x^2, y^2, "+"))
z <- cos(r^2)*exp(-r/6)
image(x,y,z,col=gray((0:32)/32))

## zoom in on one portion of that image and
## re-display
zoom <- scaleZoom(x,y,z, zoom=c(-5,5,-1,10), logxy="")
image(zoom$x, zoom$y, zoom$z, col=gray((0:32)/32))
```

solveODE *Numerical integration of ODEs*

Description

Solve a system of ordinary differential equations via a fourth order Runge-Kutta numerical integration scheme.

Usage

```
solveODE(FUN, initial=NULL, step=0.01, n.sample=1000, n.transient=100, ...)
```

Arguments

FUN	a function defining the sytem of ODEs. This function should have as an input X, where X is a vector whose length is equal to the order of the ODEs. It should return a value for each order (state) of the system.
...	additional arguments sent directly to FUN.
initial	a vector of initial values, one element for each state of the system defined by the ODEs. By default, this value is NULL, in which case FUN is called with a X=NULL. Therefore, FUN should be able to handle a NULL value input if you do not specify an initial condition. Default: NULL.
n.sample	the number of desired samples for each state beyond that specified for the transient ala n.transient. Default: 1000.
n.transient	the numeber of transient points. These points are excluded from the output. Default: 100.
step	a numerical integration time step. Default: 0.1.

Value

a data.frame containing the estimated system response variables.

See Also

par.

Examples

```
## estimate response of the chaotic Lorenz system
"lorode" <- function(x, sigma = 10, r = 28, b = 8/3){
    c(sigma * (x[2] - x[1]), x[1] * (r - x[3]) - x[2],  - b * x[3]
       + x[1] * x[2])
}

z <- solveODE(lorode,  initial=c(0.1,0.3,1), n.transient=1500,
    n.sample=2000)
nms <- c("X","Y","Z")
```

```
## plot the results
stackPlot(x=seq(150, by=0.1, length=2000), y=z,
    ylab=nms, main="Lorenz System in Chaos", xlab="Time")
```

`sparsestQuadrant` *Estimates sparsest quadrant in data set*

Description

Finds the sparsest are in the current plot for placing an object.

Usage

```
sparsestQuadrant(x, y=NULL, nquadrant=5)
```

Arguments

`x`	if `y` is not `NULL`, this argument is a numeric vector containing the abscissa (x-axis) values for the current plot. If `y=NULL`, this argument is taken as a numeric vector of ordinate (y-axis) values.
`nquadrant`	an integer denoting the number of quadrants to partition the space spanned by x and y. For example, if `nquadrants=3`, the x-y space is partitioned into a 3x3 grid of equisized blocks. Default: 5.
`y`	a numeric vector containing the ordinate (y-axis) values in the current plot. If `NULL`, the `x` input argument is taken as the y-axis values while the x-axis values are extracted from `x` via the `positions` function. If there are no positions available, then the sequence `1:length(x)` is used as the x-axis positions. Default: `NULL`.

Details

Partitions the space spanned by the input x-y data coordinates into a uniform `nquadrant` by `nquadrant` grid and finds the "sparsest quadrant": defined as the quadrant that contains a minimum population (pmin) of x-y data points and is and farthest (in an L-inf sense) from quadrants containing a population greater than pmin.

Value

a list containing the `x` and `y`-coordinates of the sparsest data location in the current plot. In addition, a two-element `corner` vector is returned and should be used directly in placing a key at the returned coordinates via the `autoKey` function.

See Also

`autoKey`, `autoText`.

Examples

```
## find the sparset quadrant in a circle
tt <- seq(0,2*pi,length=100)
x  <- sin(tt)
y  <- cos(tt)
sparsestQuadrant(x,y)
```

`splitplot` *Partitions plot space into rectangular grid*

Description

Uses the `plt` argument of the `par` function to divide the space according to the input grid.

Usage

