# Package 'bsts'

July 2, 2014

**Date** 2011-01-06

**Title** Bayesian structural time series

**Author** Steven L. Scott <stevescott@google.com>

**Maintainer** Steven L. Scott <stevescott@google.com>

**Description** Time series regression using dynamic linear models fit using MCMC.

**Depends** BoomSpikeSlab, zoo, xts, Boom, R(>= 3.0.2)

**LinkingTo** Boom, BH (>= 1.15.0-2)

**Version** 0.5.1

**License** LGPL-2.1 | file LICENSE

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2014-06-27 13:56:24

## R topics documented:

---

`add.ar` *AR(p) state component*

---

## Description

Add an AR(p) state component to the state specification.

## Usage

```
AddAr(state.specification,
      y,
      lags = 1,
      sigma.prior,
      initial.state.prior = NULL,
      sdy)
```

## Arguments

| | |
|---|---|
| `state.specification` | A list of state components. If omitted, an empty list is assumed. |
| `y` | A numeric vector. The time series to be modeled. |
| `lags` | The number of lags ("p") in the AR(p) process. |
| `sigma.prior` | An object created by SdPrior. The prior for the standard deviation of the process increments. |
| `initial.state.prior` | An object of class MvnPrior describing the values of the state at time 0. This argument can be `NULL`, in which case the stationary distribution of the AR(p) process will be used as the initial state distribution. |
| `sdy` | The sample standard deviation of the time series to be modeled. Used to scale the prior distribution. This can be omitted if `y` is supplied. |

## Details

The model is

$$\alpha_t = \phi_1 \alpha_{i,t-1} + \cdots + \phi_p \alpha_{t-p} + \epsilon_{t-1} \qquad \epsilon_t \sim \mathcal{N}(0, \sigma^2)$$

The state consists of the last `p` lags of `alpha`. The state transition matrix has `phi` in its first row, ones along its first subdiagonal, and zeros elsewhere. The state variance matrix has `sigma^2` in its upper left corner and is zero elsewhere. The observation matrix has 1 in its first element and is zero otherwise.

## Value

Returns `state.specification` with an AR(p) state component added to the end.

## Author(s)

Steven L. Scott <stevescott@google.com>

## References

Harvey (1990), "Forecasting, structural time series, and the Kalman filter", Cambridge University Press.

Durbin and Koopman (2001), "Time series analysis by state space methods", Oxford University Press.

## See Also

bsts. SdPrior

## Examples

```
n <- 100
residual.sd <- .001

# Actual values of the AR coefficients
true.phi <- c(-.7, .3, .15)
ar <- arima.sim(model = list(ar = true.phi),
                n = n,
                sd = 3)

## Layer some noise on top of the AR process.
y <- ar + rnorm(n, 0, residual.sd)
ss <- AddAr(list(), lags = 3, sigma.prior = SdPrior(3.0, 1.0))

# Fit the model with knowledge with residual.sd essentially fixed at the
# true value.
model <- bsts(y, state.specification=ss, niter = 500, prior = SdPrior(residual.sd, 100000))

# Now compare the empirical ACF to the true ACF.
acf(y, lag.max = 30)
points(0:30, ARMAacf(ar = true.phi, lag.max = 30), pch = "+")
points(0:30, ARMAacf(ar = colMeans(model$AR3.phi), lag.max = 30))
legend("topright", leg = c("empirical", "truth", "MCMC"), pch = c(NA, "+", "o"))
```

---

`add.dynamic.regression`

*Dynamic regression state component*

---

## Description

Add a dynamic regression model to a state specification.

## Usage

```
AddDynamicRegression(
   state.specification,
   formula,
   data,
   sigma.mean.prior = NULL,
   shrinkage.parameter.prior = NULL,
   contrasts = NULL,
   na.action = na.pass)
```

## Arguments

`state.specification`
: A list of state components that you wish to add to. If omitted, an empty list will be assumed.

`formula` A formula describing the regression portion of the relationship between y and X. If no regressors are desired then the formula can be replaced by a numeric vector giving the time series to be modeled.

`data` An optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from 'environment(formula)', typically the environment from which `AddDynamicRegression` is called.

`sigma.mean.prior` An object created by `GammaPrior` describing the prior distribution of b/a (see details below).

`shrinkage.parameter.prior` An object of class `GammaPrior` describing the shrinkage parameter, a (see details below).

`contrasts` An optional list. See the `contrasts.arg` of `model.matrix.default`. This argument is only used if a model formula is specified. It can usually be ignored even then.

`na.action` What to do about missing values. The default is to allow missing responses, but no missing predictors. Set this to na.omit or na.exclude if you want to omit missing responses altogether.

## Details

The model is

$$\beta_{i,t+1} = beta_{i,t} + \epsilon_t \qquad \epsilon_t \sim \mathcal{N}(0, \sigma_i^2/variance_{xi})$$

$$\frac{1}{\sigma_i^2} \sim Ga(a, b)$$

$$\sqrt{b/a} \sim sigma.mean.prior$$

$$a \sim shrinkage.parameter.prior$$

That is, each coefficient evolves independently, with its own variance term which is scaled by the variance of the i'th column of X. The parameters of the hyperprior are interpretable as: sqrt(b/a) typical amount that a coefficient might change in a single time period, and 'a' is the 'sample size' or 'shrinkage parameter' measuring the degree of similarity in sigma[i] among the arms.

In most cases we hope b/a is small, so that sigma[i]'s will be small and the series will be forecastable. We also hope that 'a' is large because it means that the sigma[i]'s will be similar to one another.

The default prior distribution is a pair of independent Gamma priors for sqrt(b/a) and a. The mean of sigma[i] is set to .01 * sd(y) with shape parameter equal to 1. The mean of the shrinkage parameter is set to 10, but with shape parameter equal to 1.

## Value

Returns a list with the elements necessary to specify a dynamic regression model.

## Author(s)

Steven L. Scott <stevescott@google.com>

## References

Harvey (1990), "Forecasting, structural time series, and the Kalman filter", Cambridge University Press.

Durbin and Koopman (2001), "Time series analysis by state space methods", Oxford University Press.

## See Also

bsts. SdPrior NormalPrior

## Examples

```
n <- 1000
x <- matrix(rnorm(n))

# beta follows a random walk with sd = .1 starting at -12.
beta <- cumsum(rnorm(n, 0, .1)) - 12

# level is a local level model with sd = 1 starting at 18.
level <- cumsum(rnorm(n)) + 18

# sigma.obs is .1
error <- rnorm(n, 0, .1)

y <- level + x * beta + error
par(mfrow = c(1, 3))
plot(y, main = "Raw Data")
plot(x, y - level, main = "True Regression Effect")
plot(y - x * beta, main = "Local Level Effect")

ss <- list()
ss <- AddLocalLevel(ss, y)
ss <- AddDynamicRegression(ss, y ~ x)
model <- bsts(y, state.specification = ss, niter = 500)
plot(model, "dynamic")
```

`add.generalized.local.linear.trend` *A generalized local linear trend state component*

## Description

The generalized local linear trend model is more complicated. It assumes the level moves according to a random walk, but the slope moves according to an AR1 process centered on some potentially nonzero value $D$. The equation for the mean is

$$\mu_{t+1} = \mu_t + \delta_t + \epsilon_t \qquad \epsilon_t \sim \mathcal{N}(\emptyset, \sigma_\mu).$$

The equation for the slope is

$$\delta_{t+1} = D + \phi(\delta_t - D) + \eta_t \qquad \eta_t \sim \mathcal{N}(\emptyset, \sigma_\delta).$$

The prior distribution for this model has four independent components. There is an inverse gamma prior on the level standard deviation $\sigma_\mu$, an inverse gamma prior on the slope standard deviation $\sigma_\delta$, a Gaussian prior on the long run slope parameter $D$, and a potentially truncated Gaussian prior on the AR1 coefficient $\phi$. If the prior on $\phi$ is truncated to (-1, 1), then the slope will exhibit short term stationary variation around the long run slope $D$.

## Usage

```
AddGeneralizedLocalLinearTrend(
   state.specification,
   y,
   level.sigma.prior,
   slope.mean.prior,
   slope.ar1.prior,
   slope.sigma.prior,
   initial.level.prior,
   initial.slope.prior,
   sdy,
   initial.y)
```

## Arguments

`state.specification`
: A list of state components that you wish to add to. If omitted, an empty list will be assumed.

`y`
: The time series to be modeled, as a numeric vector.

`level.sigma.prior`
: An object created by `SdPrior` describing the prior distribution for the standard deviation of the level component.

`slope.mean.prior`
: An object created by `NormalPrior` giving the prior distribution for the mean parameter in the generalized local linear trend model (see below).

`slope.ar1.prior`
: An object created by `Ar1CoefficientPrior` giving the prior distribution for the ar1 coefficient parameter in the generalized local linear trend model (see below).

`slope.sigma.prior`
: An object created by `SdPrior` describing the prior distribution of the standard deviation of the slope component.

`initial.level.prior`
: An object created by `NormalPrior` describing the initial distribution of the level portion of the initial state vector.

`initial.slope.prior`
: An object created by `NormalPrior` describing the prior distribution for the slope portion of the initial state vector.

`sdy`
: The standard deviation of the series to be modeled. This will be ignored if `y` is provided, or if all the required prior distributions are supplied directly.

`initial.y`
: The initial value of the series being modeled. This will be ignored if `y` is provided, or if the priors for the initial state are all provided directly.

## Value

Returns a list with the elements necessary to specify a generalized local linear trend state model.

## Author(s)

Steven L. Scott <stevescott@google.com>

## References

Harvey (1990), "Forecasting, structural time series, and the Kalman filter", Cambridge University Press.

Durbin and Koopman (2001), "Time series analysis by state space methods", Oxford University Press.

## See Also

`bsts`. `SdPrior` `NormalPrior`

add.holiday *Holiday state models*

## Description

Add a random-walk holiday model to a state specification.

This model allows each active day in a holiday window to move according to a random walk relative to the day's value the last time the holiday occurred.

