

# R documentation

of all in ‘.’

December 24, 2025

## Contents

all.blocks . . . . .	2
armafit . . . . .	3
armaxfit . . . . .	4
dates . . . . .	5
delta.sum . . . . .	5
drop_first . . . . .	7
drop_last . . . . .	7
dummy.after . . . . .	8
extract.month . . . . .	9
extract.quarter . . . . .	9
first . . . . .	10
head.mts . . . . .	11
head.ts . . . . .	12
hstep . . . . .	12
import.tsformat . . . . .	13
lags . . . . .	15
last . . . . .	15
make.dates . . . . .	16
make.trend . . . . .	17
meanreg . . . . .	17
month . . . . .	18
month.dummy . . . . .	19
predict.tsreg . . . . .	20
quarter . . . . .	21
quarter.dummy . . . . .	21
recursive.forecast . . . . .	22
residuals.tsreg . . . . .	23
rest . . . . .	24
seasonal.dummy . . . . .	24
tail.mts . . . . .	25
tail.ts . . . . .	26

trend.after . . . . .	26
tsobs . . . . .	27
tsreg . . . . .	28
tsToList . . . . .	29
vec . . . . .	29
vec.list . . . . .	30
with.ts . . . . .	31
year . . . . .	32
yq . . . . .	32

**Index****33****all.blocks***Get All Blocks of a Time Series***Description**

Returns a list of all contiguous blocks of a specified length from a time series.

**Usage**

```
all.blocks(x, k)
```

**Arguments**

- x            A `ts` or `mts` object.
- k            The length of the blocks to extract.

**Value**

A list of `ts` or `mts` objects, where each element is a block of length `k` from the original series `x`.

**Examples**

```
y <- ts(1:10, start=2001)
all.blocks(y, 3)
```

---

**armafit***Estimate an ARMA model*

---

## Description

Estimates an ARMA model. Optionally selects the lag length automatically using auto.arima in the forecast package.

## Usage

```
armafit(x, ar=0, ma=0, auto=FALSE)
```

## Arguments

x	ts object
ar	Number of AR lags if auto=FALSE, otherwise the maximum number of AR lags to try
ma	Number of MA lags if auto=FALSE, otherwise the maximum number of MA lags to try
auto	If TRUE, uses auto.arima to select the lag length

## Value

Returns the same output as a call to arima, but with an additional argument, par, that has an intercept with the conventional interpretation. The intercept in a call to arima is actually the mean, which is quite confusing for someone learning time series, and is honestly just a terrible label for that statistic.

## Author(s)

Lance Bachmeier

## Examples

```
set.seed(400)
x <- arima.sim(n=250, list(ar=c(0.5, 0.2, 0.1), ma=0.4))
fit <- armafita(x, ar=3, ma=3, auto=TRUE)
fit$par
```

**armaxfit***Estimate an ARMAX model***Description**

Estimates an ARMAX model

**Usage**

```
armaxfit(y, xreg=NULL, ar=0, ma=0, trend=0, seasonal=FALSE,
auto=FALSE)
```

**Arguments**

y	ts object
xreg	A vector or matrix of exogenous regressors. Note that xreg should not include polynomial time trend terms or seasonal dummies. Those are handled by options trend and seasonal, which are more convenient for prediction.
ar	Number of AR lags
ma	Number of MA lags
trend	Degree of polynomial trend to include. Defaults to 0 (no trend).
seasonal	Set to TRUE to add a seasonal dummy variable if you are using monthly or quarterly data. If you are using a different frequency of data such as weekly, you should use a procedure such as STL to seasonally adjust the data before estimating the model.
auto	Not yet implemented

**Value**

Returns an S3 class object of type *armaxfit*. Is generally only of interest for prediction purposes using *predict* or *predictions*. Note that this function calls into base R's *arima*. It estimates a linear regression model with ARMA forecast errors. If you want to work with the coefficients of an ARX model, estimate the model manually using *tsreg*, as that will give you that coefficients you're after.

**Author(s)**

Lance Bachmeier

---

dates	<i>Generate Numeric Time Series Dates</i>
-------	---

---

## Description

Generates a sequence of numeric dates for a time series given start, end, and frequency.

## Usage

```
dates(start, end, freq)
```

## Arguments

- |       |   |
|-------|---|
| start | The start date of the time series (e.g., c(1990, 1) or 1990).                           |
| end   | The end date of the time series (e.g., c(2000, 12) or 2000).                            |
| freq  | The frequency of the time series (e.g., 4 for quarterly, 12 for monthly, 1 for annual). |

## Value

A numeric vector representing the time points of the generated time series.

