

# Package ‘sspir’

February 15, 2012

**Title** State Space Models in R

**Version** 0.2.8

**Date** 2009-11-19

**Encoding** latin1

**Author** Claus Dethlefsen, Soren Lundbye-Christensen and Anette Luther Christensen

**Description** A glm-like formula language to define dynamic generalized linear models (state space models). Includes functions for Kalman filtering and smoothing. Estimation of variance matrices can be performed using the EM algorithm in case of Gaussian models. Read help(sspir) to get started.

**Maintainer** Claus Dethlefsen <cld@rn.dk>

**URL** <http://www.math.aau.dk/~dethlef/sspir>

**License** GPL (>= 2)

**Depends** R (>= 2.1.0),MASS,mvtnorm

**Repository** CRAN

**Date/Publication** 2009-11-19 13:00:55

## R topics documented:

EMalgo . . . . .	2
extended . . . . .	4
kfilter . . . . .	5
kfs . . . . .	6
ksimulate . . . . .	7
kurit . . . . .	8
mumps . . . . .	9
recursion . . . . .	9
smoother . . . . .	10

Specials . . . . .	11
SS . . . . .	12
ssm . . . . .	16
sspir . . . . .	18
vandrivars . . . . .	20

<b>Index</b>	<b>21</b>
--------------	-----------

---

EMalgo	<i>Estimation of variance matrices in a Gaussian state space model</i>
--------	--

---

### Description

Estimates variance matrices of the observation- and latent process in a Gaussian state space model given as input, using the EM algorithm.

### Usage

```
EMalgo(ss, epsilon = 1e-06, maxiter = 50, Vstruc =
      function(V) { return(V)}, Wstruc = function(W) {
      return(W)}, print.ite = F)
```

### Arguments

ss	an object of class <a href="#">SS</a> .
epsilon	a (small) positive numeric giving the tolerance of the maximum relative differences of Vmat and Wmat between iterations.
maxiter	a positive integer giving the maximum number of iterations to run.
Vstruc	a function specifying the structure of the variance matrix of the observation model if such is available.
Wstruc	a function specifying the structure of the variance matrix of the state model if such is available.
print.ite	A logical specifying whether information should be printed after each iteration.

### Details

The initial variance matrices are to be specified in the model specification and structures of the variance matrices may be specified by the user by the functions Vstruc and Wstruc. As default these are assigned NA, if not specified they are not estimated, hence it is possible to estimate only the observation variance matrix or the evolution variance matrix. The EM algorithm requires the variance matrices to be estimated are constant, however if a variance matrix is not be estimated it may be non-constant.

The output provided by the function is the smoothed Gaussian model along with the estimated variance matrices, maximum values of the log likelihood function for each iterations and the number of iteration upon convergence.

**Value**

ss	the value from smoother.
Vmat.est	the estimate of the observation variance matrix, which is provided if the input of Wstruc is of class function, otherwise as input of Vmat.
Wmat.est	the estimate of the observation variance matrix, which is provided if the input of Wstruc is of class function, otherwise as input of Wmat.
loglik	maximum value of log likelihood function for each iteration.
iteration	number of iterations upon convergence.

**Author(s)**

Anette Luther Christensen

**See Also**

[kfilter](#), [smoother](#), [recursion](#).

**Examples**

```

m1 <- SS(

Fmat = function(tt,x,phi) {
Fmat      <- matrix(NA,3,1)
Fmat[1,1] <- 1
Fmat[2,1] <- cos(2*pi*tt/12)
Fmat[3,1] <- sin(2*pi*tt/12)
return(Fmat) },

Gmat = function(tt,x,phi) {
return(matrix(c(1,0,0,0,1,0,0,0,1),nrow=3)) },

Wmat = matrix(c(0.01,0,0,0,0.1,0,0,0,0.1),nrow=3),
Vmat = matrix(1),
m0   = matrix(c(0,0,0),nrow=1),
C0   = matrix(c(1,0,0,0,0.001,0,0,0,0.001),nrow=3,ncol=3)
)

m1 <- recursion(m1,100)

Wstruc <- function(W){
W[1,2:3]<-0
W[2:3,1]<-0
W[2,2]<-(W[2,2]+W[3,3])/2
W[3,3]<-W[2,2]
W[2,3]<-0
W[3,2]<-0
return(W)}

estimates <- EMalgo(m1,Wstruc=Wstruc)

```

```
plot(estimates$model)
```

---

extended

*Iterated Extended Kalman Smoothing*

---

## Description

An iterative procedure for calculation of the conditional mean and variance of the latent process in non-Gaussian state space models. The method calculates an approximating Gaussian state space model.

