

# Package ‘splus2R’

January 2, 2012

**Type** Package

**Title** Supplemental S-PLUS functionality in R

**Version** 1.1-0

**Date** 2011-10-16

**Author** William Constantine, Tim Hesterberg, Knut Wittkowski, Tingting Song

**Maintainer** William Constantine <wlbconstan@gmail.com>

**Depends** R (>= 2.5.1), methods

**Description** Currently there are many functions in S-PLUS that are missing in R. To facilitate the conversion of S-PLUS packages to R packages, this package provides some missing S-PLUS functionality in R.

**License** GPL-2

**Collate** Swrappers.R bits.per.integer.R doTest.R muS2RC.R  
showStructure.R sigseriesS3.R sigseriesS4.R splus2R\_is.R splus2R\_pkg.R

**Repository** CRAN

**Date/Publication** 2011-10-17 18:42:54

## R topics documented:

allTrue . . . . .	2
anyMissing . . . . .	4
as.rectangular . . . . .	4
asSeriesData . . . . .	5
bits.per.integer . . . . .	5
collIds . . . . .	6
deparseText . . . . .	7
do.test . . . . .	8
expectStop . . . . .	10
ifelse1 . . . . .	12

is.inf . . . . .	13
is.missing . . . . .	14
is.number . . . . .	15
is.numeric.atomic.vector . . . . .	15
is.orderable . . . . .	16
is.rectangular . . . . .	17
lowerCase . . . . .	18
MC . . . . .	18
nDotArgs . . . . .	19
numCols . . . . .	20
numericSequence . . . . .	20
numericSequence-methods . . . . .	21
oldUnclass . . . . .	22
peaks . . . . .	23
positions . . . . .	23
rmvnorm . . . . .	24
seriesData . . . . .	25
seriesDataNew . . . . .	26
seriesDataValid . . . . .	26
showStructure . . . . .	27
signalSeries . . . . .	28
signalSeries-methods . . . . .	29
stdev . . . . .	30
subscript2d . . . . .	31
vecnorm . . . . .	31
which.na . . . . .	32

**Index** **34**

---

allTrue	<i>Test whether all expressions return TRUE</i>
---------	---

---

**Description**

This is typically used to combine multiple [all.equal](#) tests into a single test, in a test file called by [do.test](#).

**Usage**

```
allTrue(...)
```

**Arguments**

... Each argument is typically a call to `do.test` or another expression that returns a logical value.

**Details**

This is intended for use in test run by `do.test`. A typical test may contain lines that create one or more objects, followed by commands to check that those objects have the expected structure and/or that calculations were correct. By using `allTrue`, the tests can all be combined into the same expression that created the objects, so that if an error occurs it is easier to see where it occurred.

**Value**

if all inputs are TRUE the value is TRUE. Otherwise a list indicating which arguments did not return TRUE, containing the actual values.

**Author(s)**

Tim Hesterberg

**See Also**

[all.equal](#), [do.test](#), [expectStop](#), [expectWarnings](#), [identical](#).

**Examples**

```
# This is the type of expression that may be found in a test file
# to be run by do.test -- inside {} are lines that create one or
# more objects, followed by multiple tests (inside allTrue) that
# check those objects.
{
  y <- rnorm(30)
  x <- matrix(rnorm(60), ncol=2)
  fit <- lm(y~x)
  allTrue(# are important components included?
    all(is.element(c("coefficients", "residuals", "effects", "rank",
      "fitted.values", "assign", "df.residual", "call"),
      names(fit))),
    {
      # do coefficients match the algebraic form?
      # The algebraic form is inaccurate, so allow greater tolerance
      X <- cbind(1, x)
      all.equal(unname(fit$coefficients),
        drop(solve( t(X) %*% X, t(X) %*% y)),
        tol = 1e-5)
    },
    # are residuals computed correctly?
    all.equal(fit$residuals, y - X %*% fit$coefficients))
}
# The second test uses 'unname' to remove names and 'drop' to change a
# matrix to a vector, so the test should pass.
# The third test fails because fit$residuals is a vector with names
# while the %*% calculation returns a matrix.
```

---

anyMissing	<i>Returns TRUE if missing values are round, otherwise FALSE</i>
------------	--

---

**Description**

Detection of missing values.

**Usage**

```
anyMissing(x)
```

**Arguments**

x                    any object (though not all are currently supported).

**Value**

logical, TRUE is missing values are detected.

**See Also**

[is.missing](#), [which.na](#).

**Examples**

```
anyMissing(1:5)
anyMissing(c(1, NA, 2))
anyMissing(list(a=1:3, b=NA))
anyMissing(data.frame(a=1:3, b=c(NA, 5:6)))
```

---

as.rectangular	<i>Convert to rectangular object</i>
----------------	--------------------------------------

---

**Description**

Rectangular data objects include matrices, data frames, and atomic vectors.

**Arguments**

x                    object to be converted to rectangular data.  
i                    first (row) subscript  
j                    second (column) subscript.

**Value**

as.rectangular(x) returns x if x is already rectangular, or as.data.frame(x) if it is not.

**See Also**

[is.rectangular](#).

**Examples**

```
##  
as.rectangular(list(a=1:10, b=11:20))
```

---

asSeriesData	<i>Convert to a seriesData</i>
--------------	--------------------------------

---

**Description**

Essentially just converts data to rectangular form, if possible.

**Usage**

```
asSeriesData(x)
```

**Arguments**

x                   input object to coerce.

**Value**

rectangular conversion of input argument.

**See Also**

[numericSequence](#), [seriesData](#), [seriesDataNew](#), [seriesDataValid](#), [signalSeries](#), [as.data.frame.signalSeries](#), [as.matrix.signalSeries](#), [cumsum.signalSeries](#), [deltat.signalSeries](#), [diff.signalSeries](#), [plot.signalSeries](#).

---

bits.per.integer	<i>Internal Size of an integer</i>
------------------	------------------------------------

---

**Description**

Reports the size of an integer in bits.

