Package ‘bsts’

July 2, 2021

Version 0.9.7
Date 2021-06-21
Title Bayesian Structural Time Series
Author Steven L. Scott <steve.the.bayesian@gmail.com>
Maintainer Steven L. Scott <steve.the.bayesian@gmail.com>
Description Time series regression using dynamic linear models fit using
other sources.
Depends BoomSpikeSlab (>= 1.2.4), zoo (>= 1.8), xts, Boom (>= 0.9.7),
R(>= 3.4.0)
Suggests testthat
LinkingTo Boom (>= 0.9.7)
License LGPL-2.1 | file LICENSE
Encoding UTF-8
NeedsCompilation yes
Repository CRAN
Date/Publication 2021-07-02 04:30:07 UTC

R topics documented:

add.ar ................................................................. 3
add.dynamic.regression ........................................... 5
add.local.level .................................................... 8
add.local.linear.trend ............................................ 10
add.monthly.annual.cycle ....................................... 11
add.random.walk.holiday ........................................ 13
add.seasonal ....................................................... 16
add.semiLocal.linear.trend ...................................... 17
add.shared.local.level .......................................... 19
add.static.intercept ............................................ 21
add.student.local.linear.trend ................................ 22
<table>
<thead>
<tr>
<th>R topics documented:</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>add.trig</td>
<td>24</td>
</tr>
<tr>
<td>aggregate.time.series</td>
<td>26</td>
</tr>
<tr>
<td>aggregate.weeks.to.months</td>
<td>28</td>
</tr>
<tr>
<td>auto.ar</td>
<td>29</td>
</tr>
<tr>
<td>bsts</td>
<td>31</td>
</tr>
<tr>
<td>bsts.options.Rd</td>
<td>36</td>
</tr>
<tr>
<td>compare.bsts.models</td>
<td>38</td>
</tr>
<tr>
<td>date.range</td>
<td>39</td>
</tr>
<tr>
<td>descriptive-plots</td>
<td>40</td>
</tr>
<tr>
<td>diagnostic-plots</td>
<td>41</td>
</tr>
<tr>
<td>dirm</td>
<td>42</td>
</tr>
<tr>
<td>dirm-model-optoins</td>
<td>45</td>
</tr>
<tr>
<td>estimate.time.scale</td>
<td>46</td>
</tr>
<tr>
<td>extend.time</td>
<td>47</td>
</tr>
<tr>
<td>format.timestamps</td>
<td>48</td>
</tr>
<tr>
<td>gdp</td>
<td>49</td>
</tr>
<tr>
<td>geometric.sequence</td>
<td>49</td>
</tr>
<tr>
<td>get.fraction</td>
<td>50</td>
</tr>
<tr>
<td>goog</td>
<td>51</td>
</tr>
<tr>
<td>HarveyCumulator</td>
<td>52</td>
</tr>
<tr>
<td>holiday</td>
<td>53</td>
</tr>
<tr>
<td>iclaims</td>
<td>55</td>
</tr>
<tr>
<td>last.day.in.month</td>
<td>56</td>
</tr>
<tr>
<td>MATCH.NumericTimestamps</td>
<td>57</td>
</tr>
<tr>
<td>match.week.to.month</td>
<td>58</td>
</tr>
<tr>
<td>max.window.width</td>
<td>59</td>
</tr>
<tr>
<td>mixed.frequency</td>
<td>60</td>
</tr>
<tr>
<td>month.distance</td>
<td>62</td>
</tr>
<tr>
<td>named.holidays</td>
<td>63</td>
</tr>
<tr>
<td>new.home.sales</td>
<td>64</td>
</tr>
<tr>
<td>one.step.prediction.errors</td>
<td>64</td>
</tr>
<tr>
<td>plot.bsts</td>
<td>66</td>
</tr>
<tr>
<td>plot.bsts.mixed</td>
<td>70</td>
</tr>
<tr>
<td>plot.bsts.prediction</td>
<td>72</td>
</tr>
<tr>
<td>plot.bsts.predictors</td>
<td>74</td>
</tr>
<tr>
<td>plot.holiday</td>
<td>75</td>
</tr>
<tr>
<td>predict.bsts</td>
<td>77</td>
</tr>
<tr>
<td>quarter</td>
<td>80</td>
</tr>
<tr>
<td>regression.holiday</td>
<td>81</td>
</tr>
<tr>
<td>regularize.timestamps</td>
<td>84</td>
</tr>
<tr>
<td>residuals.bsts</td>
<td>85</td>
</tr>
<tr>
<td>rsxfs</td>
<td>86</td>
</tr>
<tr>
<td>shark</td>
<td>87</td>
</tr>
<tr>
<td>shorten</td>
<td>87</td>
</tr>
<tr>
<td>simulate.fake.mixed.frequency.data</td>
<td>88</td>
</tr>
<tr>
<td>spike.slab.ar:prior</td>
<td>90</td>
</tr>
<tr>
<td>state.sizes</td>
<td>91</td>
</tr>
<tr>
<td>StateSpecification</td>
<td>92</td>
</tr>
</tbody>
</table>
Add an AR(p) state component to the state specification.

**Usage**

```r
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

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

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 \texttt{state.specification} with an AR\((p)\) state component added to the end.

Author(s)

Steven L. Scott <steve.the.bayesian@gmail.com>

References

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


See Also

\texttt{bsts, SdPrior}

Examples

```r
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)
```
add.dynamic.regression

Dynamic Regression State Component

Description

Add a dynamic regression component to the state specification of a bsts model. A dynamic regression is a regression model where the coefficients change over time according to a random walk.

Usage

AddDynamicRegression(
  state.specification,
  formula,
  data,
  model.options = NULL,
  sigma.mean.prior.DEPRECATED = NULL,
  shrinkage.parameter.prior.DEPRECATED = GammaPrior(a = 10, b = 1),
  sigma.max.DEPRECATED = NULL,
  contrasts = NULL,
  na.action = na.pass)

DynamicRegressionRandomWalkOptions(
  sigma.prior = NULL,
  sdy = NULL,
  sdx = NULL)

DynamicRegressionHierarchicalRandomWalkOptions(
  sdy = NULL,
  sigma.mean.prior = NULL,
  shrinkage.parameter.prior = GammaPrior(a = 10, b = 1),
  sigma.max = NULL)

DynamicRegressionArOptions(lags = 1, sigma.prior = SdPrior(1, 1))

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 \texttt{as.data.frame} to a data frame) containing the variables in the model. If not found in data, the variables are taken from \texttt{environment(formula)}, typically the environment from which AddDynamicRegression is called.

model.options An object inheriting from \texttt{DynamicRegressionOptions} giving the specific transition model for the dynamic regression coefficients, and the prior distribution for any hyperparameters associated with the transition model.

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

sigma.mean.prior.DEPRECATED This option should be set using model.options. It will be removed in a future release.

shrinkage.parameter.prior An object of class \texttt{GammaPrior} describing the shrinkage parameter, \( a \) (see details below).

shrinkage.parameter.prior.DEPRECATED This option should be set using model.options. It will be removed in a future release.

sigma.max The largest supported value of each \( \sigma[i] \). Truncating the support of \( \sigma \) can keep ill-conditioned models from crashing. This must be a positive number (\( \text{Inf} \) is okay), or \texttt{NULL}. A \texttt{NULL} value will set \( \sigma_{\text{max}} = \text{sd}(y) \), which is a substantially larger value than one would expect, so in well behaved models this constraint will not affect the analysis.

sigma.max.DEPRECATED This option should be set using model.options. It will be removed in a future release.

contrasts An optional list. See the \texttt{contrasts.arg} of \texttt{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 \texttt{na.omit} or \texttt{na.exclude} if you want to omit missing responses altogether.

sdy The standard deviation of the response variable. This is used to scale default priors and \( \sigma_{\text{max}} \) if other arguments are left \texttt{NULL}. If all other arguments are \texttt{non-NULL} then \texttt{sdy} is not used.

sdx The vector of standard deviations of each predictor variable in the dynamic regression. Used only to scale the default prior. This argument is not used if a prior is specified directly.

lags The number of lags in the autoregressive process for the coefficients.

sigma.prior Either an object of class \texttt{SdPrior} or a list of such objects. If a single \texttt{SdPrior} is given then it specifies the prior on the innovation variance for all the coefficients. If a list of \texttt{SdPrior} objects is given, then each element gives the prior distribution for the corresponding regression coefficient. The length of such a list must match the number of predictors in the dynamic regression part of the model.
Details

For the standard "random walk" coefficient model, the model is

\[ \beta_{i,t+1} = \beta_{i,t} + \epsilon_t \quad \epsilon_t \sim N(0, \sigma_i^2 / \text{variance}_{xi}) \]

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

\[ \sqrt{\frac{b}{a}} \sim sigma.mean.prior \]

\( a \sim shrinkage.parameter.prior \)

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.

If the coefficients have AR dynamics, then the model is that each coefficient independently follows an AR(p) process, where p is given by the lags argument. Independent priors are assumed for each coefficient's model, with a uniform prior on AR coefficients (with support restricted to the finite region where the process is stationary), while the sigma.prior argument gives the prior for each coefficient's innovation variance.

Value

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

Author(s)

Steven L. Scott

References

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


See Also

bsts.SdPrior NormalPrior
### 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:

\[ \alpha_{t+1} = \alpha_t + \epsilon_t \quad \epsilon_t \sim N(0, \sigma). \]

The prior is on the \( \sigma \) parameter.

#### Usage

```r
AddLocalLevel(state.specification, 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 \texttt{SdPrior} describing the prior distribution for the standard deviation of the random walk increments.

initial.state.prior
An object created using \texttt{NormalPrior}, describing the prior distribution of 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 <steve.the.bayesian@gmail.com>

References

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


See Also

\texttt{bsts}, \texttt{SdPrior}, \texttt{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

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

The equation for the slope is

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

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

Usage

```r
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.
add.monthly.annual.cycle

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 <steve.the.bayesian@gmail.com>

References

Harvey (1990), "Forecasting, structural time series, and the Kalman filter", Cambridge 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.monthly.annual.cycle

Monthly Annual Cycle State Component

Description

A seasonal state component for daily data, representing the contribution of each month to the annual seasonal cycle. i.e. this is the "January, February, March, ..." effect, with 12 seasons. There is a step change at the start of each month, and then the contribution of that month is constant over the course of the month.

Note that if you have anything other than daily data, then you’re probably looking for AddSeasonal instead.
The state of this model is an 11-vector $\gamma_t$ where the first element is the contribution to the mean for the current month, and the remaining elements are the values for the 10 most recent months. When $t$ is the first day in the month then

$$\gamma_{t+1} = -\sum_{i=2}^{11} \gamma_{t,i} + \epsilon_t$$

\(\epsilon_t \sim \mathcal{N}(0, \sigma)\)

And the remaining elements are $\gamma_t$ shifted down one. When $t$ is any other day then $\gamma_{t+1} = \gamma_t$.

Usage

```r
AddMonthlyAnnualCycle(state.specification, y, date.of.first.observation = NULL, sigma.prior = NULL, initial.state.prior = NULL, sdy)
```

Arguments

- `state.specification`:
  A list of state components, to which the monthly annual cycle will be added. If omitted, an empty list will be assumed.
- `y`:
  The time series to be modeled, as a numeric vector.
- `date.of.first.observation`:
  The time stamp of the first observation in `y`, as a `Date` or `POSIXt` object. If `y` is a `zoo` object with appropriate time stamps then this argument will be deduced.
- `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.

Examples