## Examples

```
# Monthly dates from Jan 2000 to Dec 2000  
dates(c(2000, 1), c(2000, 12), 12)  
  
# Annual dates from 1990 to 2000  
dates(1990, 2000, 1)
```

---

delta.sum	<i>Standard error of a sum of coefficients</i>
-----------	--

---

## Description

Calculates the standard error of a sum of coefficients using the delta method.

## Usage

```
delta.sum(theta, v)  
delta.cumsum(theta, v)  
se.sum(fit, m=NULL)  
se.cumsum(fit, m=NULL)
```

## Arguments

<code>theta</code>	Vector of coefficients
<code>v</code>	Matrix holding covariance terms corresponding to theta
<code>fit</code>	An estimated lm object
<code>m</code>	A vector holding the index values of the coefficients of fit that are of interest. If not specified, it will select all coefficients.

## Details

`delta.sum` and `delta.cumsum` take a vector of coefficients and a matrix of covariance terms as arguments and return the standard error of the sum and cumulative sum, respectively, of those coefficients.

`se.sum` and `se.cumsum` do the same thing, but the arguments are a fitted lm object and an index indicating the location of the coefficients used to calculate the sum.

`se.sum` and `se.cumsum` are more convenient when they work, but `delta.sum` and `delta.cumsum` will work in any case that you have a coefficient vector and a coefficient covariance matrix.

## Value

Returns a standard error for the sum of coefficients when taking the sum, and a vector of standard errors when taking the cumulative sum. The standard error is calculated using the `deltamethod` function from the `msm` package.

## Author(s)

Lance Bachmeier

## Examples

```
set.seed(200)
y <- rnorm(400)
x <- matrix(rnorm(1200), ncol=3)
fit <- lm(y ~ x)
# Calculate the standard error of the sum of all coefficients
# including the intercept
delta.sum(coef(fit), vcov(fit))
# Same thing, but se.sum will extract the coefficients and by default
# takes the sum of all coefficients
se.sum(fit)
# Standard error of the sum of only the slope coefficients
delta.sum(coef(fit)[2:4], vcov(fit)[2:4, 2:4])
se.sum(fit, 2:4)
# Vector of standard errors of the cumulative sums
delta.cumsum(coef(fit)[2:4], vcov(fit)[2:4, 2:4])
se.cumsum(fit, 2:4)
```

---

drop_first	<i>Drop the First N Elements</i>
------------	----------------------------------

---

## Description

Removes the first n elements from a vector or time series object.

## Usage

```
drop_first(x, n = 1L)
```

## Arguments

- |   |   |
|---|---|
| x | A vector or a time series object.   |
| n | A single integer greater than zero. The number of initial elements to drop. Defaults to 1L. |

## Value

An object of the same class as x with the first n elements removed.

## See Also

[drop\\_last](#), [first](#), [tail](#)

## Examples

```
vec <- 1:5
drop_first(vec)
drop_first(vec, 3)

ts_object <- ts(1:10, start = c(2000, 1), frequency = 4)
drop_first(ts_object, 2)
```

---

drop_last	<i>Drop the Last N Elements</i>
-----------	---------------------------------

---

## Description

Removes the last n elements from a vector or time series object.

## Usage

```
drop_last(x, n = 1L)
```

**Arguments**

- x A vector or a time series object.
- n A single integer greater than zero. The number of trailing elements to drop. Defaults to 1L.

**Value**

An object of the same class as x with the last n elements removed.

**See Also**

[drop\\_first](#), [last](#), [head](#)

**Examples**

```
vec <- 1:5
drop_last(vec)
drop_last(vec, 2)

ts_object <- ts(1:10, start = c(2000, 1), frequency = 4)
drop_last(ts_object, 3)
```

**dummy.after**

*Generate Seasonal Dummies for Forecast Horizon*

**Description**

Generates seasonal dummy variables (monthly or quarterly) for a forecast horizon starting after a given time series.

**Usage**

```
dummy.after(x, n, omit=1)
```

**Arguments**

- x A time series object (monthly or quarterly).
- n The number of periods for the forecast horizon.
- omit Optional: For monthly, a month name or number to omit. For quarterly, an integer (1-4) to omit. Defaults to 1.

**Value**

An mts object of seasonal dummy variables for the forecast horizon.

**Examples**

```
x <- ts(rnorm(24), start=c(2000, 1), frequency=12)
dummy.after(x, 3)
```

---

**extract.month**      *Extract Data for a Specific Month*

---

**Description**

Extracts all observations corresponding to a specific month from a monthly time series.

**Usage**

```
extract.month(x, k)
```

**Arguments**

- |   |   |
|---|---|
| x | A monthly time series object.                     |
| k | An integer (1-12) representing the desired month. |

**Value**

A time series object containing only the observations for the specified month, with an annual frequency.

