Package 'SSsimple'

July 21, 2025

Type Package

Title State Space Models

Version 0.6.6

Date 2019-12-06

Author Dave Zes

Maintainer Dave Zes <zesdave@gmail.com>

Description Simulate, solve state space models.

License GPL (>= 2)

Depends mvtnorm

Suggests maps

Imports methods

NeedsCompilation no

Repository CRAN

Date/Publication 2019-12-07 01:10:10 UTC

Contents

SSsimple-package *Simple State Space Models*

Description

Simulate, solve (estimate), fit state space models

Details

Package:	SSsimple
Type:	Package
Version:	0.6.6
Date:	2019-12-06
License:	GPL (>= 2)
LazyLoad:	yes

If you wish to parameterize a state space model given only data, `Z`, use the function `SS.ID`. If you wish to simulate data, use `SS.sim` or `SS.sim.tv`. If you have data, know the model parameters, and wish to solve the lowest L2 estimate, use `SS.solve` or `SS.solve.tv`.

The two functions, `H.omega.sincos` and `H.omega.cos.2D`, provide a means of introducing response curvature over a domain (probably space) through the common sine bases expansion.

Finally, `SS.stst` and `SS.stst.tv`, attempt to find the time at which a system acheives a "steady-state."

The system of interest is defined as

$\mathbf{b}(t) = \mathbf{F}\,\mathbf{b}(t\text{-}1) + \mathbf{n}(t)$, $\mathbf{n}(t) \sim N[\mathbf{0}, \mathbf{Q}]$

$\mathbf{z}'(t) = \mathbf{H}\,\mathbf{b}(t) + \mathbf{e}(t)$, $\mathbf{e}(t) \sim N[\mathbf{0}, \mathbf{R}]$

Functions whose names end in ".tv" provide for the usage of time-varying `F`, `H`, `Q`, `R`.

Author(s)

Dave Zes

Maintainer: Dave Zes <zesdave@gmail.com>

`H.omega.cos.2D` *Bases Transformation*

Description

Create H as cosine bases expansion over R^2

Usage

```
H.omega.cos.2D(x, y, u.x, u.y, phs.x, phs.y)
```

Arguments

`x`	A vector of locations on *x* of length *n*.
`y`	A vector of locations on *y* of length *n*.
`u.x`	A vector of frequencies on *x*.
`u.y`	A vector of frequencies on *y*.
`phs.x`	A vector of phase shifts on *x*. Should be same length as `u.x`.
`phs.y`	A vector of phase shifts on *y*. Should be same length as `u.y`.

Value

An *n* x *d* matrix, with *d* = `length(u.x) * length(u.y)`.

See Also

`H.omega.sincos`

Examples

```
x <- rep( I(0:10) / 10,   11 )
y <- rep( I(0:10) / 10,   each=11 )
u.x <- I(1:2) * pi
u.y <- I(1:2) * pi
H <- H.omega.cos.2D(x, y, u.x, u.y, c(0,0), c(0,0))

b <- rep(1, ncol(H))

z <- H %*% b

plot(x, y, cex=z-min(z))
```

H.omega.sincos *Bases Transformation*

Description

Create H as sine-cosine bases expansion over R^1

Usage

```
H.omega.sincos(x, u)
```

Arguments

x	A vector of locations of length *n*.
u	A vector of frequencies of length *d*/2.

Value

An *n* x *d* matrix.

See Also

H.omega.cos.2D

Examples

```
x <- I(0:10) / 10
u <- I(1:4) * pi
H.omega.sincos(x, u)
```

SS.ID *System Identification*

Description

Perform non-iterative, subspace grey-box system identification

Usage

```
SS.ID(Z, d, rsN = NULL)
```

Arguments

Z	A *T* x *n* data matrix
d	A scalar integer. The system "order."
rsN	A 3-element integer vector, containing *r*, *s*, *N* as described by Ljung.