## Usage

```
AddFixedDateHoliday(state.specification = NULL,
                    holiday.name,
                    month,
                    day,
                    y,
                    sigma.prior = NULL,
                    initial.state.prior = NULL,
                    sdy = sd(as.numeric(y), na.rm = TRUE),
                    time0 = NULL,
                    days.before = 1,
                    days.after = 1)

AddNthWeekdayInMonthHoliday(state.specification = NULL,
                            holiday.name,
                            month,
                            day.of.week,
                            which.week,
                            y,
                            sigma.prior = NULL,
                            initial.state.prior = NULL,
                            sdy = sd(as.numeric(y), na.rm = TRUE),
                            time0 = NULL,
                            days.before = 1,
                            days.after = 1)

AddLastWeekdayInMonthHoliday(state.specification = NULL,
                             holiday.name,
                             month,
                             day.of.week,
                             y,
                             sigma.prior = NULL,
                             initial.state.prior = NULL,
                             sdy = sd(as.numeric(y), na.rm = TRUE),
                             time0 = NULL,
```

```
                              days.before = 1,
                              days.after = 1)

NamedHolidays(except = NULL)

AddNamedHolidays(state.specification = NULL,
                 named.holidays = NamedHolidays(),
                 y,
                 sigma.prior = NULL,
                 initial.state.prior = NULL,
                 sdy = sd(as.numeric(y), na.rm = TRUE),
                 time0 = NULL,
                 days.before = 1,
                 days.after = 1)
```

## Arguments

| | |
|---|---|
| state.specification | A list of state components that you wish augment. If omitted, an empty list will be assumed. |
| holiday.name | A string that can be used to label the holiday in output. |
| named.holidays | A character vector containing one or more recognized holiday names. |
| y | The time series to be modeled, as a numeric vector convertible to xts. This state model assumes y contains daily data. |
| sigma.prior | An object created by SdPrior describing the prior distribution for the standard deviation of the random walk increments. |
| initial.state.prior | An object created using NormalPrior, describing the prior distribution of the the initial state vector (at time 1). |
| sdy | The standard deviation of the series to be modeled. This will be ignored if y is provided, or if all the required prior distributions are supplied directly. |
| time0 | An object convertible to POSIXt containing the date of the initial observation in the training data. If omitted and y is a zoo or xts object, then time0 will be obtained from the index of y[1]. |
| days.before | An integer giving the number of days the influence of the holiday extends prior to the actual holiday. |
| days.after | An integer giving the number of days the influence of the holiday extends after the actual holiday. |
| month | A string naming the month in which the holiday occurs. Unambiguous partial matches are acceptable. Capitalize the first letter. |
| day | An integer specifying the day of the month on which the FixedDateHoliday occurs. |
| day.of.week | A string giving the weekday on which the NthWeekdayInMonthHoliday occurs. |
| which.week | An integer specifying the week of the month on which the NthWeekdayInMonthHoliday occurs. If which.week <= 0 then the holiday is assumed to occur on the last day.of.week in month. |

except — If NULL then all named holidays are returned. If except is a character vector containing partial matches to holiday names, the matched names will be omitted from the returned list.

## Value

NamedHolidays returns a character vector with the names of the recognized holiday.

The other functions return state.specification, after adding the requested state components.

AddNthWeekdayInMonthHoliday, AddLastWeekdayInMonthHoliday, and AddFixedDateHoliday can each add one holiday at a time. AddNamedHolidays will add several holidays at once, if named.holidays is a vector.

## Author(s)

Steven L. Scott <stevescott@google.com>

## References

Harvey (1990), "Forecasting, structural time series, and the Kalman filter", Cambridge University Press.

Durbin and Koopman (2001), "Time series analysis by state space methods", Oxford University Press.

## See Also

bsts. SdPrior NormalPrior

## Examples

```
## TODO(stevescott):  add examples
```

---

# add.local.level *Local level trend state component*

---

## Description

Add a local level model to a state specification. The local level model assumes the trend is a random walk:

$$\alpha_{t+1} = \alpha_t + \epsilon_t \qquad \epsilon_t \sim \mathcal{N}(0, \sigma).$$

The prior is on the $\sigma$ parameter.

## Usage

```
AddLocalLevel(
   state.specification,
   y,
   sigma.prior,
   initial.state.prior,
   sdy,
   initial.y)
```

## Arguments

| | |
|---|---|
| state.specification | A list of state components that you wish to add to. If omitted, an empty list will be assumed. |
| y | The time series to be modeled, as a numeric vector. |
| sigma.prior | An object created by SdPrior describing the prior distribution for the standard deviation of the random walk increments. |
| initial.state.prior | An object created using NormalPrior, describing the prior distribution of the the initial state vector (at time 1). |
| sdy | The standard deviation of the series to be modeled. This will be ignored if y is provided, or if all the required prior distributions are supplied directly. |
| initial.y | The initial value of the series being modeled. This will be ignored if y is provided, or if the priors for the initial state are all provided directly. |

## Value

Returns a list with the elements necessary to specify a local linear trend state model.

## Author(s)

Steven L. Scott <stevescott@google.com>

## References

Harvey (1990), "Forecasting, structural time series, and the Kalman filter", Cambridge University Press.

Durbin and Koopman (2001), "Time series analysis by state space methods", Oxford University Press.

## See Also

bsts. SdPrior NormalPrior

`add.local.linear.trend` *Local linear trend state component*

## Description

Add a local linear trend model to a state specification. The local linear trend model assumes that both the mean and the slope of the trend follow random walks. The equation for the mean is

$$\mu_{t+1} = \mu_t + \delta_t + \epsilon_t \qquad \epsilon_t \sim \mathcal{N}(0, \sigma_\mu).$$

The equation for the slope is

$$\delta_{t+1} = \delta_t + \eta_t \qquad \eta_t \sim \mathcal{N}(0, \sigma_\delta).$$

The prior distribution is on the level standard deviation $\sigma_\mu$ and the slope standard deviation $\sigma_\delta$.

## Usage

```
AddLocalLinearTrend(
   state.specification = NULL,
   y,
   level.sigma.prior = NULL,
   slope.sigma.prior = NULL,
   initial.level.prior = NULL,
   initial.slope.prior = NULL,
   sdy,
   initial.y)
```

## Arguments

`state.specification`
: A list of state components that you wish to add to. If omitted, an empty list will be assumed.

`y`
: The time series to be modeled, as a numeric vector.

`level.sigma.prior`
: An object created by `SdPrior` describing the prior distribution for the standard deviation of the level component.

`slope.sigma.prior`
: An object created by `SdPrior` describing the prior distribution of the standard deviation of the slope component.

`initial.level.prior`
: An object created by `NormalPrior` describing the initial distribution of the level portion of the initial state vector.

`initial.slope.prior`
: An object created by `NormalPrior` describing the prior distribution for the slope portion of the initial state vector.

| | |
|---|---|
| sdy | The standard deviation of the series to be modeled. This will be ignored if y is provided, or if all the required prior distributions are supplied directly. |
| initial.y | The initial value of the series being modeled. This will be ignored if y is provided, or if the priors for the initial state are all provided directly. |

## Value

Returns a list with the elements necessary to specify a local linear trend state model.

## Author(s)

Steven L. Scott <stevescott@google.com>

## References

Harvey (1990), "Forecasting, structural time series, and the Kalman filter", Cambridge University Press.

Durbin and Koopman (2001), "Time series analysis by state space methods", Oxford University Press.

## See Also

bsts. SdPrior NormalPrior

## Examples

```
data(AirPassengers)
y <- log(AirPassengers)
ss <- AddLocalLinearTrend(list(), y)
ss <- AddSeasonal(ss, y, nseasons = 12)
model <- bsts(y, state.specification = ss, niter = 500)
pred <- predict(model, horizon = 12, burn = 100)
plot(pred)
```

---

# add.seasonal *Seasonal state component*

---

## Description

Add a seasonal model to a state specification.

The seasonal model can be thought of as a regression on nseasons dummy variables with coefficients constrained to sum to 1 (in expectation). If there are S seasons then the state vector $\gamma$ is of dimension S-1. The first element of the state vector obeys

$$\gamma_{t+1,1} = -\sum_{i=2}^{S} \gamma_{t,i} + \epsilon_t \qquad \epsilon_t \sim \mathcal{N}(0, \sigma)$$

## Usage

```
AddSeasonal(
   state.specification,
   y,
   nseasons,
   season.duration = 1,
   sigma.prior,
   initial.state.prior,
   sdy)
```

## Arguments

| | |
|---|---|
| state.specification | A list of state components that you wish to add to. If omitted, an empty list will be assumed. |
| y | The time series to be modeled, as a numeric vector. |
| nseasons | The number of seasons to be modeled. |
| season.duration | The number of time periods in each season. |
| sigma.prior | An object created by SdPrior describing the prior distribution for the standard deviation of the random walk increments. |
| initial.state.prior | An object created using NormalPrior, describing the prior distribution of the the initial state vector (at time 1). |
| sdy | The standard deviation of the series to be modeled. This will be ignored if y is provided, or if all the required prior distributions are supplied directly. |

## Value

Returns a list with the elements necessary to specify a seasonal state model.

## Author(s)

Steven L. Scott <stevescott@google.com>

## References

Harvey (1990), "Forecasting, structural time series, and the Kalman filter", Cambridge University Press.

Durbin and Koopman (2001), "Time series analysis by state space methods", Oxford University Press.

## See Also

bsts. SdPrior NormalPrior

## Examples

```
data(AirPassengers)
y <- log(AirPassengers)
ss <- AddLocalLinearTrend(list(), y)
ss <- AddSeasonal(ss, y, nseasons = 12)
model <- bsts(y, state.specification = ss, niter = 500)
pred <- predict(model, horizon = 12, burn = 100)
plot(pred)
```

---

# add.student.local.linear.trend

*Robust local linear trend*

---

## Description

Add a local level model to a state specification. The local linear trend model assumes that both the mean and the slope of the trend follow random walks. The equation for the mean is

$$\mu_{t+1} = \mu_t + \delta_t + \epsilon_t \qquad \epsilon_t \sim \mathcal{T}_{\nu_\mu}(0, \sigma_\mu).$$

The equation for the slope is

$$\delta_{t+1} = \delta_t + \eta_t \qquad \eta_t \sim \mathcal{T}_{\nu_\delta}(0, \sigma_\delta).$$

Independent prior distributions are assumed on the level standard deviation, $\sigma_\mu$ the slope standard deviation $\sigma_\delta$, the level tail thickness $\nu_\mu$, and the slope tail thickness $\nu_\delta$.