```r
## Let's simulate some fake daily data with a monthly cycle.
## Not run:
residuals <- rnorm(365 * 5)

n <- length(residuals)
dates <- seq.Date(from = as.Date("2014-01-01"),
  len = n,
  by = 1)
```
add.random.walk.holiday

```
monthly.cycle <- rnorm(12)
monthly.cycle <- monthly.cycle - mean(monthly.cycle)
timestamps <- as.POSIXlt(dates)
month <- timestamps$mon + 1

new.month <- c(TRUE, diff(timestamps$mon) != 0)
month.effect <- cumsum(new.month)
month.effect[month.effect == 0] <- 12

response <- monthly.cycle[month] + residuals
response <- zoo(response, timestamps)

## Now let's fit a bsts model to the daily data with a monthly annual cycle.
ss <- AddLocalLevel(list(), response)
ss <- AddMonthlyAnnualCycle(ss, response)

## In real life you'll probably want more iterations.
model <- bsts(response, state.specification = ss, niter = 200)
plot(model)
plot(model, "monthly")
```

---

**add.random.walk.holiday**

**Random Walk Holiday State Model**

**Description**

Adds a random walk holiday state model to the state specification. This model says

\[ y_t = \alpha_{d(t),t} + \epsilon_t \]

where there is one element in \( \alpha_t \) for each day in the holiday influence window. The transition equation is

\[ \alpha_{d(t+1),t+1} = \alpha_{d(t),t} + \epsilon_{t+1} \]

if \( t+1 \) occurs on day \( d(t+1) \) of the influence window, and

\[ \alpha_{d(t+1),t+1} = \alpha_{d(t+1),t} \]

otherwise.
Usage

AddRandomWalkHoliday(state.specification = NULL,
y, holiday,
time0 = NULL,
sigma.prior = NULL,
initial.state.prior = NULL,
sdy = sd(as.numeric(y), na.rm = TRUE))

Arguments

state.specification A list of state components that you wish augment. If omitted, an empty list will be assumed.
y The time series to be modeled, as a numeric vector convertible to xts. This state model assumes y contains daily data.
holiday An object of class Holiday describing the influence window of the holiday being modeled.
time0 An object convertible to Date 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].
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 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

A list describing the specification of the random walk holiday state model, formatted as expected by the underlying C++ code.

Author(s)

Steven L. Scott <steve.the.bayesian@gmail.com>

References

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

See Also

bsts.RegressionHolidayStateModel HierarchicalRegressionHolidayStateModel
Examples

trend <- cumsum(rnorm(730, 0, .1))
dates <- seq.Date(from = as.Date("2014-01-01"), length = length(trend),
                 by = "day")
y <- zoo(trend + rnorm(length(trend), 0, .2), dates)

AddHolidayEffect <- function(y, dates, effect) {
  ## Adds a holiday effect to simulated data.
  ## Args:
  ##   y: A zoo time series, with Dates for indices.
  ##   dates: The dates of the holidays.
  ##   effect: A vector of holiday effects of odd length. The central effect is
  ##     the main holiday, with a symmetric influence window on either side.
  ## Returns:
  ##   y, with the holiday effects added.
  time <- dates - (length(effect) - 1) / 2
  for (i in 1:length(effect)) {
    y[time] <- y[time] + effect[i]
    time <- time + 1
  }
  return(y)
}

## Define some holidays.
memorial.day <- NamedHoliday("MemorialDay")
memorial.day.effect <- c(.3, 3, .5)
memorial.day.dates <- as.Date(c("2014-05-26", "2015-05-25"))
y <- AddHolidayEffect(y, memorial.day.dates, memorial.day.effect)

presidents.day <- NamedHoliday("PresidentsDay")
presidents.day.effect <- c(.5, 2, .25)
presidents.day.dates <- as.Date(c("2014-02-17", "2015-02-16"))
y <- AddHolidayEffect(y, presidents.day.dates, presidents.day.effect)

labor.day <- NamedHoliday("LaborDay")
labor.day.effect <- c(1, 2, 1)
labor.day.dates <- as.Date(c("2014-09-01", "2015-09-07"))
y <- AddHolidayEffect(y, labor.day.dates, labor.day.effect)

## The holidays can be in any order.
holiday.list <- list(memorial.day, labor.day, presidents.day)
number.of.holidays <- length(holiday.list)

## In a real example you'd want more than 100 MCMC iterations.
niter <- 100
ss <- AddLocalLevel(list(), y)
ss <- AddRandomWalkHoliday(ss, y, memorial.day)
ss <- AddRandomWalkHoliday(ss, y, labor.day)
ss <- AddRandomWalkHoliday(ss, y, presidents.day)
model <- bsts(y, state.specification = ss, niter = niter, seed = 8675309)

## Plot model components.
### 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

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

### Usage

```r
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 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.
add.semilocal.linear.trend

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

Author(s)
Steven L. Scott <steve.the.bayesian@gmail.com>

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

See Also
bsts, SdPrior, NormalPrior

Examples
```r
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.semilocal.linear.trend

Semilocal Linear Trend

Description
The semi-local linear trend model is similar to the local linear trend, but more useful for long-term forecasting. It assumes the level component moves according to a random walk, but the slope component moves according to an AR1 process centered on a potentially nonzero value $D$. The equation for the level is

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

The equation for the slope is

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

This model differs from the local linear trend model in that the latter assumes the slope $\delta_t$ follows a random walk. A stationary AR(1) process is less variable than a random walk when making
projections far into the future, so this model often gives more reasonable uncertainty estimates when making long term forecasts.

The prior distribution for the semi-local linear trend has four independent components. These are:

- 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

```r
AddSemilocalLinearTrend(
  state.specification = list(),
  y = NULL,
  level.sigma.prior = NULL,
  slope.mean.prior = NULL,
  slope.ar1.prior = NULL,
  slope.sigma.prior = NULL,
  initial.level.prior = NULL,
  initial.slope.prior = NULL,
  sdy = NULL,
  initial.y = NULL)
```

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. This can be omitted if sdy and initial.y are supplied, or if all prior distributions are supplied directly.
- **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.
**add.shared.local.level**

**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 <steve.the.bayesian@gmail.com>

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

**See Also**
`bsts.SdPrior NormalPrior`

---

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

\[
\alpha_{t+1} = \alpha_t + \eta_t \quad \eta_{tj} \sim \mathcal{N}(0, \sigma_j).
\]

The contribution to the mean of the observed series obeys

\[
y_t = B\alpha_t + \epsilon_t.
\]

plus observation error. Identifiability constraints imply that the observation coefficients $B$ form a rectangular lower triangular matrix with diagonal 1.0.
Usage

```
AddSharedLocalLevel(
  state.specification,  # A pre-existing list of state components that you wish to add to. If omitted, an empty list will be assumed.
  response,            # The time series to be modeled. This can either be a matrix with rows as time and columns as series, or it can be a numeric vector. If a vector is passed then timestamps and series.id are required. Otherwise they are unused.
  nfactors,            # The number of latent factors to include in the model. This is the dimension of the state for this model component.
  coefficient.prior = NULL,  # Prior distribution on the observation coefficients. This currently must be a ScaledMatrixNormalPrior. Expect other choices in the future.
  initial.state.prior = NULL,  # An object of class MvnPrior, describing the prior distribution of the initial state vector (at time 1).
  timestamps = NULL,       # If response is in long format (i.e. a vector instead of a matrix) this argument is a vector of the same length indicating the time index to which each element of response belongs.
  series.id = NULL,        # If response is in long format (i.e. a vector instead of a matrix) this argument is a vector of the same length indicating the time series to which each element of response belongs.
  sdy,                    # A vector giving the standard deviation of each series to be modeled. This argument is only necessary if response cannot be supplied directly.
  ...                     # Extra arguments passed to ConditionalZellnerPrior, used to create a default prior for the observation coefficients when coefficient.prior is left as NULL.
)
```

Arguments

- **state.specification**: A pre-existing list of state components that you wish to add to. If omitted, an empty list will be assumed.
- **response**: The time series to be modeled. This can either be a matrix with rows as time and columns as series, or it can be a numeric vector. If a vector is passed then timestamps and series.id are required. Otherwise they are unused.
- **nfactors**: The number of latent factors to include in the model. This is the dimension of the state for this model component.
- **coefficient.prior**: Prior distribution on the observation coefficients. This currently must be a ScaledMatrixNormalPrior. Expect other choices in the future.
- **initial.state.prior**: An object of class MvnPrior, describing the prior distribution of the initial state vector (at time 1).
- **timestamps**: If response is in long format (i.e. a vector instead of a matrix) this argument is a vector of the same length indicating the time index to which each element of response belongs.
- **series.id**: If response is in long format (i.e. a vector instead of a matrix) this argument is a vector of the same length indicating the time series to which each element of response belongs.
- **sdy**: A vector giving the standard deviation of each series to be modeled. This argument is only necessary if response cannot be supplied directly.
- **...**: Extra arguments passed to ConditionalZellnerPrior, used to create a default prior for the observation coefficients when coefficient.prior is left as NULL.

Value

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

Author(s)

Steven L. Scott <steve.the.bayesian@gmail.com>
add.static.intercept

References

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


See Also

bsts, SdPrior, NormalPrior

add.static.intercept  Static Intercept State Component

Description

Adds a static intercept term to a state space model. If the model includes a traditional trend component (e.g. local level, local linear trend, etc) then a separate intercept is not needed (and will probably cause trouble, as it will be confounded with the initial state of the trend model). However, if there is no trend, or the trend is an AR process centered around zero, then adding a static intercept will shift the center to a data-determined value.

Usage

AddStaticIntercept(
  state.specification,
  y,
  initial.state.prior = NormalPrior(y[1], sd(y, na.rm = TRUE))
)

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.

initial.state.prior
  An object created using NormalPrior, describing the prior distribution of the intercept term.

Value

Returns a list with the information required to specify the state component. If initial.state.prior is specified then y is unused.

Author(s)

Steven L. Scott
add.student.local.linear.trend

References

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


See Also

bsts. SdPrior NormalPrior

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

\[ \mu_{t+1} = \mu_t + \delta_t + \epsilon_t \quad \epsilon_t \sim T_{\nu_{\mu}}(0, \sigma_{\mu}). \]

The equation for the slope is

\[ \delta_{t+1} = \delta_t + \eta_t \quad \eta_t \sim T_{\nu_{\delta}}(0, \sigma_{\delta}). \]

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

```r
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 inheriting 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 inheriting 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 <steve.the.bayesian@gmail.com>

References

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

See Also

`bsts`, `SdPrior`, `NormalPrior`

Examples

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

Description

Add a trigonometric seasonal model to a state specification.

Usage

```r
AddTrig(
  state.specification = NULL,
  y, period, frequencies,
  sigma.prior = NULL,
  initial.state.prior = NULL,
  sdy = sd(y, na.rm = TRUE),
  method = c("harmonic", "direct"))
```

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.
- `period`: A positive scalar giving the number of time steps required for the longest cycle to repeat.
- `frequencies`: A vector of positive real numbers giving the number of times each cyclic component repeats in a period. One sine and one cosine term will be added for each frequency.
- `sigma.prior`: An object created by `SdPrior` describing the prior distribution for the standard deviation of the increments for the harmonic coefficients.
- `initial.state.prior`: An object created using `NormalPrior`, describing the prior distribution of 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.

method

The method of including the sinusoids. The "harmonic" method is strongly preferred, with "direct" offered mainly for teaching purposes.