Details

Only works when $T \gg n$ (one resolved to using `SS.ID` when this is not true is free to pluck columns from Z until it is).

Complaints issued from this function to the effect that a matrix that is some function of "PP" cannot be inverted might be remedied by turning r and s down (the first two elements of the `rsN` argument), or perhaps by adding a small amount of noise to Z.

This is subspace estimation. `SS.ID` estimates system hyperparameters from data. One can usually henceforth solve (using `SS.solve`) for good quality observation-space estimates, but should not assume the resulting state estimates are anywhere near truth. One may wish to use estimates generated with this function as initial values for iterative estimation techniques, e.g., package `Stem`.

Value

A named list.

`F.hat`	A d x d matrix.
`H.hat`	An n x d matrix.
`Q.hat`	A d x d matrix.
`R.hat`	An n x n matrix.

References

Lennart Ljung. *System Identification, Theory for the User*. Prentice Hall, 1999.

Examples

```
Q <- diag(1/10, 2)
R <- diag(2, 3)
H <- matrix(1, 3, 2)
F <- diag(0.99, 2)

set.seed(9999)
xs <- SS.sim(F, H, Q, R, 2000, rep(0, 2))

## notice that while the parameter estimates appear somewhat inaccurate ...
ssid <- SS.ID( xs$Z , 2,  c(3, 6, 900) ) ; ssid

## the observation estimate:
sss <- SS.solve( xs$Z, ssid$F.hat, ssid$H.hat, ssid$Q.hat, ssid$R.hat, nrow(xs$Z), 10^5, c(0,0))
Z.hat <- t( ssid$H.hat %*% t( sss$B.apri ) )
sqrt( mean( (xs$Z - Z.hat)^2 ) )

## is nontheless very close to that using true hyperparameter values:
sss.true <- SS.solve( xs$Z, F, H, Q, R, nrow(xs$Z), 10^5, c(0,0))
Z.hat <- t( H %*% t( sss.true$B.apri ) )
sqrt( mean( (xs$Z - Z.hat)^2 ) )
```

SS.sim *Simulation*

Description

Simulate a state space system

Usage

```
SS.sim(F, H, Q, R, length.out, beta0=0)
```

Arguments

F	The state matrix. A scalar, or vector of length d, or a d x d matrix. When scalar, F is constant diagonal. When a vector, F is diagonal.
H	The measurement matrix. Must be n x d.
Q	The state variance. A scalar, or vector of length d, or a d x d matrix. When scalar, Q is constant diagonal. When a vector, Q is diagonal.
R	The measurement variance. A scalar, or vector of length n, or an n x n matrix. When scalar, R is constant diagonal. When a vector, R is diagonal.
length.out	Scalar integer.
beta0	Initial state value. A scalar, or a vector of length d.

Details

H is the master argument from which system dimensionality is determined.

Value

A named list.

Beta	A T x d matrix, the ith row of which is the state at time i.
Y	A T x n matrix, the ith row of which is the noiseless observation at time i.
Z	A T x n matrix, the ith row of which is the observation at time i.

Note

For a definition of the system of interest, please see SSsimple.

Examples

```
tau <- 30

x <- I( 0:10 / 10 )

H <- H.omega.sincos( x, c( 1*pi, 4*pi ) )

xs <- SS.sim( 0.99, H, 1, 2, tau, rep(0, ncol(H)) )

## Not run:
for(i in 1:nrow(xs$Z)) {
plot(x, xs$Z[ i, ], ylim=range(xs$Z), main=i)
Sys.sleep(1/10)
}

## End(Not run)
```

`SS.sim.chol` *Simulation*

Description

Simulate a state space system by supplying measurement variance Cholesky decomposition

Usage

```
SS.sim.chol(F, H, Q, R.chol, length.out, beta0=0)
```

Arguments

`F`	The state matrix. A scalar, or vector of length *d*, or a *d* x *d* matrix. When scalar, `F` is constant diagonal. When a vector, `F` is diagonal.
`H`	The measurement matrix. Must be *n* x *d*.
`Q`	The state variance. A scalar, or vector of length *d*, or a *d* x *d* matrix. When scalar, `Q` is constant diagonal. When a vector, `Q` is diagonal.
`R.chol`	The Cholesky decomposition of the measurement variance (must possess pivot), must be *n* x *n*.
`length.out`	Scalar integer.
`beta0`	Initial state value. A scalar, or a vector of length *d*.