## Usage

```
extended(ss, maxiter = 50, epsilon = 1e-06, debug = FALSE)
```

## Arguments

<code>ss</code>	an object of class <a href="#">SS</a> .
<code>maxiter</code>	a positive integer giving the maximum number of iterations to run.
<code>epsilon</code>	a (small) positive numeric giving the tolerance of the maximum relative differences of $m$ and $C$ between iterations.
<code>debug</code>	a logical. If TRUE, some extra information is printed.

## Details

This is the default method when using [kfs](#) on an object of class [ssm](#) when the family is not gaussian. The conditional mean and variance can be retrieved using [getFit](#) and are then stored in the attributes `m` and `C`, respectively.

## Value

The object `ss` with updated components `m`, `C`, `loglik`, `iteration`, `ytilde`, `x$vtilde`, `mu`. These describe the approximating Gaussian state space model.

## Author(s)

Claus Dethlefsen and Søren Lundbye-Christensen.

## References

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

## See Also

[ssm](#), [kfilter](#), [smoother](#).

**Examples**

```

data(mumps)
index <- 1:length(mumps) # use 'index' instead of time
model <- ssm( mumps ~ -1 + tvar(polytime(index,1)),
              family=poisson(link=log))
results <- getFit(model)
plot(mumps,type='l',ylab='Number of Cases',xlab='',axes=FALSE)
lines( exp(results$m[,1]), lwd=2)
## Alternatives:
## results2 <- extended(model$ss)
## results3 <- kfs(model) ## yields the same

```

---

kfilter

*Kalman filter for Gaussian state space model*


---

**Description**

From an SS object, runs the Kalman filter to produce the conditional means and variances of the state vectors given the current time point.

**Usage**

```
kfilter(ss)
```

**Arguments**

ss                    object of class [SS](#).

**Details**

The Kalman filter yields the distribution

$$(\theta_t | y_1, \dots, y_t) \sim N(m_t, C_t)$$

through the recursion for  $t = 1, \dots, n$ ,

$$\begin{aligned}
 a_t &= G_t m_{t-1} \\
 R_t &= G_t C_{t-1} G_t^T + W_t \\
 f_t &= F_t^T a_t \\
 Q_t &= F_t^T R_t F_t + V_t \\
 e_t &= y_t - f_t \\
 A_t &= R_t F_t Q_t^{-1} \\
 m_t &= a_t + A_t e_t \\
 C_t &= R_t - A_t Q_t A_t^T
 \end{aligned}$$

Also, the log-likelihood is calculated.

**Value**

An object of class `SS` with the components `m`, `C`, and `loglik` updated.

**Author(s)**

Claus Dethlefsen and Søren Lundbye-Christensen.

**See Also**

`SS`, `smoother`

**Examples**

```
data(kurit)
m1 <- SS(kurit)
phi(m1) <- c(100,5)
m0(m1) <- matrix(130)
C0(m1) <- matrix(400)

m1.f <- kfilter(m1)
plot(m1$y)
lines(m1.f$m, lty=2)
```

---

kfs

*(Iterated extended) Kalman smoother*

---

**Description**

Runs the Kalman smoother on an `ssm` object if the family-attribute of the object is `gaussian`. If not, the iterated extended Kalman smoother is used.

**Usage**

```
kfs(ss, ...)
```

**Arguments**

`ss` an object of class `ssm`.  
`...` further arguments passed to the Kalman smoother.

**Details**

The function is a wrapper for `kfilter` followed by `smoother` in the Gaussian case. For other family types, `extended` is called. The `kfs` function is called by default in the call to `ssm` unless the option `fit=FALSE` is given.

**Value**

The returned value from either `smoother` (Gaussian case) or `extended`.

**Author(s)**

Claus Dethlefsen and Søren Lundbye-Christensen.