Details

**Harmonic Method:**

Each frequency $\lambda_j = 2\pi j / S$ where S is the period (number of time points in a full cycle) is associated with two time-varying random components: $\gamma_{jt}$, and $\gamma^*_{jt}$. They evolve through time as

$$
\gamma_{j,t+1} = \gamma_{jt} \cos(\lambda_j) + \gamma^*_{j,t} \sin(\lambda_j) + \epsilon_0
$$

$$
\gamma^*_{j,t+1} = \gamma^*[j, t] \cos(\lambda_j) - \gamma_{jt}^* \sin(\lambda_j) + \epsilon_1
$$

where $\epsilon_0$ and $\epsilon_1$ are independent with the same variance. This is the real-valued version of a harmonic function: $\gamma \exp(i\theta)$.

The transition matrix multiplies the function by $\exp(i\lambda_j)$, so that after 't' steps the harmonic's value is $\gamma \exp(i\lambda_j t)$.

The model dynamics allows gamma to drift over time in a random walk.

The state of the model is $(\gamma_{jt}, \gamma^*_{jt})$, for $j = 1, \ldots$ number of frequencies.

The state transition matrix is a block diagonal matrix, where block 'j' is

$$
\begin{pmatrix}
\cos(\lambda_j) & \sin(\lambda_j) \\
-\sin(\lambda_j) & \cos(\lambda_j)
\end{pmatrix}
$$

The error variance matrix is $\sigma^2 * I$. There is a common $\sigma^2$ parameter shared by all frequencies.

The model is full rank, so the state error expander matrix $R_t$ is the identity.

The observation matrix is $(1, 0, 1, 0, \ldots)$, where the 1's pick out the 'real' part of the state contributions.

**Direct Method:**

Under the 'direct' method the trig component adds a collection of sine and cosine terms with randomly varying coefficients to the state model. The coefficients are the states, while the sine and cosine values are part of the "observation matrix".

This state component adds the sum of its terms to the observation equation,

$$
y_t = \sum_j \beta_{jt} \sin(f_j t) + \gamma_{jt} \cos(f_j t)
$$

The evolution equation is that each of the sinusoid coefficients follows a random walk with standard deviation $\sigma_j$.

$$
\beta_{jt} = \beta_{jt-1} + N(0, \sigma^2_{\beta_j})
$$

$$
\gamma_{jt} = \gamma_{j-1} + N(0, \sigma^2_{\gamma_j})
$$

The direct method is generally inferior to the harmonic method. It may be removed in the future.
aggregate.time.series

Value

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

Author(s)

Steven L. Scott <steve.the.bayesian@gmail.com>

References

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

See Also

bsts, SdPrior, MvnPrior

Examples

data(AirPassengers)
y <- log(AirPassengers)
ss <- AddLocalLinearTrend(list(), y)
ss <- AddTrig(ss, y, period = 12, frequencies = 1:3)
model <- bsts(y, state.specification = ss, niter = 200)
plot(model)

## The "harmonic" method is much more stable than the "direct" method.
ss <- AddLocalLinearTrend(list(), y)
ss <- AddTrig(ss, y, period = 12, frequencies = 1:3, method = "direct")
model2 <- bsts(y, state.specification = ss, niter = 200)
plot(model2)

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 <steve.the.bayesian@gmail.com>

Examples

```r

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

\[
\text{AggregateWeeksToMonths(weekly.series,} \\
\quad \text{membership.fraction = NULL,} \\
\quad \text{trim.left = TRUE,} \\
\quad \text{trim.right = NULL)}
\]

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 <steve.the.bayesian@gmail.com>

See Also

`AggregateTimeSeries`
Examples

```r
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)
```

---

**auto.ar**

Sparse AR(p)

**Description**

Add a sparse AR(p) process to the state distribution. A sparse AR(p) is an AR(p) process with a spike and slab prior on the autoregression coefficients.

**Usage**

```r
AddAutoAr(state.specification, 
  y, 
  lags = 1, 
  prior = NULL, 
  sdy = NULL, 
  ...)```

**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. This can be omitted if `sdy` is supplied.

- `lags`  
  The maximum number of lags ("p") to be considered in the AR(p) process.

- `prior`  
  An object inheriting from `SpikeSlabArPrior`, or NULL. If the latter, then a default `SpikeSlabArPrior` will be created.

- `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.

- `...`  
  Extra arguments passed to `SpikeSlabArPrior`. 
Details

The model contributes alpha[t] to the expected value of y[t], where the transition equation is

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

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.

This model differs from the one in `AddAr` only in that some of its coefficients may be set to zero.

Value

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

Author(s)

Steven L. Scott <steve.the.bayesian@gmail.com>

References

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


See Also

`bsts`, `SdPrior`

Examples

```r
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 <- AddAutoAr(list(), y, lags = 6)

# 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$AR6.coefficients), lag.max = 30))
legend("topright", leg = c("empirical", "truth", "MCMC"), pch = c(NA, "+", "o"))

---

**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.

The model allows for several useful extensions beyond standard Bayesian dynamic linear models.

- A spike-and-slab prior is used for the (static) regression component of models that include predictor variables. This is especially useful with large numbers of regressor series.
- Both the spike-and-slab component (for static regressors) and the Kalman filter (for components of time series state) require observations and state variables to be Gaussian. The *bsts* package allows for non-Gaussian error families in the observation equation (as well as some state components) by using data augmentation to express these families as conditionally Gaussian.
- As of version 0.7.0, *bsts* supports having multiple observations at the same time point. In this case the basic model is taken to be

\[ y_{t,j} = Z_T \alpha_t + \beta^T x_{t,j} + \epsilon_{t,j}. \]

This is a regression model where all observations with the same time point share a common time series effect.

**Usage**

```r
bsts(formula, 
    state.specification, 
    family = c("gaussian", "logit", "poisson", "student"), 
    data, 
    prior, 
    contrasts = NULL, 
    na.action = na.pass, 
    niter, 
```
ping = niter / 10,
model.options = BstsOptions(),
timestamps = NULL,
seed = NULL,
...)

Arguments

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. Missing values are not allowed in predictors, but they are allowed in the response variable.
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.

family The model family for the observation equation. Non-Gaussian model families use data augmentation to recover a conditionally Gaussian model.

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.
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 containing the names of contrast functions to use when converting factors numeric variables in a regression formula. This argument works exactly as it does in lm. The names of the list elements correspond to factor variables in your model formula. The list elements themselves are the names of contrast functions (see help(contr.treatment) and the contrasts.arg argument to model.matrix.default). This argument is only used if a model formula is specified, and even then the default is probably 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.

model.options An object (list) returned by BstsOptions. See that function for details.
The timestamp associated with each value of the response. This argument is primarily useful in cases where the response has missing gaps, or where there are multiple observations per time point. If the response is a "regular" time series with a single observation per time point then you can leave this argument as NULL. In that case, if either the response or the data argument is a type convertible to zoo then timestamps will be inferred.

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).

Details

If the model family is logit, then there are two ways one can format the response variable. If the response is 0/1, TRUE/FALSE, or 1/-1, then the response variable can be passed as with any other model family. If the response is a set of counts out of a specified number of trials then it can be passed as a two-column matrix, where the first column contains the counts of successes and the second contains the count of failures.

Likewise, if the model family is Poisson, the response can be passed as a single vector of counts, under the assumption that each observation has unit exposure. If the exposures differ across observations, then the response can be a two column matrix, with the first column containing the event counts and the second containing exposure times.

Value

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

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.

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 <steve.the.bayesian@gmail.com>
References

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


See Also

bsts, AddLocalLevel, AddLocalLinearTrend, AddSemilocalLinearTrend, 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 <- AddSemilocalLinearTrend(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 <- AddSemilocalLinearTrend(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)

\dontrun{
  ## Example 6: Including regressors
data(iclaims)
ss <- AddLocalLinearTrend(list(), initial.claims$iclaimsNSA)
ss <- AddSeasonal(ss, initial.claims$iclaimsNSA, nseasons = 52)
model <- bsts(iclaimsNSA ~ ., state.specification = ss, data =
    initial.claims, niter = 1000)
plot(model)
plot(model, "components")
plot(model, "coefficients")
plot(model, "predictors")
}\dontrun{

## End(Not run)

## Not run:

## Example 7: Regressors with multiple time stamps.
number.of.time.points <- 50
sample.size.per.time.point <- 10
total.sample.size <- number.of.time.points * sample.size.per.time.point
sigma.level <- .1
sigma.obs <- 1

trend <- cumsum(rnorm(number.of.time.points, 0, sigma.level))
predictors <- matrix(rnorm(total.sample.size * 2), ncol = 2)
rownames(predictors) <- c("X1", "X2")
coefficients <- c(-10, 10)
regression <- as.numeric(predictors %*% coefficients)
y.hat <- rep(trend, sample.size.per.time.point) + regression
y <- rnorm(length(y.hat), y.hat, sigma.obs)

## Create some time stamps, with multiple observations per time stamp.
first <- as.POSIXct("2013-03-24")
dates <- seq(from = first, length = number.of.time.points, by = "month")
timestamps <- rep(dates, sample.size.per.time.point)

## Run the model with a local level trend, and an unnecessary seasonal component.
ss <- AddLocalLevel(list(), y)
ss <- AddSeasonal(ss, y, nseasons = 7)
model <- bsts(y ~ predictors, ss, niter = 250, timestamps = timestamps,
             seed = 8675309)
plot(model)
plot(model, "components")

## End(Not run)

## Example 8: Non-Gaussian data
## Poisson counts of shark attacks in Florida.
data(shark)
logshark <- log1p(shark$Attacks)
ss.level <- AddLocalLevel(list(), y = logshark)
model <- bsts(shark$Attacks, ss.level, niter = 500,
            ping = 250, family = "poisson", seed = 8675309)

## Poisson data can have an 'exposure' as the second column of a
## two-column matrix.
model <- bsts(cbind(shark$Attacks, shark$Population / 1000),
            state.specification = ss.level, niter = 500,
            family = "poisson", ping = 250, seed = 8675309)

---

**bsts.options.Rd**

**Bsts Model Options**

**Description**

Rarely used modeling options for bsts models.

**Usage**

```r
BstsOptions(save.state.contributions = TRUE,
            save.prediction.errors = TRUE,
            bma.method = c("SSVS", "ODA"),
            oda.options = list(  
                fallback.probability = 0.0,
                eigenvalue.fudge.factor = 0.01),
            timeout.seconds = Inf,
            save.full.state = FALSE)
```
Arguments

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.

bma.method
If the model contains a regression component, this argument specifies the method to use for Bayesian model averaging. "SSVS" is stochastic search variable selection, which is the classic approach from George and McCulloch (1997). "ODA" is orthogonal data augmentation, from Ghosh and Clyde (2011). It adds a set of latent observations that make the $X^TX$ matrix diagonal, vastly simplifying complete data MCMC for model selection.

oda.options
If bma.method == "ODA" then these are some options for fine tuning the ODA algorithm.

• fallback.probability: Each MCMC iteration will use SSVS instead of ODA with this probability. In cases where the latent data have high leverage, ODA mixing can suffer. Mixing in a few SSVS steps can help keep an errant algorithm on track.

• eigenvalue.fudge.factor: The latent X's will be chosen so that the complete data $X^TX$ matrix (after scaling) is a constant diagonal matrix equal to the largest eigenvalue of the observed (scaled) $X^TX$ times (1 + eigenvalue.fudge.factor). This should be a small positive number.

timeout.seconds
The number of seconds that sampler will be allowed to run. If the timeout is exceeded the returned object will be truncated to the final draw that took place before the timeout occurred, as if that had been the requested number of iterations.

save.full.state
Logical. If TRUE then the full distribution of the state vector will be preserved. It will be stored in the model under the name full.state, which is a 3-way array with dimensions corresponding to MCMC iteration, state dimension, and time.

Value

The arguments are checked to make sure they have legal types and values, then a list is returned containing the arguments.

Author(s)

Steven L. Scott <steve.the.bayesian@gmail.com>
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

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

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` palette 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?
- **cutpoint**: Either NULL, or an integer giving the observation number used to define a holdout sample. Prediction errors occurring after the cutpoint will be true out of sample errors. If NULL then all prediction errors are "in sample". See the discussion in `bsts.prediction.errors`.

Value

Invisibly returns the matrix of cumulative one-step ahead prediction errors (the lines in the top panel of the plot). Each row in the matrix corresponds to a model in model.list.

Author(s)

Steven L. Scott <steve.the.bayesian@gmail.com>
Examples