Details

`H` is the master argument from which system dimensionality is determined. Spiritually identical to `SS.sim`. This method can be used to speed up simulating multiple systems with the same parameterization.

Value

A named list.

Beta	A T x d matrix, the ith row of which is the state at time i.
Y	A T x n matrix, the ith row of which is the noiseless observation at time i.
Z	A T x n matrix, the ith row of which is the observation at time i.

Note

For a definition of the system of interest, please see SSsimple.

Examples

```
tau <- 30

x <- I( 0:10 / 10 )

H <- H.omega.sincos( x, c( 1*pi, 4*pi ) )

R <- diag(7, length(x))
R.chol <- chol(R, pivot=TRUE)

xs <- SS.sim.chol( 0.99, H, 1, R.chol, tau, rep(0, ncol(H)) )

## Not run:
for(i in 1:nrow(xs$Z)) {
plot(x, xs$Z[ i, ], ylim=range(xs$Z), main=i)
Sys.sleep(1/10)
}

## End(Not run)
```

SS.sim.tv *Simulation*

Description

Simulate a time-varying state space system

Usage

```
SS.sim.tv(F, H, Q, R, length.out, beta0=0)
```

Arguments

F	A list of d x d matrices.
H	A list of n x d matrices.
Q	A list of d x d matrices.
R	A list of n x n matrices.
length.out	A scalar integer.
beta0	Initial state value. A scalar, or a vector of length d.

Details

This function is a more general, and slower, implementation of SS.sim. This function can also accept arguments in non-time-varying fashion (*a la* SS.sim).

Value

A named list.

Beta	A T x d matrix, the ith row of which is the state at time i.
Y	A T x n matrix, the ith row of which is noiseless observation at time i.
Z	A T x n matrix, the ith row of which is observation at time i.

Examples

```
set.seed(9999)

H.tv <- list()
for(i in 1:200) {
H.tv[[i]] <- matrix( c( sin(i * 0.05), cos(i * 0.05) ), 1, 2 )
}

ssx <- SS.sim.tv( 0.99, H.tv, 0.001, 1, 200, c(4,4) )

plot(ssx$Z[ ,1], type="l")
```

SS.solve *Optimal Estimation*

Description

Solve a state space system using the Kalman Filter

Usage

```
SS.solve(Z, F, H, Q, R, length.out, P0, beta0=0)
```

Arguments

Z	A T x n data matrix.
F	The state matrix. A scalar, or vector of length d, or a d x d matrix. When scalar, F is constant diagonal. When a vector, F is diagonal.
H	The measurement matrix. Must be n x d.
Q	The state variance. A scalar, or vector of length d, or a d x d matrix. When scalar, Q is constant diagonal. When a vector, Q is diagonal.
R	The measurement variance. A scalar, or vector of length n, or an n x n matrix. When scalar, R is constant diagonal. When a vector, R is diagonal.
length.out	Scalar integer.
P0	Initial *a priori* prediction error.
beta0	Initial state value. A scalar, or a vector of length d.

Details

H is the master argument from which system dimensionality is determined.

Value

A named list.

B.apri	A T x d matrix, the ith row of which is the best state estimate prior to observing data at time i.
B.apos	A T x d matrix, the ith row of which is the best state estimate given the observation at time i.

Note

For a definition of the system of interest, please see SSsimple.