## Usage

```
AddStudentLocalLinearTrend(
   state.specification = NULL,
   y,
   save.weights = FALSE,
   level.sigma.prior = NULL,
   level.nu.prior = NULL,
   slope.sigma.prior = NULL,
   slope.nu.prior = NULL,
   initial.level.prior = NULL,
   initial.slope.prior = NULL,
   sdy,
   initial.y)
```

## Arguments

`state.specification`
: A list of state components that you wish to add to. If omitted, an empty list will be assumed.

`y`
: The time series to be modeled, as a numeric vector.

`save.weights` A logical value indicating whether to save the draws of the weights from the normal mixture representation.

`level.sigma.prior`
An object created by `SdPrior` describing the prior distribution for the standard deviation of the level component.

`level.nu.prior` An object inheritng from the class `DoubleModel`, representing the prior distribution on the `nu` tail thickness parameter of the T distribution for errors in the evolution equation for the level component.

`slope.sigma.prior`
An object created by `SdPrior` describing the prior distribution of the standard deviation of the slope component.

`slope.nu.prior` An object inheritng from the class `DoubleModel`, representing the prior distribution on the `nu` tail thickness parameter of the T distribution for errors in the evolution equation for the slope component.

`initial.level.prior`
An object created by `NormalPrior` describing the initial distribution of the level portion of the initial state vector.

`initial.slope.prior`
An object created by `NormalPrior` describing the prior distribution for the slope portion of the initial state vector.

`sdy` The standard deviation of the series to be modeled. This will be ignored if `y` is provided, or if all the required prior distributions are supplied directly.

`initial.y` The initial value of the series being modeled. This will be ignored if `y` is provided, or if the priors for the initial state are all provided directly.

## Value

Returns a list with the elements necessary to specify a local linear trend state model.

## Author(s)

Steven L. Scott <stevescott@google.com>

## References

Harvey (1990), "Forecasting, structural time series, and the Kalman filter", Cambridge University Press.

Durbin and Koopman (2001), "Time series analysis by state space methods", Oxford University Press.

## See Also

`bsts`. `SdPrior` `NormalPrior`

## Examples

```
data(rsxfs)
ss <- AddStudentLocalLinearTrend(list(), rsxfs)
model <- bsts(rsxfs, state.specification = ss, niter = 500)
pred <- predict(model, horizon = 12, burn = 100)
plot(pred)
```

---

`aggregate.time.series` *Aggregate a fine time series to a coarse summary*

---

## Description

Aggregate measurements from a fine scaled time series into a coarse time series. This is similar to functions from the `xts` package, but it can handle aggregation from weeks to months.

## Usage

```
AggregateTimeSeries(fine.series,
                    contains.end,
                    membership.fraction,
                    trim.left = any(membership.fraction < 1),
                    trim.right = NULL,
                    byrow = TRUE)
```

## Arguments

| | |
|---|---|
| `fine.series` | A numeric vector or matrix giving the fine scale time series to be aggregated. |
| `contains.end` | A logical vector corresponding to `fine.series` indicating whether each fine time interval contains the end of a coarse time interval. |
| `membership.fraction` | A numeric vector corresponding to `fine.series`, giving the fraction of each time interval's observation attributable to the coarse interval containing the fine interval's first day. This will usually be a vector of 1's, unless `fine.series` is weekly. |
| `trim.left` | Logical indicating whether the first observation in the coarse aggregate should be removed. |
| `trim.right` | Logical indicating whether the final observation in the coarse aggregate should be removed. |
| `byrow` | Logical. If `fine.series` is a matrix, this argument indicates whether rows (`TRUE`) or columns (`FALSE`) correspond to time points. |

## Value

A matrix (if `fine.series` is a matrix) or vector (otherwise) containing the aggregated values of `fine.series`.

### Author(s)

Steven L. Scott <stevescott@google.com>

### Examples

```
week.ending <- as.Date(c("2011-11-05",
                         "2011-11-12",
                         "2011-11-19",
                         "2011-11-26",
                         "2011-12-03",
                         "2011-12-10",
                         "2011-12-17",
                         "2011-12-24",
                         "2011-12-31"))
membership.fraction <- GetFractionOfDaysInInitialMonth(week.ending)
which.month <- MatchWeekToMonth(week.ending, as.Date("2011-11-01"))
contains.end <- WeekEndsMonth(week.ending)

weekly.values <- rnorm(length(week.ending))
monthly.values <- AggregateTimeSeries(weekly.values, contains.end, membership.fraction)
```

---

## aggregate.weeks.to.months

*Aggregate a weekly time series to monthly*

---

### Description

Aggregate measurements from a weekly time series into a monthly time series.

### Usage

```
AggregateWeeksToMonths(weekly.series,
                       membership.fraction = NULL,
                       trim.left = TRUE,
                       trim.right = NULL)
```

### Arguments

| | |
|---|---|
| weekly.series | A numeric vector or matrix of class zoo giving the weekly time series to be aggregated. The index must be convertible to class Date. |
| membership.fraction | A optional numeric vector corresponding to weekly.series, giving the fraction of each week's observation attributable to the month containing the week's first day. If missing, then weeks will be split across months in proportion to the number of days in each month. |

| | |
|---|---|
| trim.left | Logical indicating whether the first observation in the monthly aggregate should be removed. |
| trim.right | Logical indicating whether the final observation in the monthly aggregate should be removed. |

## Value

A zoo-matrix (if weekly.series is a matrix) or vector (otherwise) containing the aggregated values of weekly.series.

## Author(s)

Steven L. Scott <stevescott@google.com>

## See Also

AggregateTimeSeries

## Examples

```
week.ending <- as.Date(c("2011-11-05",
                         "2011-11-12",
                         "2011-11-19",
                         "2011-11-26",
                         "2011-12-03",
                         "2011-12-10",
                         "2011-12-17",
                         "2011-12-24",
                         "2011-12-31"))

weekly.values <- zoo(rnorm(length(week.ending)), week.ending)
monthly.values <- AggregateWeeksToMonths(weekly.values)
```

---

# bsts *Bayesian structural time series*

---

## Description

Uses MCMC to sample from the posterior distribution of a Bayesian structural time series model. This function can be used either with or without contemporaneous predictor variables (in a time series regression).

If predictor variables are present, the regression coefficients are fixed (as opposed to time varying, though time varying coefficients might be added as state component). The predictors and response in the formula are contemporaneous, so if you want lags and differences you need to put them in the predictor matrix yourself.

If no predictor variables are used, then the model is an ordinary state space time series model.

## Usage

```
bsts(formula,
     state.specification,
     save.state.contributions = TRUE,
     save.prediction.errors = TRUE,
     data,
     prior,
     contrasts = NULL,
     na.action = na.pass,
     niter,
     ping = niter / 10,
     seed = NULL,
     ...)
```

## Arguments

| | |
|---|---|
| formula | A formula describing the regression portion of the relationship between y and X.<br>If no regressors are desired then the formula can be replaced by a numeric vector giving the time series to be modeled. Missing values are not allowed.<br>If the response variable is of class zoo, xts, or ts, then the time series information it contains will be used in many of the plotting methods called from plot.bsts. |
| state.specification | A list with elements created by AddLocalLinearTrend, AddSeasonal, and similar functions for adding components of state. See the help page for state.specification. |
| save.state.contributions | Logical. If TRUE then a 3-way array named state.contributions will be stored in the returned object. The indices correspond to MCMC iteration, state model number, and time. Setting save.state.contributions to FALSE yields a smaller object, but plot will not be able to plot the the "state", "components", or "residuals" for the fitted model. |
| save.prediction.errors | Logical. If TRUE then a matrix named one.step.prediction.errors will be saved as part of the model object. The rows of the matrix represent MCMC iterations, and the columns represent time. The matrix entries are the one-step-ahead prediction errors from the Kalman filter. |
| data | An optional data frame, list or environment (or object coercible by as.data.frame to a data frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which bsts is called. |
| prior | If regressors are supplied in the model formula, then this is a prior distribution for the regression component of the model, as created by SpikeSlabPrior. The prior for the time series component of the model will be specified during the creation of state.specification. This argument is only used if a formula is specified.<br>If the model contains no regressors, then this is simply the prior on the residual standard deviation, expressed as an object created by SdPrior. |

| | |
|---|---|
| contrasts | An optional list. See the contrasts.arg of model.matrix.default. This argument is only used if a model formula is specified. Even then the default is usually what you want. |
| na.action | What to do about missing values. The default is to allow missing responses, but no missing predictors. Set this to na.omit or na.exclude if you want to omit missing responses altogether. |
| niter | A positive integer giving the desired number of MCMC draws. |
| ping | A scalar giving the desired frequency of status messages. If ping > 0 then the program will print a status message to the screen every ping MCMC iterations. |
| seed | An integer to use as the random seed for the underlying C++ code. If NULL then the seed will be set using the clock. |
| ... | Extra arguments to be passed to SpikeSlabPrior (see the entry for the prior argument, above). |

## Value

An object of class bsts which is a list with the following components

| | |
|---|---|
| coefficients | A niter by ncol(X) matrix of MCMC draws of the regression coefficients, where X is the design matrix implied by formula. This is only present if a model formula was supplied. |
| sigma.obs | A vector of length niter containing MCMC draws of the residual standard deviation. |

The returned object will also contain named elements holding the MCMC draws of model parameters belonging to the state models. The names of each component are supplied by the entries in state.specification. If a model parameter is a scalar, then the list element is a vector with niter elements. If the parameter is a vector then the list element is a matrix with niter rows. If the parameter is a matrix then the list element is a 3-way array with first dimension niter.

Finally, if a model formula was supplied, then the returned object will contain the information necessary for the predict method to build the design matrix when a new prediction is made.

## Author(s)

Steven L. Scott <stevescott@google.com>

## References

Harvey (1990), "Forecasting, structural time series, and the Kalman filter", Cambridge University Press.

Durbin and Koopman (2001), "Time series analysis by state space methods", Oxford University Press.

## See Also

bsts, AddLocalLevel, AddLocalLinearTrend, AddGeneralizedLocalLinearTrend, AddSeasonal AddDynamicRegression SpikeSlabPrior, SdPrior.

