COMPUTATIONAL FINANCE & RISK MANAGEMENT
UNIVERSITY of WASHINGTON
Department of Applied Mathematics

Time Series Forecasting with State Space Models

Eric Zivot
University of Washington
Guy Yollin
University of Washington

Outline

1. Introduction to state space models and the dlm package
2. DLM estimation and forecasting examples
3. Structural time series models and StructTS
4. Exponential smoothing models and the forecast package
5. Time series cross validation
6. Summary

Lecture references

G. Petris, S. Petrone, and P. Campagnoli
Dynamic Linear Models with R.
Springer, 2009.

J. Durbin and S. J. Koopman
Time Series Analysis by State Space Methods.
Oxford University Press, 2001.

J. J. F. Commandeur and S. J. Koopman
An Introduction to State Space Time Series Analysis.
Oxford University Press, 2007.

Rob Hyndman
Forecasting with Exponential Smoothing: The State Space Approach.
Springer, 2008

Outline

1. Introduction to state space models and the dlm package
2. DLM estimation and forecasting examples
3. Structural time series models and StructTS
4. Exponential smoothing models and the forecast package
5. Time series cross validation
6. Summary

Linear-Gaussian State Space Models

Linear-Gaussian state space model

A linear-Gaussian *state space model* for an $m-$dimensional time series $\mathbf{y}_t$ consists of a *measurement equation* relating the observed data to an $p-$dimensional state vector θ_t, and a Markovian transition equation that describes the evolution of the state vector over time.

The *measurement equation* has the form

$$\underset{m\times 1}{\mathbf{y}_t} = \underset{(m\times p)}{\mathbf{F}_t}\underset{(p\times 1)}{\theta_t} + \underset{m\times 1}{\mathbf{v}_t}, \qquad \mathbf{v}_t \sim \text{iid } N(\mathbf{0}, \mathbf{V}_t)$$

The *transition equation* for the state vector θ_t is the first order Markov process

$$\underset{p\times 1}{\theta_t} = \underset{(p\times p)}{\mathbf{G}_t}\underset{(p\times 1)}{\theta_{t-1}} + \underset{(p\times 1)}{\mathbf{w}_t} \qquad \mathbf{w}_t \sim \text{iid } N(\mathbf{0}, \mathbf{W}_t)$$

$$E[\mathbf{v}_t\mathbf{w}_s'] = \mathbf{0} \text{ for all } s, t = 1, \ldots, T$$

Linear-Gaussian State Space Models

- The matrices $\mathbf{F}_t, \mathbf{V}_t, \mathbf{G}_t$,and $\mathbf{W}_t$ are called the *system matrices*, and contain non-random elements.
- If these matrices do not depend deterministically on t the state space system is called *time invariant*.
- Note: If $\mathbf{y}_t$ is covariance stationary, then the state space system will be time invariant.

Specification of Initial State Distribution

$$\theta_0 \sim N(\mathbf{m}_0, \mathbf{C}_0)$$

$$E[\mathbf{v}_t \theta_0'] = \mathbf{0}, \; E[\mathbf{w}_t \theta_0'] = \mathbf{0} \text{ for } t = 1, \ldots, T$$

- If some or all of the elements of θ_t are covariance stationary, then we can typically solve for the corresponding elements of $\mathbf{m}_0$ and $\mathbf{C}_0$ analytically from the elements of the system matrices
- For deterministic elements of θ (e.g., mean of a series), the corresponding element of $\mathbf{C}_0$ is defined to be zero.
- For non-stationary elements of θ, it is customary to set the corresponding element of $\mathbf{C}_0$ to a very large positive number, say 10^6.

Mean-zero covariance stationary AR(2) model

$$\text{ME}: y_t = c_t$$

$$\text{TE}: c_t = \phi_1 c_{t-1} + \phi_2 c_{t-2} + \eta_t,\ \eta_t \sim N(0, \sigma_\eta^2)$$

The state vector is $\theta_t = (c_t, \phi_2 c_{t-1})'$ and the transition equation is

$$\begin{pmatrix} c_t \\ \phi_2 c_{t-1} \end{pmatrix} = \begin{pmatrix} \phi_1 & 1 \\ \phi_2 & 0 \end{pmatrix} \begin{pmatrix} c_{t-1} \\ \phi_2 c_{t-2} \end{pmatrix} + \begin{pmatrix} \eta_t \\ 0 \end{pmatrix}$$

The transition equation system matrices are

$$\mathbf{G} = \begin{pmatrix} \phi_1 & 1 \\ \phi_2 & 0 \end{pmatrix},\ \mathbf{W} = \begin{pmatrix} \sigma_\eta^2 & 0 \\ 0 & 0 \end{pmatrix},\ \mathbf{w}_t = \begin{pmatrix} \eta_t \\ 0 \end{pmatrix}$$