Examples

```
set.seed(999)
H <- matrix(1)
x <- SS.sim( 1, H, 1, 1, 100, 0 )
y <- SS.solve( x$Z, 1, H, 1, 1, 100, 10^5, 0 )

z.hat <- t( H %*% t( y$B.apri ) )

plot( x$Z, type="l", col="blue" )
points( z.hat[ ,1], type="l", col="red" )
```

SS.solve.SMW *Optimal Estimation*

Description

Solve a state space system using the Kalman Filter.

Usage

```
SS.solve.SMW(Z, F, H, Q, inv.R, length.out, P0, beta0=0)
```

Arguments

Z	A T x n data matrix.
F	The state matrix. A scalar, or vector of length d, or a d x d matrix. When scalar, F is constant diagonal. When a vector, F is diagonal.
H	The measurement matrix. Must be n x d.
Q	The state variance. A scalar, or vector of length d, or a d x d matrix. When scalar, Q is constant diagonal. When a vector, Q is diagonal.
inv.R	The inverse of the measurement variance. A scalar, or vector of length n, or a n x n matrix. When scalar, inv.R is constant diagonal. When a vector, inv.R is diagonal.
length.out	Scalar integer.
P0	Initial *a priori* prediction error.
beta0	Initial state value. A scalar, or a vector of length d.

Details

H is the master argument from which system dimensionality is determined. Otherwise identical to SS.solve, except that the Woodbury identity is used for inversion. This method offers a computationally reduced means of solving the system realization of interest; however, this method must be supplied with the inverse of the measurement variance matrix, R – not R.

Value

A named list.

B.apri	A T x d matrix, the ith row of which is the best state estimate prior to observing data at time i.
B.apos	A T x d matrix, the ith row of which is the best state estimate given the observation at time i.

Note

For a definition of the system of interest, please see SSsimple.

Examples

```
set.seed(999)
H <- matrix(1)
R <- 7
inv.R <- 1 / R
x <- SS.sim( 1, H, 1, R, 100, 0 )
y <- SS.solve.SMW( x$Z, 1, H, 1, inv.R, 100, 10^5, 0 )

z.hat <- t( H %*% t( y$B.apri ) )

plot( x$Z, type="l", col="blue" )
points( z.hat[ ,1], type="l", col="red" )
```

`SS.solve.tv` *Optimal Estimation*

Description

Solve a time-varying state space system using the Kalman Filter

Usage

```
SS.solve.tv(Z, F, H, Q, R, length.out, P0, beta0)
```

Arguments

`Z`	A T x n data matrix
`F`	A list of d x d matrices.
`H`	A list of n x d matrices.
`Q`	A list of d x d matrices.
`R`	A list of n x n matrices.
`length.out`	A scalar integer.
`P0`	Initial *a priori* prediction error.
`beta0`	Initial state value. A scalar, or a vector of length d.

Details

This function is a more general, and slower, implementation of `SS.solve`. This function can also accept arguments in non-time-varying fashion (*a la* `SS.solve`).

Value

A named list.

`B.apri`	A T x d matrix, the ith row of which is the best state estimate prior to observing data at time i.
`B.apos`	A T x d matrix, the ith row of which is the best state estimate given the observation at time i.

See Also

SS.solve

SS.stst *Steady State*

Description

Find steady state of system, i.e., locate when Kalman gain converges

Usage

```
SS.stst(F, H, Q, R, P0, epsilon, verbosity=0)
```

Arguments

F	The state matrix. A scalar, or vector of length *d*, or a *d* x *d* matrix. When scalar, F is constant diagonal. When a vector, F is diagonal.
H	The measurement matrix. Must be *n* x *d*.
Q	The state variance. A scalar, or vector of length *d*, or a *d* x *d* matrix. When scalar, Q is constant diagonal. When a vector, Q is diagonal.
R	The measurement variance. A scalar, or vector of length *n*, or a *n* x *n* matrix. When scalar, R is constant diagonal. When a vector, R is diagonal.
P0	Initial *a priori* prediction error.
epsilon	A small scalar number.
verbosity	0, 1 or 2.