**See Also**

[kfilter](#), [smoother](#), [extended](#), [ssm](#).

**Examples**

```
data(mumps)
index <- 1:length(mumps)
phi.start <- c(0,0,0.0005,0.0001)
m3 <- ssm( mumps ~ -1 + tvar(polytime(index,1)) +
           tvar(polytrig(index,12,1)),
           family=poisson(link=log),
           phi=phi.start, C0 = diag(4),
           fit=FALSE
         )

## The option "fit=FALSE" means that the Kalman Filter/Smoother is not
## run.
## At this point you may inspect/change the setup before running 'kfs'
C0(m3)
C0(m3) <- 10*diag(4)
## incorporate possible structural 'jump' at timepoint 10
Wold <- Wmat(m3)
Wmat(m3) <- function(tt,x,phi) {
  W <- Wold(tt,x,phi)
  if (tt==10) {W[2,2] <- 100*W[2,2]; return(W)}
  else return(W)
}

m3.fit <- kfs(m3)

plot(mumps,type='l',ylab='Number of Cases',xlab='')
lines(exp(m3.fit$m[,1]),type='l',lwd=2)
```

---

ksimulate

*Forwards filtering Backwards sampling*


---

**Description**

Draws  $N$  samples from the conditional distribution of  $\theta_1, \dots, \theta_n$  given  $y_1, \dots, y_n$  in a Gaussian state space model.

**Usage**

```
ksimulate(ss, N = 1)
```

**Arguments**

`ss` an object of class `SS` with components `m` and `C` obtained from running `kfilter`.  
`N` an integer giving the number of simulated realizations wanted.

**Value**

An array of dimension  $n \times p \times N$ . The  $i$ 'th simulated state vector at time  $t$  is located in `[t, , i]`.

**Author(s)**

Claus Dethlefsen and Søren Lundbye-Christensen.

**References**

C.K. Carter and R. Kohn (1994). On Gibbs Sampling for State Space Models. *Biometrika*, 3, 541-553.

**See Also**

`ssm`, `kfilter`, `smoother`.

**Examples**

```
data(kurit)
m1 <- SS(kurit)
phi(m1) <- c(100,5)
m0(m1) <- matrix(130)
C0(m1) <- matrix(400)

m1 <- kfilter(m1)
m1.s <- smoother(m1)
sim <- ksimulate(m1,10)

plot(kurit)
for (i in 1:10) lines(sim[, ,i],lty=2,col=2)

lines(smoother(m1)$m,lwd=2)
```

---

kurit

*Kurit*

---

**Description**

A hypothetical example from West and Harrison, page 40.

**Usage**

```
data(kurit)
```

**Format**

A time series.

---

mumps

*Mumps*

---

**Description**

Monthly registered cases of mumps in New York City, January 1928 through June 1972.

**Usage**

data(mumps)

**Format**

534 observations

**mumps** a numeric vector of registred cases.

**References**

Hipel KW, McLeod IA (1994). Time Series Modeling of Water Resources and Environmental Systems. Elsevier Science Publishers B.V. (North-Holland).

---

recursion

*Simulate from a Gaussian state space model*

---

**Description**

Draw a sample of  $y_1, \dots, y_n$ , and  $\theta_1, \dots, \theta_n$  from the state space model given as input.

**Usage**

recursion(ss, n)

**Arguments**

ss an object of class [SS](#) defining the state space model.

n an integer specifying the number of time steps to be simulated.

**Value**

The object ss with updated components y, mu and truetheta.

**Author(s)**

Claus Dethlefsen and Søren Lundbye-Christensen.

**See Also**

[ssm](#), [kfilter](#), [smoother](#), [getFamily](#).

**Examples**

```
data(kurit)
m1 <- SS(kurit)
phi(m1) <- c(100,5)
m0(m1) <- matrix(130)
C0(m1) <- matrix(400)

par(mfrow=c(2,1))
plot(recursion(m1,100))
phi(m1) <- c(5,100)
plot(recursion(m1,100))
```

---

 smoother

*Kalman smoother for Gaussian state space model*


---

**Description**

Based on the output from [kfilter](#), this function runs the Kalman smoother to produce the conditional means and variances of the state vectors given all observations.