Mean-zero covariance stationary AR(2) model

The *measurement equation* is

$$y_t = (1, 0)\theta_t$$

which has system matrices

$$\mathbf{F}_t = (1, 0), \;\; V = 0 \Rightarrow v_t = 0$$

Initial state distribution

$$\theta_0 \sim N(\mathbf{m}_0, \mathbf{C}_0)$$

Since $\theta_t = (c_t, \phi_2 c_{t-1})'$ is stationary, we find $\mathbf{m}_0$ using

$$\theta_0 = E[\theta_t] = \mathbf{G}E[\theta_{t-1}] + E[\mathbf{w}_t] = \mathbf{G}E[\theta_t]$$

$$\Rightarrow E[\theta_t](\mathbf{I}_2 - \mathbf{G}) = \mathbf{0}$$

$$\Rightarrow m_0 = E[\theta_t] = \mathbf{0}$$

Mean-zero covariance stationary AR(2) model

For the state variance, stationarity of $\theta_t = \mathbf{G}\theta_{t-1} + \mathbf{w}_t$ implies that for all t

$$\mathrm{var}(\theta_t) = \mathbf{G}\mathrm{var}(\theta_t)\mathbf{G}' + \mathrm{var}(\mathbf{w}_t) \Rightarrow$$

$$\mathbf{C}_0 = \mathbf{G}\mathbf{C}_0\mathbf{G}' + \mathbf{W}$$

Stacking columns via the $\mathrm{vec}(\cdot)$ operator then gives

$$\mathrm{vec}(\mathbf{C}_0) = (\mathbf{G} \otimes \mathbf{G})\,\mathrm{vec}(\mathbf{C}_0) + \mathrm{vec}(\mathbf{W})$$

$$\Rightarrow \mathrm{vec}(\mathbf{C}_0) = (\mathbf{I}_4 - \mathbf{G} \otimes \mathbf{G})^{-1}\mathrm{vec}(\mathbf{W})$$

Mean zero ARMA(1,1) model

$$\text{TE} : y_t = c_t$$

$$\text{ME} : c_t = \phi c_{t-1} + \eta_t + \theta \eta_{t-1}, \;\; \eta_t \sim N(0, \sigma_\eta^2)$$

Define $\theta_t = (c_t, \theta\eta_t)'$ and write

$$y_t = \begin{pmatrix} 1 & 0 \end{pmatrix} \theta_t$$

$$\begin{pmatrix} c_t \\ \theta\eta_t \end{pmatrix} = \begin{pmatrix} \phi & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} c_{t-1} \\ \theta\eta_{t-1} \end{pmatrix} + \begin{pmatrix} \eta_t \\ \theta\eta_t \end{pmatrix}$$

so that the system matrices are

$$\mathbf{F} = (1, 0), \;\; V = 0$$

$$\mathbf{G} = \begin{pmatrix} \phi & 1 \\ 0 & 0 \end{pmatrix}, \; \mathbf{W} = \sigma_\eta^2 \begin{pmatrix} 1 & \theta \\ \theta & \theta^2 \end{pmatrix}$$

Linear regression with time varying parameters

$$\text{ME}: y_t = \alpha_t + \beta_t x_t + v_t,\ \ v_t \sim N(0, \sigma_v^2)$$

$$\text{TE}: \alpha_t = \alpha_{t-1} + w_{\alpha,t},\ \ w_{\alpha,t} \sim N(0, \sigma_\alpha^2)$$

$$\text{TE}: \beta_t = \beta_{t-1} + w_{\beta,t},\ \ w_{\beta,t} \sim N(0, \sigma_\beta^2)$$

Define $\theta_t = (\alpha_t, \beta_t)'$

$$y_t = \begin{pmatrix} 1 & x_t \end{pmatrix} \theta_t + v_t$$

$$\begin{pmatrix} \alpha_t \\ \beta_t \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \alpha_{t-1} \\ \beta_{t-1} \end{pmatrix} + \begin{pmatrix} w_{\alpha,t} \\ w_{\beta,t} \end{pmatrix}$$

Linear regression with time varying parameters

The system matrices are

$$\mathbf{F}_t = \begin{pmatrix} 1 & x_t \end{pmatrix},\ V_t = \sigma_v^2,$$

$$\mathbf{G} = \mathbf{I}_2,\ \mathbf{W} = \begin{pmatrix} \sigma_\alpha^2 & 0 \\ 0 & \sigma_\beta^2 \end{pmatrix}$$

Notice that $\mathbf{F}_t$ is time varying.
Because the θ_t is non-stationary the initial distribution is

$$\theta_0 \sim N(\mathbf{m}_0, \mathbf{C}_0)$$

$$\mathbf{m}_0 = \mathbf{0},\ \mathbf{C}_0 = k \times \mathbf{I}_2,\ k = 10^7$$

Log-Normal Stochastic Volatility Model

$$r_t = \sigma_t u_t, \ u_t \sim N(0,1)$$

$$\ln \sigma_t = \omega + \phi \ln \sigma_{t-1} + \eta_t, \ \eta_t \sim N(0, \sigma_\eta^2), \ |\phi| < 1$$

Notice that $|r_t| = \sigma_t |u_t|$ so that

$$\ln |r_t| = \ln \sigma_t + \ln |u_t|$$

$$E[|u_t|] = -0.63518, \ var(|u_t|) = \pi^2/8$$

Hence

$$\ln |r_t| \ = -0.63518 + \ln \sigma_t + v_t, \ v_t \sim (0, \pi^2/8)$$

$$\ln \sigma_t = \omega + \phi \ln \sigma_{t-1} + \eta_t, \ \eta_t \sim N(0, \sigma_\eta^2)$$