## Examples

```
## Example 1:  Time series (ts) data
data(AirPassengers)
y <- log(AirPassengers)
ss <- AddLocalLinearTrend(list(), y)
ss <- AddSeasonal(ss, y, nseasons = 12)
model <- bsts(y, state.specification = ss, niter = 500)
pred <- predict(model, horizon = 12, burn = 100)
par(mfrow = c(1,2))
plot(model)
plot(pred)

## Not run:

MakePlots <- function(model, ask = TRUE) {
  ## Make all the plots callable by plot.bsts.
  opar <- par(ask = ask)
  on.exit(par(opar))
  plot.types <- c("state", "components", "residuals",
                  "prediction.errors", "forecast.distribution")
  for (plot.type in plot.types) {
    plot(model, plot.type)
  }
  if (model$has.regression) {
    regression.plot.types <- c("coefficients", "predictors", "size")
    for (plot.type in regression.plot.types) {
      plot(model, plot.type)
    }
  }
}

## Example 2: GOOG is the Google stock price, an xts series of daily
##            data.
data(goog)
ss <- AddGeneralizedLocalLinearTrend(list(), goog)
model <- bsts(goog, state.specification = ss, niter = 500)
MakePlots(model)

## Example 3:  Change GOOG to be zoo, and not xts.
goog <- zoo(goog, index(goog))
ss <- AddGeneralizedLocalLinearTrend(list(), goog)
model <- bsts(goog, state.specification = ss, niter = 500)
MakePlots(model)

## Example 4:  Naked numeric data works too
y <- rnorm(100)
ss <- AddLocalLinearTrend(list(), y)
model <- bsts(y, state.specification = ss, niter = 500)
MakePlots(model)

## Example 5:  zoo data with intra-day measurements
y <- zoo(rnorm(100),
```

```
          seq(from = as.POSIXct("2012-01-01 7:00 EST"), len = 100, by = 100))
  ss <- AddLocalLinearTrend(list(), y)
  model <- bsts(y, state.specification = ss, niter = 500)
  MakePlots(model)

## End(Not run)
```

---

`bsts.holdout.prediction.errors` *One step prediction errors on a holdout sample*

---

## Description

Computes the one-step-ahead prediction errors for a model of class `bsts` on a holdout sample.

## Usage

```
bsts.holdout.prediction.errors(bsts.object,
                               newdata,
                               burn = SuggestBurn(.1, bsts.object),
                               na.action = na.omit)
```

## Arguments

| | |
|---|---|
| `bsts.object` | An object of class `bsts`. |
| `newdata` | The holdout sample of data. If `bsts.object` contains a regression component then this must be a `data.frame` with all the relevant variables in the model formula for `bsts.object`. Otherwise this should be a numeric vector. |
| `burn` | An integer giving the number of MCMC iterations to discard as burn-in. If `burn <= 0` then no burn-in sample will be discarded. |
| `na.action` | A function determining what should be done with missing values in `newdata`. |

## Value

A matrix of draws of the one-step-ahead prediction errors. Rows of the matrix correspond to MCMC draws. Columns correspond to time.

## Author(s)

Steven L. Scott <stevescott@google.com>

## References

Harvey (1990), "Forecasting, structural time series, and the Kalman filter", Cambridge University Press.

Durbin and Koopman (2001), "Time series analysis by state space methods", Oxford University Press.

## See Also

bsts, AddLocalLevel, AddLocalLinearTrend, AddGeneralizedLocalLinearTrend, SpikeSlabPrior, SdPrior, bsts.prediction.errors.

## Examples

```
data(AirPassengers)
y <- log(AirPassengers)
ss <- AddLocalLinearTrend(list(), y)
ss <- AddSeasonal(ss, y, nseasons = 12)
model <- bsts(y, state.specification = ss, niter = 500)
errors <- bsts.prediction.errors(model, burn = 100)
PlotDynamicDistribution(errors)
```

---

# bsts.prediction.errors

*One step prediction errors*

---

## Description

Computes the one-step-ahead prediction errors for a model of class bsts.

## Usage

```
bsts.prediction.errors(bsts.object,
                       newdata,
                       burn = SuggestBurn(.1, bsts.object),
                       na.action = na.omit)
```

## Arguments

| | |
|---|---|
| bsts.object | An object of class bsts. |
| newdata | An optional holdout sample of data. If bsts.object contains a regression component then this must be a data.frame with all the relevant variables in the model formula for bsts.object. Otherwise this should be a numeric vector. If newdata is omitted then the prediction errors will be with respect to the training data. |
| burn | An integer giving the number of MCMC iterations to discard as burn-in. If burn <= 0 then no burn-in sample will be discarded. |
| na.action | A function determining what should be done with missing values in newdata. |

## Value

A matrix of draws of the one-step-ahead prediction errors. Rows of the matrix correspond to MCMC draws. Columns correspond to time.

### Author(s)

Steven L. Scott <stevescott@google.com>

### References

Harvey (1990), "Forecasting, structural time series, and the Kalman filter", Cambridge University Press.

Durbin and Koopman (2001), "Time series analysis by state space methods", Oxford University Press.

### See Also

bsts, AddLocalLevel, AddLocalLinearTrend, AddGeneralizedLocalLinearTrend, SpikeSlabPrior, SdPrior, bsts.holdout.prediction.errors.

### Examples

```
data(AirPassengers)
y <- log(AirPassengers)
ss <- AddLocalLinearTrend(list(), y)
ss <- AddSeasonal(ss, y, nseasons = 12)
model <- bsts(y, state.specification = ss, niter = 500)
errors <- bsts.prediction.errors(model, burn = 100)
PlotDynamicDistribution(errors)
```

---

## compare.bsts.models *Compare bsts models*

---

### Description

Produce a set of line plots showing the cumulative absolute one step ahead prediction errors for different models. This plot not only shows which model is doing the best job predicting the data, it highlights regions of the data where the predictions are particularly good or bad.

### Usage

```
CompareBstsModels(model.list,
                  burn = SuggestBurn(.1, model.list[[1]]),
                  filename = "",
                  colors = NULL,
                  lwd = 2,
                  xlab = "Time",
                  main = "",
                  grid = TRUE)
```

### Arguments

| | |
|---|---|
| model.list | A list of bsts models. |
| burn | The number of initial MCMC iterations to remove from each model as burn-in. |
| filename | A string. If non-empty string then a pdf of the plot will be saved in the specified file. |
| colors | A vector of colors to use for the different lines in the plot. If NULL then the rainbow pallette will be used. |
| lwd | The width of the lines to be drawn. |
| xlab | Label for the horizontal axis. |
| main | Main title for the plot. |
| grid | Logical. Should gridlines be drawn in the background? |

### Value

Returns invisible NULL.

### Author(s)

Steven L. Scott <stevescott@google.com>

### Examples

```
data(AirPassengers)
y <- log(AirPassengers)
ss <- AddLocalLinearTrend(list(), y)
trend.only <- bsts(y, ss, niter = 500)

ss <- AddSeasonal(ss, y, nseasons = 12)
trend.and.seasonal <- bsts(y, ss, niter = 500)

CompareBstsModels(list(trend = trend.only,
                       "trend and seasonal" = trend.and.seasonal))
```

---

## estimate.time.scale *Intervals between dates*

---

### Description

Estimate the time scale used in time series data.

### Usage

```
EstimateTimeScale(dates)
```

### Arguments

| | |
|---|---|
| dates | A sorted vector of class Date. |

## Value

A character string. Either "daily", "weekly", "yearly", "monthly", "quarterly", or "other". The value is determined based on counting the number of days between successive observations in dates.

## Author(s)

Steven L. Scott <stevescott@google.com>

## Examples

```
weekly.data <- as.Date(c("2011-10-01",
                         "2011-10-08",
                         "2011-10-15",
                         "2011-10-22",
                         "2011-10-29",
                         "2011-11-05"))

EstimateTimeScale(weekly.data) # "weekly"

almost.weekly.data <- as.Date(c("2011-10-01",
                                "2011-10-08",
                                "2011-10-15",
                                "2011-10-22",
                                "2011-10-29",
                                "2011-11-06"))  # last day is one later

EstimateTimeScale(weekly.data) # "other"
```

---

# extend.time *Extends a vector of dates to a given length*

---

## Description

Pads a vector of dates to a specified length.

## Usage

```
ExtendTime(dates, number.of.periods, dt = NULL)
```

## Arguments

| | |
|---|---|
| dates | An ordered vector of class Date. |
| number.of.periods | The desired length of the output. |
| dt | A character string describing the frequency of the dates in dates. Possible values are "daily", "weekly", "monthly", "quarterly", "yearly", or "other". An attempt to deduce dt will be made if it is missing. |

### Value

If number.of.periods is longer than length(dates), then dates will be padded to the desired length. Extra dates are added at time intervals matching the average interval in dates. Thus they may not be

### Author(s)

Steven L. Scott <stevescott@google.com>

### See Also

bsts.mixed.

### Examples

```
origin.month <- as.Date("2011-09-01")
week.ending <- as.Date(c("2011-10-01",   ## 1
                         "2011-10-08",   ## 2
                         "2011-12-03",   ## 3
                         "2011-12-31"))  ## 4
MatchWeekToMonth(week.ending, origin.month) == 1:4
```

---

## get.fraction *Compute membership fractions*

---

### Description

Returns the fraction of days in a week that occur in the ear

### Usage

```
GetFractionOfDaysInInitialMonth(week.ending)
GetFractionOfDaysInInitialQuarter(week.ending)
```

### Arguments

| | |
|---|---|
| week.ending | A vector of class Date. Each entry contains the date of the last day in a week. |

### Value

Returns a numeric vector of the same length as week.ending. Each entry gives the fraction of days in the week that occur in the coarse time interval (month or quarter) containing the start of the week (i.e the date 6 days before).

### Author(s)

Steven L. Scott <stevescott@google.com>

## See Also

bsts.mixed.

## Examples

```
dates <- as.Date(c("2003-03-31",
                   "2003-04-01",
                   "2003-04-02",
                   "2003-04-03",
                   "2003-04-04",
                   "2003-04-05",
                   "2003-04-06",
                   "2003-04-07"))
fraction <- GetFractionOfDaysInInitialMonth(dates)
fraction == c(1, 6/7, 5/7, 4/7, 3/7, 2/7, 1/7, 1)
```

---

# goog *Google stock price*

---

## Description

Daily closing price of Google stock.