**Examples**

```
x <- ts(rnorm(24), start=c(2000, 1), frequency=12)
extract.month(x, 1) # January data
```

---

**extract.quarter**      *Extract Data for a Specific Quarter*

---

**Description**

Extracts all observations corresponding to a specific quarter from a quarterly time series.

**Usage**

```
extract.quarter(x, k)
```

**Arguments**

- |   |  |
|---|--|
| x | A quarterly time series object.                    |
| k | An integer (1-4) representing the desired quarter. |

**Value**

A time series object containing only the observations for the specified quarter, with an annual frequency.

## Examples

```
x <- ts(rnorm(12), start=c(2000, 1), frequency=4)
extract.quarter(x, 1) # Q1 data
```

**first**

*Get the First N Elements*

## Description

Returns the first n elements of a vector or time series. It is a wrapper for head for time series and a convenience function for vectors.

## Usage

```
first(x, n = 1)
```

## Arguments

- x                    A vector or a time series object.
- n                    A single integer. The number of elements to return. Defaults to 1.

## Value

An object of the same class as x containing the first n elements.

## See Also

[last](#), [head](#), [tail](#)

## Examples

```
vec <- c(1, 2, 3, 4, 5)
first(vec)
first(vec, 3)

ts_object <- ts(1:10, start = c(2000, 1), frequency = 4)
first(ts_object)
first(ts_object, 4)
```

---

**head.mts***Extract the First Part of a Multiple Time Series Object*

---

## Description

An S3 method for the generic [head](#) function, specifically for objects of class "mts" (multiple time series). It returns the first n observations of a multiple time series, preserving its time series attributes. This function is an alias for [head.ts](#).

## Usage

```
## S3 method for class 'mts'  
head(x, n = 6L)
```

## Arguments

- x A multiple time series object (i.e., an object of class "mts").  
n A single integer. The number of observations to return from the beginning of the time series. Defaults to 6L.

## Value

A multiple time series object ("mts") containing the first n observations of x.

## See Also

[head](#), [head.ts](#), [tail.mts](#)

## Examples

```
# Create a sample mts object  
data(presidents)  
pres_mts <- ts(cbind(presidents, diff(presidents)), start=start(presidents), frequency=frequency(presidents))  
colnames(pres_mts) <- c("Approval", "Change")  
  
head(pres_mts)  
head(pres_mts, n = 10)
```

head.ts

*Extract the First Part of a Time Series Object***Description**

An S3 method for the generic [head](#) function, tailored for objects of class "ts". It returns the first n observations of a time series, preserving its time series attributes.

**Usage**

```
## S3 method for class 'ts'
head(x, n = 6L)
```

**Arguments**

- x A time series object (i.e., an object of class "ts").
- n A single integer. The number of observations to return from the beginning of the time series. Defaults to 6L.

**Value**

A time series object ("ts") containing the first n observations of x.

**See Also**

[head](#), [tail.ts](#), [first](#)

**Examples**

```
data(AirPassengers)
head(AirPassengers)
head(AirPassengers, n = 12)
```

hstep

*Estimate a forecasting model using an h-step ahead projection***Description**

Estimate an h-step ahead forecasting model using direct estimation.

**Usage**

```
hstep(lhs, rhs, k=1, h=1)
```

**Arguments**

<code>lhs</code>	A ts object that is the dependent variable in the regression
<code>rhs</code>	A ts object or mts object holding the regressors.
<code>k</code>	The number of lags of each element of rhs that will be used in the regression.
<code>h</code>	The horizon of the regression. If <code>h=3</code> , then the time t values of lhs will be regressed on the time t-3 and later values of rhs.
<code>ect</code>	If there is an error correction term in the model, or multiple error correction terms, they should be added here. Note that this term should not be lagged - the function handles all lagging.

**Details**

There is a ‘predict’ method for the hstep object: ‘`predict(fit)`’.

**Value**

Returns a list of class hstep holding:

<code>fit</code>	The output returned by <code>tsreg</code> after estimating the model.
<code>rhs</code>	
<code>k</code>	
<code>h</code>	

**Author(s)**

Lance Bachmeier

`import.tsformat`      *Import time series data*

**Description**

Import time series data stored in a .csv file, with the metadata (frequency and start date) stored in the first two rows. Currently works with monthly, quarterly, and annual data. Optionally, you can include a line of documentation (see details).

**Usage**

```
import.tsformat(fn, skip=0, header=FALSE, ...)
```

**Arguments**

<code>fn</code>	The name of a .csv file.
<code>skip</code>	Number of lines to skip in the file after removing the metadata in the first two lines.
<code>...</code>	Additional arguments passed on to <code>read.csv</code> and applied to the data after removing the metadata in the first two lines.