**Usage**

```
bits.per.integer()
```

**Details**

Included for S-PLUS compatibility; R currently uses 32-bit integers, even on 64-bit machines.

**Author(s)**

Dunlap, W.M., and Taylor, C.F.

**Examples**

```
bits.per.integer()
```

---

colIds

*Column and row summary function*

---

**Description**

Summarizes the columns or rows of a rectangular object.

**Usage**

```
colIds(x, do.NULL=TRUE, prefix="col")
colMaxs(x, na.rm = FALSE, dims = 1, n = NULL)
colMedians(x, na.rm=FALSE)
colMins(x, na.rm = FALSE, dims = 1, n = NULL)
colRanges(x, na.rm = FALSE, dims = 1, n = NULL)
colStdevs(x, ...)
colVars(x, na.rm = FALSE, dims = 1, unbiased = TRUE,
        SumSquares = FALSE, weights = NULL, freq = NULL, n = NULL)
rowIds(x, do.NULL=TRUE, prefix="row")
rowMaxs(x, na.rm = FALSE, dims = 1, n = NULL)
rowMins(x, na.rm = FALSE, dims = 1, n = NULL)
rowRanges(x, na.rm = FALSE, dims = 1, n = NULL)
rowStdevs(x, ...)
rowVars(x, na.rm = FALSE, dims = 1, unbiased = TRUE,
        SumSquares = FALSE, weights = NULL, freq = NULL, n = NULL)
```

**Arguments**

x	rectangular input object such as a matrix or data.frame.
...	optional arguments for colStdevs function.
SumSquares	if TRUE, then unbiased is ignored and unnormalized sums of squares are returned.
dims	if x has dimension higher than 2, dims determines what dimensions are summarized.
do.NULL	logical for rowIds or colIds.
freq	vector of positive integers, the same number of observations as x. If present, the kth row of x is repeated k times. The effect is similar to the weights argument, except this does not cause the unbiased argument to be ignored, and division is by (sum(freq)-1) rather than (n-1).

n	number of rows; treat x as a matrix with n rows.
na.rm	logical, NA values are removed if TRUE.
prefix	character string preface for column IDs returned by rowIds or colIds.
unbiased	logical, unbiased variance is returned if TRUE.
weights	vector, with the same number of observations as x. If present, argument unbiased is ignored and the definition used is $\text{sum}(\text{weights} * (\text{x} - \text{mean}(\text{x}, \text{weights} = \text{weights}))^2)$ if <code>SumSquares=TRUE</code> and $\text{sum}(\text{weights} * (\text{x} - \text{mean}(\text{x}, \text{weights} = \text{weights}))^2) / \text{sum}(\text{weights})$ otherwise.

**Value**

corresponding summary by row or by column.

**Examples**

```
## create a matrix, add dimensions, and obtain
## various summaries
x <- matrix(sin(1:20), nrow=4)
dimnames(x) <- list(c("a","b","c","d"), paste("col", 1:5))
colIds(x)
colMaxs(x)
colMedians(x)
colMins(x)
colRanges(x)
colStdevs(x)
colVars(x)
rowIds(x)
rowMaxs(x)
rowMins(x)
rowRanges(x)
rowStdevs(x)
rowVars(x)
```

---

deparseText	<i>Deparses input argument</i>
-------------	--------------------------------

---

**Description**

Deparse the argument into a single string, with at most `maxchars` characters. New lines are turned into blanks, and truncated results end in "...".

**Usage**

```
deparseText(expr, maxchars=30)
```

**Arguments**

expr	any expression.
maxchars	maximum number of characters returned

**Value**

deparsed character string.

**Examples**

```
deparseText(args(1m), maxchars=20)
```

---

do.test

*Test Functions and Expressions - for automated testing*

---

**Description**

Expressions are parsed and evaluated from file. Each expression should evaluate to a logical TRUE. Otherwise, `do.test()` prints the expression and its value.

**Usage**

```
do.test(file, verbose=FALSE, strict=FALSE, local=FALSE, check)
```

**Arguments**

file	a file or connection containing code to test.
verbose	logical flag. If TRUE, all expressions are printed, not just those that fail. Regardless of this flag, the value is also printed for failures.
strict	logical flag. If TRUE, any validity failures cause an error; that is, you get to debug after the first failed assertion.
local	logical flag controlling where the evaluation takes place: by default ( <code>local=FALSE</code> ), in the environment that called <code>do.test</code> , typically the global environment, (objects created remain there after <code>do.test</code> is finished). <code>local=TRUE</code> , causes <code>do.test</code> to create and work in a new environment.
check	an unevaluated expression. If <code>check</code> is supplied, <code>do.test</code> evaluates this expression (it should be given via <code>Quote()</code> ) between each parse and evaluation. (This is for when you need to check some global information.)

**Details**

A test file typically contains a sequence of expressions to test different aspects of a function or set of functions, including testing that each input argument is handled appropriately, error handling, the output has the expected structure, correct output under a number of combinations of inputs, and error handling (warning and stop invoked when appropriate and with appropriate messages). Each expression may contain multiple lines grouped using `{}`, where early lines may do computations and the last line checks for expected results, usually using [all.equal](#).

Some expressions may be included that aren't intended to test anything by finishing them with TRUE, e.g. to read data: `{read.table("data.txt"); TRUE}` or to remove objects at the end of a test file: `{rm(a, b, x, y); TRUE}`.

We recommend including comments inside expressions to indicate the purpose of each test; then if errors occur the comments are printed too.

To compare just numbers, not names or matrix dimensions, functions `unname` and `drop` are useful.

To exclude certain components or attributes from the comparison the function `all.equal.excluding` is useful. This is defined in the examples below.

Each test should run silently if everything is working correctly; there should be nothing printed. `expectWarnings` can be used to intercept `warning` statements.