```
splitplot(nrows, ncols, i=1, new=as.logical(i > 1 && i <= nrows*ncols), gap=0.15)
```

Arguments

`nrows`	an integer defining the number of desired rows in the plot grid.
`ncols`	an integer defining the number of desired column in the plot grid.
`gap`	a numeric scalar used a nudge factor for the gap between plots in both the x- and y-directions. Default: `0.15`.
`i`	selects the `i`th plot of the current `nrow` by `ncol` plot grid for the next plot. The upper left plot region is denoted by `i=1` and increments moving from left to right, then top to bottom. Default: `1`.
`new`	a logical flag. If `TRUE`, a new plot grid is established and previously defined plot regions are erases. Default: `as.logical(i > 1)`.

Value

the original `plt` value of the `par` function prior to the call. The user can use this to reset `plt` to its original values.

Note

The `plt` option of the `par` function is altered.

See Also

`stackPlot`.

Examples

```
## establish a 2x2 plot grid and select the first
## for plotting
old.plt <- splitplot(2,2,1)

## plot the data
for (i in seq(4)){
    if (i > 1)
       splitplot(2,2,i)

    plot(rnorm(100)*i)
    mtext(paste("i=", i, sep=""), side=3, line=0.5)
}

## return the original plot state of plt in par
par(old.plt)
```

---

stackPlot *Stack plot*

Description

Plots input data as a stack of plots.

Usage

```
stackPlot(x, y = NULL, xlty = NULL, bty = "n", lty = 1, col = 1:8,
    lwd = 1, rescale = TRUE, add = FALSE, cex = 1, xaxs = "r", xpd = TRUE,
    yaxis = list(add = TRUE, ndigit = 3,
    col = 1:8, lty = 1, lwd = 3, side = "left", cex = 1),
    xlab = list(text = "", cex = 1, srt = 0, col = 1), ylab = list(text = NULL,
    cex = 1, srt = 0, col = 1:8, side = "right"), main = list(text = "",
    cex = 1, srt = 0, col = 1, adj = 0.5), ylim = NULL)
```

Arguments

x	a vector of numeric values corresponding to a common abscissa (x-axis) for all ordinate (y-axis) values.
y	a numeric vector, matrix, data.frame, or rectangular list containing the ordinate (y-axis) values.
add	a logical value. If TRUE, the plot is added using the current par layout. Otherwise a new plot is produced. Default: FALSE.
bty	a character string defining the box type (ala par's bty parameter) for each plot. Default: "n" (no boxes).
cex	a numeric value defining the character expansion for the plot labels (ala par's cex parameter). Default: 1.

`col` an integer or vector of integers denoting the color of each plotted series (ala `par`'s `col` parameter). This input can be a vector, one color for each series plotted. If the length of this vector is less than the number of series, then `col` is repeated as many times as necessary. Default: `1:8`.

`lty` the line type for each plot (ala `par`'s `lty` parameter). This input can be a vector, one line type for each series plotted. If the length of this vector is less than the number of series, then `lty=1` is used for those plots where lty is not directly specified. Default: 1.

`lwd` an integer or vector of integers denoting the line width of each plotted series (ala `par`'s `lwd` parameter). This input can be a vector, one line width for each series plotted. If the length of this vector is less than the number of series, then `lwd` is repeated as many times as necessary. Default: 1.

`main` the main label as defined by a list of the following named objects:

text A character string defining the label. Default: `""` (no label).

cex A character expansion value used to scale the label. Default: 1.

srt A numerical value specifying the rotation of the label in degrees. Default: 0.

col An integer defining the color of the label. Default: 1.

adj A numeric value on [0,1] defining the justification of the label relative to the entire width of the plot window. The value 0, 0.5, and 1 represent left, center, and right text alignment, respectively. Default: 0.5.

If a partial list of the above named objects is supplied, those objects are merged with the default list defined above.

`rescale` a logical value. If `TRUE`, the data in each plot is scaled so that visually the height of each subplot is approximately the same (the y-axis labels denote the unscaled/original values. Default: `TRUE`.

`xaxs` a character string defining the style of the x-axis interval calculation (ala `par`'s `xaxs` parameter). Default: `"e"` (extended axes).

`xlab` the x-axis label as defined by a list of the following named objects:

text A character string defining the label. Default: `""` (no label).

cex A character expansion value used to scale the label. Default: 1.

srt A numerical value specifying the rotation of the label in degrees. Default: 0.

col An integer defining the color of the label. Default: 1.

If a partial list of the above named objects is supplied, those objects are merged with the default list defined above.

`xlty` the line type (ala `par`'s `lty` parameter) of horizontal lines used to separate stacked plots. Default: `NULL` (no separator lines).

`xpd` a character string defining the style of the plot clipping (ala `par`'s `xpd` parameter). Default: `TRUE`.

`yaxis` the y-axis style as defined by a list of the following named objects:

add A logical value. If `TRUE`, a y-axis is drawn for each plot. Default: `TRUE`.

ndigit An integer defining the number of digits to use in labeling y-axis data ranges. Default: 3.

lwd An integer defining the line width of the y-axis. Default: 3.

side A character string (either `"left"` or `"right"` denoting the side to place the y-axis. Default: `"left"`.

cex A character expansion value used to scale the labels on the y-axis. Default: 1.

If a partial list of the above named objects is supplied, those objects are merged with the default list defined above.

`ylab` the y-axis label(s) as defined by a list of the following named objects:

text A (vector of) character string(s) defining the label(s). Default: the names of the y-axis labels as labeled in the primary `y` input.

cex A character expansion value used to scale the label(s). Default: 1.

srt A numerical value specifying the rotation of the label(s) in degrees. Default: 0.

col A (vector of) integer(s) defining the color of the label(s). Default: `1:8`.

side A character string (either `"left"` or `"right"` denoting the side to place the y-axis label(s). Default: `"right"`.

If a partial list of the above named objects is supplied, those objects are merged with the default list defined above.

`ylim` a two-element numeric vector containing the y-axis range of each series to plot. The range of the specified `ylim` values must span that of all the series, otherwise an error is returned. Default: `NULL` (y-axis limits are calculated automatically).

Value

no output.

Note

A stack plot is produced.

See Also

`splitplot`.

Examples