**Usage**

```
smoother(ss)
```

**Arguments**

`ss` object of class [SS](#), where the components `m` and `C` have been set by the [kfilter](#).

**Details**

The Kalman smoother yields the distribution

$$(\theta_t | y_1, \dots, y_n) \sim N(m_t^*, C_t^*)$$

through the backward recursion for  $t = n..1$ ,

$$R_{t+1} = G_{t+1} C_t G_{t+1}^T + W_{t+1}$$

$$B_t = C_t G_{t+1}^T R_{t+1}^{-1}$$

$$m_t^* = m_t + B_t (m_{t+1}^* - G_{t+1} m_t)$$

$$C_t^* = C_t + B_t(C_{t+1}^* - R_{t+1})B_t^T$$

where the matrices  $F$ ,  $G$ ,  $V$ ,  $W$  are stored in the SS object as functions, eg. `Fmat(tt,x,phi)`, see [SS](#). The vectors  $m$  and matrices  $C$  are set by the [kfilter](#) and are overwritten by the Kalman smoother.

The smoother also calculates the signal,  $\mu_t = F_t^T m_t^*$ .

### Value

An object of class [SS](#) with the components `m`, `C`, and `mu` updated.

### Author(s)

Claus Dethlefsen and Søren Lundbye-Christensen.

### See Also

[SS](#), [kfilter](#)

### Examples

```
data(kurit)
m1 <- SS(kurit)
phi(m1) <- c(100,5)
m0(m1) <- matrix(130)
C0(m1) <- matrix(400)

m1.s <- smoother(kfilter(m1))
plot(m1$y)
lines(m1.s$m,lty=2)
```

---

Specials

*Special functions used in ssm formulas*

---

### Description

Auxiliary function for modelling the time-dependent trend patterns when formulating [ssm](#) models.

### Usage

```
polytrig(time, period = 365.25, degree = 1)
polytime(time, degree = 1)
sumseason(time, period = 12)
season(time, period = 12)
```

**Arguments**

time	a vector giving the observation times, eg. 1:n. The variable 'time' is by default set to time(y), where y is the response in the formula in the <code>ssm</code> call.
period	the period of the seasonal pattern.
degree	the degree of the trigonometric polynomial or the degree of the polynomial time trend.

**Details**

`polytrig` defines a trigonometric polynomial in time, `polytime` defines a polynomial time trend, `sumseason` produces a seasonal pattern in which the factors sum to zero over the period, and `season` defines a factor. The functions `polytrig` and `season` can be used outside the `ssm` call. However, `polytime` and `sumseason` are treated mainly inside the `ssm` call.

**Author(s)**

Claus Dethlefsen and Søren Lundbye-Christensen.

**See Also**

[ssm](#), [kfilter](#), [smoother](#), [getFamily](#).

**Examples**

```
polytrig(1:10,degree=2)
season(1:12,period=4)
```

---

SS

*Representation of Gaussian State Space Model*


---

**Description**

Creates an SS-object describing a Gaussian state space model.

**Usage**

```
SS(y = NA, x = list(x = NA),
  Fmat = function(tt,x,phi) { return(matrix(1)) },
  Gmat = function(tt,x,phi) { return(matrix(1)) },
  Vmat = function(tt,x,phi) { return(matrix(phi[1])) },
  Wmat = function(tt,x,phi) { return(matrix(phi[2])) },
  m0 = matrix(0),
  C0 = matrix(100),
  phi = c(1,1))

## S3 method for class 'SS'
C0(ssm)
```

```

## S3 method for class 'SS'
m0(ssm)
## S3 method for class 'SS'
Fmat(ssm)
## S3 method for class 'SS'
Gmat(ssm)
## S3 method for class 'SS'
Vmat(ssm)
## S3 method for class 'SS'
Wmat(ssm)
## S3 method for class 'SS'
phi(ssm)
## S3 replacement method for class 'SS'
C0(ssm) <- value
## S3 replacement method for class 'SS'
m0(ssm) <- value
## S3 replacement method for class 'SS'
Fmat(ssm) <- value
## S3 replacement method for class 'SS'
Gmat(ssm) <- value
## S3 replacement method for class 'SS'
Vmat(ssm) <- value
## S3 replacement method for class 'SS'
Wmat(ssm) <- value
## S3 replacement method for class 'SS'
phi(ssm) <- value