## Value

NULL

## See Also

`all.equal`, `allTrue`, `drop`, `expectStop`, `expectWarnings`, `identical`, `Quote`, `unname`. An extensive set of examples is available at <http://www.insightful.com/Hesterberg/bootstrap/resampleLoop.zip>

## Examples

```
## Not run:
# Create a toy test file, and run it
cat('{all.equal(24/8, 3)}',
    '{all.equal(5, 6)}',      # this one will fail
    'expectWarnings( { # Test subscript replacement ',
    ' x <- data.frame(a=1:3,b=2:4)',
    ' x[,3] <- x',
    ' all.equal(ncol(x), 3)',
    '}, expected = "provided 2 variables to replace 1 var")',
    'expectStop(lm(5), expected = "invalid formula")',
    '{ rm(x) ; TRUE }',      # cleanup at end of test
    sep="\n", file = "testfile.t")
do.test("testfile.t")
## ----- Test file: testfile.t -----
## {all.equal(5, 6)}
## [1] "Mean relative difference: 0.2"
#
# The test that fails is printed, with the results of the test.
# In R 2.6.1 the subscript replacement test above also fails
# (bug reported 14 Jan 2008), resulting in the additional printout:
## expectWarnings( {
##   x <- data.frame(a=1:3,b=2:4)
##   x[,3] <- x
##   all.equal(ncol(x), 3)
## }, expected = "provided 2 variables to replace 1 var")
## $'Test result'
## [1] "Mean relative difference: 0.25"

## End(Not run)

# This function is useful in some tests:
```

```

all.equal.excluding <- function(x, y, ..., excluding=NULL, attrs=NULL){
  # Like all.equal, but exclude components in 'excluding',
  #   and excluding attributes named in 'attrs'.
  #
  # 'excluding' and 'attrs' should be character, names of components
  #   and attributes.
  #
  # For example:
  #   all.equal.excluding(obj1, obj2, excluding = c("call", "x"))
  for(i in intersect(names(x), excluding)) x[[i]] <- NULL
  for(i in intersect(names(y), excluding)) y[[i]] <- NULL
  for(i in intersect(names(attributes(x)), attrs)) attr(x,i) <- NULL
  for(i in intersect(names(attributes(y)), attrs)) attr(y,i) <- NULL
  all.equal(x,y, ...)
}
# Test if two objects are the same except for "call" and "x":
data <- data.frame(x = 1:20, y = exp(1:20/20))
fit1 <- lm(y ~ x, data = data, x=TRUE)
fit2 <- update(fit1, x=)
all.equal.excluding(fit1, fit2, excluding = c("call", "x"))

```

---

expectStop

*Test whether expected stop() or warning() messages are produced.*

---

## Description

These functions are for use in automated testing using `do.test`, to test whether function give specified stop and warning messages.

## Usage

```

expectStop(expr, expected = NULL)
expectWarnings(expr, expected)

```

## Arguments

<code>expr</code>	An expression, that should result in a call to <code>stop()</code> or <code>warning()</code> .
<code>expected</code>	NULL, or a character string containing (part of) the message expected from <code>stop</code> . For <code>expectWarnings</code> a vector of character strings containing (parts of) all expected warnings.

## Details

`expectStop` is useful for checking error checking; that a function stops when it should, and gives the right message. For example, this may be in a file called `do.test`:

```

{
  expectStop(var(1:5, 1:4),
             if(is.R()) "incompatible"

```

```

        else "x and y must have the same number of")
    }

```

The function returns TRUE if

- a stop() occurs, and
- the error message is expected.

Otherwise it returns appropriate messages.

expectStop intercepts the error. Execution continues, and assignments made earlier are committed.

Similarly, expectWarnings is useful to check that a function gives appropriate warnings. For example, this may be in a file called by do.test:

```

expectWarnings(
  {
    object1 <- (code generating warning messages);
    object2 <- (code generating possibly other warning messages);
    all.equal(object1, object2)
  },
  c("expected warning 1",
    "expected warning 2"))

```

The function returns TRUE if

- expr evaluates to TRUE; and
- each warning message produced by evaluating expr contains as a substring an element of expected, and each element of expected is a substring of at least one of the produced warning messages.

Otherwise it returns a list with components describing the test failures. Normal printing of warning messages is suppressed.

It is possible to test for warnings and a stop in a single expression, by nesting calls to the two functions.

## Value

If all tests pass, then TRUE. Otherwise expectStop returns character strings describing the failure, while expectWarnings returns a list with one or more of the following components:

- 'Test result' the value (if not TRUE) returned by evaluating expr.
- 'Unexpected warnings' character vector of actual warning messages that were not listed in expected.
- 'Warnings expected but not found' character vector of messages in expected that were not produced.

## Author(s)

Tim Hesterberg

**See Also**[do.test](#)**Examples**

```

# Expressions like the following would typically be included in a file
# that is called by do.test

expectStop(lm(5), expected = "invalid formula")

expectStop(cov2cor( matrix(2:1) ),
            expected = "'V' is not a square numeric matrix")

expectWarnings( # Test subscript replacement; should discard extra
                # column and give a warning
                {
  x <- data.frame(a=1:3,b=2:4)
  x[,3] <- x
  all.equal(ncol(x), 3)
},
            expected = "provided 2 variables to replace 1 var")

# Test for a warning and stop together:
{
  f <- function(x){
    warning("a warning")
    stop("a stop")
  }
  expectStop( expectWarnings( f(3), expected = "a warning"),
            expected = "a stop")
}
# The definition of f and the call to expectStop are included here
# within {} because that is how they would typically be grouped in
# a file that is called by do.test. Also note that f has been saved
# (the assignment of f is committed, rather than aborted).

```

---

ifelse1*Conditional Data Selection*

---

**Description**

This is equivalent to `if(test) x else y`. The main advantage of using this function is better formatting, and a more natural syntax when the result is being assigned; see examples below.

With 5 arguments, this is equivalent to `if(test1) x else if(test2) u else v` (where arguments are given by name, not position).