## Usage

```
data(goog)
```

## Format

xts time series

## Source

The Internets

---

# HarveyCumulator *HarveyCumulator*

---

## Description

Given a state space model on a fine scale, the Harvey cumulator aggregates the model to a coarser scale (e.g. from days to weeks, or weeks to months).

## Usage

```
HarveyCumulator(fine.series,
                contains.end,
                membership.fraction)
```

## Arguments

| | |
|---|---|
| `fine.series` | The fine-scale time series to be aggregated. |
| `contains.end` | A logical vector, with length matching `fine.series` indicating whether each fine scale time interval contains the end of a coarse time interval. For example, months don't contain a fixed number of weeks, so when cumulating a weekly time series into a monthly series, you need to know which weeks contain the end of a month. |
| `membership.fraction` | The fraction of each fine-scale time observation belonging to the coarse scale time observation at the beginning of the time interval. For example, if week i started in March and ended in April, `membership.fraction[i]` is the fraction of fine.series[i] that should be attributed to March. This should be 1 for most observations. |

## Value

Returns a vector containing the course scale partial aggregates of `fine.series`.

## Author(s)

Steven L. Scott <stevescott@google.com>

## References

Harvey (1990), "Forecasting, structural time series, and the Kalman filter", Cambridge University Press.

Durbin and Koopman (2001), "Time series analysis by state space methods", Oxford University Press.

## See Also

`bsts.mixed`,

## Examples

```
data(goog)
days <- factor(weekdays(index(goog)),
               levels = c("Monday", "Tuesday", "Wednesday",
                          "Thursday", "Friday"),
               ordered = TRUE)

## Because of holidays, etc the days do not always go in sequence.
```

```
## (Sorry, Rebecca Black! https://www.youtube.com/watch?v=kfVsfOSbJY0)
## diff.days[i] is the number of days between days[i-1] and days[i].
## We know that days[i] is the end of a week if diff.days[i] < 0.
diff.days <- tail(as.numeric(days), -1) - head(as.numeric(days), -1)
contains.end <- c(FALSE, diff.days < 0)

goog.weekly <- HarveyCumulator(goog, contains.end, 1)
```

---

`last.day.in.month` *Find the last day in a month*

---

## Description

Finds the last day in the month containing a specefied date.

## Usage

```
LastDayInMonth(dates)
```

## Arguments

| | |
|---|---|
| `dates` | A vector of class `Date`. |

## Value

A vector of class `Date` where each entry is the last day in the month containing the corresponding entry in `dates`.

## Author(s)

Steven L. Scott <stevescott@google.com>

## Examples

```
inputs <- as.Date(c("2007-01-01",
                    "2007-01-31",
                    "2008-02-01",
                    "2008-02-29",
                    "2008-03-14",
                    "2008-12-01",
                    "2008-12-31"))
expected.outputs <- as.Date(c("2007-01-31",
                              "2007-01-31",
                              "2008-02-29",
                              "2008-02-29",
                              "2008-03-31",
                              "2008-12-31",
                              "2008-12-31"))
LastDayInMonth(inputs) == expected.outputs
```

`match.week.to.month` *Find the month containing a week*

## Description

Returns the index of a month, in a sequence of months, that contains a given week.

## Usage

```
MatchWeekToMonth(week.ending, origin.month)
```

## Arguments

| | |
|---|---|
| `week.ending` | A vector of class `Date`. Each entry contains the date of the last day in a week. |
| `origin.month` | A `Date`, giving any day in the month to use as the origin of the sequence (month 1). |

## Value

The index of the month matching the month containing the first day in `week.ending`. The origin is month 1. It is the caller's responsibility to ensure that these indices correspond to legal values in a particular vector of months.

## Author(s)

Steven L. Scott <stevescott@google.com>

## See Also

`bsts.mixed`.

## Examples

```
origin.month <- as.Date("2011-09-01")
week.ending <- as.Date(c("2011-10-01",   ## 1
                         "2011-10-08",   ## 2
                         "2011-12-03",   ## 3
                         "2011-12-31"))  ## 4
MatchWeekToMonth(week.ending, origin.month) == 1:4
```

`mixed.frequency` *Models for mixed frequency time series*

## Description

Fit a structured time series to mixed frequncy data.

## Usage

```
bsts.mixed(target.series,
           predictors,
           which.coarse.interval,
           membership.fraction,
           contains.end,
           state.specification,
           regression.prior,
           niter,
           ping = niter / 10,
           seed = NULL,
           truth = NULL,
           ...)
```

## Arguments

| | |
|---|---|
| `target.series` | A vector object of class zoo indexed by calendar dates. The date associated with each element is the LAST DAY in the time interval measured by the corresponding value. The value is what Harvey (1989) calls a 'flow' variable. It is a number that can be viewed as an accumulation over the measured time interval. |
| `predictors` | A matrix of class zoo indexed by calendar dates. The date associated with each row is the LAST DAY in the time interval encompasing the measurement. The dates are expected to be at a finer scale than the dates in `target.series`. Any predictors should be at sufficient lags to be able to predict the rest of the cycle. |
| `which.coarse.interval` | A numeric vector of length `nrow(predictors)` giving the index of the coarse interval corresponding to the end of each fine interval. |
| `membership.fraction` | A numeric vector of length `nrow(predictors)` giving the fraction of activity attributed to the coarse interval corresponding to the beginning of each fine interval. This is always positive, and will be 1 except when a fine interval spans the boundary between two coarse intervals. |
| `contains.end` | A logical vector of length `nrow(predictors)` indicating whether each fine interval contains the end of a coarse interval. |
| `state.specification` | A state specification like that required by `bsts`. |

`regression.prior`
: A prior distribution created by `SpikeSlabPrior`. A default prior will be generated if none is specified.

`niter`
: The desired number of MCMC iterations.

`ping`
: An integer indicating the frequency with which progress reports get printed. E.g. setting `ping = 100` will print a status message with a time and iteration stamp every 100 iterations. If you don't want these messages set `ping < 0`.

`seed`
: An integer to use as the random seed for the underlying C++ code. If `NULL` then the seed will be set using the clock.

`truth`
: For debugging purposes only. A list containing one or more of the following elements. If any are present then corresponding values will be held fixed in the MCMC algorithm.
  - A matrix named `state` containing the state of the coarse model from a fake-data simulation.
  - A vector named `beta` of regression coefficients.
  - A scalar named `sigma.obs`.

`...`
: Extra arguments passed to SpikeSlabPrior

## Value

An object of class `bsts.mixed`, which is a list with the following elements. Many of these are arrays, in which case the first index of the array corresponds to the MCMC iteration number.

`coefficients`
: A matrix containing the MCMC draws of the regression coefficients. Rows correspond to MCMC draws, and columns correspond to variables.

`sigma.obs`
: The standard deviation of the weekly latent observations.

`state.contributions`
: A three-dimensional array containing the MCMC draws of each state model's contributions to the state of the weekly model. The three dimensions are MCMC iteration, state model, and week number.

`weekly`
: A matrix of MCMC draws of the weekly latent observations. Rows are MCMC iterations, and columns are weekly time points.

`cumulator`
: A matrix of MCMC draws of the cumulator variable.

The returned object also contains MCMC draws for the parameters of the state models supplied as part of `state.specification`, relevant information passed to the function call, and other supplemental information.

## Author(s)

Steven L. Scott <stevescott@google.com>

## References

Harvey (1990), "Forecasting, structural time series, and the Kalman filter", Cambridge University Press.

Durbin and Koopman (2001), "Time series analysis by state space methods", Oxford University Press.

## See Also

bsts, AddLocalLevel, AddLocalLinearTrend, AddGeneralizedLocalLinearTrend, SpikeSlabPrior, SdPrior.

## Examples

```
data <- SimulateFakeMixedFrequencyData(nweeks = 104, xdim = 20)

## Setting an upper limit on the standard deviations can help keep the
## MCMC from flying off to infinity.
sd.limit <- sd(data$coarse.target)
state.specification <-
     AddLocalLinearTrend(list(),
                  data$coarse.target,
                  level.sigma.prior = SdPrior(1.0, 5, upper.limit = sd.limit),
                  slope.sigma.prior = SdPrior(.5, 5, upper.limit = sd.limit))
weeks <- index(data$predictor)
months <- index(data$coarse.target)
which.month <- MatchWeekToMonth(weeks, months[1])
membership.fraction <- GetFractionOfDaysInInitialMonth(weeks)
contains.end <- WeekEndsMonth(weeks)

model <- bsts.mixed(target.series = data$coarse.target,
                    predictors = data$predictors,
                    membership.fraction = membership.fraction,
                    contains.end = contains.end,
                    which.coarse = which.month,
                    state.specification = state.specification,
                    niter = 500,
                    expected.r2 = .999,
                    prior.df = 1)

plot(model, "state")
plot(model, "components")
```

---

# month.distance *Elapsed time in months*

---

## Description

The (integer) number of months between dates.

## Usage

```
MonthDistance(dates, origin)
```

### Arguments

| | |
|---|---|
| dates | A vector of class Date to be measured. |
| origin | A scalar of class Date. |

### Value

Returns a numeric vector giving the integer number of months that have elapsed between origin and each element in dates. The daily component of each date is ignored, so two dates that are in the same month will have the same measured distance. Distances are signed, so months that occur before origin will have negative values.

### Author(s)

Steven L. Scott <stevescott@google.com>

### Examples

```
dates <- as.Date(c("2008-04-17",
                   "2008-05-01",
                   "2008-05-31",
                   "2008-06-01"))
origin <- as.Date("2008-05-15")
MonthDistance(dates, origin) ==  c(-1, 0, 0, 1)
```

---

## new.home.sales *New home sales and Google trends*

---

### Description

The first column, HSN1FNSA is a time series of new home sales in the US, obtained from the FRED online data base. The series has been manually deseasonalized. The remaining columns contain search terms from Google trends (obtained from http://trends.google.com/correlate). These show the relative popularity of each search term among all serach terms typed into Google. All series in this data set have been standardized by subtracting off their mean and dividing by their standard deviation.

### Usage

```
data(new.home.sales)
```

### Format

zoo time series

### Source

FRED and trends.google.com

plot.bsts *Plotting functions for Bayesian structural time series*

## Description

Functions to plot the results of a model fit using bsts.