```

### Arguments

<code>y</code>	a matrix giving a multivariate time series of observations. The observation at time <code>tt</code> is <code>y[tt, ]</code> . The dimension of <code>y</code> is $n \times d$ . Preferably, <code>y</code> is a <code>ts</code> object.
<code>x</code>	a list of entities (eg. covariates) passed as argument to the functions <code>Fmat</code> , <code>Gmat</code> , <code>Vmat</code> , and <code>Wmat</code> .
<code>Fmat</code>	a function depending on the parameter-vector <code>phi</code> , covariates <code>x</code> and returns the $p \times d$ design matrix at time <code>tt</code> .
<code>Gmat</code>	a function depending on the parameter-vector <code>phi</code> , covariates <code>x</code> and returns the $p \times p$ evolution matrix at time <code>tt</code> .
<code>Vmat</code>	a function depending on the parameter-vector <code>phi</code> , covariates <code>x</code> and returns the $d \times d$ (positive definit) variance matrix at time <code>tt</code> . The matrix <code>Vmat</code> may be specified as a constant matrix not depending on <code>phi</code> coded with <code>matrix</code> .
<code>Wmat</code>	a function depending on the parameter-vector <code>phi</code> , covariates <code>x</code> and returns the $p \times p$ (positive semidefinite) evolution variance matrix at time <code>tt</code> . The matrix <code>Wmat</code> may be specified as a constant matrix not depending on <code>phi</code> coded with <code>matrix</code> .
<code>m0</code>	a $1 \times p$ matrix giving the initial state.
<code>C0</code>	a $p \times p$ variance matrix giving the variance matrix of the initial state.

phi	a (hyper) parameter vector passed as argument to the functions Fmat, Gmat, Vmat, and Wmat.
ssm	an object of class SS.
value	an object to be assigned to the element of the state space model.

### Details

The state space model is given by

$$Y_t = F_t^T \theta_t + v_t, v_t \sim N(0, V_t)$$

$$\theta_t = G_t \theta_{t-1} + w_t, w_t \sim N(0, W_t)$$

for  $t = 1, \dots, n$ . The matrices  $F_t$ ,  $G_t$ ,  $V_t$ , and  $W_t$  may depend on a parameter vector  $\phi$ . The initialization is given as

$$\theta_0 \sim N(m_0, C_0).$$

### Value

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

y	as input.
x	as input.
Fmat	as input.
Gmat	as input.
Vmat	as input.
Wmat	as input.
m0	as input.
C0	as input.
phi	as input.
n	the number of time points
d	the dimension of each observation.
p	the dimension of the state vector at each timepoint.
ytilde	adjusted observations for use in the extended Kalman filter, see <a href="#">extended</a> .
iteration	an integer giving the number of iterations used in the extended Kalman filter, see <a href="#">extended</a> .
m	after Kalman filtering (or smoothing), holds the conditional mean of the state vectors given the observations up till time $t$ (filtering) or all observations (smoothing). This is organised in a $n \times p$ dimensional matrix holding $m_t$ ( $m_t^*$ ) in rows. Is returned as a <b>ts</b> object.
C	after Kalman filtering (or smoothing), holds the conditional variance of the state vectors given the observations up til time $t$ (filtering) or all observations (smoothing). This is organised in a list holding the $p \times p$ dimensional matrices $C_t$ ( $C_t^*$ ).
mu	after Kalman smoothing, holds the conditional mean of the signal ( $\mu_t = F_t^T \theta_t$ ) given all observations. This is organised in a $n \times d$ dimensional matrix holding $\mu_t$ in rows.
loglik	the log-likelihood value after Kalman filtering.

**Author(s)**

Claus Dethlefsen, Søren Lundbye-Christensen and Anette Luther Christensen

**See Also**

[ssm](#) for a glm-like interface of specifying models, [kfilter](#) for Kalman filter and [smoother](#) for Kalman smoother.