Log-Normal Stochastic Volatility Model

Define $\theta_t = (-0.63518, \omega, \ln \sigma_t)'$. Then the state-space representation is

$$\text{ME} : \ln |r_t| = \begin{pmatrix} 1 & 0 & 1 \end{pmatrix} \theta_t + v_t$$

$$\text{TE} : \begin{pmatrix} -0.63518 \\ \omega \\ \ln \sigma_t \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & \phi \end{pmatrix} \begin{pmatrix} -0.63518 \\ \omega \\ \ln \sigma_{t-1} \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ \eta_t \end{pmatrix}$$

The system matrices are

$$\mathbf{F} = \begin{pmatrix} 1 & 0 & 1 \end{pmatrix}, \; V = \pi^2/8$$

$$\mathbf{G} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & \phi \end{pmatrix}, \; \mathbf{W} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \sigma_\eta^2 \end{pmatrix}$$

Log-Normal Stochastic Volatility Model

Because $\ln \sigma_t$ follows a stationary AR(1)

$$E[\ln \sigma_t] = \frac{\omega}{1-\phi}, \; var(\ln \sigma_t) = \frac{\sigma_\eta^2}{1-\phi^2}$$

The initial distribution is

$$\theta_0 \sim N(\mathbf{m}_0, \mathbf{C}_0)$$

$$\mathbf{m}_0 = \begin{pmatrix} -0.63518 \\ \omega \\ \omega/(1-\phi) \end{pmatrix}, \; \mathbf{C}_0 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{\sigma_\eta^2}{1-\phi^2} \end{pmatrix}$$

Note: in dlm, you can't have elements of $\mathbf{C}_0$ exactly zero; use very small number 1e-7 instead.

Specifying a State Space Model with the `dlm` package

- State space models in dlm are represented as lists with named components associated with the system matrices and initial value parameters

Model Parameter	**List Name**	**Time Varying Name**
$\mathbf{F}$	`FF`	`JFF`
$\mathbf{V}$	`V`	`JV`
$\mathbf{G}$	`GG`	`JFF`
$\mathbf{W}$	`W`	`JW`
$\mathbf{C}_0$	`C0`	
$\mathbf{m}_0$	`m0`	
data		`X`

$$\mathbf{y}_t = \mathbf{F}_t \theta_t + \mathbf{v}_t \qquad \mathbf{v}_t \sim NID(\mathbf{0}, \mathbf{V}_t)$$

$$\theta_t = \mathbf{G}_t \theta_{t-1} + \mathbf{w}_t \qquad \mathbf{w}_t \sim NID(\mathbf{0}, \mathbf{W}_t)$$

Specifying a State Space Model with the `dlm` package

Function	Model
`dlm`	generic DLM
`dlmModARMA`	ARMA process
`dlmModPoly`	*n*th order polynomial DLM
`dlmModReg`	Linear regression
`dlmModSeas`	Periodic – Seasonal factors
`dlmModTrig`	Periodic – Trigonometric form

Table: Functions to create dlm objects

Example Models: ARMA(1,1)

R Code: Create an ARMA model with DLM

```
> ########################################
> # EX.1 state space for ARMA(1,1)
> ########################################
> # ARMA(1,1) with phi=0.8, theta=0.2, sig2=1
> library(dlm)
> library(methods)
> arma11.dlm = dlmModARMA(ar=0.8, ma=0.2, sigma2=1)
> class(arma11.dlm)

[1] "dlm"

> names(arma11.dlm)

 [1] "m0"   "C0"   "FF"   "V"    "GG"   "W"    "JFF" "JV"   "JGG" "JW"
```

Example Models: ARMA(1,1)

R Code: Create an ARMA model with DLM

```
> arma11.dlm

$FF
     [,1] [,2]
[1,]    1    0

$V
     [,1]
[1,]    0

$GG
     [,1] [,2]
[1,]  0.8    1
[2,]  0.0    0

$W
     [,1] [,2]
[1,]  1.0 0.20
[2,]  0.2 0.04

$m0
[1] 0 0

$C0
      [,1]  [,2]
[1,] 1e+07 0e+00
[2,] 0e+00 1e+07
```

Example Models: Regression with time-varying parameters

R Code: TVP Regression Model

```
> library(PerformanceAnalytics)
> data(managers)
> # extract HAM1 and SP500 excess returns
> HAM1 = 100*(managers[,"HAM1", drop=FALSE] - managers[,"US 3m TR", drop=FALSE])
> sp500 = 100*(managers[,"SP500 TR", drop=FALSE] - managers[,"US 3m TR",
   drop=FALSE])
> colnames(sp500) = "SP500"
> s2v = 1
> s2a = 0.01
> s2b = 0.01
> tvp.dlm = dlmModReg(X=sp500, addInt=TRUE,
                      dV=s2v, dW=c(s2a, s2b))
```

Example Models: Regression with time-varying parameters

R Code: TVP Regression Model

```
> tvp.dlm[c("FF","V","GG","W","m0","C0")]

$FF
     [,1] [,2]
[1,]    1    1

$V
     [,1]
[1,]    1

$GG
     [,1] [,2]
[1,]    1    0
[2,]    0    1

$W
     [,1] [,2]
[1,] 0.01 0.00
[2,] 0.00 0.01

$m0
[1] 0 0

$C0
      [,1]  [,2]
[1,] 1e+07 0e+00
[2,] 0e+00 1e+07
```

Example Models: Regression with time-varying parameters

R Code: TVP Regression Model

```
> tvp.dlm[c("JFF","JV","JGG","JW")]

$JFF
     [,1] [,2]
[1,]    0    1

$JV
NULL

$JGG
NULL

$JW
NULL

> head(tvp.dlm$X)

            SP500
1996-01-30  2.944
1996-02-28  0.532
1996-03-30  0.589
1996-04-29  1.042
1996-05-30  2.137
1996-06-29 -0.032
```