```r
data(AirPassengers)
y <- log(AirPassengers)
ss <- AddLocalLinearTrend(list(), y)
trend.only <- bsts(y, ss, niter = 250)

ss <- AddSeasonal(ss, y, nseasons = 12)
trend.and.seasonal <- bsts(y, ss, niter = 250)

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

CompareBstsModels(list(trend = trend.only, 
  "trend and seasonal" = trend.and.seasonal), 
  cutpoint = 100)
```

---

**Date Range**

<table>
<thead>
<tr>
<th>date.range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns the first and last dates of the influence window for the given holiday, among the given timestamps.</td>
</tr>
</tbody>
</table>

**Usage**

```r
DateRange(holiday, timestamps)
```

**Arguments**

- `holiday`: An object of class `Holiday`.
- `timestamps`: A vector of timestamps of class `Date` or class `POSIXt`. This function assumes daily data. Use with care in other settings.

**Value**

Returns a two-column data frame giving the first and last dates of the influence window for the holiday in the period covered by timestamps.

**Author(s)**

Steven L. Scott <steve.the.bayesian@gmail.com>
Examples

```r
descriptive-plots

Examples

global <- NamedGlobal("GlobalHoliday", days.before = 2, days.after = 2)
timestamps <- seq.Date(from = as.Date("2001-01-01"), by = "day", 
length.out = 365 * 10)
influence <- DateRange(global, timestamps)
```

Description

Plots for describing time series data.

Usage

```r
DayPlot(y, colors = NULL, ylab = NULL, ...) 
MonthPlot(y, seasonal.identifier = months, colors = NULL, ylab = NULL, ...) 
YearPlot(y, colors = NULL, ylab = NULL, ylim = NULL, legend = TRUE, ...) 
```

Arguments

- **y**: A time series to plot. Must be of class `ts`, or `zoo`. If a zoo object then the timestamps must be of type `Date`, `yearmon`, or `POSIXt`.
- **seasonal.identifier**: A function that takes a vector of class `POSIXt` (date/time) and returns a character vector indicating the season to which each element belongs. Each unique element returned by this function returns a "season" to be plotted. See `weekdays`, `months`, and `quarters` for examples of how this should work.
- **colors**: A vector of colors to use for the lines.
- **legend**: Logical. If TRUE then a legend is added to the plot.
- **ylab**: Label for the vertical axis.
- **ylim**: Limits for the vertical axis. (a 2-vector)
- **...**: Extra arguments passed to `plot` or `lines`.

Details

DayPlot and MonthPlot plot the time series one season at a time, on the same set of axes. The intent is to use DayPlot for daily data and MonthPlot for monthly or quarterly data. YearPlot plots each year of the time series as a separate line on the same set of axes. Both sets of plots help visualize seasonal patterns.

Value

Returns `invisible(NULL)`. 
See Also

`monthplot` is a base R function for plotting time series of type `ts`.

Examples

```r
## Plot a 'ts' time series.
data(AirPassengers)
par(mfrow = c(1,2))
MonthPlot(AirPassengers)
YearPlot(AirPassengers)

## Plot a 'zoo' time series.
data(turkish)
par(mfrow = c(1,2))
YearPlot(turkish)
DayPlot(turkish)
```

---

diagnostic-plots  Diagnostic Plots

diagnostic-plots  Diagnostic Plots

Description

Diagnostic plots for distributions of residuals.

Usage

```r
qqdist(draws, ...)
AcfDist(draws, lag.max = NULL, xlab = "Lag", ylab = "Autocorrelation", ...)
```

Arguments

- `draws`: A matrix of Monte Carlo draws of residual errors. Each row is a Monte Carlo draw, and each column is an observation. In the case of `AcfDist` successive observations are assumed to be sequential in time.
- `lag.max`: The number of lags to plot in the autocorrelation function. See `acf`.
- `xlab`: Label for the horizontal axis.
- `ylab`: Label for the vertical axis.
- `...`: Extra arguments passed to either `boxplot` (for `AcfDist`) or `PlotDynamicDistribution` (for `qqdist`).

Details

`qqdist` sorts the columns of `draws` by their mean, and plots the resulting set of curves against the quantiles of the standard normal distribution. A reference line is added, and the mean of each column of draws is represented by a blue dot. The dots and the line are the transpose of what you get with `qqnorm` and `qqline`.

`AcfDist` plots the posterior distribution of the autocorrelation function using a set of side-by-side boxplots.
Examples

```r
data(AirPassengers)
y <- log(AirPassengers)

ss <- AddLocalLinearTrend(list(), y)
ss <- AddSeasonal(ss, y, nseasons = 12)
model <- bsts(y, ss, niter = 500)

r <- residuals(model)
par(mfrow = c(1,2))
qqdist(r)  ## A bit of departure in the upper tail
AcfDist(r)
```

---

**dirm**  
*Dynamic intercept regression model*

---

**Description**

A dynamic intercept regression is a regression model where the intercept term is a state space model. This model differs from `bsts` in that there can be multiple observations per time point.

**Usage**

```r
dirm(formula,  
     state.specification,  
     data,  
     prior = NULL,  
     contrasts = NULL,  
     na.action = na.pass,  
     niter,  
     ping = niter / 10,  
     model.options = DirmModelOptions(),  
     timestamps = NULL,  
     seed = NULL,  
     ...)```

**Arguments**

- `formula`  
  A formula, as you would supply to `lm` describing the regression portion of the relationship between y and X.

- `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`. The state specification describes the dynamic intercept term in the regression model.
data
An optional data frame, list or environment (or object coercible by \texttt{as.data.frame} to a data frame) containing the variables in the model. If not found in \texttt{data}, the variables are taken from \texttt{environment(formula)}, typically the environment from which \texttt{dirm} is called.

prior
A prior distribution for the regression component of the model, as created by \texttt{SpikeSlabPrior}. The prior for the time series component of the model will be specified during the creation of \texttt{state.specification}.

contrasts
An optional list containing the names of contrast functions to use when converting factors numeric variables in a regression formula. This argument works exactly as it does in \texttt{lm}. The names of the list elements correspond to factor variables in your model formula. The list elements themselves are the names of contrast functions (see \texttt{help(contr.treatment)} and the \texttt{contrasts.arg} argument to \texttt{model.matrix.default}). This argument can usually be omitted.

na.action
What to do about missing values. The default is to allow missing responses, but no missing predictors. Set this to \texttt{na.omit} or \texttt{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 \texttt{ping > 0} then the program will print a status message to the screen every \texttt{ping} MCMC iterations.

model.options
An object created by \texttt{DirmModelOptions} specifying the desired model options.

timestamps
The timestamp associated with each value of the response. This is most likely a \texttt{Date} or \texttt{POSIXt}. It is expected that there will be multiple observations per time point (otherwise \texttt{’bsts’} should be used instead of \texttt{’dirm’}), and thus the \texttt{’timestamps’} argument will contain many duplicate values.

seed
An integer to use as the random seed for the underlying C++ code. If \texttt{NULL} then the seed will be set using the clock.

... Extra arguments to be passed to \texttt{SpikeSlabPrior} (see the entry for the \texttt{prior} argument, above).

Details
The fitted model is a regression model with an intercept term given by a structural time series model. This is similar to the model fit by \texttt{bsts}, but it allows for multiple observations per time period.

Currently \texttt{dirm} only supports Gaussian observation errors, but look for that to change in future releases.

Value
An object of class \texttt{bsts} which is a list with the following components

\begin{description}
\item[coefficients]A \texttt{niter by ncol(X)} matrix of MCMC draws of the regression coefficients, where \texttt{X} is the design matrix implied by \texttt{formula}. This is only present if a model formula was supplied.
\item[sigma.obs]A vector of length \texttt{niter} containing MCMC draws of the residual standard deviation.
\end{description}
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 <steve.the.bayesian@gmail.com>

References

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


See Also

`bsts`, `AddLocalLevel`, `AddLocalLinearTrend`, `AddSemilocalLinearTrend`, `AddSeasonal`, `AddDynamicRegression`, `SpikeSlabPrior`, `SdPrior`.

Examples

```r
SimulateDirmData <- function(observation.sd = 1, trend.sd = .1, time.dimension = 100, nobs.per.period = 3, xdim = 4) {
  trend <- cumsum(rnorm(time.dimension, 0, trend.sd))
  total.sample.size <- nobs.per.period * time.dimension
  predictors <- matrix(rnorm(total.sample.size * xdim), nrow = total.sample.size)
  coefficients <- rnorm(xdim)
  expanded.trend <- rep(trend, each = nobs.per.period)
  response <- expanded.trend + predictors %*% coefficients + rnorm(total.sample.size, 0, observation.sd)
  timestamps <- seq.Date(from = as.Date("2008-01-01"), len = time.dimension, by = "day")
  extended.timestamps <- rep(timestamps, each = nobs.per.period)
  return(list(response = response, predictors = predictors, timestamps = extended.timestamps, trend = trend, coefficients = coefficients))
}
```
data <- SimulateDirmData(time.dimension = 20)
ss <- AddLocalLevel(list(), data$response)

# In real life you'd want more than 50 MCMC iterations.
model <- dirm(data$response ~ data$predictors, ss, niter = 50,
              timestamps = data$timestamps)

---

dirm-model-optoins

Specify Options for a Dynamic Intercept Regression Model

Description

Specify modeling options for a dynamic intercept regression model.

Usage

DirmModelOptions(timeout.seconds = Inf,
                  high.dimensional.threshold.factor = 1.0)

Arguments

timeout.seconds

The overall time budget for model fitting. If the MCMC algorithm takes longer
than this number, the current iteration will complete, and then the fitting algo-
rithm will return with however many MCMC iterations were managed during
the allotted time.

high.dimensional.threshold.factor

When doing Kalman filter updates for the model, Sherman-Morrisson-Woodbury
style updates are applied for high dimensional data, while direct linear algebra is
used for low dimensional data. The definition of "high dimensional" is relative
to the dimension of the state. An observation is considered high dimensional if
its dimension exceeds the state dimension times this factor.

Value

An object of class DirmModelOptions, which is simply a list containing values of the function
arguments.

The value of using this function instead of making a list "by hand" is that argument types are
properly checked, and list names are sure to be correct.
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 <steve.the.bayesian@gmail.com>

Examples


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(almost.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 <steve.the.bayesian@gmail.com>

See Also

bsts.mixed.