**Examples**

```

data(kurit) ## West & Harrison, page 40
m1 <- SS(y=kurit,
        Fmat=function(tt,x,phi) return(matrix(1)),
        Gmat=function(tt,x,phi) return(matrix(1)),
        Wmat=function(tt,x,phi) return(matrix(5)), ## Alternatively Wmat=matrix(5)
        Vmat=function(tt,x,phi) return(matrix(100)), ## Alternatively Vmat=matrix(100)
        m0=matrix(130),C0=matrix(400)
        )

plot(m1$y)
m1.f <- kfilter(m1)
m1.s <- smoother(m1.f)
lines(m1.f$m,lty=2,col=2)
lines(m1.s$m,lty=2,col=2)

## make a model with an intervention at time 10
m2 <- m1
Wmat(m2) <- function(tt,x,phi) {
  if (tt==10) return(matrix(900))
  else return(matrix(5))
}

m2.f <- kfilter(m2)
m2.s <- smoother(m2.f)
lines(m2.f$m,lty=2,col=4)
lines(m2.s$m,lty=2,col=4)

## Use 'ssm' to construct an SS skeleton
phi.start <- StructTS(log10(UKgas),type="BSM")$coef[c(4,1,2,3)]
gasmodel <- ssm( log10(UKgas) ~ -1+
                tvar(polytime(time,1))+
                tvar(sumseason(time,12)),
                phi=phi.start)

m0(gasmodel)
C0(gasmodel)
phi(gasmodel)

fit <- getFit(gasmodel)
plot( fit$m[,1:3] )

```

---

 ssm

*Define state-space model in a glm-style call.*


---

## Description

Use a glm-style formula and family arguments to setup a state space model.

## Usage

```

ssm(formula, family = gaussian, data = list(), subset =
    NULL, fit = TRUE, phi = NULL, m0 = NULL, C0 = NULL,
        Fmat = NULL, Gmat = NULL, Vmat = NULL, Wmat = NULL)
## S3 method for class 'ssm'
C0(ssm)
## S3 method for class 'ssm'
m0(ssm)
## S3 method for class 'ssm'
Fmat(ssm)
## S3 method for class 'ssm'
Gmat(ssm)
## S3 method for class 'ssm'
Vmat(ssm)
## S3 method for class 'ssm'
Wmat(ssm)
## S3 method for class 'ssm'
phi(ssm)
## S3 replacement method for class 'ssm'
C0(ssm) <- value
## S3 replacement method for class 'ssm'
m0(ssm) <- value
## S3 replacement method for class 'ssm'
Fmat(ssm) <- value
## S3 replacement method for class 'ssm'
Gmat(ssm) <- value
## S3 replacement method for class 'ssm'
Vmat(ssm) <- value
## S3 replacement method for class 'ssm'
Wmat(ssm) <- value
## S3 replacement method for class 'ssm'
phi(ssm) <- value
getFit(ssm)

```

## Arguments

formula	a formula with univariate response on the lefthand side. The righthand side is a sum of terms and the special functions <a href="#">sumseason</a> , <a href="#">polytime</a> , <a href="#">polytrig</a> , and <a href="#">season</a> can be used. Terms can be marked by the tvar-function to create a term
---------	---

	with time-varying coefficients. A special case is <code>tvar(1)</code> meaning a random walk.
<code>family</code>	a description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. (See <a href="#">family</a> for details of family functions.)
<code>data</code>	an optional data frame containing the variables in the model. If not found in <code>data</code> , the variables are taken from <code>environment(formula)</code> , typically the environment from which <code>ssm</code> is called.
<code>subset</code>	an optional vector specifying a subset of observations to be used in the fitting process.
<code>ssm</code>	an object of class <code>ssm</code> .
<code>fit</code>	a logical. If <code>TRUE</code> , the model is fit using the <code>kfs</code> method. Otherwise, the model is returned as is.
<code>phi</code>	a vector of initial values of hyperparamters. Note that these have to be in the right order. Best advice is to leave this option to be <code>NULL</code> and then inspect the returned result using <code>phi(ssm)</code> .
<code>m0</code>	a vector with the initial state vector.
<code>C0</code>	a matrix with the variance matrix of the initial state.
<code>Fmat</code>	a function giving the regression matrix at a given timepoint.
<code>Gmat</code>	a function giving the evolution matrix at a given timepoint.
<code>Wmat</code>	a function giving the evolution variance matrix at a given timepoint.
<code>Vmat</code>	a function giving the observation variance matrix at a given timepoint.
<code>value</code>	an object to be assigned to the element of the state space model.