In `ifelse1`, `is.numeric.atomic.vector`, `test` should be a single value, and the calculations for `y` (or `x`) are not performed if it is not selected. In contrast, for `ifelse`, `test` is normally a vector, both `x` and `y` are evaluated, even if not used, and `x` and `y` are vectors the same length as `test`.

**Usage**

```
ifelse1(test, x, y, ...)
```

**Arguments**

test	logical value; if TRUE return x.
x	any object; this is returned if test is TRUE.
y	any object; this is returned if test is FALSE.
...	there should be 3, 5, 7, etc. arguments to this function; arguments 1, 3, 5, etc. should be logical values; the other arguments (even numbered, and last) are objects that may be returned.

**Value**

with three arguments, one of x or y. With k arguments, one of arguments 2, 4, ..., k-1, k.

**See Also**

[ifelse](#), [if](#).

**Examples**

```
ifelse1(TRUE, "cat", "dog")
ifelse1(FALSE, "one", FALSE, "two", "three")
```

---

is.inf

*Infinite*

---

**Description**

is.inf returns a vector of the same length as the input object, indicating which elements are infinite (not missing).

**Usage**

```
is.inf(x)
```

**Arguments**

x	object to be tested
---	---------------------

**Details**

This calls [is.infinite](#).

This returns a vector of the same length as x; the jth element is TRUE if x[j] is infinite (i.e., equal to one of Inf or -Inf). This will be FALSE if x is not numeric or complex. Complex numbers are infinite if either the real and imaginary part is.

**See Also**[is.infinite](#)**Examples**

```
is.inf(Inf)
# [1] TRUE
is.inf(NA)
# [1] FALSE
is.inf(1)
# [1] FALSE
```

---

is.missing	<i>Check for missing values</i>
------------	---------------------------------

---

**Description**

Check to see whether the input is either NA or a vector of length 0.

**Usage**

```
is.missing(x)
```

**Arguments**

x                    object to check.

**Value**

TRUE if the input is a vector of length 0; `is.na(x)` otherwise.

**See Also**[anyMissing.](#)**Examples**

```
is.missing(numeric(0))
is.missing(NA)
is.missing(c(1,2,3,NA,5))
```

is.number

*Check Values***Description**

Returns a logical vector describing if a numeric elements is a number.

**Usage**

```
is.number(x)
```

**Arguments**

x                    numeric vector

**Details**

is.number is TRUE if the value is numeric or complex and is not missing (NA or NaN).

**Examples**

```
is.number(32)
# [1] TRUE
is.number(matrix(1:20, nrow=2))
#      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
# [1,] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
# [2,] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
is.number(list(matrix(1:20, nrow=2), 1:4))
# [1] TRUE TRUE
is.number('s')
# [1] TRUE
```

is.numeric.atomic.vector

*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 interpretation of the definition of a vector. Contrarily, the `is.numeric.atomic.vector(z)` function returns TRUE for each of the above examples. Thus, `is.numeric.atomic.vector` expands the basic definition of a vector to allow matrices containing a single row or column and named vectors. Also, unlike `is.vector`, `is.numeric.atomic.vector` returns FALSE for objects of class `list`.

### Usage

```
is.numeric.atomic.vector(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)`.

### Examples

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

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

---

<code>is.orderable</code>	<i>If a value can be ordered</i>
---------------------------	----------------------------------

---

### Description

`is.orderable(x)` returns `!is.na()`.

### Usage

```
is.orderable(x)
```

### Arguments

`x` object to be tested.

**Details**

x should not be a list; in that case the behavior currently differs between S-PLUS and R.

**Value**

is.orderable returns a logical vector of the same length as x.

**See Also**

[is.na](#)

**Examples**

```
x <- c(1, 4, NA, 0, 5)
is.orderable(x)
# [1] TRUE TRUE FALSE TRUE TRUE
```

---

<code>is.rectangular</code>	<i>Checks for object rectangularity</i>
-----------------------------	---

---

**Description**

Returns TRUE if the input object is rectangular.

**Usage**

```
is.rectangular(x)
```

**Arguments**

x                    any object.

**Value**

logical, returns TRUE if input object is rectangular.

**See Also**

[as.rectangular](#).

**Examples**

```
is.rectangular(matrix(1:12, nrow=3))
is.rectangular(list(1:3,2:4))
is.rectangular(as.rectangular(list(1:3,2:4)))
```

lowerCase

*Case conversion*

---

**Description**

Convert text to lower or upper case.

**Usage**

```
lowerCase(x)
upperCase(x)
```

**Arguments**

x                    a character string.

**Value**

a character string coerced to the specified case.

**See Also**

[casefold](#), [tolower](#), [toupper](#).

**Examples**

```
x <- "A dog and a cat"
lowerCase(x)
upperCase(x)
```

---

MC*Make Closure for functions*

---

**Description**

MC makes closures for defining functions in a function.

**Usage**

```
MC(f, env=NULL)
```

**Arguments**

f                    function  
env                  a list containing functions to be used in f

**Details**

MC declares functions to be used in f. When f is defined inside of a function, say fun, it cannot call other functions defined in fun. MC can enclose the functions needed by f and make it possible for f to call other functions defined in fun.

**Author(s)**

Knut M. Wittkowski <kmw@rockefeller.edu>

**Examples**

```
f1 <- function(x, y) x+y
f2 <- MC(function(x, y) x*y, list(f1=f1))
```

---

nDotArgs

*Determine Number of Arguments to Function*

---

**Description**

count the number of ... arguments passed.

**Usage**

```
nDotArgs(...)
```

**Arguments**

...                   ... arguments or real arguments in the call to the function which calls ttnDotArgs.

**Value**

the number of ... arguments in the call to the function which calls nDotArgs.

**See Also**

[nargs](#).

**Examples**