Example Models: Log-Normal AR(1) SV Model

R Code: Log-Normal AR(1) SV Model

```
> #########################################
> # EX 3. state space for SV model
> #########################################
> # ln|r(t)| = -0.63518 + lns(t) + v(t), v(t) ~ (0,pi^2/8)
> # lns(t) = w + phi*lns(t-1) + w(t), w(t) ~ N(0, s2w)
> # theta = (-0.63518, w, lns(t))'
> # m0 = (-0.63518, w, w/(1-phi))'
> # C0 = I*1e-7, C0[3,3] = sw2/(1-phi^2)
> phi = 0.9
> sig2n = 1
> omega = 0.1
> F.mat = matrix(c(1,0,1),1,3)
> V.val = pi^2/8
> G.mat = matrix(c(1,0,0,0,1,0,0,1,phi),3,3)
> W.mat = diag(0,3)
> W.mat[3,3] = sig2n
> m0.vec = c(-0.63518, omega, omega/(1-phi))
> C0.mat = diag(1,3)*1e-7
> C0.mat[3,3] = sig2n/(1-phi^2)
> SV.dlm = dlm(FF=F.mat, V=V.val, GG=G.mat,
            W=W.mat, m0=m0.vec, C0=C0.mat)
```

Example Models: Log-Normal AR(1) SV Model

R Code: Log-Normal AR(1) SV Model

```
> SV.dlm

$FF
     [,1] [,2] [,3]
[1,]    1    0    1

$V
         [,1]
[1,] 1.233701

$GG
     [,1] [,2] [,3]
[1,]    1    0  0.0
[2,]    0    1  1.0
[3,]    0    0  0.9

$W
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
[3,]    0    0    1

$m0
[1] -0.63518  0.10000  1.00000

$C0
      [,1]   [,2]     [,3]
[1,] 1e-07 0e+00 0.000000
[2,] 0e+00 1e-07 0.000000
[3,] 0e+00 0e+00 5.263158
```

Signal Extraction and Prediction

In a state space model, the unobserved state vector θ_t is the signal and the measurement error $\mathbf{w}_t$ is the noise. Given observed data $\mathbf{y}_1, \ldots, \mathbf{y}_T$ the goals of state space estimation are:

1. Optimal signal extraction
2. Optimal $h-$step ahead prediction of states and data

Filtering and Smoothing

There are two types of signal extraction:

1. Filtering: Optimal estimates of θ_t given information available at time t, $I_t = \{\mathbf{y}_1, \ldots, \mathbf{y}_t\}$:

$$E[\theta_t | I_t] = \text{ filtered estimate of } \theta$$

2. Smoothing: Optimal estimates of θ_t given information available at time T, $I_T = \{\mathbf{y}_1, \ldots, \mathbf{y}_T\}$

$$E[\theta_t | I_T] = \text{ smoothed estimate of } \theta$$

The Kalman Filter

The *Kalman filter* is a set of recursion equations for determining the optimal estimates of the state vector θ_t given information available at time t, I_t. The filter consists of two sets of equations:

1. *Prediction equations*
2. *Updating equations*

To describe the filter, let

$$\mathbf{m}_t = E[\theta_t | I_t] = \text{ optimal estimator of } \theta_t \text{ based on } I_t$$

$$\mathbf{C}_t = E[(\theta_t - \mathbf{m}_t)(\theta_t - \mathbf{m}_t)' | I_t] = \text{ MSE matrix of } \mathbf{m}_t$$

Prediction Equations

Given $\mathbf{m}_{t-1}$ and $\mathbf{C}_{t-1}$ at time $t-1$, the optimal predictor of θ_t and its associated MSE matrix are

$$\mathbf{m}_{t|t-1} = E[\theta_t|I_{t-1}] = \mathbf{G}_t\mathbf{m}_{t-1}$$

$$\mathbf{C}_{t|t-1} = E[(\theta_t - \mathbf{m}_{t-1})(\theta_t - \mathbf{m}_{t-1})'|I_{t-1}]$$

$$= \mathbf{G}_t\mathbf{C}_{t-1}\mathbf{G}_t' + \mathbf{W}_t$$

The corresponding optimal predictor of $\mathbf{y}_t$ give information at $t-1$ is

$$\mathbf{y}_{t|t-1} = E[\mathbf{y}_t|I_{t-1}] = \mathbf{F}_t\mathbf{m}_{t|t-1}$$

The *prediction error* and its MSE matrix are

$$\mathbf{e}_t = \mathbf{y}_t - \mathbf{y}_{t|t-1} = \mathbf{y}_t - \mathbf{F}_t\mathbf{m}_{t|t-1}$$

$$= \mathbf{F}_t(\theta_t - \mathbf{m}_{t|t-1}) + \mathbf{v}_t$$

$$E[\mathbf{e}_t\mathbf{e}_t'] = \mathbf{Q}_t = \mathbf{F}_t\mathbf{C}_{t|t-1}\mathbf{F}_t' + \mathbf{V}_t$$