## Usage

```
  ## S3 method for class 'bsts'
plot(x, y = c("state", "components", "residuals",
              "coefficients", "prediction.errors",
              "forecast.distribution",
              "predictors", "size", "dynamic"),
     ...)

  PlotBstsCoefficients(bsts.object, burn = SuggestBurn(.1, bsts.object),
                       inclusion.threshold = 0, number.of.variables = NULL, ...)
  PlotBstsComponents(bsts.object, burn = SuggestBurn(.1, bsts.object),
                     time, same.scale = TRUE,
                     layout = c("square", "horizontal", "vertical"),
                     style = c("dynamic", "boxplot"),
                     ylim = NULL, ...)
  PlotDynamicRegression(bsts.object, burn = SuggestBurn(.1, bsts.object),
                        time = NULL, style = c("dynamic", "boxplot"),
                        layout = c("square", "horizontal", "vertical"),
                        ...)
  PlotBstsState(bsts.object, burn = SuggestBurn(.1, bsts.object),
                     time, show.actuals = TRUE,
                     style = c("dynamic", "boxplot"), ...)
  PlotBstsResiduals(bsts.object, burn = SuggestBurn(.1, bsts.object),
                     time, style = c("dynamic", "boxplot"), ...)
  PlotBstsPredictionErrors(bsts.object, burn = SuggestBurn(.1, bsts.object),
                     time, style = c("dynamic", "boxplot"), ...)
  PlotBstsSize(bsts.object, burn = SuggestBurn(.1, bsts.object), style =
                     c("histogram", "ts"), ...)
```

## Arguments

| | |
|---|---|
| x | An object of class bsts. |
| bsts.object | An object of class bsts. |
| y | A character string indicating the aspect of the model that should be plotted. |
| burn | The number of MCMC iterations to discard as burn-in. |
| time | An optional vector of values to plot against. If missing, the default is to diagnose the time scale of the original time series. |

| | |
|---|---|
| same.scale | Logical. If TRUE then all the state components will be plotted with the same scale on the vertical axis. If FALSE then each component will get its own scale for the vertical axis. |
| show.actuals | Logical. If TRUE then actual values from the fitted series will be shown on the plot. |
| style | The desired plot style. Partial matching is allowed, so "dyn" would match "dynamic", for example. |
| layout | For controlling the layout of functions that generate mutiple plots. |
| inclusion.threshold | A inclusion probability that individual coefficients must exceed in order to be displayed when what == "coefficients". See the help file for plot.lm.spike. |
| number.of.variables | If non-NULL this specifies the number of coefficients to plot, taking precedence over inclusion.threshold. See plot.lm.spike. |
| ylim | Limits for the vertical axis. If NULL these will be inferred from the state components and the same.scale argument. Otherwise all plots will be created with the same ylim values. |
| ... | Additional arguments to be passed to PlotDynamicDistribution |

## Details

PlotBstsState, PlotBstsComponents, and PlotBstsResiduals all produce dynamic distribution plots. PlotBstsState plots the aggregate state contribution (including regression effects) to the mean, while PlotBstsComponents plots the contribution of each state component. PlotBstsResiduals plots the posterior distribution of the residuals given complete data (i.e. looking forward and backward in time). PlotBstsPredictionErrors plots filtering errors (i.e. the one-step-ahead prediction errors given data up to the previous time point). PlotBstsForecastDistribution plots the one-step-ahead forecasts instead of the prediction errors.

PlotBstsCoefficients creates a significance plot for the predictors used in the state space regression model. It is obviously not useful for models with no regressors.

PlotBstsSize plots the distribution of the number of predictors included in the model.

## Value

These functions are called for their side effect, which is to produce a plot on the current graphics device.

## See Also

bsts PlotDynamicDistribution plot.lm.spike

## Examples

```
data(AirPassengers)
y <- log(AirPassengers)
ss <- AddLocalLinearTrend(list(), y)
```

```
ss <- AddSeasonal(ss, y, nseasons = 12)
model <- bsts(y, state.specification = ss, niter = 500)
plot(model, burn = 100)
plot(model, "residuals", burn = 100)
plot(model, "components", burn = 100)
plot(model, "forecast.distribution", burn = 100)
```

---

`plot.bsts.mixed` *Plotting functions for mixed frequency Bayesian structural time series*

---

## Description

Functions for plotting the output of a mixed frequency time series regression.

## Usage

```
  ## S3 method for class 'bsts.mixed'
plot(x,
                           y = c("state", "components",
                                 "coefficients", "predictors", "size"),
                           ...)

   PlotBstsMixedState(bsts.mixed.object,
                      burn = SuggestBurn(.1, bsts.mixed.object),
                      time = NULL,
                      fine.scale = FALSE,
                      style = c("dynamic", "boxplot"),
                      trim.left = NULL,
                      trim.right = NULL,
                      ...)

   PlotBstsMixedComponents(bsts.mixed.object,
                           burn = SuggestBurn(.1, bsts.mixed.object),
                           time = NULL,
                           same.scale = TRUE,
                           fine.scale = FALSE,
                           style = c("dynamic", "boxplot"),
                           layout = c("square", "horizontal", "vertical"),
                           ylim = NULL,
                           trim.left = NULL,
                           trim.right = NULL,
                           ...)
```

## Arguments

| | |
|---|---|
| `x` | An object of class `bsts.mixed`. |
| `bsts.mixed.object` | An object of class `bsts.mixed`. |

| | |
|---|---|
| y | A character string indicating the aspect of the model that should be plotted. |
| burn | The number of MCMC iterations to discard as burn-in. |
| time | An optional vector of values to plot against. If missing, the default is to obtain the time scale from the original time series. |
| fine.scale | Logical. If TRUE then the plots will be at the weekly level of granularity. If FALSE they will be at the monthly level. |
| same.scale | Logical. If TRUE then all the state components will be plotted with the same scale on the vertical axis. If FALSE then each component will get its own scale for the vertical axis. |
| style | character. If "dynamic" then a dynamic distribution plot will be shown. If "box" then boxplots will be shown. |
| layout | A character string indicating whether the plots showing components of state should be laid out in a square, horizontally, or vertically. |
| trim.left | A logical indicating whether the first (presumedly partial) observation in the aggregated state time series should be removed. |
| trim.right | A logical indicating whether the last (presumedly partial) observation in the aggregated state time series should be removed. |
| ylim | Limits for the vertical axis. Optional. |
| ... | Additional arguments to be passed to PlotDynamicDistribution or TimeSeriesBoxplot |

## Details

PlotBstsMixedState plots the aggregate state contribution (including regression effects) to the mean, while PlotBstsComponents plots the contribution of each state component separately. PlotBstsCoefficients creates a significance plot for the predictors used in the state space regression model.

## Value

These functions are called for their side effect, which is to produce a plot on the current graphics device.

## See Also

bsts.mixed PlotDynamicDistribution plot.lm.spike PlotBstsSize

## Examples

```
## Not run:
## This example is flaky and needs to be fixed
  data <- SimulateFakeMixedFrequencyData(nweeks = 104, xdim = 20)
  state.specification <- AddLocalLinearTrend(list(), data$coarse.target)
  weeks <- index(data$predictor)
  months <- index(data$coarse.target)
  which.month <- MatchWeekToMonth(weeks, months[1])
  membership.fraction <- GetFractionOfDaysInInitialMonth(weeks)
  contains.end <- WeekEndsMonth(weeks)
```

```
  model <- bsts.mixed(target.series = data$coarse.target,
                      predictors = data$predictors,
                      membership.fraction = membership.fraction,
                      contains.end = contains.end,
                      which.coarse = which.month,
                      state.specification = state.specification,
                      niter = 500)

  plot(model, "state")
  plot(model, "components")

## End(Not run)
```

---

`plot.bsts.prediction` *Plot predictions from Bayesian structural time series*

---

## Description

Plot the posterior predictive distribution from a `bsts` prediction object.

## Usage

```
  ## S3 method for class 'bsts.prediction'
plot(x,
                                  y = NULL,
                                  burn = 0,
                                  plot.original = TRUE,
                                  median.color = "blue",
                                  median.type = 1,
                                  median.width = 3,
                                  interval.quantiles = c(.025, .975),
                                  interval.color = "green",
                                  interval.type = 2,
                                  interval.width = 2,
                                  style = c("dynamic", "boxplot"),
                                  ylim = NULL,
                                  ...)
```

## Arguments

| | |
|---|---|
| `x` | An object of class `bsts.prediction` created by calling `predict` on a `bsts` object. |
| `y` | A dummy argument necessary to match the signature of the `plot` generic function. This argument is unused. |
| `plot.original` | Logical. If `TRUE` then the prediction is plotted after a time series plot of the original series. Otherwise, the prediction fills the entire plot. |

| | |
|---|---|
| burn | The number of observations you wish to discard as burn-in from the posterior predictive distribution. This is in addition to the burn-in discarded using predict.bsts. |
| median.color | The color to use for the posterior median of the prediction. |
| median.type | The type of line (lty) to use for the posterior median of the prediction. |
| median.width | The width of line (lwd) to use for the posterior median of the prediction. |
| interval.quantiles | The lower and upper limits of the credible interval to be plotted. |
| interval.color | The color to use for the upper and lower limits of the 95% credible interval for the prediction. |
| interval.type | The type of line (lty) to use for the upper and lower limits of the 95% credible inerval for of the prediction. |
| interval.width | The width of line (lwd) to use for the upper and lower limits of the 95% credible inerval for of the prediction. |
| style | Either "dynamic", for dynamic distribution plots, or "boxplot", for box plots. Partial matching is allowed, so "dyn" or "box" would work, for example. |
| ylim | Limits on the vertical axis. |
| ... | Extra arguments to be passed to PlotDynamicDistribution and lines. |

## Details

Plots the posterior predictive distribution described by x using a dynamic distribution plot generated by PlotDynamicDistribution. Overlays the posterior median and 95% prediction limits for the predictive distribution.

## Value

Returns NULL.

## See Also

bsts PlotDynamicDistribution plot.lm.spike

## Examples

```
data(AirPassengers)
y <- log(AirPassengers)
ss <- AddLocalLinearTrend(list(), y)
ss <- AddSeasonal(ss, y, nseasons = 12)
model <- bsts(y, state.specification = ss, niter = 500)
pred <- predict(model, horizon = 12, burn = 100)
plot(pred)
```

`plot.bsts.predictors` *Plot the most likely predictors*

## Description

Creates a time series plot showing the most likely predictors of a time series used to fit a `bsts` object.