## Details

The first two lines hold the frequency and start date. If the series is monthly starting in January 1979, the file will look like this:

```
# frequency: 12\cr # start: 1979 1\cr 1.5
2.6
.
.
.
```

If the series is annual starting in 1934, the file will look like this:

```
# frequency: 1\cr # start: 1934\cr 1.5
2.6
.
.
.
```

There can optionally be a line of documentation below the frequency and start date. It's one line that starts with `# doc:`. The documentation, if present, goes to the next line feed, and is added as an attribute to the returned time series. For the previous example, it would look like this:

```
# frequency: 1\cr # start: 1934\cr # doc: This is a line of documentation. I can write as
much as I want, as long as there is no line feed. If you need a line feed, you're doing something
wrong. This should include information like the source of the data. It's not a novel. 1.5
2.6
.
.
.
```

**WARNING:** `header=FALSE` by default, which is the opposite of `read.csv`. That's because it's much harder to identify errors if `header=TRUE` is incorrect (you lose the first observation) than if `header=FALSE` is incorrect (you're trying to read the variable name as an observation). This is a source of bugs when reading in data, because the default value of the argument is used any time the user forgets to specify it, and even a careful data analyst might not do a thorough check of the data every time it's read.

## Value

Returns a `ts` object holding the output of a call to `read.csv`. Then frequency and start date specified in the csv file are added to the return value. If a documentation string is included in line 3, it's added as an attribute, and can be read using `attr(v, "doc")`, where `v` is the name of the variable holding the data.

## Author(s)

Lance Bachmeier

---

lags	<i>Create multiple lags of a ts object</i>
------	--

---

## Description

Creates an mts object from a ts object and a vector of lags

## Usage

```
lags(x, k)
```

## Arguments

x	ts object
k	Vector of lags

## Value

Returns an mts object with lags of x as given by k. Follows the usual convention that a positive value in k means to take the lag rather than the lead, unlike R's builtin `lag` function. Observations with missing values at the beginning or end are dropped.

## Author(s)

Lance Bachmeier

## See Also

[lag](#), [ts.intersect](#)

## Examples

```
y <- ts(rnorm(100))
lags(y, 1:12)
```

---

last	<i>Get the Last N Elements</i>
------	--------------------------------

---

## Description

Returns the last n non-missing elements of a vector or time series object. For time series objects, it acts as a wrapper for `tail`. For vectors, it first removes NA values before extracting the last n elements.

## Usage

```
last(x, n = 1L)
```

**Arguments**

- x A vector or a time series object.
- n A single integer. The number of elements to return. Defaults to 1L.

**Value**

An object of the same class as x containing the last n non-missing elements.

**See Also**

[first](#), [head](#), [tail](#)

**Examples**

```
vec <- c(1, 2, NA, 3, 4, 5)
last(vec)
last(vec, 3)

ts_object <- ts(c(1:9, NA), start = c(2000, 1), frequency = 4)
last(ts_object)
last(ts_object, 4)
```

**make.dates**

*Generate Numeric Dates for Time Series*

**Description**

Generates a vector of numeric dates corresponding to a specified time series range and frequency, with an optional lag.

**Usage**

```
make.dates(start, end, frequency, lagged=0)
```

**Arguments**

- start The start date of the time series (e.g., c(2000, 1)).
- end The end date of the time series (e.g., c(2002, 12)).
- frequency The frequency of the time series (e.g., 4 for quarterly, 12 for monthly).
- lagged Optional: An integer representing the lag to apply to the dates. Defaults to 0.

**Value**

A numeric vector of dates.

**Examples**

```
make.dates(start=c(2000, 1), end=c(2000, 3), frequency=12)
```

---

**make.trend***Create a Time Trend Variable*

---

**Description**

Creates a time trend variable or a matrix of polynomial time trends.

**Usage**

```
make.trend(x, k = 1)
```

**Arguments**

- |                |   |
|----------------|---|
| <code>x</code> | A time series object. The data in the object is not used, but its time series properties (start, end, frequency) are.   |
| <code>k</code> | An integer specifying the highest power of the trend to create. If <code>k=1</code> (the default), a simple linear trend is created. If <code>k &gt; 1</code> , a matrix with columns for trend, <code>trend^2</code> , ..., <code>trend^k</code> is created. |

**Value**

A time series object containing the trend(s). If `k=1`, it's a univariate time series. If `k>1`, it's a multivariate time series (`mts`) object.

**Examples**

```
y <- ts(rnorm(100), start=c(1990,1), frequency=12)
trend <- make.trend(y)
plot(trend)

trends <- make.trend(y, k=3)
head(trends)
```

---

**meanreg***Mean Regression*

---

**Description**

Computes the mean of a time series over a specified sample period by running a regression of the series on an intercept.