Examples

orign.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
Description

Tools for checking if a series of timestamps is 'regular' meaning that it has no duplicates, and no gaps. Checking for regularity can be tricky. For example, if you have monthly observations with `Date` or `POSIXt` timestamps then gaps between timestamps can be 28, 29, 30, or 31 days, but the series is still "regular".

Usage

```
NoDuplicates(timestamps)
NoGaps(timestamps)
IsRegular(timestamps)

HasDuplicateTimestamps(bsts.object)
```

Arguments

- `timestamps`: A set of (possibly irregular or non-unique) timestamps. This could be a set of integers (like 1, 2, 3...), a set of numeric like (1945, 1945.083, 1945.167, ...) indicating years and fractions of years, a `Date` object, or a `POSIXt` object.
- `bsts.object`: A bsts model object.

Value

All four functions return scalar logical values. `NoDuplicates` returns `TRUE` if all elements of `timestamps` are unique.

`NoGaps` examines the smallest nonzero gap between time points. As long as no gaps between time points are more than twice as wide as the smallest gap, it returns `TRUE`, indicating that there are no missing timestamps. Otherwise it returns `FALSE`.

`IsRegular` returns `TRUE` if `NoDuplicates` and `NoGaps` both return `TRUE`.

`HasDuplicateTimestamps` returns `FALSE` if the data used to fit bsts.model either has NULL timestamps, or if the timestamps contain no duplicate values.

Author(s)

Steven L. Scott <steve.the.bayesian@gmail.com>

Examples

```
first <- as.POSIXct("2015-04-19 08:00:04")
monthly <- seq(from = first, length.out = 24, by = "month")
IsRegular(monthly) ## TRUE

skip.one <- monthly[-8]
```
gdp

IsRegular(skip.one) ## FALSE

has.duplicates <- monthly
has.duplicates[1] <- has.duplicates[2]
IsRegular(has.duplicates) ## FALSE

---

gdp

**Gross Domestic Product for 57 Countries**

---

**Description**

Annual gross domestic product for 57 countries, as produced by the OECD.

Fields:

- **LOCATION**: Three letter country code.
- **MEASURE**: MLN_USD signifies a total GDP number in millions of US dollars. USD_CAP is per capita GDP in US dollars.
- **TIME**: The year of the measurement.
- **Value**: The measured value.
- **Flag Codes**: P for provisional data, B for a break in the series, and E for an estimated value.

**Usage**

data(gdp)

**Format**

data frame

**Source**


---

geometric.sequence

**Create a Geometric Sequence**

---

**Description**

Create a geometric sequence.

**Usage**

GeometricSequence(length, initial.value = 1, discount.factor = .5)
Arguments

- `length` A positive integer giving the length of the desired sequence.
- `initial.value` The first term in the sequence. Cannot be zero.
- `discount.factor` The ratio between a sequence term and the preceding term. Cannot be zero.

Value

A numeric vector containing the desired sequence.

Author(s)

Steven L. Scott <steve.the.bayesian@gmail.com>

Examples

```r
GeometricSequence(4, .8, .6)
# [1] 0.8000 0.4800 0.2880 0.1728

GeometricSequence(5, 2, 3)
# [1] 2 6 18 54 162

## Not run:
GeometricSequence(0, -1, -2)
# Error: length > 0 is not TRUE
## End(Not run)
```

get.fraction

Compute membership fractions

Description

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

Usage

```r
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 <steve.the.bayesian@gmail.com>

**See Also**

`bsts.mixed`.

**Examples**

```r
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**

```r
data(goog)
```

**Format**

xts time series

**Source**

The Internets
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 <steve.the.bayesian@gmail.com>

References

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

See Also

bsts.mixed,
Examples

```r
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)
```

---

**holiday**  

**Specifying Holidays**

Description

Specify holidays for use with holiday state models.

Usage

```r
FixedDateHoliday(holiday.name,  
   month = base::month.name,  
   day,  
   days.before = 1,  
   days.after = 1)
```

```r
NthWeekdayInMonthHoliday(holiday.name,  
   month = base::month.name,  
   day.of.week = weekday.names,  
   week.number = 1,  
   days.before = 1,  
   days.after = 1)
```

```r
LastWeekdayInMonthHoliday(holiday.name,  
   month = base::month.name,  
   day.of.week = weekday.names,  
   days.before = 1,  
   days.after = 1)
```
NamedHoliday(holiday.name = named.holidays,
    days.before = 1,
    days.after = 1)

DateRangeHoliday(holiday.name,
    start.date,
    end.date)

Arguments

holiday.name A string that can be used to label the holiday in output.
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 day of the week on which the holiday occurs.
week.number An integer specifying the week of the month on which the NthWeekdayInMonthHoliday occurs.
days.before An integer giving the number of days of influence that the holiday exerts prior to the actual holiday.
days.after An integer giving the number of days of influence that holiday exerts after the actual holiday.
named.holidays A character vector containing one or more recognized holiday names.
start.date A vector of starting dates for the holiday. Each instance of the holiday in the training data or the forecast period must be represented by an element in this vector. Thus if this is an annual holiday and, there are 10 years of training data, and a 1-year forecast is needed, then this will be a vector of length 11.
end.date A vector of ending dates for the holiday. Each date must occur on or after the corresponding element of start.date, and end.date[i] must come before start.date[i+1].

Value

Each function returns a list containing the information from the function arguments, formatted as expected by the underlying C++ code. State models that focus on holidays, such as AddRandomWalkHoliday, AddRegressionHoliday, and AddHierarchicalRegressionHoliday, will expect one or more holiday objects as arguments.

- FixedDateHoliday describes a holiday that occurs on the same date each year, like US independence day (July 4).
- NthWeekdayInMonthHoliday describes a holiday that occurs a particular weekday of a particular week of a particular month. For example, US Labor Day is the first Monday in September.
- LastWeekdayInMonthHoliday describes a holiday that occurs on the last instance of a particular weekday in a particular month. For example, US Memorial Day is the last Monday in May.
• DateRangeHoliday describes an irregular holiday that might not follow a particular pattern. You can handle this type of holiday by manually specifying a range of dates for each instance of the holiday in your data set. NOTE: If you plan on using the model to forecast, be sure to include date ranges in the forecast period as well as the period covered by the training data.

• NamedHoliday is a convenience class for describing several important holidays in the US.

Author(s)

Steven L. Scott <steve.the.bayesian@gmail.com>

See Also

AddRandomWalkHoliday, AddRegressionHoliday, AddHierarchicalRegressionHoliday

Examples

july4 <- FixedDateHoliday("July4", "July", 4)
memorial.day <- LastWeekdayInMonthHoliday("MemorialDay", "May", "Monday")
labor.day <- NthWeekdayInMonthHoliday("LaborDay", "September", "Monday", 1)
another.way.to.get.memorial.day <- NamedHoliday("MemorialDay")
easter <- NamedHoliday("Easter")
winter.olympics <- DateRangeHoliday("WinterOlympicsSince2000",
    start = as.Date(c("2002-02-08",
    "2006-02-10",
    "2010-02-12",
    "2014-02-07",
    "2018-02-07")),
    end = as.Date(c("2002-02-24",
    "2006-02-26",
    "2010-02-28",
    "2014-02-23",
    "2018-02-25")))

iclaims

Initial Claims Data

Description

A weekly time series of US initial claims for unemployment. The first column contains the initial claims numbers from FRED. The others contain a measure of the relative popularity of various search queries identified by Google Correlate.

Usage

data(iclaims)
last.day.in.month

Format

zoo time series

Source

FRED. http://research.stlouisfed.org/fred2/series/ICNSA,

See Also

bsts

Examples

data(iclaims)
plot(initial.claims)

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

Description

Finds the last day in the month containing a specified 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 <steve.the.bayesian@gmail.com>
Examples

```r
LastDayInMonth(inputs) == expected.outputs
```

---

**MATCH.NumericTimestamps**

*Match Numeric Timestamps*

**Description**

S3 generic method for MATCH function supplied in the zoo package.

**Usage**

```r
## S3 method for class 'NumericTimestamps'
MATCH(x, table, nomatch = NA, ...)
```

**Arguments**

- `x`: A numeric set of timestamps.
- `table`: A set of regular numeric timestamps to match against.
- `nomatch`: The value to be returned in the case when no match is found. Note that it is coerced to integer.
- `...`: Additional arguments passed to `match`.

**Details**

Numeric timestamps match if they agree to 8 significant digits.

**Value**

Returns the index of the entry in `table` matched by each argument in `x`. If an entry has no match then `nomatch` is returned at that position.
See Also

MATCH

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 <steve.the.bayesian@gmail.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
Description

The maximum width of a holiday’s influence window

Usage

```r
## Default S3 method:
MaxWindowWidth(holiday, ...)
## S3 method for class 'DateRangeHoliday'
MaxWindowWidth(holiday, ...)
```

Arguments

- `holiday`: An object of class `Holiday`.
- `...`: Other arguments (not used).

Value

Returns the number of days in a holiday’s influence window.

Author(s)

Steven L. Scott <steve.the.bayesian@gmail.com>

See Also

`Holiday. AddRegressionHoliday. AddRandomWalkHoliday. AddHierarchicalRegressionHoliday`

Examples

```r
easter <- NamedHoliday("Easter", days.before = 2, days.after = 1)
if (MaxWindowWidth(easter) == 4) {
    print("That's the right answer!
")
}

## This holiday lasts two days longer in 2005 than in 2004.
may18 <- DateRangeHoliday("May18",
    start = as.Date(c("2004-05-17", "2005-05-16")),
    end = as.Date(c("2004-05-19", "2005-05-20")))

if (MaxWindowWidth(may18) == 5) {
    print("Right again!
")
}
```
mixed.frequency

Models for mixed frequency time series

Description

Fit a structured time series to mixed frequency 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 encompassing 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.
mixed.frequency

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 <steve.the.bayesian@gmail.com>

References

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

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 <steve.the.bayesian@gmail.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)

---

Some text is highlighted:

**named.holidays**

*Holidays Recognized by Name*

Description

A character vector listing the names of pre-specified holidays.

Usage

named.holidays

Value

"NewYearsDay" "SuperBowlSunday" "MartinLutherKingDay" "PresidentsDay" "ValentinesDay"
"SaintPatricksDay" "USDaylightSavingsTimeBegins" "USDaylightSavingsTimeEnds" "EasterSunday"
"USMothersDay" "IndependenceDay" "LaborDay" "ColumbusDay" "Halloween" "Thanksgiving"
"MemorialDay" "VeteransDay" "Christmas"
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

one.step.prediction.errors

Prediction Errors

Description

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

Usage

bsts.prediction.errors(bsts.object,
cutpoints = NULL,
burn = SuggestBurn(.1, bsts.object),
standardize = FALSE)
Arguments

bsts.object An object of class bsts.
cutpoints An increasing sequence of integers between 1 and the number of time points in the training data for bsts.object, or NULL. If NULL then the in-sample one-step prediction errors from the bsts object will be extracted and returned. Otherwise the model will be re-fit with a separate MCMC run for each entry in ‘cutpoints’. Data up to each cutpoint will be included in the fit, and one-step prediction errors for data after the cutpoint will be computed.
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.
standardize Logical. If TRUE then the prediction errors are divided by the square root of the one-step-ahead forecast variance. If FALSE the raw errors are returned.