```
myfun <- function(..., a=4) nDotArgs(...)
myfun()                   ## returns 0
myfun(1:3, "bear")       ## returns 2
myfun(a=5, 1:3, "bear") ## returns 2 (excludes a)
```

numCols

*Object dimensions*

---

**Description**

Returns number of rows or number of columns of rectangular input object.

**Usage**

```
numCols(x)
numRows(x)
```

**Arguments**

x                    rectangular object.

**Value**

the number of rows or columns of the input object.

**See Also**

[as.data.frame](#), [matrix](#), [Subscript](#), [nrow](#), [dimnames](#).

**Examples**

```
x <- matrix(1:12, nrow=3)
numCols(x)
numRows(x)
```

---

numericSequence*Constructor for numericSequence Class*

---

**Description**

Constructor function for numericSequence objects. At least three of the four arguments must be supplied, unless the function is called with no arguments.

**Usage**

```
numericSequence(from, to, by, length.)
```

**Arguments**

from	start of the sequence.
to	end of the sequence.
by	increment for the sequence.
length.	length of the sequence, a non-negative integer.

**Value**

a numericSequence object with properties given by the arguments, or the default numericSequence if no arguments are supplied.

**S3 METHODS**

**S4** supported S4 methods include: Math, Math2, Ops, Summary, [, [[, duplicated, is.na, length, match, mean, median, quantile, show, sort, summary, unique, which.na. There are also [ and [[ S4 style replacement methods available.

**as** s4 style conversion methods ala as(x, foo) where foo is one of the following conversion classes: "character", "integer", "numeric".

**See Also**

[seriesData](#), [asSeriesData](#), [seriesDataNew](#), [seriesDataValid](#), [signalSeries](#), [as.data.frame.signalSeries](#), [as.matrix.signalSeries](#), [cumsum.signalSeries](#), [deltat.signalSeries](#), [diff.signalSeries](#), [plot.signalSeries](#).

---

numericSequence-methods

*Methods for class signalSeries in package splus2R*

---

**Description**

Methods for class signalSeries in package splus2R

**Details**

Supported (generic) methods include:

**as** Target classes: character, integer, numeric.

**Math** Math functions

**Math2** Math2 functions

**Ops** Ops functions

**Data access and replacement** Single- and double-bracket access and replacement functions

**duplicated** Find duplicated entries in sequence

**is.na** Test for NA entries in sequence

**length** Length of sequence  
**match** Matching functions for sequence  
**mean** Mean of sequence  
**median** Median of sequence  
**quantile** Quantile of sequence  
**rev** Reverse sequence  
**show** Show the sequence  
**sort** Sort the sequence  
**unique** Find unique sequence entries  
**summary** Summarize the sequence  
**unique** Find unique elements of a sequence  
**which.na** Find NA entries in the sequence

**See Also**

[signalSeries](#).

---

oldUnclass

*Class conversion*

---

**Description**

the old-style version of function unclass; it sets oldClass to NULL, rather than class.

**Usage**

```
oldUnclass(x)
```

**Arguments**

x                    any object.

**Value**

unclassed version of input object.

**See Also**

[oldClass](#), [unclass](#).

**Examples**

```
oldUnclass(matrix(1:10))
```

---

peaks	<i>Local maxima</i>
-------	---------------------

---

**Description**

Finds the local maxima in a vector, or time series, or in each column of a matrix.

**Usage**

```
peaks(x, span=3, strict=TRUE)
```

**Arguments**

x	vector or matrix.
span	a peak is defined as an element in a sequence which is greater than all other elements within a window of width span centered at that element. The default value is 3, meaning that a peak is bigger than both of its neighbors. Default: 3.
strict	logical flag: if TRUE, an element must be strictly greater than all other values in its window to be considered a peak. Default: TRUE.

**Value**

an object like x of logical values. Values that are TRUE correspond to local peaks in the data.

**See Also**

[max](#), [cummax](#), [pmax](#).

**Examples**

```
x <- as.vector(sunspots)
z <- peaks(x, span=51)
plot(x, type="l")
abline(v=which(z), col="red", lty="dashed")
```

---

positions	<i>Positions of signalSeries objects</i>
-----------	--

---

**Description**

Access the positions of series objects.

**Usage**

```
positions(object)
```

**Arguments**

object            an object of class `signalSeries`.

**Value**

the positions associated with the input time series: an object of class `numericSequence`.

**See Also**

[seriesData](#), [signalSeries](#).

**Examples**

```
x <- signalSeries(1:10, from=pi, by=0.1)
positions(x)
```

---

 rmvnorm

---

*Multivariate Normal (Gaussian) Distribution*


---

**Description**

Random generation for the multivariate normal (also called Gaussian) distribution.

**Usage**

```
rmvnorm(n, mean=rep(0,d), cov=diag(d), sd, rho, d=2)
```

**Arguments**

n	sample size – number of random vectors of length d to return (as rows in a matrix).
cov	covariance or correlation matrix with d rows and columns.
d	dimension of the multivariate normal.
mean	vector of length d, or matrix with n rows and d columns.
rho	scalar, vector, or <code>bdVector</code> of length n, containing correlations for bivariate data. This is ignored if cov is supplied.
sd	vector of length d, or matrix with n rows and d columns, containing standard deviations. If supplied, the rows and columns of cov are multiplied by sd. In particular, if cov is a correlation matrix and sd is a vector of standard deviations, the result is a covariance matrix. If sd is a matrix then one row is used for each observation.

**Value**

random sample ( `rmvnorm`) for the multivariate normal distribution.

**See Also**

[rnorm, set.seed.](#)

**Examples**

```
## 5 rows and 2 independent columns
rmvnorm(5)

## 5 rows and 3 independent columns
rmvnorm(5, mean=c(9,3,1))

## 2 columns, std. dev. 1, correlation .9
rmvnorm(5, rho=.9)

## specify variable means and covariance matrix
rmvnorm(5, mean=c(9,3), cov=matrix(c(4,1,1,2), 2))
```

---

seriesData

*Access Data Of series Objects*

---

**Description**

Access the data slot of series objects.

**Usage**