Updating Equations

When new observations $\mathbf{y}_t$ become available, the optimal predictor $\mathbf{m}_{t|t-1}$ and its MSE matrix are updated using

$$\begin{aligned} \mathbf{m}_t &= \mathbf{m}_{t|t-1} + \mathbf{C}_{t|t-1}\mathbf{F}_t'\mathbf{Q}_t^{-1}(\mathbf{y}_t - \mathbf{F}_t\mathbf{m}_{t|t-1}) \\ &= \mathbf{m}_{t|t-1} + \mathbf{C}_{t|t-1}\mathbf{F}_t'\mathbf{Q}_t^{-1}\mathbf{v}_t \\ \mathbf{C}_t &= \mathbf{C}_{t|t-1} - \mathbf{C}_{t|t-1}\mathbf{F}_t'\mathbf{Q}_t^{-1}\mathbf{F}_t\mathbf{C}_{t|t-1} \end{aligned}$$

Note: $\mathbf{K}_t = \mathbf{C}_{t|t-1}\mathbf{F}_t'\mathbf{Q}_t^{-1}$ = Kalman gain matrix. It gives the weight on new information $\mathbf{e}_t = \mathbf{y}_t - \mathbf{F}_t\mathbf{m}_{t|t-1}$ in the updating equation for $\mathbf{m}_t$.

Kalman Smoother

Once all data I_T is observed, the optimal estimates $E[\theta_t | I_T]$ can be computed using the backwards Kalman smoothing recursions

$$E[\theta_t | I_T] = \mathbf{m}_{t|T} = \mathbf{m}_t + \mathbf{C}_t^* \left(\mathbf{m}_{t+1|T} - \mathbf{G}_{t+1} \mathbf{m}_t \right)$$

$$E[(\theta_t - \mathbf{m}_{t|T})(\theta_t - \mathbf{m}_{t|T})' | I_T] = \mathbf{C}_{t|T} = \mathbf{C}_t + \mathbf{C}_t^* (\mathbf{C}_{t+1|T} - \mathbf{C}_{t+1|t}) \mathbf{C}_t^{*\prime}$$

$$\mathbf{C}_t^* = \mathbf{C}_t \mathbf{G}'_{t+1} \mathbf{C}_{t+1|t}^{-1}$$

The algorithm starts by setting $\mathbf{m}_{T|T} = \mathbf{m}_T$ and $\mathbf{C}_{T|T} = \mathbf{C}_T$ and then proceeds backwards for $t = T-1, \ldots, 1$.

Maximum likelihood estimation

- Let ψ denote the parameters of the state space model, which are embedded in the system matrices $\mathbf{F}_t$, $\mathbf{G}_t$, $\mathbf{W}_t$ and $\mathbf{V}_t$. These parameters are typically unknown and must be estimated from the data $\mathbf{y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_T\}$.
- In the linear-Gaussian state space model, the parameter vector ψ can be estimated be estimated by maximum likelihood using the prediction error decomposition of the log-likelihood

$$\hat{\psi}_{MLE} = argmax_{\psi} \ln L(\psi|\mathbf{y}) = \sum_{t=1}^{T} \ln f(y_t|I_{t-1};\psi)$$

where $f(\mathbf{y}_t|I_{t-1};\psi)$ is the conditional density of y_t given I_{t-1}

Prediction Error Decomposition

- From the Kalman filter equations with a fixed value of ψ we have that

$$\mathbf{y}_{t|t-1} \sim N(\mathbf{F}_t(\psi)\mathbf{m}_{t|t-1}(\psi), \mathbf{Q}_t(\psi))$$

and so

$$f(\mathbf{y}_t|I_{t-1};\psi) = (2\pi\mathbf{Q}_t(\psi))^{-1/2}\exp\left\{-\frac{1}{2}\mathbf{e}_t(\psi)'\mathbf{Q}_t^{-1}\mathbf{e}_t(\psi)\right\}$$

- The *prediction error decomposition* of the Gaussian log-likelihood function follows immediately:

$$\ln L(\psi|\mathbf{y}) = -\frac{NT}{2}\ln(2\pi) - \frac{1}{2}\sum_{t=1}^{T}\ln|\mathbf{Q}_t(\psi)|$$

$$-\frac{1}{2}\sum_{t=1}^{T}\mathbf{e}_t'(\psi)\mathbf{Q}_t^{-1}(\psi)\mathbf{e}_t(\psi)$$

Forecasting

- The Kalman filter prediction equations produces in-sample 1-step ahead forecasts and MSE matrices
- Out-of-sample $h-$step ahead predictions and MSE matrices can be computed from the prediction equations by extending the data set $\mathbf{y}_1, \ldots, \mathbf{y}_T$ with a set of h missing values
  - When y_τ is missing the Kalman filter reduces to the prediction step so a sequence of h missing values at the end of the sample will produce a set of $h-$step ahead forecasts for $j = 1, \ldots, h$

Kalman filtering functions in the dlm package

Function	Task
dlmFilter	Kalman filtering
dlmSmooth	Kalman smoothing
dlmForecast	Forecasting
dlmLL	Likelihood
dlmMLE	ML estimation

Table: Kalman filtering related functions in package dlm

Outline

Regression model with time varying parameters

R Code: Fit regression model via OLS

```
> # ols fit - constant equity beta
> ols.fit = lm(HAM1 ~ sp500)
> summary(ols.fit)

Call:
lm(formula = HAM1 ~ sp500)

Residuals:
    Min      1Q  Median      3Q     Max
-5.1782 -1.3871 -0.2147  1.2626  5.7441

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.57747    0.16971   3.403 0.000887 ***
sp500        0.39007    0.03908   9.981  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.934 on 130 degrees of freedom
Multiple R-squared: 0.4339,        Adjusted R-squared: 0.4295
F-statistic: 99.63 on 1 and 130 DF,  p-value: < 2.2e-16
```