Details

Returns the posterior distribution of the one-step-ahead prediction errors from the bsts.object. The errors are computing using the Kalman filter, and are of two types.

Purely in-sample errors are computed as a by-product of the Kalman filter as a result of fitting the model. These are stored in the bsts.object assuming the save.prediction.errors option is TRUE, which is the default (See BstsOptions). The in-sample errors are ‘in-sample’ in the sense that the parameter values used to run the Kalman filter are drawn from their posterior distribution given complete data. Conditional on the parameters in that MCMC iteration, each ‘error’ is the difference between the observed y[t] and its expectation given data to t-1.

Purely out-of-sample errors can be computed by specifying the ‘cutpoints’ argument. If cutpoints are supplied then a separate MCMC is run using just data up to the cutpoint. The Kalman filter is then run on the remaining data, again finding the difference between y[t] and its expectation given data to t-1, but conditional on parameters estimated using data up to the cutpoint.

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 <steve.the.bayesian@gmail.com>

References

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

See Also

bsts, AddLocalLevel, AddLocalLinearTrend, AddSemilocalLinearTrend, SpikeSlabPrior, SdPrior.
Examples

```r
data(AirPassengers)
y <- log(AirPassengers)
ss <- AddLocalLinearTrend(list(), y)
ss <- AddSeasonal(ss, y, nseasons = 12)

## Not run:
model <- bsts(y, state.specification = ss, niter = 500)
## End(Not run)

errors <- bsts.prediction.errors(model, burn = 100)
PlotDynamicDistribution(errors$in.sample)

## Compute out of sample prediction errors beyond times 80 and 120.
errors <- bstsPrediction.errors(model, cutpoints = c(80, 120))
standardized.errors <- bstsPrediction.errors(
  model, cutpoints = c(80, 120), standardize = TRUE)
plot(model, "prediction.errors", cutpoints = c(80, 120))
str(errors)  # three matrices, with 400 (= 500 - 100) rows
            # and length(y) columns
```

---

plot.bsts

Plotting functions for Bayesian structural time series

Description

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

Usage

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

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,
components = 1:length(bsts.object$state.specification),
...

PlotDynamicRegression(bsts.object,
    burn = SuggestBurn(.1, bsts.object),
    time = NULL,
    same.scale = FALSE,
    style = c("dynamic", "boxplot"),
    layout = c("square", "horizontal", "vertical"),
    ylim = NULL,
    zero.width = 2,
    zero.color = "green",
    ...
)

PlotBstsState(bsts.object, burn = SuggestBurn(.1, bsts.object),
    time, show.actuals = TRUE,
    style = c("dynamic", "boxplot"),
    scale = c("linear", "mean"),
    ylim = NULL,
    ...
)

PlotBstsResiduals(bsts.object, burn = SuggestBurn(.1, bsts.object),
    time, style = c("dynamic", "boxplot"), means = TRUE, ...
)

PlotBstsPredictionErrors(bsts.object, cutpoints = NULL,
    burn = SuggestBurn(.1, bsts.object),
    style = c("dynamic", "boxplot"),
    xlab = "Time", ylab = "", main = "",
    ...
)

PlotBstsForecastDistribution(bsts.object, cutpoints = NULL,
    burn = SuggestBurn(.1, bsts.object),
    style = c("dynamic", "boxplot"),
    xlab = "Time",
    ylab = "",
    main = "",
    show.actuals = TRUE,
    col.actuals = "blue",
    ...
)

PlotBstsSize(bsts.object, burn = SuggestBurn(.1, bsts.object), style =
    c("histogram", "ts"), ...
)

PlotSeasonalEffect(bsts.object, nseasons = 7, season.duration = 1,
    same.scale = TRUE, ylim = NULL, get.season.name = NULL,
    burn = SuggestBurn(.1, bsts.object), ...)
PlotMonthlyAnnualCycle(bsts.object, ylim = NULL, same.scale = TRUE, burn = SuggestBurn(.1, bsts.object), ...)

**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.
- **col.actuals**: The color to use for the actual data when comparing actuals vs forecasts.
- **components**: A numeric vector indicating which components to plot. Component indices correspond to elements of the state specification that was used to build the bsts model being plotted.
- **cutpoints**: A numeric vector of integers, or `NULL`. For diagnostic plots of prediction errors or forecast distributions, the model will be re-fit with a separate MCMC run for each entry in 'cutpoints'. Data up to each cutpoint will be included in the fit, and one-step prediction errors for data after the cutpoint will be computed.
- **get.season.name**: A function that can be used to infer the title of each seasonal plot. It should take a single `POSIXt`, `Date`, or similar object as an argument, and return a single string that can be used as a panel title. If `get.season.name` is `NULL` and `nseasons` is specified or inferred to be one of the following values, then the following functions will be used.
  - 4: `quarters`
  - 7: `weekdays`
  - 12: `months`
- **inclusion.threshold**: An inclusion probability that individual coefficients must exceed in order to be displayed when `what == "coefficients"`. See the help file for `plot.lm.spike`.
- **layout**: For controlling the layout of functions that generate multiple plots.
- **main**: Main title for the plot.
- **means**: Logical. If TRUE then the mean of each residual is plotted as a blue dot. If false only the distribution of the residuals is plotted.
- **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.
- **number.of.variables**: If non-NULL this specifies the number of coefficients to plot, taking precedence over `inclusion.threshold`. See `plot.lm.spike`.
- **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.
scale

The scale on which to plot the state. If the error family is "logit" or "poisson" then the state can either be plotted on the scale of the linear predictor (e.g. trend + seasonal + regression) or the linear predictor can be passed through the link function so as to plot the distribution of the conditional mean.

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.

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.

time

An optional vector of values to plot against. If missing, the default is to diagnose the time scale of the original time series.

xlab

Label for the horizontal axis.

ylab

Label for the vertical axis.

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.

zero.width

A numerical value for the width of the reference line at zero. If NULL then the line will be omitted.

zero.color

A color for the width of the reference line at zero. If NULL then the line will be omitted.

...

Additional arguments to be passed to PlotDynamicDistribution, or TimeSeriesBoxplot.

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.

PlotSeasonalEffect generates an array of plots showing how the distribution of the seasonal effect changes, for each season, for models that include a seasonal state component.

PlotMonthlyAnnualCycle produces an array of plots much like PlotSeasonalEffect, for models that include a MonthlyAnnualCycle state component.
Value

These functions are called for their side effect, which is to produce a plot on the current graphics
device.
PlotBstsState invisibly returns the state object being plotted.

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)
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 (presumably partial) observation in the aggregated state time series should be removed.
- **trim.right**: A logical indicating whether the last (presumably 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

```r
## 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

```r
## 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,
```
```r
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 or numeric. If `TRUE` then the prediction is plotted after a time series plot of the original series. If `FALSE`, the prediction fills the entire plot. If numeric, then it specifies the number of trailing observations of the original time series to plot in addition to the predictions.
- **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 interval for the prediction.
- **interval.width**: The width of line (lwd) to use for the upper and lower limits of the 95% credible interval for 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)

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.
plot.holiday

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.holiday  

Plot Holiday Effects

Description

Plot the estimated effect of the given holiday.

Usage

PlotHoliday(holiday, model, show.raw.data = TRUE, ylim = NULL, ...)

Arguments

holiday  
An object of class Holiday.

model  
A model fit by bsts containing either a RegressionHolidayStateModel or HierarchicalRegressionHolidayStateModel that includes holiday.

show.raw.data  
Logical indicating if the raw data corresponding to holiday should be superimposed on the plot. The 'raw data' are the actual values of the target series, minus the value of the target series the day before the holiday began, which is a (somewhat poor) proxy for remaining state elements. The raw data can appear artificially noisy if there are other strong state effects such as a day-of-week effect for holidays that don’t always occur on the same day of the week.

ylim  
Limits on the vertical axis of the plots.

...  
Extra arguments passed to boxplot.

Value

Returns invisible(NULL).

See Also

bsts AddRandomWalkHoliday
Examples

trend <- cumsum(rnorm(730, 0, .1))
dates <- seq.Date(from = as.Date("2014-01-01"), length = length(trend), by = "day")
y <- zoo(trend + rnorm(length(trend), 0, .2), dates)

AddHolidayEffect <- function(y, dates, effect) {
  ## Adds a holiday effect to simulated data.
  ## Args:
  ## y: A zoo time series, with Dates for indices.
  ## dates: The dates of the holidays.
  ## effect: A vector of holiday effects of odd length. The central effect is
  ## the main holiday, with a symmetric influence window on either side.
  ## Returns:
  ## y, with the holiday effects added.
  time <- dates - (length(effect) - 1) / 2
  for (i in 1:length(effect)) {
    y[time] <- y[time] + effect[i]
    time <- time + 1
  }
  return(y)
}

## Define some holidays.
memorial.day <- NamedHoliday("MemorialDay")
memorial.day.effect <- c(.3, 3, .5)
memorial.day.dates <- as.Date(c("2014-05-26", "2015-05-25"))
y <- AddHolidayEffect(y, memorial.day.dates, memorial.day.effect)

presidents.day <- NamedHoliday("PresidentsDay")
presidents.day.effect <- c(.5, 2, .25)
presidents.day.dates <- as.Date(c("2014-02-17", "2015-02-16"))
y <- AddHolidayEffect(y, presidents.day.dates, presidents.day.effect)

labor.day <- NamedHoliday("LaborDay")
labor.day.effect <- c(1, 2, 1)
labor.day.dates <- as.Date(c("2014-09-01", "2015-09-07"))
y <- AddHolidayEffect(y, labor.day.dates, labor.day.effect)

## The holidays can be in any order.
holiday.list <- list(memorial.day, labor.day, presidents.day)
number.of.holidays <- length(holiday.list)

## In a real example you'd want more than 100 MCMC iterations.
niter <- 100
ss <- AddLocalLevel(list(), y)
ss <- AddRegressionHoliday(ss, y, holiday.list = holiday.list)
model <- bsts(y, state.specification = ss, niter = niter)

PlotHoliday(memorial.day, model)
predict.bsts

**Description**

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

**Usage**

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

**Arguments**

- **object**: An object of class `bsts` created by a call to the function `bsts`.
- **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.
- **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 component. 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`.
- **timestamps**: A vector of time stamps (of the same type as the timestamps used to fit `object`), with one per row of `newdata` (or element of `newdata`, if `newdata` is a vector). The time stamps give the time points as which each prediction is desired. They must be interpretable as integer (0 or larger) time steps following the last time stamp in `object`. If NULL, then the requested predictions are interpreted as being at 1, 2, 3, ... steps following the training data.
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.

The value for olddata depends on whether or not object contains a regression component.

- 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.
- If no regression component is present, then olddata is a vector containing historical values of a time series.

olddata.timestamps  A set of timestamps corresponding to the observations supplied in olddata. If olddata is NULL then this argument is not used. If olddata is supplied and this is NULL then trivial timestamps (1, 2, ...) will be assumed. Otherwise this argument behaves like the timestamps argument to the bsts function.

trials.or.exposure  For logit or Poisson models, the number of binomial trials (or the exposure time) to assume at each time point in the forecast period. This can either be a scalar (if the number of trials is to be the same for each time period), or it can be a vector with length equal to horizon (if the model contains no regression term) or nrow(newdata) if the model contains a regression term.

quantiles  A numeric vector of length 2 giving the lower and upper quantiles to use for the forecast interval estimate.

seed  An integer to use as the C++ random seed. If NULL then the C++ seed will be set using the clock.