```
seriesData(object)
```

**Arguments**

object            object with which to find data.

**Value**

the data slot of object.

**S3 METHODS**

[<- single level data replacement method.  
Usage: x[1:4] <- 1:4)

**See Also**

[numericSequence](#), [asSeriesData](#), [seriesDataNew](#), [seriesDataValid](#), [signalSeries](#), [as.data.frame.signalSeries](#), [as.matrix.signalSeries](#), [cumsum.signalSeries](#), [deltat.signalSeries](#), [diff.signalSeries](#), [plot.signalSeries](#).

---

seriesDataNew	<i>Creates template for new seriesData object</i>
---------------	---

---

**Description**

Creates template for new seriesData object., basically a NULL matrix.

**Usage**

```
seriesDataNew()
```

**Value**

nULL matrix.

**See Also**

[numericSequence](#), [seriesData](#), [asSeriesData](#), [seriesDataValid](#), [signalSeries](#), [as.data.frame.signalSeries](#), [as.matrix.signalSeries](#), [cumsum.signalSeries](#), [deltat.signalSeries](#), [diff.signalSeries](#), [plot.signalSeries](#).

---

seriesDataValid	<i>Validates the structure of seriesData object</i>
-----------------	---

---

**Description**

Checks to see if input object is rectangular.

**Usage**

```
seriesDataValid(object)
```

**Arguments**

object            the object to check.

**Value**

logical value, TRUE is a valid seriesData object.

**See Also**

[numericSequence](#), [seriesData](#), [asSeriesData](#), [seriesDataNew](#), [signalSeries](#), [as.data.frame.signalSeries](#), [as.matrix.signalSeries](#), [cumsum.signalSeries](#), [deltat.signalSeries](#), [diff.signalSeries](#), [plot.signalSeries](#).

---

showStructure	<i>Describe the structure of an object</i>
---------------	--

---

**Description**

Describe the structure of an object, recursively.

**Usage**

```
showStructure(x, maxlen = 20, describeAttributes = TRUE,  
             short = NULL, prefix = "", attri = FALSE, ...)
```

**Arguments**

x	any object
maxlen	integer; if x is a list with more than maxlen components, only the names are printed. This may be a vector, in which case the Kth element is used at the Kth level of recursion.
describeAttributes	logical; if FALSE then only the names of attributes are printed; the structure of the attributes is not shown.
short	NULL or logical; this may be used by methods, to indicate whether to print a shorter description. It is currently used by bdFrame and bdVector methods
prefix	for internal use in recursive calls. This is used for indenting in recursive calls.
attri	for internal use in recursive calls. This is TRUE if the curent object being described is a list of attributes.
...	Additional argument that may be passed to methods; not currently used.

**Details**

This supports recursive objects, using recursive calls. Each level of recursion is indented two additional spaces. List components are shown with \$, slots with @, and attributes with &.

**Value**

This prints a description; it doesn't return anything useful.

**Author(s)**

Tim Hesterberg

**See Also**

[names](#), [str](#).

**Examples**

```

a <- c(m=1, n=2)
b <- diag(1:3)
cc <- cbind(a=1:5, b=2:6, c=letters[1:5])
d <- data.frame(cc)
attr(d, "dup.row.names") <- TRUE
e <- ts(1:10, frequency = 4, start = c(1959, 2))
f <- list(a,b=b)
setClass("track", representation(x="numeric", y="numeric"))
g <- new("track", x=1:5, y=1:5)

showStructure(a)
showStructure(b)
showStructure(cc)
showStructure(d)
showStructure(e)
showStructure(f)
showStructure(g) # prints with @ rather than $
showStructure(list(a=a, b=b))
showStructure(list(cc=cc, d, list(a,e)))

```

---

signalSeries

*Constructor function for the signalSeries class*


---

**Description**

Construct a signalSeries object from positions and data, or return an empty signalSeries object.

**Usage**

```
signalSeries(data, positions., units, units.position, from=1, by=1)
```

**Arguments**

by	amount to skip for positions.
data	variable data, which will be converted to a rectangular object with the as.rectangular function.
from	starting value of positions.
positions.	numeric or numeric sequence object to use as the time/position values.
units	units for variable data.
units.position	units for positions.

**Details**

If no arguments are supplied, the default (empty) signalSeries object is returned. Otherwise, a signalSeries object is created with the given positions and data, and units if they are supplied. As an alternative to supplying the positions directly, they can be supplied by giving from and by, in which case the positions are generated as a numeric sequence with the right length to match the data.

**Value**

a signalSeries object with the given data and positions.

**S3 METHODS**

**as** s4 style conversion to another class ala `as(x, foo)` where `foo` is any of the following: "character", "complex", "data.frame", "integer", "logical", "matrix", "numeric", "vector".

**as.data.frame** convert to a data.frame.

**as.matrix** convert to a matrix.

**cumsum** cumulative summation over series.

**deltat** samlig intervals of series.

**diff** differencing operation applied to the series. Usage: `diff(x, ...)` where the ... are additional arguments sent directly to the `diff` function.

**plot** plots the series.

**See Also**

[numericSequence](#).

---

signalSeries-methods *Methods for class signalSeries in package splus2R*

---

**Description**

Methods for class signalSeries in package splus2R

**Details**

Supported (generic) methods include:

**as** Target classes: complex, character, matrix, numeric, logical, integer, vector, and data.frame. S3 style methods include `as.numeric` and `as.vector`.

**Arith** Arithmetic functions

**Compare** Comparison functions

**Logic** Logical functions

**Math** Math functions

**Math2** Math2 functions

**Ops** Ops functions

**dim** Dimension of series (NULL is returned)

**length** Length of series

**mean** Mean of series

**min** Minimum of series

**ncol** Number of columns of series (1 is returned)  
**plot** Plot the series  
**show** Display the series  
**sum** Sum the series.  
**summary** Sumamrize the series object

### See Also

[numericSequence.](#)

---

stdev	<i>Standard deviation</i>
-------	---------------------------

---

### Description

Calculates the standard deviation of a series.

### Usage