TVP model: estimate parameters via MLE

R Code: Fit dlm TVP model

```
> # function to build TVP ss model
> buildTVP <- function(parm, x.mat){
   parm <- exp(parm)
   return( dlmModReg(X=x.mat, dV=parm[1],
                     dW=c(parm[2], parm[3])) )
 }
> # maximize over log-variances
> start.vals = c(0,0,0)
> names(start.vals) = c("lns2v", "lns2a", "lns2b")
> TVP.mle = dlmMLE(y=HAM1, parm=start.vals,
                  x.mat=sp500, build=buildTVP,
                  hessian=T)
> class(TVP.mle)

[1] "list"

> names(TVP.mle)

[1] "par"          "value"        "counts"       "convergence" "message"
[6] "hessian"
```

TVP model: MLE estimates

R Code: dlm model fit

```
> TVP.mle

$par
    lns2v      lns2a      lns2b
  1.137778 -13.902591  -5.787831

$value
[1] 167.6016

$counts
function gradient
      28       28

$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"

$hessian
             lns2v        lns2a        lns2b
lns2v 5.943783e+01 2.871303e-05 1.853667e+00
lns2a 2.871303e-05 1.566605e-04 3.391776e-05
lns2b 1.853667e+00 3.391776e-05 1.860590e+00
```

TVP model: Kalman filtering and smoothing

R Code: Filter and smooth

```
> # get sd estimates
> se2 <- sqrt(exp(TVP.mle$par))
> names(se2) = c("sv", "sa", "sb")
> sqrt(se2)

        sv         sa         sb
1.32902368 0.03094178 0.23528498

> # fitted ss model
> TVP.dlm <- buildTVP(TVP.mle$par, sp500)
> # filtering
> TVP.f <- dlmFilter(HAM1, TVP.dlm)
> class(TVP.f)

[1] "dlmFiltered"

> names(TVP.f)

[1] "y"   "mod" "m"   "U.C" "D.C" "a"   "U.R" "D.R" "f"

> # smoothing
> TVP.s <- dlmSmooth(TVP.f)
> class(TVP.s)

[1] "list"

> names(TVP.s)

[1] "s"   "U.S" "D.S"
```

TVP model: compute confidence intervals

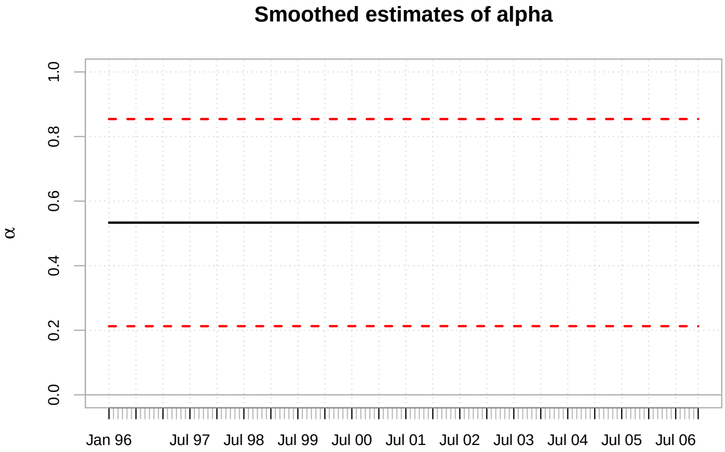
R Code: Compute confidence intervals

```
> # extract smoothed states - intercept and slope coefs
> alpha.s = xts(TVP.s$s[-1,1,drop=FALSE],
   as.Date(rownames(TVP.s$s[-1,])))
> beta.s = xts(TVP.s$s[-1,2,drop=FALSE],
   as.Date(rownames(TVP.s$s[-1,])))
> colnames(alpha.s) = "alpha"
> colnames(beta.s) = "beta"
> # extract std errors - dlmSvd2var gives list of MSE matrices
> mse.list = dlmSvd2var(TVP.s$U.S, TVP.s$D.S)
> se.mat = t(sapply(mse.list, FUN=function(x) sqrt(diag(x))))
> se.xts = xts(se.mat[-1, ], index(beta.s))
> colnames(se.xts) = c("alpha", "beta")
> a.u = alpha.s + 1.96*se.xts[,"alpha"]
> a.l = alpha.s - 1.96*se.xts[, "alpha"]
> b.u = beta.s + 1.96*se.xts[,"beta"]
> b.l = beta.s - 1.96*se.xts[, "beta"]
```