...  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 \texttt{bsts.prediction}, which is a list with the following components.

- \texttt{mean}: A vector giving the posterior mean of the prediction.
- \texttt{interval}: A two (column/row) matrix giving the upper and lower bounds of the 95 percent credible interval for the prediction.
- \texttt{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

References

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


See Also

\texttt{bsts. AddLocalLevel. AddLocalLinearTrend. AddSemilocalLinearTrend}.

Examples

# The number of MCMC draws in the following examples is artificially low.

```
## Making predictions when there is no regression component.
data(AirPassengers)
y <- log(AirPassengers)
ss <- AddLocalLinearTrend(list(), y)
ss <- AddSeasonal(ss, y, nseasons = 12)
model <- bsts(y, state.specification = ss, niter = 250)
pred <- predict(model, horizon = 12, burn = 100)
plot(pred)
```

```
## An example using the olddata argument.
full.pred <- pred
training <- window(y, end = c(1959, 12))
model <- bsts(training, state.specification = ss, niter = 250)
## Predict the next 12 months.
pred <- predict(model, horizon = 12)
## Compare the predictions to the actual data.
plot(pred)
lines(as.numeric(y, col = "red", lty = 2, lwd = 2))
```

```
## Predict the 12 months of 1961 based on the posterior distribution
```
quarter

Find the quarter in which a date occurs

Description

Returns the quarter and year in which a date occurs.

Usage

Quarter(date)
Regression Based Holiday Models

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 <steve.the.bayesian@gmail.com>

Examples

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

Description

Add a regression-based holiday model to the state specification.

Usage

AddRegressionHoliday(
  state.specification = NULL,
  y,
  holiday.list,
  time0 = NULL,
  prior = NULL,
  sdy = sd(as.numeric(y), na.rm = TRUE))

AddHierarchicalRegressionHoliday(
  state.specification = NULL,
  y,
  holiday.list,
  coefficient.mean.prior = NULL,
  coefficient.variance.prior = NULL,
  time0 = NULL,
  sdy = sd(as.numeric(y), na.rm = TRUE))
Arguments

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

holiday.list
A list of objects of type Holiday. The width of the influence window should be the same number of days for all the holidays in this list. If the data contains many instances of holidays with different window widths, then multiple instances HierarchicalRegressionHolidayModel can be used as long as all holidays in the same state component model have the same sized window width.

y
The time series to be modeled, as a numeric vector convertible to xts. This state model assumes y contains daily data.

prior
An object of class NormalPrior describing the expected variation among daily holiday effects.

coefficient.mean.prior
An object of type MvnPrior giving the hyperprior for the average effect of a holiday in each day of the influence window.

coefficient.variance.prior
An object of type InverseWishartPrior describing the prior belief about the variation in holiday effects from one holiday to the next.

time0
An object convertible to Date 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].

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.

Details

The model assumes that

\[ y_t = \beta_{d(t)} + \epsilon_t \]

The regression state model assumes vector of regression coefficients \( \beta \) contains elements \( \beta_d \sim N(0, \sigma) \).

The HierarchicalRegressionHolidayModel assumes \( \beta \) is composed of holiday-specific sub-vectors \( \beta_h \sim N(b_0, V) \), where each \( \beta_h \) contains coefficients describing the days in the influence window of holiday \( h \). The hierarchical version of the model treats \( b_0 \) and \( V \) as parameters to be learned, with prior distributions

\[ b_0 \sim N(\tilde{b}, \Omega) \]

and

\[ V \sim IW(\nu, S) \]

where \( IW \) represents the inverse Wishart distribution.

Value

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

Author(s)
Steven L. Scott <steve.the.bayesian@gmail.com>

References

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


See Also

*bsts*. RandomWalkHolidayStateModel. SdPrior NormalPrior

Examples

trend <- cumsum(rnorm(730, 0, .1))
dates <- seq.Date(from = as.Date("2014-01-01"), length = length(trend), by = "day")
y <- zoo(trend + rnorm(length(trend), 0, .2), dates)

AddHolidayEffect <- function(y, dates, effect) {
  ## Adds a holiday effect to simulated data.
  ## Args:
  ## y: A zoo time series, with Dates for indices.
  ## dates: The dates of the holidays.
  ## effect: A vector of holiday effects of odd length. The central effect is
  ## the main holiday, with a symmetric influence window on either side.
  ## Returns:
  ## y, with the holiday effects added.
  time <- dates - (length(effect) - 1) / 2
  for (i in 1:length(effect)) {
    y[time] <- y[time] + effect[i]
    time <- time + 1
  }
  return(y)
}

## Define some holidays.
memorial.day <- NamedHoliday("MemorialDay")
memorial.day.effect <- c(.3, 3, .5)
memorial.day.dates <- as.Date(c("2014-05-26", "2015-05-25"))
y <- AddHolidayEffect(y, memorial.day.dates, memorial.day.effect)

presidents.day <- NamedHoliday("PresidentsDay")
presidents.day.effect <- c(.5, 2, .25)
presidents.day.dates <- as.Date(c("2014-02-17", "2015-02-16"))
y <- AddHolidayEffect(y, presidents.day.dates, presidents.day.effect)

labor.day <- NamedHoliday("LaborDay")
labor.day.effect <- c(1, 2, 1)
labor.day.dates <- as.Date(c("2014-09-01", "2015-09-07"))
regularize.timestamps  Produce a Regular Series of Time Stamps

Description

Given an set of timestamps that might contain duplicates and gaps, produce a set of timestamps that has no duplicates and no gaps.

Usage

RegularizeTimestamps(timestamps)

## Default S3 method:
RegularizeTimestamps(timestamps)

## S3 method for class 'numeric'
RegularizeTimestamps(timestamps)

## S3 method for class 'Date'

y <- AddHolidayEffect(y, labor.day.dates, labor.day.effect)

## The holidays can be in any order.
holiday.list <- list(memorial.day, labor.day, presidents.day)

## In a real example you'd want more than 100 MCMC iterations.
niter <- 100

## Fit the model
ss <- AddLocalLevel(list(), y)
ss <- AddRegressionHoliday(ss, y, holiday.list = holiday.list)
model <- bsts(y, state.specification = ss, niter = niter)

## Plot all model state components.
plot(model, "comp")

## Plot the specific holiday state component.
plot(ss[[2]], model)

## Try again with some shrinkage. With only 3 holidays there won't be much
## shrinkage.
ss2 <- AddLocalLevel(list(), y)

## Plot the specific holiday state component.
ss2 <- AddHierarchicalRegressionHoliday(ss2, y, holiday.list = holiday.list)
model2 <- bsts(y, state.specification = ss2, niter = niter)

plot(model2, "comp")
plot(ss2[[2]], model2)
RegularizeTimestamps(timestamps)

## S3 method for class 'POSIXt'
RegularizeTimestamps(timestamps)

Arguments

timestamps  A set of (possibly irregular or non-unique) timestamps. This could be a set of integers (like 1, 2, 3...), a set of numeric like (1945, 1945.083, 1945.167, ...) indicating years and fractions of years, a Date object, or a POSIXt object. If the argument is NULL a NULL will be returned.

Value

A set of regularly spaced timestamps of the same class as the argument (which might be NULL).

Author(s)

Steven L. Scott <steve.the.bayesian@gmail.com>

Examples

```r
first <- as.POSIXct("2015-04-19 08:00:04")
monthly <- seq(from = first, length.out = 24, by = "month")
skip.one <- monthly[-8]
has.duplicates <- monthly
has.duplicates[2] <- has.duplicates[3]

reg1 <- RegularizeTimestamps(skip.one)
all.equal(reg1, monthly) ## TRUE

reg2 <- RegularizeTimestamps(has.duplicates)
all.equal(reg2, monthly) ## TRUE
```

residuals.bsts  Residuals from a bsts Object

Description

Residuals (or posterior distribution of residuals) from a bsts object.

Usage

```r
## S3 method for class 'bsts'
residuals(object, 
  burn = SuggestBurn(.1, object), 
  mean.only = FALSE, 
  ...)
```
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.
- **mean.only**: Logical. If `TRUE` then the mean residual for each time period is returned. If `FALSE` then the full posterior distribution is returned.
- **...**: Not used. This argument is here to comply with the signature of the generic residuals function.

Value

If `mean.only` is `TRUE` then this function returns a vector of residuals with the same "time stamp" as the original series. If `mean.only` is `FALSE` then the posterior distribution of the residuals is returned instead, as a matrix of draws. Each row of the matrix is an MCMC draw, and each column is a time point. The colnames of the returned matrix will be the timestamps of the original series, as text.

See Also

- `bsts`, `plot.bsts`.

---

**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)`
shark

Shark Attacks in Florida.

Description
An annual time series of shark attacks and fatalities in Florida.

Usage
data(shark)

Format
zoo time series

Source
From Jeffrey Simonoff "Analysis of Categorical Data". http://people.stern.nyu.edu/jsimonof/AnalCatData/Data/Comma_separated/floridashark.csv

Examples
data(shark)
head(shark)

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 <steve.the.bayesian@gmail.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.
simulate.fake.mixed.frequency.data

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

course.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.

ture.beta  The vector of "true" regression coefficients used to simulate fine.target.

ture.sigma.obs  The residual standard deviation that was used to simulate fine.target.

ture.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.

ture.trend  The combined contribution of the simulated latent state on fine.target, including regression effects.

ture.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 <steve.the.bayesian@gmail.com>

References

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

Spike and Slab Priors for AR Processes

Returns a spike and slab prior for the parameters of an AR(p) process.

Usage

SpikeSlabArPrior(
  lags,
  prior.inclusion.probabilities =
    GeometricSequence( lags, initial.value = .8, discount.factor = .8),
  prior.mean = rep(0, lags),
  prior.sd =
    GeometricSequence(lags, initial.value = .5, discount.factor = .8),
  sdy,
  prior.df = 1,
  expected.r2 = .5,
  sigma.upper.limit = Inf,
  truncate = TRUE)

Arguments

  lags  A positive integer giving the maximum number of lags to consider.
  prior.inclusion.probabilities  A vector of length lags giving the prior probability that the corresponding AR coefficient is nonzero.
  prior.mean  A vector of length lags giving the prior mean of the AR coefficients. This should almost surely stay set at zero.
  prior.sd  A vector of length lags giving the prior standard deviations of the AR coefficients, which are modeled as a-priori independent of one another.
  sdy  The sample standard deviation of the series being modeled.
  expected.r2  The expected fraction of variation in the response explained by this AR process.
  prior.df  A positive number indicating the number of observations (time points) worth of weight to assign to the guess at expected.r2.
**state.sizes**

sigma.upper.limit

A positive number less than infinity truncates the support of the prior distribution to regions where the residual standard deviation is less than the specified limit. Any other value indicates support over the entire positive real line.

truncate

If TRUE then the support of the distribution is truncated to the region where the AR coefficients imply a stationary process. If FALSE the coefficients are unconstrained.

Value

A list of class SpikeSlabArPrior containing the information needed for the underlying C++ code to instantiate this prior.