**Value**

An object of class `ssm` with the following components

<code>ss</code>	an object of class <code>SS</code> describing the state space model. In addition, the <code>ss</code> object contains the components <code>family</code> and <code>ntotal</code> (for binomial case).
-----------------	---

**Author(s)**

Claus Dethlefsen and Søren Lundbye-Christensen.

**See Also**

[SS](#), [extended](#)

**Examples**

```
data(vanddrivers)
vanddrivers$y <- ts(vanddrivers$y, start=1969, frequency=12)
vd.time <- time(vanddrivers$y)
vd <- ssm( y ~ tvar(1) + seatbelt + sumseason(vd.time,12),
          family=poisson(link="log"),
```

```

        data=vanddrivers,
        phi = c(1,0.0004),
        C0=diag(13)*100,
        fit=FALSE
      )
phi(vd)["(Intercept)"] <- exp(- 2*3.703307 )
C0(vd) <- diag(13)*1000
vd.res <- kfs(vd)

plot( vd.res$m[,1:3] )

attach(vanddrivers)
plot(y,ylim=c(0,20))
lines(exp(vd.res$m[,1]+vd.res$m[,2]*seatbelt),lwd=2 )
detach(vanddrivers)

```

---

sspir

*State Space Models in R*


---

## Description

The main contribution of this package is to give a formula language for specifying dynamic generalized linear models. That is, an extension of glm formulae by marking terms with `tvar` to specify that their coefficients are time-varying. The package also provides (extended) Kalman filter and Kalman smoother for models within Gaussian, Poisson and binomial families. To get started, try `demo(gas)`, `demo(vanddrivers)` and `demo(mumps)`.

Also, read the paper "C. Dethlefsen and S. Lundbye-Christensen. Formulating state space models in r with focus on longitudinal regression models. *Journal of Statistical Software*, 16(1), 2006."

## What `sspir` does not include

- Optimization To keep full generality, the Kalman filter and smoother use the `SS` functions `Fmat`, `Gmat`, `Vmat`, `Wmat`. The special cases where these matrices are time-invariant, the algorithms can be considerably speeded up by implementing these special cases eg. in C. For simple models, see [StructTS](#).
- Diffuse initialization We use  $m_0$  and  $C_0$  as initialization of the state process. This may cause numeric problems which may be solved using diffuse initialization (see Durbin and Koopman (2000)).
- ARIMA models These are not directly supported, but since they can be expressed as state space models, it is possible to specify them as `SS` objects.
- Importance sampling We use iterated extended Kalman smoothing and use the likelihood from the approximating Gaussian state space model. This may be improved using importance sampling.
- Multivariate observations We have plans including methods for combining `ssm` objects for different time-series so that they can possibly share components in the latent process.

- Multi-process models We have plans for including methods for combining `ssm` objects defining different models for the same time-series. Providing prior probabilities for these models, it is possible to calculate posterior probabilities for the models, thereby discriminating between the models.
- Markov chain Monte Carlo A state space model may be part of a hierarchical model giving priors on hyper-parameters. Inference may be done using MCMC methods and for the state space model, the Forwards filtering, Backwards sampling method may be used. We plan to include at least some support for this.
- Missing values Missing values are not allowed in the covariates.
- Family Currently, only three combinations of distribution/link functions are supported: Gaussian/identity, Poisson/log, Binomial/logit. To add new variations, edit the function `getFamily`