**Usage**

```
meanreg(y, start = NULL, end = NULL)
```

## Arguments

<code>y</code>	A univariate time series object.
<code>start</code>	The start date for the computation. If <code>NULL</code> , the start date of <code>y</code> is used.
<code>end</code>	The end date for the computation. If <code>NULL</code> , the end date of <code>y</code> is used.

## Details

This function is a simple wrapper around [tsreg](#). It regresses the time series `y` on a constant. The result is an object of class `tsreg`, which inherits from `lm`. The main purpose is to provide a consistent interface for mean estimation that aligns with other regression functions in the `tstools` package.

## Value

An object of class "tsreg", which inherits from "lm". The object contains the results of regressing `y` on a constant. The estimated coefficient for the intercept is the sample mean of `y` over the specified period.

## See Also

[tsreg](#)

## Examples

```
y <- ts(rnorm(50, mean=5), start=c(2000, 1), frequency=4)
# Calculate the mean of the entire series
fit <- meanreg(y)
print(fit)
# The coefficient is the mean
coef(fit)

# Calculate the mean for a subset of the data
fit_sub <- meanreg(y, start=c(2005,1), end=c(2010,4))
coef(fit_sub)
mean(window(y, start=c(2005,1), end=c(2010,4)))
```

## Description

Extracts the month number (1-12) from a time series object.

## Usage

`month(x)`

**Arguments**

- x A time series object.

**Value**

An integer representing the month (1-12).

**Examples**

```
x <- ts(rnorm(24), start=c(2000, 1), frequency=12)
month(x)
```

---

month.dummy

*Generate Monthly Dummy Variables*

---

**Description**

Generates a time series of monthly dummy variables (0 or 1) for a given monthly time series. Allows one month to be omitted (e.g., for use as a reference category in regression).

**Usage**

```
month.dummy(x, omit=NULL)
```

**Arguments**

- x A monthly time series object.  
omit Optional: A character string (e.g., "Jan") or an integer (1-12) representing the month to omit. If NULL, all 12 dummies are returned.

**Value**

An mts object containing the monthly dummy variables.

**Examples**

```
x <- ts(rnorm(24), start=c(2000, 1), frequency=12)
month.dummy(x, omit="Jan")
```

**predict.tsreg***Predict Method for tsreg Objects***Description**

Obtains predictions from a fitted `tsreg` object.

**Usage**

```
## S3 method for class 'tsreg'
predict(obj, newdata = NULL)
```

**Arguments**

<code>obj</code>	An object of class <code>tsreg</code> , typically the result of a call to <a href="#">tsreg</a> .
<code>newdata</code>	An optional data structure containing the values of the predictor variables for which to predict. This can be a vector, matrix, <code>mts</code> , data frame, or list. If omitted, the prediction is for the period immediately following the estimation sample, using the last available observation of the predictors.

**Details**

This function is the predict method for objects returned by `tsreg`. If `newdata` is not provided, it produces a one-step-ahead forecast using the last observation in the model matrix of the fitted object. This is useful for h-step ahead forecasting schemes.

If `newdata` is provided, the function computes the predictions based on the new data. The function handles the inclusion of an intercept automatically, based on the original model fit.

**Value**

A numeric value or vector of predicted values.

**See Also**

[tsreg](#)

**Examples**

```
# Example usage:
y <- ts(rnorm(100), start=c(1990,1), frequency=4)
x <- lag(y, -1) # x is a predictor for y
fit <- tsreg(y, x, start=c(1991,1), end=c(2000,1))

# One-step-ahead forecast
predict(fit)

# Predict with new data
newdata <- ts(c(0.5, 0.8), start=c(2000,2), frequency=4)
predict(fit, newdata=newdata)
```

---

**quarter***Extract Quarter from Time Series*

---

**Description**

Extracts the quarter number (1-4) from a time series object.

**Usage**

```
quarter(x)
```

**Arguments**

x                   A time series object.

**Value**

An integer representing the quarter (1-4).

**Examples**

```
x <- ts(rnorm(12), start=c(2000, 1), frequency=4)
quarter(x)
```

---

**quarter.dummy***Generate Quarterly Dummy Variables*

---

**Description**

Generates a time series of quarterly dummy variables (0 or 1) for a given quarterly time series.  
Allows one quarter to be omitted (e.g., for use as a reference category in regression).

**Usage**

```
quarter.dummy(x, omit=1)
```

**Arguments**

x                   A quarterly time series object.

omit               Optional: An integer (1-4) representing the quarter to omit. Defaults to 1.

**Value**

An mts object containing the quarterly dummy variables.