TVP model: estimated alpha

R Code: plot smoothed estimates with +/- 2*SE bands

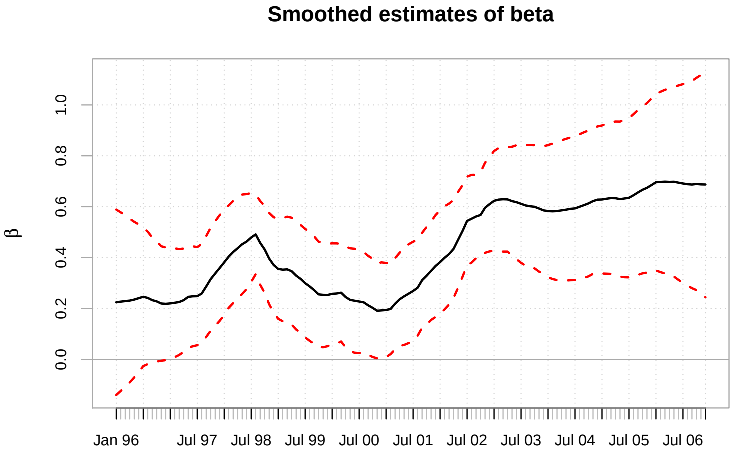
```
> chart.TimeSeries(cbind(alpha.s, a.l, a.u), main="Smoothed estimates of alpha",
  ylim=c(0,1), colorset=c(1,2,2), lty=c(1,2,2),ylab=expression(alpha),xlab="")
```



TVP model: estimated beta

R Code: plot smoothed estimates with +/- 2*SE bands

```
> chart.TimeSeries(cbind(beta.s, b.l, b.u), main="Smoothed estimates of beta",
   colorset=c(1,2,2), lty=c(1,2,2),ylab=expression(beta),xlab="")
```



TVP model: forecast of alpha and beta

R Code: Forecast alpha and beta

```
> # forecasting using dlmFilter
> # add 10 missing values to end of sample
> new.xts = xts(rep(NA, 10),
   seq.Date(from=end(HAM1), by="months", length.out=11)[-1])
> HAM1.ext = merge(HAM1, new.xts)[,1]
> TVP.ext.f = dlmFilter(HAM1.ext, TVP.dlm)
> # extract h-step ahead forecasts of state vector
> TVP.ext.f$m[as.character(index(new.xts)),]

                [,1]      [,2]
2007-01-30 0.5333932 0.6872751
2007-03-02 0.5333932 0.6872751
2007-03-30 0.5333932 0.6872751
2007-04-30 0.5333932 0.6872751
2007-05-30 0.5333932 0.6872751
2007-06-30 0.5333932 0.6872751
2007-07-30 0.5333932 0.6872751
2007-08-30 0.5333932 0.6872751
2007-09-30 0.5333932 0.6872751
2007-10-30 0.5333932 0.6872751
```

Estimate stochastic volatility model for S&P 500 returns

R Code: Download S&P 500 data

```
> library(quantmod)
> library(PerformanceAnalytics)
> getSymbols("^GSPC", from ="2000-01-03", to = "2012-04-03")

> GSPC = GSPC[, "GSPC.Adjusted", drop=F]
> GSPC.ret = CalculateReturns(GSPC, method="compound")
> GSPC.ret = GSPC.ret[-1,]*100
> colnames(GSPC.ret) = "GSPC"
> lnabs.ret = log(abs(GSPC.ret[GSPC.ret !=0]))
> lnabsadj.ret = lnabs.ret + 0.63518
```

Stochastic volatility example: build function

R Code: SV model build function

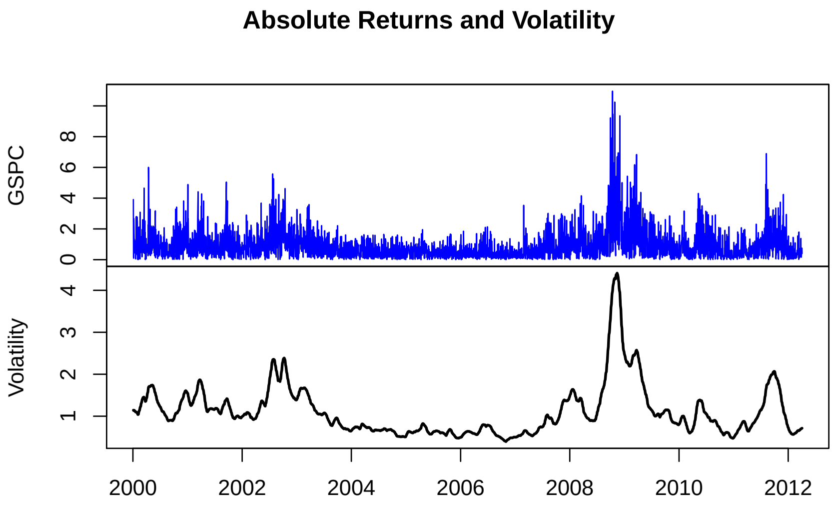
```
> # create state space
> # ln(r(t)) = -0.63518 + ln(s(t-1)) + v(t)
> # ln(s(t)) = w + phi*ln(s(t-1)) + n(t)
> buildSV = function(parm) {
   # parm[1]=phi, parm[2]=omega, parm[3]=lnsig2n
   parm[3] = exp(parm[3])
   F.mat = matrix(c(1,0,1),1,3)
   V.val = pi^2/8
   G.mat = matrix(c(1,0,0,0,1,0,0,1,parm[1]),3,3, byrow=TRUE)
   W.mat = diag(0,3)
   W.mat[3,3] = parm[3]
   m0.vec = c(-0.63518, parm[2], parm[2]/(1-parm[1]))
   C0.mat = diag(1,3)*1e7
   C0.mat[1,1] = 1e-7
   C0.mat[2,2] = 1e-7
   C0.mat[3,3] = parm[3]/(1-parm[1]^2)
   SV.dlm = dlm(FF=F.mat, V=V.val, GG=G.mat, W=W.mat,
                m0=m0.vec, C0=C0.mat)
   return(SV.dlm)
 }
```