## Usage

```
PlotBstsPredictors(bsts.object,
                   burn = SuggestBurn(.1, bsts.object),
                   inclusion.threshold = .1,
                   ylim = NULL,
                   flip.signs = TRUE,
                   show.legend = TRUE,
                   grayscale = TRUE,
                   short.names = TRUE,
                   ...)
```

## Arguments

| | |
|---|---|
| `bsts.object` | An object of class `bsts`. |
| `burn` | The number of observations you wish to discard as burn-in. |
| `inclusion.threshold` | Plot predictors with marginal inclusion probabilities above this threshold. |
| `ylim` | Scale for the vertical axis. |
| `flip.signs` | If true then a predictor with a negative sign will be flipped before being plotted, to better align visually with the target series. |
| `show.legend` | Should a legend be shown indicating which predictors are plotted? |
| `grayscale` | Logical. If `TRUE` then lines for different predictors grow progressively lighter as their inclusion probability decreases. If `FALSE` then lines are drawn in black. |
| `short.names` | Logical. If `TRUE` then a common prefix or suffix shared by all the variables will be discarded. |
| `...` | Extra arguments to be passed to `plot`. |

## See Also

`bsts` `PlotDynamicDistribution` `plot.lm.spike`

## Examples

```
data(AirPassengers)
y <- log(AirPassengers)
lag.y <- c(NA, head(y, -1))
ss <- AddLocalLinearTrend(list(), y)
ss <- AddSeasonal(ss, y, nseasons = 12)
## Call bsts with na.action = na.omit to omit the leading NA in lag.y
model <- bsts(y ~ lag.y, state.specification = ss, niter = 500,
              na.action = na.omit)
plot(model, "predictors")
```

---

`plot.holidays` *Plot bsts holidays*

---

## Description

Makes a series of side-by-side boxplots for all the holiday state components in model.

## Usage

```
PlotHolidays(model, ylim = NULL, same.scale = TRUE, ...)
```

## Arguments

| | |
|---|---|
| `model` | A model fit by `bsts` containing one or more holiday state components. |
| `ylim` | Limits on the vertical axis of the plots. If `ylim` is specified, all plots will have the same vertical axis. |
| `same.scale` | If ylim is `NULL`, this flag determines whether all plots share the same scale for there vertical axis (`same.scale == TRUE`), or each plot is independently scaled (`same.scale == FALSE`). |
| `...` | Extra arguments passed to `boxplot`. |

## Value

Returns `invisible{NULL}`.

## See Also

`bsts` `AddNamedHolidays` `AddFixedDateHoliday` `AddNthWeekdayInMonthHoliday`

## Examples

```
## TODO(stevescott):  add examples
```

plot.seasonal.effect *Plot bsts seasonal effects*

## Description

Plot the seasonal effect from a bsts model.

## Usage

```
PlotSeasonalEffect(model, nseasons = 7, season.duration = 1,
                   same.scale = TRUE, ylim = NULL, getname = NULL, ...)
```

## Arguments

| | |
|---|---|
| model | A model fit by bsts, containing a seasonal component. |
| nseasons | If there is only one seasonal component in the model, this argument is ignored. If there are multiple seasonal components then nseasons and season.duration are used to select the desired one. |
| season.duration | If there is only one seasonal component in the model, this argument is ignored. If there are multiple seasonal components then nseasons and season.duration are used to select the desired one. |
| same.scale | A logical indicating how to scale the vertical axis if ylim is unspecified. If TRUE then all panels will have the same scale. If FALSE then each panel is scaled independently. If ylim is specified then this argument is ignored. |
| ylim | A length 2 vector giving the limits of the vertical axis. |
| getname | A function taking a single POSIXt, Date, or similar object as an argument, and returning a single string that can be used as a panel title. |
| ... | Extra arguments passed to plot.dynamic.distribution. |

## Value

Returns invisible{NULL}.

## See Also

bsts PlotDynamicDistribution

## Examples

```
data(AirPassengers)
y <- log(AirPassengers)
ss <- AddLocalLinearTrend(list(), y)
ss <- AddSeasonal(ss, y, nseasons = 12)
model <- bsts(y, state.specification = ss, niter = 500)
PlotSeasonalEffect(model)
```

PlotDynamicDistribution

*Plots the evolution of a distribution over time.*

## Description

Produces an dynamic distribution plot where gray scale shading is used to show the evolution of a distribution over time. This function is useful when there are too many time points to do side-by-side boxplots.

## Usage

```
PlotDynamicDistribution(curves, time = 1:ncol(curves), quantile.step=.01,
                        xlim = range(time), xlab = "time",
                        ylim = range(curves), ylab = "distribution",
                        add=FALSE, ...)
```

## Arguments

| | |
|---|---|
| curves | A matrix where each row represents a curve (e.g. a simulation of a time series from a posterior distribution) and columns represent time. A long time series would be a wide matrix. |
| time | An optional vector of time points that 'curves' will be plotted against. Good choices for time are numeric, or Date (see as.Date). |
| quantile.step | Each color step in the plot corresponds to this difference in quantiles.. Smaller values make prettier plots, but the plots take longer to produce. |
| xlim | The x limits (x1, x2) of the plot. Note that x1 > x2 is allowed and leads to a "reversed axis". |
| xlab | Label for the horzontal axis. |
| ylim | The y limits (y1, y2) of the plot. Note that y1 > y2 is allowed and leads to a "reversed axis". |
| ylab | Label for the vertical axis. |
| add | Logical. If true then add the plot to the current plot. Otherwise a fresh plot will be created. |
| ... | Extra arguments to pass on to plot |

## Details

The function works by passing many calls to polygon. Each polygon is associated with a quantile level, with darker shading near the median.

## Value

This function is called for its side effect, which is to produce a plot on the current graphics device.

## Author(s)

Steven L. Scott <stevescott@google.com>

## Examples

```
x <- t(matrix(rnorm(1000 * 100, 1:100, 1:100), nrow=100))
## x has 1000 rows, and 100 columns.  Column i is N(i, i^2) noise.

PlotDynamicDistribution(x)
time <- as.Date("2010-01-01", format = "%Y-%m-%d") + (0:99 - 50)*7
PlotDynamicDistribution(x, time)
```

---

`predict.bsts` *Prediction for bayesian structural time series*

---

## Description

Generated draws from the posterior predictive distribution of a `bsts` object.

## Usage

```
## S3 method for class 'bsts'
predict(object,
        newdata,
        horizon = 1,
        burn = SuggestBurn(.1, object),
        na.action = na.exclude,
        olddata,
        quantiles = c(.025, .975),
        ...)
```

## Arguments

| | |
|---|---|
| `object` | An object of class `bsts` created by a call to the function `bsts`. |
| `newdata` | a vector, matrix, or data frame containing the predictor variables to use in making the prediction. This is only required if `object` contains a regression compoent. If a data frame, it must include variables with the same names as the data used to fit `object`. The first observation in `newdata` is assumed to be one time unit after the end of the last observation used in fitting `object`, and the subsequent observations are sequential time points. If the regression part of `object` contains only a single predictor then `newdata` can be a vector. If `newdata` is passed as a matrix it is the caller's responsibility to ensure that it contains the correct number of columns and that the columns correspond to those in `object$coefficients`. |

| | |
|---|---|
| horizon | An integer specifying the number of periods into the future you wish to predict. If object contains a regression component then the forecast horizon is nrow(X), and this argument is not used. |
| burn | An integer describing the number of MCMC iterations in object to be discarded as burn-in. If burn <= 0 then no burn-in period will be discarded. |
| na.action | A function determining what should be done with missing values in newdata. |
| olddata | This is an optional component allowing predictions to be made conditional on data other than the data used to fit the model. If omitted, then it is assumed that forecasts are to be made relative to the final observation in the training data. If olddata is supplied then it will be filtered to get the distribution of the next state before a prediction is made, and it is assumed that the first entry in newdata comes immediately after the last entry in olddata.<br>The value for olddata depends on whether or not object contains a regression component.<br>• If a regression component is present, then olddata is a data.frame including variables with the same names as the data used to fit object, including the response .<br>• If no regression component is present, then olddata is a vector containing historical values of a time series. |
| quantiles | A numeric vector of length 2 giving the lower and upper quantiles to use for the forecast interval estimate. |
| ... | This is a dummy argument included to match the signature of the generic predict function. It is not used. |

## Details

Samples from the posterior distribution of a Bayesian structural time series model. This function can be used either with or without contemporaneous predictor variables (in a time series regression).

If predictor variables are present, the regression coefficients are fixed (as opposed to time varying, though time varying coefficients might be added as state component). The predictors and response in the formula are contemporaneous, so if you want lags and differences you need to put them in the predictor matrix yourself.

If no predictor variables are used, then the model is an ordinary state space time series model.

## Value

Returns an object of class bsts.prediction, which is a list with the following components.

| | |
|---|---|
| mean | A vector giving the posterior mean of the prediction. |
| interval | A two (column/row?) matrix giving the upper and lower bounds of the 95 percent credible interval for the prediction. |
| distribution | A matrix of draws from the posterior predictive distribution. Each row in the matrix is one MCMC draw. Columns represent time. |

## Author(s)

Steven L. Scott <stevescott@google.com>

## References

Harvey (1990), "Forecasting, structural time series, and the Kalman filter", Cambridge University Press.

Durbin and Koopman (2001), "Time series analysis by state space methods", Oxford University Press.

## See Also

bsts. AddLocalLevel. AddLocalLinearTrend. AddGeneralizedLocalLinearTrend.

## Examples

```
data(AirPassengers)
y <- log(AirPassengers)
ss <- AddLocalLinearTrend(list(), y)
ss <- AddSeasonal(ss, y, nseasons = 12)
model <- bsts(y, state.specification = ss, niter = 500)
pred <- predict(model, horizon = 12, burn = 100)
plot(pred)
```

---

# quarter *Find the quarter in which a date occurs*

---

## Description

Returns the quarter and year in which a date occurs.

## Usage

```
Quarter(date)
```

## Arguments

| | |
|---|---|
| date | A vector convertible to POSIXlt. A Date or character is fine. |

## Value

A numeric vector identifying the quarter that each element of date corresponds to, expressed as a number of years since 1900. Thus Q1-2000 is 100.00, and Q3-2007 is 107.50.