## Examples

```
x <- ts(rnorm(12), start=c(2000, 1), frequency=4)
quarter.dummy(x, omit=4)
```

`recursive.forecast`      *Make pseudo-OOS forecasts*

## Description

Calculate pseudo-OOS forecasts to be used in an out-of-sample forecast evaluation or comparison. Forecasts are calculated recursively using the data that would have been available at the time the forecast would have been made. Supports increasing or fixed window estimation.

## Usage

```
recursive.forecast(data, f, pct, first.date, P, h, window)
```

## Arguments

<code>data</code>	[ts] The variable being forecast
<code>f</code>	[function] The function that takes the data and creates a forecast
<code>pct</code>	[double] If provided, the percentage of the sample to use for out-of-sample evaluation
<code>first.date</code>	[date] If provided and pct is not, the first date to make a forecast
<code>P</code>	[int] If provided and pct and first.date are not, the number of observations to use for out-of-sample evaluation
<code>h</code>	[int=1] Forecast horizon. If set, it affects the data that's passed to <code>f</code> .
<code>window</code>	[int] If provided, use a rolling window with this many observations. Otherwise use an increasing window with all observations available at the time the forecast is made.

## Value

[ts] A time series that holds all the pseudo-OOS forecasts made

---

**residuals.tsreg**      *Extract Residuals from a tsreg Object*

---

## Description

Extracts the residuals from a time series regression model fitted by **tsreg**.

## Usage

```
## S3 method for class 'tsreg'  
residuals(x)
```

## Arguments

**x**      An object of class **tsreg**, the result of a call to **tsreg**.

## Details

This is the residuals method for **tsreg** objects. It returns the residuals as a time series (**ts**) object, preserving the time series properties of the original data.

## Value

A time series (**ts**) object containing the model residuals.

## See Also

[tsreg](#)

## Examples

```
y <- ts(rnorm(100), start=c(1990,1), frequency=12)  
x <- lag(y, -1)  
fit <- tsreg(y, x)  
res <- residuals(fit)  
class(res)  
tsinfo(res)
```

<code>rest</code>	<i>Get All Elements Except the First</i>
-------------------	--

### Description

Returns all elements of a vector or time series object except for the first one. This function is a convenience wrapper around `tail(x, -1)`.

### Usage

```
rest(x)
```

### Arguments

<code>x</code>	A vector or a time series object.
----------------	-----------------------------------

### Value

An object of the same class as `x` with the first element removed.

### See Also

[first](#), [drop\\_first](#)

### Examples

```
vec <- 1:5
rest(vec)

ts_object <- ts(1:10, start = c(2000, 1), frequency = 4)
rest(ts_object)
```

<code>seasonal.dummy</code>	<i>Generate Seasonal Dummy Variables</i>
-----------------------------	--

### Description

Generates seasonal dummy variables (monthly or quarterly) for a given time series.

### Usage

```
seasonal.dummy(x, omit=1)
```

### Arguments

<code>x</code>	A time series object (monthly or quarterly).
<code>omit</code>	Optional: For monthly, a month name or number to omit. For quarterly, an integer (1-4) to omit. Defaults to 1.

**Value**

An `mts` object of seasonal dummy variables.

**Examples**

```
x <- ts(rnorm(24), start=c(2000, 1), frequency=12)
seasonal.dummy(x, omit="Dec")
```

---

**tail.mts***Extract the Last Part of a Multiple Time Series Object*

---

**Description**

An S3 method for the generic `tail` function, specifically for objects of class "mts" (multiple time series). It returns the last n observations of a multiple time series, preserving its time series attributes. This function is an alias for `tail.ts`.

**Usage**

```
## S3 method for class 'mts'
tail(x, n = 6L)
```

**Arguments**

- x A multiple time series object (i.e., an object of class "mts").
- n A single integer. The number of observations to return from the end of the time series. Defaults to 6L.

**Value**

A multiple time series object ("mts") containing the last n observations of x.

**See Also**

`tail`, `tail.ts`, `head.mts`

**Examples**

```
# Create a sample mts object
data(presidents)
pres_mts <- ts(cbind(presidents, diff(presidents)), start=start(presidents), frequency=frequency(presidents))
colnames(pres_mts) <- c("Approval", "Change")

tail(pres_mts)
tail(pres_mts, n = 10)
```

**tail.ts***Extract the Last Part of a Time Series Object***Description**

An S3 method for the generic [tail](#) function, tailored for objects of class "ts". It returns the last n observations of a time series, preserving its time series attributes.

**Usage**

```
## S3 method for class 'ts'
tail(x, n = 6L)
```

**Arguments**

- x A time series object (i.e., an object of class "ts").
- n A single integer. The number of observations to return from the end of the time series. Defaults to 6L.

**Value**

A time series object ("ts") containing the last n observations of x.