```
stdev(x, ...)
```

### Arguments

**x** input series.  
**...** optional arguments sent directly to the `colVars` function. You can control for example the removal of NA values prior to analysis via the `na.rm` argument, and whether or not an unbiased estimate is returned ala the `unbiased` argument.

### Value

the standard deviation of the input series.

### See Also

[var,colVars,colStdevs.](#)

### Examples

```
stdev(c(pi, 1, 3))
```

---

`subscript2d`*Uniform Rectangular Data Subscripting Function*

---

**Description**

`subscript2d` is for subscripting matrices and data frames.

**Usage**

```
subscript2d(x, i, j)
subscript2d.matrix(x, i, j)
subscript2d.data.frame(x, i, j)
```

**Arguments**

<code>x</code>	a matrix or data frame
<code>i</code>	first (row) subscript.
<code>j</code>	second (column) subscript.

**Value**

`subscript2d(x, i, j)` is like `x[i, j, drop=F]`, except that it allows `x[, 1]` (for example) for atomic vectors as well, and it always returns an object of the same class as `x` (that is, it does not support a `drop` argument).

**See Also**

[as.data.frame](#), [matrix](#).

**Examples**

```
x <- 1:10
subscript2d(x, 3, 1)
subscript2d(data.frame(x), 3, 1)
subscript2d(matrix(x), 3, 1)
```

---

`vecnorm`*p-norm of a vector*

---

**Description**

Computes the p-norm of a vector

**Usage**

```
vecnorm(x, p=2)
```

**Arguments**

`x` the vector whose norm is sought (either numeric or complex).  
`p` a number or character string indicating the type of norm desired. Possible values include real number greater or equal to 1, Inf, or character strings "euclidean" or "maximum". Default: 2.

**Value**

requested p-norm of input vector.

**See Also**

[rnorm](#).

**Examples**

```
## compare 2-norm calculations
x <- rnorm(100)
sqrt(sum(x*x))
vecnorm(x)

## compare 2-norm of series which sums to Inf. The
## vecnorm returns a finite value in this case.
x <- rep(sqrt(.Machine$double.xmax), 4)
sqrt(sum(x*x))
vecnorm(x)

## 1-norm comparison
sum(abs(x))
vecnorm(x, p=1)

## L-infinity norm comparison
max(abs(x))
vecnorm(x, p=Inf)
```

---

which.na

*Determine Which Values are Missing Values*

---

**Description**

Returns an integer vector describing which values in the input vector, if any, are missing.

**Usage**

```
which.na(x)
```

**Arguments**

`x` an R object, which should be of mode "logical", "numeric", or "complex".

**Value**

an integer vector describing which values in the input vector, if any, are missing.

**See Also**

[is.na.](#)

**Examples**

```
## A non-zero number divided by zero creates
## infinity, zero over zero creates a NaN
weird.values <- c(1/0, -20.9/0, 0/0, NA)