### Other software

- Ox/SsfPack SsfPack is a package for Ox by SJ Koopman, N Shephard and JA Doornik. Multivariate Gaussian state space models with some support for non-Gaussian models. <http://www.ssfpack.com/>. Closely related to S+FinMetrics, a package for S-Plus, developed by Insightful. <http://www.insightful.com/products/finmetrics/default.asp>.
- StructTS StructTS by BD Ripley is a part of R. Univariate Gaussian state space models of the class structural time series models.
- KEE Kalman Estimating Equations is a package for S-Plus by SJ Knudsen. Poisson-Tweedie log-linear State-Space model. <http://www.statdem.sdu.dk/~sjk/kee/index.html>.
- IEKS IEKS library is a package for S-Plus and R by B Klein. Implementation of the iterated extended Kalman filter and smoother to discrete data from an exponential family. The latent process is assumed to be linear and Gaussian. <http://genetics.agrsci.dk/~bmk/>.
- Matlab Kalman filter toolbox for Matlab by K Murphy. <http://www.ai.mit.edu/~murphyk/Software/Kalman/kalman.html>.
- Bats BATS software by A Pole, M West and J Harrison. <http://www.isds.duke.edu/~mw/bats.html>.
- dse Dynamic Systems Estimation library (dse1 and dse2) are packages for R by P Gilbert. Multivariate Gaussian state space models with focus on ARMA models. See <http://www.bank-banque-canada.ca/pgilbert>.
- CTSM Continuous Time Stochastic Modelling (CTSM) is a stand alone program by NR Kristensen, LE Christiansen and H Madsen. Stochastic differential equations, linear and non-linear Gaussian state space models. <http://www.imm.dtu.dk/ctsm/>.

Probably this list is incomplete. Feel free to contribute with more links to packages.

### Author(s)

Claus Dethlefsen and Søren Lundbye-Christensen.

### See Also

[ssm](#)

---

vandriviers

*Vandriviers*

---

**Description**

The monthly numbers of light goods van drivers killed in road accidents, from January 1969 to December 1984

**Usage**

```
data(vandriviers)
```

**Format**

A data frame with 192 observations on the following variables.

**y** a numeric vector giving the number of deaths.

**seatbelt** a numeric vector with values 0 before February 1983 and 1 after. This indicates when the seat belt legislation law was introduced.

# Index

## \*Topic **datasets**

kurit, 8  
mumps, 9  
vandriviers, 20

## \*Topic **models**

EMalgo, 2  
extended, 4  
kfilter, 5  
kfs, 6  
ksimulate, 7  
recursion, 9  
smoother, 10  
Specials, 11  
SS, 12  
ssm, 16  
sspir, 18

C0 (ssm), 16

C0.SS (SS), 12

C0<- (ssm), 16

C0<- .SS (SS), 12

EMalgo, 2

extended, 4, 6, 7, 14, 17

family, 17

filterstep (kfilter), 5

Fmat (ssm), 16

Fmat.SS (SS), 12

Fmat<- (ssm), 16

Fmat<- .SS (SS), 12

getFamily, 10, 12

getFit, 4

getFit (ssm), 16

Gmat (ssm), 16

Gmat.SS (SS), 12

Gmat<- (ssm), 16

Gmat<- .SS (SS), 12

kfilter, 3, 4, 5, 6–8, 10–12, 15

kfilter.ssm (ssm), 16

kfs, 4, 6, 17

ksimulate, 7

kurit, 8

m0 (ssm), 16

m0.SS (SS), 12

m0<- (ssm), 16

m0<- .SS (SS), 12

mumps, 9

phi (ssm), 16

phi.SS (SS), 12

phi<- (ssm), 16

phi<- .SS (SS), 12

plot.SS (SS), 12

polytime, 16

polytime (Specials), 11

polytrig, 16

polytrig (Specials), 11

print.extended (extended), 4

print.Smoothed (smoother), 10

print.SS (SS), 12

print.ssm (ssm), 16

recursion, 3, 9

season, 16

season (Specials), 11

smoother, 3, 4, 6–8, 10, 10, 12, 15

smoother.ssm (ssm), 16

smootherstep (smoother), 10

Specials, 11

SS, 2, 4–6, 8–11, 12, 17, 18

SS-class (SS), 12

ssm, 4, 6–8, 10–12, 15, 16, 18, 19

sspir, 18

StructTS, 18

sumseason, 16

sumseason (Specials), 11

*ts*, 13, 14

vandriivers, 20

Vmat (ssm), 16

Vmat.SS (SS), 12

Vmat<- (ssm), 16

Vmat<-.SS (SS), 12

Wmat (ssm), 16

Wmat.SS (SS), 12

Wmat<- (ssm), 16

Wmat<-.SS (SS), 12