**See Also**

[tail](#), [head.ts](#), [last](#)

**Examples**

```
data(AirPassengers)
tail(AirPassengers)
tail(AirPassengers, n = 12)
```

**trend.after***Generate Trend Variables for Forecast Horizon***Description**

Generates trend variables for a specified forecast horizon, extending from an existing time series.

**Usage**

```
trend.after(x, h)
```

**Arguments**

- x A time series object.
- h The number of periods for the forecast horizon.

**Value**

A time series object containing the trend variables for the forecast horizon.

**Examples**

```
x <- ts(rnorm(24), start=c(2000, 1), frequency=12)
trend.after(x, 3)
```

tsobs

*Extract a Single Observation from a Time Series***Description**

Extracts a single observation from a time series at a specified date.

**Usage**

```
tsobs(x, d, type = "ts")
```

**Arguments**

- x A `ts` or `mts` object.
- d The date at which to extract the observation (e.g., `c(1990, 1)` or `1990.0`).
- type The desired return type: "`ts`" for a time series object (default) or "`numeric`" for a numeric vector.

**Value**

Either a `ts` object (univariate or multivariate) or a numeric vector, containing the observation(s) at date d.

**Examples**

```
y <- ts(rnorm(100), start=c(1990,1), frequency=12)

# Extract as a time series object
tsobs(y, c(1995, 6), type="ts")

# Extract as a numeric value
tsobs(y, c(1995, 6), type="numeric")

mts_data <- ts.intersect(y, x=lag(y, -1))
# Extract a row from an mts object as a ts object
```

```
tsobs(mts_data, c(1995, 6), type="ts")

# Extract a row from an mts object as numeric values
tsobs(mts_data, c(1995, 6), type="numeric")
```

---

**tsreg***Regression using time series data***Description**

Estimate a linear regression using time series data. Properly handles time series regressors and preserves the time series properties of the output.

**Usage**

```
tsreg(y, x, start=NULL, end=NULL, intercept=TRUE)
```

**Arguments**

y	A ts object that is the dependent variable in the regression
x	A ts object that holds the regressors. It can be either a single ts vector or a matrix of ts objects.
start	The start date for the regression sample. Matches the time index for y. If the first observation of the regression is later than start, this argument has no effect.
end	The end date for the regression sample. Matches the time index for y. If the last observation of the regression is earlier than start, this argument has no effect.
intercept	If FALSE, the intercept is equal to zero.

**Value**

Returns a list holding everything returned by a call to lm, plus the following additional elements:

fit	lm output
resids	Vector of residuals, with time series properties
fitted	Vector of fitted values, with time series properties
start	Start date for the estimation sample used in the regression, after eliminating rows with missing observations
end	End date for the estimation sample used in the regression, after eliminating rows with missing observations
int	TRUE if there is an intercept in the estimated model
nw	The coefficients with Newey-West corrected standard errors and t-statistics

**Author(s)**

Lance Bachmeier

---

**tsToList***Convert a Time Series Object to a List*

---

**Description**

Converts a univariate or multivariate time series object (`ts` or `mts`) into a list.

**Usage**

```
tsToList(x)
```

**Arguments**

`x` A `ts` or `mts` object.

**Value**

A list. If `x` is a univariate `ts` object, the list will have one element containing the series, and the list element will be named after the variable `x`. If `x` is an `mts` object, the list will have as many elements as `x` has columns, with each element being one of the columns and named accordingly.

**See Also**

[with.ts](#)

**Examples**

```
y <- ts(rnorm(10), start=2000)
tsToList(y)

data <- ts.intersect(gdp=ts(rnorm(10), start=2000),
                     inf=ts(rnorm(10), start=2000))
tsToList(data)
```

---

**vec***Coerce an Object to a Vector*

---

**Description**

A generic function to coerce an object to a vector.

**Usage**

```
vec(x)
```

**Arguments**

- x An R object.

**Value**

A vector representation of x.

**See Also**

[vec.list](#)

---

`vec.list`

*Coerce a List to a Numeric Vector*

---

**Description**

Coerces a list of numeric vectors into a single numeric vector.

**Usage**

```
## S3 method for class 'list'  
vec(x)
```

**Arguments**

- x A list of numeric vectors.

**Value**

A numeric vector formed by unlisting x.

**See Also**

[vec](#)

**Examples**

```
lst <- list(1:3, 4:6)  
vec(lst)
```

---

**with.ts***Evaluate an Expression in a Time Series Environment*

---

## Description

This is a method for the generic function `with` for time series objects. It evaluates an R expression in an environment constructed from a time series, potentially using column names as variables.