Author(s)

Steven L. Scott <steve.the.bayesian@gmail.com>

---

**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 <steve.the.bayesian@gmail.com>

Examples

```r
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 <steve.the.bayesian@gmail.com>

References
Harvey (1990), "Forecasting, structural time series, and the Kalman filter", Cambridge 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

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

This statistic is analogous to $R^2$ in a regression model, but the reduction in sum of squared errors is relative to a random walk with a constant drift,

$$y_{t+1} = y_t + \beta + \epsilon_t,$$

which Harvey (1989, equation 5.5.14) argues is a more relevant baseline than a simple mean. Unlike a traditional R-square statistic, this can be negative.
size
coefficients

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


See Also

*bsts*, *plot.bsts*, *summary.lm.spike*

Examples

```r
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)
```

---

**to.posixt**

.Convert to POSIXt.

**Description**

Convert an object of class Date to class POSIXct without getting bogged down in timezone calculation.

**Usage**

```r
DateToPOSIX(timestamps)
YearMonToPOSIX(timestamps)
```

**Arguments**

- `timestamps`: An object of class `yearmon` or `Date` to be converted to POSIXct.
Details

Calling `as.POSIXct` on another date/time object (e.g. `Date`) applies a timezone correction to the object. This can shift the time marker by a few hours, which can have the effect of shifting the day by one unit. If the day was the first or last in a month or year, then the month or year will be off by one as well.

Coercing the object to the character representation of a `Date` prevents this adjustment from being applied, and leaves the POSIXt return value with the intended day, month, and year.

Author(s)

Steven L. Scott <steve.the.bayesian@gmail.com>

data(turkish)

Description

A daily time series of electricity usage in Turkey.

Usage

data(turkish)

Format

zoo time series

Source

https://robjhyndman.com/data/turkey_elec.csv

See Also

`bsts`

Examples

data(turkish)
plot(turkish)
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

```r
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 <steve.the.bayesian@gmail.com>

See Also

- `bsts.mixed`

Examples

```r
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)
```
weekday.names

Days of the Week

Description
A character vector listing the names the days of the week.

Usage
weekday.names

See Also
month.name

wide.to.long

Convert Between Wide and Long Format

Description
Convert a multivariate time series between wide and long formats. In "wide" format there is one row per time point, with series organized by columns. In "long" format there is one row per observation, with variables indicating the series and time point to which an observation belongs.

Usage

WideToLong(response, na.rm = TRUE)
LongToWide(response, series.id, timestamps)

Arguments

response
For WideToLong this is a matrix, with rows representing time and columns representing variables. This can be a zoo matrix with timestamps as an index.
For LongToWide, response is a vector.

na.rm
If TRUE then missing values will be omitted from the returned data frame (their absence denoting missingness). Otherwise, missing values will be included as NA's.

series.id
A factor (or variable coercible to factor) of the same length as response, indicating the series to which each observation belongs.

timestamps
A variable of the same length as response, indicating the time period to which each observation belongs.

Value
LongToWide returns a zoo matrix with the time series in wide format. WideToLong returns a 3-column data frame with columns "time", "series", and "values".
Author(s)
Steven L. Scott <steve.the.bayesian@gmail.com>

Examples
```r
data(gdp)
gdp.wide <- LongToWide(gdp$GDP, gdp$Country, gdp$Time)
gdp.long <- WideToLong(gdp.wide)
```
Index

* character
  shorten, 87

* chron
  aggregate.time.series, 26
  aggregate.weeks.to.months, 28
  bststs.options.Rd, 36
  compare.bststs.models, 38
  date.range, 39
  estimate.time.scale, 46
  extend.time, 47
  format.timestamps, 48
  get.fraction, 50
  last.day.in.month, 56
  match.week.to.month, 58
  month.distance, 62
  quarter, 80
  regularize.timestamps, 84
  state.sizes, 91
  week.ends, 96

* datasets
  gdp, 49
  goog, 51
  iclaims, 55
  new.home.sales, 64
  rsxfs, 86
  shark, 87
  turkish, 95

* models
  add.ar, 3
  add.dynamic.regression, 5
  add.local.level, 8
  add.local.linear.trend, 10
  add.random.walk.holiday, 13
  add.seasonal, 16
  add.semilocal.linear.trend, 17
  add.shared.local.level, 19
  add.static.intercept, 21
  add.student.local.linear.trend, 22
  add.trig, 24
  auto.ar, 29
  bststs, 31
  dirm, 42
  dirm-model-optoins, 45
  HarveyCumulator, 52
  max.window.width, 59
  mixed.frequency, 60
  one.step.prediction.errors, 64
  predict.bststs, 77
  regression.holiday, 81
  simulate.fake.mixed.frequency.data, 88
  StateSpecification, 92
  to.posixt, 94

* regression
  bststs, 31
  dirm, 42
  dirm-model-optoins, 45
  HarveyCumulator, 52
  max.window.width, 59
  mixed.frequency, 60
  one.step.prediction.errors, 64
  predict.bststs, 77
  simulate.fake.mixed.frequency.data, 88
  StateSpecification, 92
  to.posixt, 94

acf, 41
AcfDist (diagnostic-plots), 41
add.ar, 3
add.dynamic.regression, 5
add.local.level, 8
add.local.linear.trend, 10
add.monthly.annual.cycle, 11
add.random.walk.holiday, 13
add.seasonal, 16
add.semilocal.linear.trend, 17
add.shared.local.level, 19
add.static.intercept, 21
add.student.local.linear.trend, 22
add.trig, 24
AddAr, 30
AddAutoAr (add.ar), 3
AddDynamicRegression (auto.ar), 29
AddDynamicRegression
  (add.dynamic.regression), 5
AddGeneralizedLocalLinearTrend
  (add.semilocal.linear.trend), 17
AddHierarchicalRegressionHoliday, 54, 55, 59
AddHierarchicalRegressionHoliday
  (regression.holiday), 81
AddLocalLevel, 34, 44, 62, 65, 79
AddLocalLevel
  (add.local.level), 8
AddLocalLinearTrend, 32, 34, 42, 44, 62, 65, 79, 90
AddLocalLinearTrend
  (add.local.linear.trend), 10
AddMonthlyAnnualCycle
  (add.monthly.annual.cycle), 11
AddRandomWalkHoliday, 54, 55, 59, 75
AddRandomWalkHoliday
  (add.random.walk.holiday), 13
AddRegressionHoliday, 54, 55, 59
AddRegressionHoliday
  (regression.holiday), 81
AddSeasonal, 11, 32, 34, 42, 44
AddSeasonal
  (add.seasonal), 16
AddSemilocalLinearTrend, 34, 44, 62, 65, 79
AddSemilocalLinearTrend
  (add.semilocal.linear.trend), 17
AddSharedLocalLevel
  (add.shared.local.level), 19
AddStatic Intercept
  (add.static.intercept), 21
AddStudentLocalLinearTrend
  (add.student.local.linear.trend), 22
AddTrig (add.trig), 24
aggregate.time.series, 26
aggregate.weeks.to.months, 28
AggregateTimeSeries, 28
AggregateTimeSeries
  (aggregate.time.series), 26
AggregateWeeksToMonths
  (aggregate.weeks.to.months), 28
Ar1CoefficientPrior, 18, 92
as.data.frame, 6, 32, 43
as.POSIXct, 95
auto.ar, 29
boxplot, 41, 75
bsts, 4, 7, 9, 11, 14, 17, 19, 21, 22, 24, 26, 30,
  31, 32–34, 38, 42–44, 56, 62, 64–66,
  68, 70, 72–75, 77–79, 83, 86, 91–95
bsts.mixed, 47, 51, 52, 58, 71, 72, 88–90, 96
bsts.mixed (mixed.frequency), 60
bsts.options.Rd, 36
bsts.prediction, 73
bsts.prediction (predict.bsts), 77
bsts.prediction.errors, 38
bsts.prediction.errors
  (one.step.prediction.errors), 64
BstsOptions, 32, 65
BstsOptions (bsts.options.Rd), 36
compare.bsts.models, 38
CompareBstsModels
  (compare.bsts.models), 38
ConditionalZellnerPrior, 20
corr.treatment, 32, 43
Date, 12, 14, 28, 39, 40, 43, 46–48, 50, 56, 58,
  63, 68, 81, 82, 85, 94, 96
date.range, 39
DateRange (date.range), 39
DateRangeHoliday (holiday), 53
DateToPOSIX (to.POSIXt), 94
dayplot (descriptive-plots), 40
descriptive-plots, 40
diagnostic-plots, 41
dirm, 42, 43
dirm-model-optoins, 45
DirmModelOptions, 43
DirmModelOptions (dirm-model-optoins), 45
DoubleModel, 23
DynamicRegressionArOptions
  (add.dynamic.regression), 5
DynamicRegressionHierarchicalRandomWalkOptions
  (add.dynamic.regression), 5
PlotBstsMixedState (plot.bsts.mixed), 70
PlotBstsPredictionErrors, 69
PlotBstsPredictionErrors (plot.bsts), 66
PlotBstsPredictors
  (plot.bsts.predictors), 74
PlotBstsResiduals, 69
PlotBstsResiduals (plot.bsts), 66
PlotBstsSize, 69, 72
PlotBstsSize (plot.bsts), 66
PlotBstsState, 69
PlotBstsState (plot.bsts), 66
PlotDynamicDistribution, 41, 69–75
PlotDynamicRegression (plot.bsts), 66
PlotHoliday (plot.holiday), 75
PlotMonthlyAnnualCycle, 69
PlotMonthlyAnnualCycle (plot.bsts), 66
PlotSeasonalEffect, 69
PlotSeasonalEffect (plot.bsts), 66
POSIXlt, 81
POSIXt, 12, 39, 40, 43, 48, 68, 85
predict, 78
predict.bsts, 73, 77
qqdist (diagnostic-plots), 41
qqline, 41
qqnorm, 41
Quarter (quarter), 80
quarter, 80
quarters, 40, 68
rainbow, 38
RandomWalkHolidayStateModel, 83
RandomWalkHolidayStateModel
  (add.random.walk.holiday), 13
regression.holiday, 81
RegressionHolidayStateModel, 14, 75
RegressionHolidayStateModel
  (regression.holiday), 81
regularize.timestamps, 84
RegularizeTimestamps
  (regularize.timestamps), 84
residuals.bsts, 85
retail.sales (rsxfs), 86
RSXFS (rsxfs), 86
rsxfs, 86
ScaledMatrixNormalPrior, 20
SdPrior, 4, 6, 7, 9–12, 14, 16–19, 21–24, 26, 30, 32, 34, 44, 62, 65, 83, 92
shark, 87
Shorten (shorten), 87
shorten, 87
simulate.fake.mixed.frequency.data, 88
SimulateFakeMixedFrequencyData
  (simulate.fake.mixed.frequency.data), 88
spike.slab.ar.prior, 90
SpikeSlabArPrior, 29
SpikeSlabArPrior (spike.slab.ar.prior), 90
SpikeSlabPrior, 32–34, 43, 44, 61, 62, 65
state.sizes, 91
state.specification, 32, 42
state.specification
  (StateSpecification), 92
StateSizes (state.sizes), 91
StateSpecification, 92
SuggestBurn, 92
summary.bsts, 93
summary.lm.spike, 93, 94
TimeSeriesBoxplot, 69, 71
to.posixt, 94
ts, 32, 40, 41
turkish, 95
week.ends, 96
weekday.names, 97
weekdays, 40, 68
WeekEndsMonth (week.ends), 96
WeekEndsQuarter (week.ends), 96
wide.to.long, 97
WideToLong (wide.to.long), 97
xts, 14, 32, 82
yearmon, 40, 94
YearMonToPOSIX (to.posixt), 94
YearPlot (descriptive-plots), 40
zoo, 12, 14, 28, 32, 33, 40, 60, 82, 89, 97