```
## stack-plot the sunspots series and a random
## walk series
set.seed(100)
ix <- seq(2048)
stackPlot(x=ix,
    y=data.frame(sunspots[ix], cumsum(rnorm(length(ix)))),
    xlty=2, ylab=list(text=c("sunspots","walk")))
```

`stringSize` *Size of a character string in current font*

Description

Calculates the approximate relative x and y spans of the input character string x in the current par("usr") coordinates. The `srt`, `adj`, and `cex` inputs are exactly those specified by the `par` function.

Usage

```
stringSize(x, srt=0, adj=0.5, cex=1)
```

Arguments

`x`	a character string.
`adj`	a numeric value on the interval [0,1] used to justify the text relative to its placement in a plot. Default: 0.5 (center aligned).
`cex`	the expansion value for the character string. Default: 1.
`srt`	a numeric value defining the amount of string rotation in degrees. Default: 0.

Value

a list of x- and y-vectors that can be used to estimate the width and height, respectively, of the input character string in the current par("usr") coordinates. Each vector contains a relative starting and ending value and the absolute difference between the two values repesents the span in the corresponding direction.

Note

A graphics window will open up if one is not already opened.

See Also

`par`, `autoKey`, `autoText`, `em`.

Examples

```
stringSize("What's for dinner?", adj=0, srt=45, cex=1.5)
```

variance *Variance computation, usable in both S-PLUS and R*

Description

A hybrid of S-PLUS and R definitions of the variance function.

Usage

```
variance(x, na.rm=TRUE, unbiased=FALSE)
```

Arguments

x	a numeric vector, matrix, or data.frame.
na.rm	a logical flag. If TRUE, NA values are removed prior to computation. Default: TRUE.
unbiased	a logical flag. If TRUE, the unbiased sample variance is returned. Default: FALSE.

Details

Allows the user to calculate the biased/unbiased variance of the input and provides the option to eliminate NA values as well.

Value

the sample variance of the input.

See Also

var.

Examples

```
set.seed(100)
x <- rnorm(100)
variance(x, unbiased=TRUE)
variance(x, unbiased=FALSE)
variance(c(x,rep(NA,30)), na.rm=TRUE)
```

Index