## Usage

```
## S3 method for class 'ts'  
with(x, expr)
```

## Arguments

- |                   |   |
|-------------------|---|
| <code>x</code>    | A <code>ts</code> or <code>mts</code> object. |
| <code>expr</code> | The expression to be evaluated.               |

## Details

`with.ts` temporarily converts the time series object into a list using `tsToList` and then calls the standard `with` function. This allows you to refer to the columns of a multivariate time series (`mts`) by name within the expression. For a univariate time series, you can refer to it by the name of the object itself.

## Value

The result of evaluating `expr`.

## See Also

`with`, `tsToList`

## Examples

```
data <- ts.intersect(gdp=ts(1:10, start=2000),  
                     consumption=ts(log(1:10), start=2000))  
  
with.ts(data, gdp / consumption)  
  
y <- ts(rnorm(10, 5), start=c(1990,1))  
with.ts(y, mean(y))
```

<code>year</code>	<i>Extract Year from Time Series</i>
-------------------	--------------------------------------

### Description

Extracts the year from a time series object.

### Usage

```
year(x)
```

### Arguments

<code>x</code>	A time series object.
----------------	-----------------------

### Value

An integer representing the year.

### Examples

```
x <- ts(rnorm(12), start=c(2000, 1), frequency=4)
year(x)
```

<code>yq</code>	<i>Combine Year and Quarter</i>
-----------------	---------------------------------

### Description

Combines the year and quarter of a time series object into a single integer (e.g., 2000Q1 becomes 200001).

### Usage

```
yq(x)
```

### Arguments

<code>x</code>	A time series object (typically quarterly).
----------------	---

### Value

An integer representing the combined year and quarter.

### Examples

```
x <- ts(rnorm(12), start=c(2000, 1), frequency=4)
yq(x)
```

# Index

- \* **forecast, evaluation, recursive, out-of-sample**
  - recursive.forecast, 22
- \* **methods**
  - vec, 29
  - vec.list, 30
- \* **predict**
  - predict.tsreg, 20
- \* **residuals**
  - residuals.tsreg, 23
- \* **time series**
  - dummy.after, 8
  - extract.month, 9
  - extract.quarter, 9
  - make.dates, 16
  - month, 18
  - month.dummy, 19
  - quarter, 21
  - quarter.dummy, 21
  - seasonal.dummy, 24
  - trend.after, 26
  - year, 32
  - yq, 32
- \* **ts**
  - all.blocks, 2
  - dates, 5
  - head.mts, 11
  - head.ts, 12
  - make.trend, 17
  - meanreg, 17
  - predict.tsreg, 20
  - residuals.tsreg, 23
  - tail.mts, 25
  - tail.ts, 26
  - tsobs, 27
  - tsToList, 29
  - with.ts, 31
- \* **utilities**
  - all.blocks, 2
- dates, 5
- dummy.after, 8
- extract.month, 9
- extract.quarter, 9
- head.mts, 11
- head.ts, 12
- make.dates, 16
- month, 18
- month.dummy, 19
- quarter, 21
- quarter.dummy, 21
- seasonal.dummy, 24
- tail.mts, 25
- tail.ts, 26
- trend.after, 26
- tsobs, 27
- tsToList, 29
- vec, 29
- vec.list, 30
- with.ts, 31
- year, 32
- yq, 32

- \* **utils**
- drop\_first, 7
- drop\_last, 7
- first, 10
- last, 15
- rest, 24
- all.blocks, 2
- armafit, 3
- armaxfit, 4
- dates, 5
- delta.cumsum(delta.sum), 5
- delta.sum, 5
- drop\_first, 7, 8, 24
- drop\_last, 7, 7
- dummy.after, 8

extract.month, 9  
extract.quarter, 9  
  
first, 7, 10, 12, 16, 24  
  
head, 8, 10–12, 16  
head.mts, 11, 25  
head.ts, 11, 12, 26  
hstep, 12  
  
import.tsformat, 13  
  
lag, 15  
lags, 15  
last, 8, 10, 15, 26  
  
make.dates, 16  
make.trend, 17  
meanreg, 17  
month, 18  
month.dummy, 19  
  
predict.tsreg, 20  
  
quarter, 21  
quarter.dummy, 21  
  
recursive.forecast, 22  
residuals.tsreg, 23  
rest, 24  
  
se.cumsum(delta.sum), 5  
se.sum(delta.sum), 5  
seasonal.dummy, 24  
  
tail, 7, 10, 15, 16, 25, 26  
tail.mts, 11, 25  
tail.ts, 12, 25, 26  
trend.after, 26  
ts.intersect, 15  
tsobs, 27  
tsreg, 18, 20, 23, 28  
tsToList, 29, 31  
  
vec, 29, 30  
vec.list, 30, 30  
  
with, 31  
with.ts, 29, 31  
  
year, 32  
yq, 32