Details

Note: The test for convergence has been (very, very slightly) modified since v0.5.1. The current test has been implemented for rigor. Users who have results based on earlier releases may observe infinitesimal differences in the resulting prediction error.

Value

A named list.

P.apri	A *d* x *d* matrix giving *a priori* prediction variance.
P.apos	A *d* x *d* matrix giving *a posteriori* prediction variance.

Examples

```
H <- matrix(1)

SS.stst(1, H, 1, 1, P0=10^5, epsilon=10^(-14), verbosity=1)
```

`SS.stst.SMW` *Steady State using the Woodbury matrix identity*

Description

Find steady state of system, i.e., locate when Kalman gain converges

Usage

```
SS.stst.SMW(F, H, Q, inv.R, P0, epsilon, verbosity=0)
```

Arguments

`F`	The state matrix. A scalar, or vector of length *d*, or a *d* x *d* matrix. When scalar, `F` is constant diagonal. When a vector, `F` is diagonal.
`H`	The measurement matrix. Must be *n* x *d*.
`Q`	The state variance. A scalar, or vector of length *d*, or a *d* x *d* matrix. When scalar, `Q` is constant diagonal. When a vector, `Q` is diagonal.
`inv.R`	The inverse of the measurement variance. A scalar, or vector of length *n*, or a *n* x *n* matrix. When scalar, `inv.R` is constant diagonal. When a vector, `inv.R` is diagonal.
`P0`	Initial *a priori* prediction error.
`epsilon`	A small scalar number.
`verbosity`	0, 1 or 2.

Details

Spiritually identical to `SS.stst`, except that the Woodbury identity is used for inversion. This method offers a computationally reduced means of finding the system steady state; however, this method must be supplied with the inverse of the measurement variance matrix, R – not R. Try comparing the example below with the evivalent example offered for `SS.stst`.

Value

A named list.

`P.apri`	A *d* x *d* matrix giving *a priori* prediction variance.
`P.apos`	A *d* x *d* matrix giving *a posteriori* prediction variance.

Examples

```
H <- matrix(1)

SS.stst.SMW(1, H, 1, 1, P0=10^5, epsilon=10^(-14), verbosity=1)
```

SS.stst.tv *Steady State*

Description

Find steady state of time-varying system, i.e., locate when Kalman gain converges

Usage

```
SS.stst.tv(F, H, Q, R, P0, epsilon, verbosity=0)
```

Arguments

F	A list of *d* x *d* matrices.
H	A list of *n* x *d* matrices.
Q	A list of *d* x *d* matrices.
R	A list of *n* x *n* matrices.
P0	Initial *a priori* prediction error.
epsilon	A small scalar number.
verbosity	0, 1 or 2.

Details

Note: The test for convergence has been (very, very slightly) modified since v0.5.1. The current test has been implemented for rigor. Users who have results based on earlier releases may observe infinitesimal differences in the resulting prediction error.

Value

A named list.

P.apri	A *d* x *d* matrix giving *a priori* prediction variance.
P.apos	A *d* x *d* matrix giving *a posteriori* prediction variance.

Examples

```
F.tv <- list()
for(i in 1:10000) {
F.tv[[i]] <- diag( c(1/(i+10), 1/(i+10)) )
}

H <- matrix(1, 2, 2)

SS.stst.tv(F.tv, H, 1, 1, 10^5, 10^(-10), verbosity=2)
```

SS_O3 *California Ozone*

Description

Daily airborne Ozone concentrations (ppb) over California, 68 fixed sensors, 2005-2006

Usage

```
data(SS_O3)
```

Format

The format is:

List of 2

$Z :'data.frame': 730 obs. of 68 variables: Ozone ppb for 68 sensors.

$locs :'data.frame': 68 obs. of 2 variables: longitude, latitude for 68 sites.

Source

The Ozone data originates from the California Air Resources Board (CARB). The interpolation grid elevations originate from the Google Elevation API.

Examples

```
data(SS_O3)
```

Index