## Author(s)

Steven L. Scott <stevescott@google.com>

## Examples

```
Quarter(c("2008-02-29", "2008-04-29"))
# [1] 108.00 108.25
```

## rsxfs *Retail sales, excluding food services*

### Description

A monthly time series of retail sales in the US, excluding food services. In millions of dollars. Seasonally adjusted.

### Usage

```
data(rsxfs)
```

### Format

zoo time series

### Source

FRED. See http://research.stlouisfed.org/fred2/series/RSXFS

### Examples

```
data(rsxfs)
plot(rsxfs)
```

## shorten *Shorten long names*

### Description

Removes common prefixes and suffixes from character vectors.

### Usage

```
Shorten(words)
```

### Arguments

| | |
|---|---|
| words | A character vector to be shortened. |

### Value

The argument words is returned, after common prefixes and suffixes have been removed. If all arguments are identical then no shortening is done.

### Author(s)

Steven L. Scott <stevescott@google.com>

### See Also

bsts.mixed.

### Examples

```
Shorten(c("/usr/common/foo.tex", "/usr/common/barbarian.tex"))
# returns c("foo", "barbarian")

Shorten(c("hello", "hellobye"))
# returns c("", "bye")

Shorten(c("hello", "hello"))
# returns c("hello", "hello")

Shorten(c("", "x", "xx"))
# returns c("", "x", "xx")

Shorten("abcde")
# returns "abcde"
```

---

## simulate.fake.mixed.frequency.data

*Simulate fake mixed frequency data*

---

### Description

Simulate a fake data set that can be used to test mixed frequency code.

### Usage

```
SimulateFakeMixedFrequencyData(nweeks,
                               xdim,
                               number.nonzero = xdim,
                               start.date = as.Date("2009-01-03"),
                               sigma.obs = 1.0,
                               sigma.slope = .5,
                               sigma.level = .5,
                               beta.sd = 10)
```

## Arguments

| | |
|---|---|
| nweeks | The number of weeks of data to simulate. |
| xdim | The dimension of the predictor variables to be simulated. |
| number.nonzero | The number nonzero coefficients. Must be less than or equal to xdim. |
| start.date | The date of the first simulated week. |
| sigma.obs | The residual standard deviation for the fine time scale model. |
| sigma.slope | The standard deviation of the slope component of the local linear trend model for the fine time scale data. |
| sigma.level | The standard deviation of the level component fo the local linear trend model for the fine time scale data. |
| beta.sd | The standard deviation of the regression coefficients to be simulated. |

## Details

The simulation begins by simulating a local linear trend model for nweeks to get the trend component.

Next a nweeks by xdim matrix of predictor variables is simulated as IID normal(0, 1) deviates, and a xdim-vector of regression coefficients is simulated as IID normal(0, beta.sd). The product of the predictor matrix and regression coefficients is added to the output of the local linear trend model to get fine.target.

Finally, fine.target is aggregated to the month level to get coarse.target.

## Value

Returns a list with the following components

| | |
|---|---|
| coarse.target | A zoo time series containing the monthly values to be modeled. |
| fine.target | A zoo time series containing the weekly observations that aggregate to coarse.target. |
| predictors | A zoo matrix corresponding to fine.target containing the set of predictors variables to use in bsts.mixed prediction. |
| true.beta | The vector of "true" regression coefficients used to simulate fine.target. |
| true.sigma.obs | The residual standard deviation that was used to simulate fine.target. |
| true.sigma.slope | The value of sigma.slope used to simulate fine.target. |
| true.sigma.level | The value of sigma.level use to simulate fine.target. |
| true.trend | The combined contribution of the simulated latent state on fine.target, including regression effects. |
| true.state | A matrix containin the fine-scale state of the model being simulated. Columns represent time (weeks). Rows correspond to regression (a constant 1), the local linear trend level, the local linear trend slope, the values of fine.target, and the weekly partial aggregates of coarse.target. |

## Author(s)

Steven L. Scott <stevescott@google.com>

## References

Harvey (1990), "Forecasting, structural time series, and the Kalman filter", Cambridge University Press.

Durbin and Koopman (2001), "Time series analysis by state space methods", Oxford University Press.

## See Also

bsts.mixed, AddLocalLinearTrend,

## Examples

```
fake.data <- SimulateFakeMixedFrequencyData(nweeks = 100, xdim = 10)
plot(fake.data$coarse.target)
```

---

# state.sizes *Compute state dimensions*

---

## Description

Returns a vector containing the size of each state component (i.e. the state dimension) in the state vector.

## Usage

```
StateSizes(state.specification)
```

## Arguments

| | |
|---|---|
| state.specification | A list containing state specification components, such as would be passed to bsts. |

## Value

A numeric vector giving the dimension of each state component.

## Author(s)

Steven L. Scott <stevescott@google.com>

## Examples

```
y <- rnorm(1000)
state.specification <- AddLocalLinearTrend(list(), y)
state.specification <- AddSeasonal(state.specification, y, 7)
StateSizes(state.specification)
```

---

`StateSpecification` *Add a state component to a Bayesian structural time series model*

---

## Description

Add a state component to the `state.specification` argument in a `bsts` model.

## Author(s)

Steven L. Scott <stevescott@google.com>

## References

Harvey (1990), "Forecasting, structural time series, and the Kalman filter", Cambridge University Press.

Durbin and Koopman (2001), "Time series analysis by state space methods", Oxford University Press.

## See Also

`bsts`. `SdPrior` `NormalPrior` `Ar1CoefficientPrior`

## Examples

```
data(AirPassengers)
y <- log(AirPassengers)
ss <- AddLocalLinearTrend(list(), y)
ss <- AddSeasonal(ss, y, nseasons = 12)
model <- bsts(y, state.specification = ss, niter = 500)
pred <- predict(model, horizon = 12, burn = 100)
plot(pred)
```

## SuggestBurn *Suggested burn-in size*

### Description

Suggest the size of an MCMC burn in sample as a proportion of the total run.

### Usage

```
SuggestBurn(proportion, bsts.object)
```

### Arguments

| | |
|---|---|
| proportion | The proportion of the MCMC run to discard as burn in. |
| bsts.object | An object of class bsts. |

### Value

An integer number of iterations to discard.

### See Also

bsts

## summary.bsts *Summarize a Bayesian structural time series object*

### Description

Print a summary of a bsts object.

### Usage

```
  ## S3 method for class 'bsts'
summary(object, burn = SuggestBurn(.1, object), ...)
```

### Arguments

| | |
|---|---|
| object | An object of class bsts created by the function of the same name. |
| burn | The number of MCMC iterations to discard as burn-in. |
| ... | Additional arguments passed to summary.lm.spike if object has a regression component. |

## Value

Returns a list with the following elements.

`residual.sd` The posterior mean of the residual standard deviation parameter.

`prediction.sd` The standard deviation of the one-step-ahead prediction errors for the training data.

`rsquare` Proportion by which the residual variance is less than the variance of the original observations.

`relative.gof` Harvey's goodness of fit statistic. Let $\nu$ denote the one step ahead prediction errors, $n$ denote the length of the series, and $y$ denote the original series. The goodness of fit statistic is

$$1 - \sum_{i=1}^{n} \nu_i^2 / \sum_{i=2} n(\Delta y_i - \Delta \bar{y})^2$$

.

This statistic is analogous to $R^2$ in a regression model, but the baseline model is a random walk with drift, instead of the mean of the data. Unlike a traditional R-square statistic, this can be negative.

`size` Distribution of the number of nonzero coefficients appearing in the model

`coefficients` If `object` contains a regression component then the output contains matrix with rows corresponding to coefficients, and columns corresponding to:

- The posterior probability the variable is included.
- The posterior probability that the variable is positive.
- The conditional expectation of the coefficient, given inclusion.
- The conditional standard deviation of the coefficient, given inclusion.

## References

Harvey's goodness of fit statistic is from Harvey (1989) *Forecasting, structural time series models, and the Kalman filter.* Page 268.

## See Also

`bsts`, `plot.bsts`, `summary.lm.spike`

## Examples

```
data(AirPassengers)
y <- log(AirPassengers)
ss <- AddLocalLinearTrend(list(), y)
ss <- AddSeasonal(ss, y, nseasons = 12)
model <- bsts(y, state.specification = ss, niter = 100)
summary(model, burn  = 20)
```

TimeSeriesBoxplot *A time series of boxplots*

## Description

Creates a series of boxplots showing the evolution of a distribution over time.

## Usage

```
TimeSeriesBoxplot(x, time, ylim = NULL, add = FALSE, ...)
```

## Arguments

| | |
|---|---|
| x | A matrix where each row represents a curve (e.g. a simulation of a time series from a posterior distribution) and columns represent time. A long time series would be a wide matrix. |
| time | A vector of class Date with lenght matching the number of columns in x. |
| ylim | limits for the y axis. |
| add | logical, if TRUE then add boxplots to current plot. |
| ... | Extra arguments to pass on to boxplot |

## Value

This function is called for its side effect, which is to produce a plot on the current graphics device.

## Author(s)

Steven L. Scott <stevescott@google.com>

## Examples

```
x <- t(matrix(rnorm(1000 * 100, 1:100, 1:100), nrow=100))
## x has 1000 rows, and 100 columns.  Column i is N(i, i^2) noise.
time <- as.Date("2010-01-01", format = "%Y-%m-%d") + (0:99 - 50)*7
TimeSeriesBoxplot(x, time)
```

week.ends *Check to see if a week contains the end of a month or quarter*

## Description

Returns a logical vector indicating whether the given week contains the end of a month or quarter.

## Usage

```
WeekEndsMonth(week.ending)
WeekEndsQuarter(week.ending)
```

## Arguments

| | |
|---|---|
| week.ending | A vector of class Date. Each entry contains the date of the last day in a week. |

## Value

A logical vector indicating whether the given week contains the end of a month or a quarter.

## Author(s)

Steven L. Scott <stevescott@google.com>

## See Also

bsts.mixed.

## Examples

```
week.ending <- as.Date(c("2011-10-01",
                         "2011-10-08",
                         "2011-12-03",
                         "2011-12-31"))
WeekEndsMonth(week.ending) == c(TRUE, FALSE, TRUE, TRUE)
WeekEndsQuarter(week.ending) == c(TRUE, FALSE, FALSE, TRUE)
```

# Index