## Produces: 3 4. In this example, the which.na
## expression and the subscript expression
## involving is.na should return the same value
which.na(weird.values)
seq(along=weird.values)[is.na(weird.values)]
```

# Index

- \*Topic **classes**
  - showStructure, 27
  - signalSeries, 28
- \*Topic **error**
  - expectStop, 10
- \*Topic **manip**
  - is.orderable, 16
  - showStructure, 27
- \*Topic **methods**
  - numericSequence-methods, 21
  - signalSeries-methods, 29
- \*Topic **programming**
  - showStructure, 27
- \*Topic **ts**
  - asSeriesData, 5
  - numericSequence, 20
  - seriesData, 25
  - seriesDataNew, 26
  - seriesDataValid, 26
- \*Topic **utilities**
  - allTrue, 2
  - anyMissing, 4
  - as.rectangular, 4
  - bits.per.integer, 5
  - colIds, 6
  - deparseText, 7
  - do.test, 8
  - expectStop, 10
  - ifelse1, 12
  - is.inf, 13
  - is.missing, 14
  - is.number, 15
  - is.numeric.atomic.vector, 15
  - is.rectangular, 17
  - lowerCase, 18
  - MC, 18
  - nDotArgs, 19
  - numCols, 20
  - oldUnclass, 22
  - peaks, 23
  - positions, 23
  - rmvnorm, 24
  - stdev, 30
  - subscript2d, 31
  - vecnorm, 31
  - which.na, 32
- [, numericSequence-method (numericSequence-methods), 21
- [, signalSeries-method (signalSeries-methods), 29
- [<-, numericSequence-method (numericSequence-methods), 21
- [<- .seriesData (seriesData), 25
- [[, numericSequence-method (numericSequence-methods), 21
- [[<-, numericSequence-method (numericSequence-methods), 21
- all.equal, 2, 3, 8, 9
- allTrue, 2, 9
- anyMissing, 4, 14
- Arith, signalSeries, ANY-method (signalSeries-methods), 29
- as.data.frame, 20, 31
- as.data.frame.signalSeries, 5, 21, 25, 26
- as.data.frame.signalSeries (signalSeries), 28
- as.matrix.signalSeries, 5, 21, 25, 26
- as.matrix.signalSeries (signalSeries), 28
- as.numeric, signalSeries-method (signalSeries-methods), 29
- as.rectangular, 4, 17
- as.vector, signalSeries-method (signalSeries-methods), 29
- asSeriesData, 5, 21, 25, 26
- bits.per.integer, 5
- casefold, 18

- coerce, list, signalSeries-method  
(signalSeries-methods), 29
- coerce, numeric, numericSequence-method  
(numericSequence-methods), 21
- coerce, numericSequence, character-method  
(numericSequence-methods), 21
- coerce, numericSequence, integer-method  
(numericSequence-methods), 21
- coerce, numericSequence, numeric-method  
(numericSequence-methods), 21
- coerce, signalSeries, character-method  
(signalSeries-methods), 29
- coerce, signalSeries, complex-method  
(signalSeries-methods), 29
- coerce, signalSeries, data.frame-method  
(signalSeries-methods), 29
- coerce, signalSeries, integer-method  
(signalSeries-methods), 29
- coerce, signalSeries, logical-method  
(signalSeries-methods), 29
- coerce, signalSeries, matrix-method  
(signalSeries-methods), 29
- coerce, signalSeries, numeric-method  
(signalSeries-methods), 29
- coerce, signalSeries, vector-method  
(signalSeries-methods), 29
- colIds, 6
- colMaxs (colIds), 6
- colMedians (colIds), 6
- colMins (colIds), 6
- colRanges (colIds), 6
- colStdevs, 30
- colStdevs (colIds), 6
- colVars, 30
- colVars (colIds), 6
- Compare, signalSeries, ANY-method  
(signalSeries-methods), 29
- cummax, 23
- cumsum, signalSeries, 5, 21, 25, 26
- cumsum, signalSeries (signalSeries), 28
- deltat, signalSeries, 5, 21, 25, 26
- deltat, signalSeries (signalSeries), 28
- deparseText, 7
- diff, signalSeries, 5, 21, 25, 26
- diff, signalSeries (signalSeries), 28
- dim, signalSeries-method  
(signalSeries-methods), 29
- dimnames, 20
- do.test, 2, 3, 8, 12
- drop, 9
- duplicated, numericSequence-method  
(numericSequence-methods), 21
- expectStop, 3, 9, 10
- expectWarnings, 3, 9
- expectWarnings (expectStop), 10
- identical, 3, 9
- if, 13
- ifelse, 13
- ifelse1, 12
- is.inf, 13
- is.infinite, 13, 14
- is.missing, 4, 14
- is.na, 17, 33
- is.na, numericSequence-method  
(numericSequence-methods), 21
- is.number, 15
- is.numeric.atomic.vector, 15
- is.orderable, 16
- is.rectangular, 5, 17
- length, numericSequence-method  
(numericSequence-methods), 21
- length, signalSeries-method  
(signalSeries-methods), 29
- Logic, signalSeries, ANY-method  
(signalSeries-methods), 29
- lowerCase, 18
- match, ANY, numericSequence-method  
(numericSequence-methods), 21
- match, numericSequence, ANY-method  
(numericSequence-methods), 21
- Math, numericSequence-method  
(numericSequence-methods), 21
- Math, signalSeries-method  
(signalSeries-methods), 29
- Math2, numericSequence-method  
(numericSequence-methods), 21
- Math2, signalSeries-method  
(signalSeries-methods), 29
- matrix, 20, 31
- max, 23
- MC, 18
- mean, numericSequence-method  
(numericSequence-methods), 21

- mean, signalSeries-method  
(signalSeries-methods), 29
- median, numericSequence-method  
(numericSequence-methods), 21
- min, signalSeries-method  
(signalSeries-methods), 29
  
- names, 27
- nargs, 19
- ncol, signalSeries-method  
(signalSeries-methods), 29
- nDotArgs, 19
- nrow, 20
- numCols, 20
- numericSequence, 5, 20, 25, 26, 29, 30
- numericSequence-class  
(numericSequence), 20
- numericSequence-methods, 21
- numRows (numCols), 20
  
- oldClass, 22
- oldUnclass, 22
- Ops, ANY, numericSequence-method  
(numericSequence-methods), 21
- Ops, numericSequence, ANY-method  
(numericSequence-methods), 21
- Ops, signalSeries, ANY-method  
(signalSeries-methods), 29
  
- peaks, 23
- plot, signalSeries-method  
(signalSeries-methods), 29
- plot.signalSeries, 5, 21, 25, 26
- plot.signalSeries (signalSeries), 28
- pmax, 23
- positions, 23
  
- quantile, numericSequence-method  
(numericSequence-methods), 21
- Quote, 9
  
- rev, numericSequence-method  
(numericSequence-methods), 21
- rmvnorm, 24
- rnorm, 25, 32
- rowIds (colIds), 6
- rowMaxs (colIds), 6
- rowMins (colIds), 6
- rowRanges (colIds), 6
  
- rowStdevs (colIds), 6
- rowVars (colIds), 6
  
- seriesData, 5, 21, 24, 25, 26
- seriesData<- (seriesData), 25
- seriesDataNew, 5, 21, 25, 26, 26
- seriesDataValid, 5, 21, 25, 26, 26
- set.seed, 25
- show, numericSequence-method  
(numericSequence-methods), 21
- show, signalSeries-method  
(signalSeries-methods), 29
- showStructure, 27
- signalSeries, 5, 21, 22, 24–26, 28
- signalSeries-class (signalSeries), 28
- signalSeries-methods, 29
- sort, numericSequence, missing-method  
(numericSequence-methods), 21
- sort, numericSequence-method  
(numericSequence-methods), 21
- stdev, 30
- str, 27
- Subscript, 20
- subscript2d, 31
- sum, signalSeries-method  
(signalSeries-methods), 29
- Summary, numericSequence-method  
(numericSequence-methods), 21
- summary, numericSequence-method  
(numericSequence-methods), 21
- Summary, signalSeries-method  
(signalSeries-methods), 29
- summary, signalSeries-method  
(signalSeries-methods), 29
  
- tolower, 18
- toupper, 18
  
- unclass, 22
- unique, numericSequence, missing-method  
(numericSequence-methods), 21
- unique, numericSequence-method  
(numericSequence-methods), 21
- unname, 9
- upperCase (lowerCase), 18
  
- var, 30
- vecnorm, 31
  
- warning, 9

`which.na`, 4, [32](#)

`which.na`, `numericSequence-method`  
(`numericSequence-methods`), [21](#)