Stochastic volatility example: MLE, filtering, smoothing

R Code: Fit, filter, smooth and plot

```
> phi.start = 0.9
> omega.start = (1-phi.start)*(mean(lnabs.ret))
> lnsig2n.start = log(0.1)
> start.vals = c(phi.start, omega.start, lnsig2n.start)
> SV.mle <- dlmMLE(y=lnabs.ret, parm=start.vals, build=buildSV, hessian=T,
   lower=c(0, -Inf, -Inf), upper=c(0.999, Inf, Inf))
> SV.dlm = buildSV(SV.mle$par)
> SV.f <- dlmFilter(lnabs.ret, SV.dlm)
> names(SV.f)

[1] "y"   "mod" "m"   "U.C" "D.C" "a"   "U.R" "D.R" "f"

> SV.s <- dlmSmooth(SV.f)
> names(SV.s)

[1] "s"   "U.S" "D.S"

> # extract smoothed estimate of logvol
> logvol.s = xts(SV.s$s[-1,3,drop=FALSE], as.Date(rownames(SV.s$s[-1,])))
> colnames(logvol.s) = "Volatility"
> # plot absolute returns with smoothed volatility
> plot.zoo(cbind(abs(GSPC.ret), exp(logvol.s)),
   main="Absolute Returns and Volatility",col=c(4,1),lwd=1:2,xlab="")
```

Stochastic volatility example: abs(returns) and vol estimate

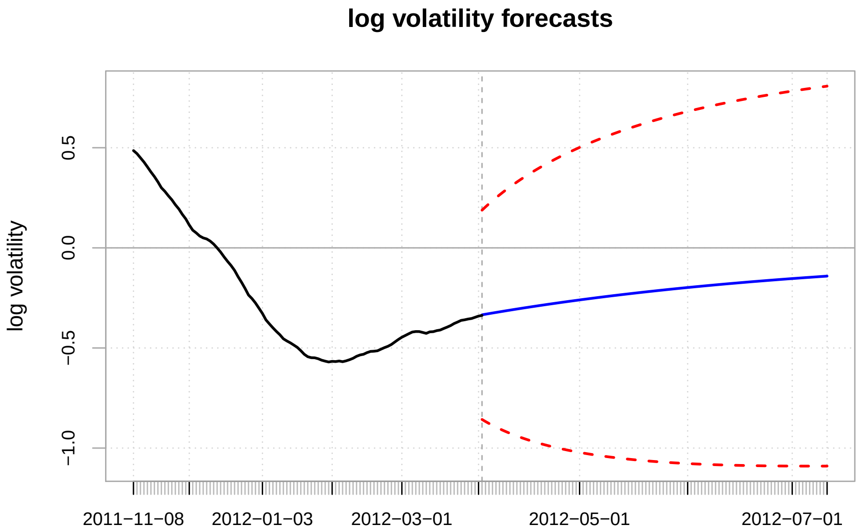


Stochastic volatility example: forecast and plot volatility

R Code: Plot code

```
> SV.fcst = dlmForecast(SV.f, nAhead=100)
> logvol.fcst = SV.fcst$a[,3]
> se.mat = t(sapply(SV.fcst$R,
                  FUN=function(x) sqrt(diag(x))))
> se.logvol = se.mat[,3]
> lvol.l = logvol.fcst - 2*se.logvol
> lvol.u = logvol.fcst + 2*se.logvol
> n.obs = length(logvol.s)
> n.hist = n.obs - 100
> new.xts = xts(cbind(logvol.fcst, lvol.l, lvol.u),
              seq.Date(from=end(logvol.s), by="days", length.out=100))
> chart.TimeSeries(merge(logvol.s[n.hist:n.obs],new.xts),
                 main="log volatility forecasts",
                 ylab="log volatility",xlab="",
                 lty=c(1,1,2,2),
                 colorset=c("black","blue", "red", "red"),
                 event.lines=list(as.character(end(logvol.s))))
```

Stochastic volatility example: volatility forecast



Outline

1. Introduction to state space models and the dlm package
2. DLM estimation and forecasting examples
3. Structural time series models and StructTS
4. Exponential smoothing models and the forecast package
5. Time series cross validation
6. Summary

The StructTS function

The `StructTS` function fits a *structural time series* model via MLE

R Code: The `StructTS` function

```
> args(StructTS)

function (x, type = c("level", "trend", "BSM"), init = NULL,
    fixed = NULL, optim.control = NULL)
NULL
```

Main arguments:

x	univariate time series (numeric vector or time series)
type	specifies local level, linear trend, or basic structural model
init	initial parameter values (optional)
fixed	specified values for fixed variables (optional)

Return value:

an object of class `StructTS`

Local level model as implemented in StructTS

$$x_t = \mu_t + \epsilon_t, \qquad \epsilon_t \sim N(0, \sigma^2_\epsilon), \qquad \text{observation equation}$$

$$\mu_{t+1} = \mu_t + \xi_t, \qquad \xi_t \sim N(0, \sigma^2_\xi), \qquad \text{state equation}$$

R Code: The `StructTS` function

```
> StructTS(GAS,type="level")

Call:
StructTS(x = GAS, type = "level")

Variances:
  level   epsilon
0.01769   0.00000
```

level — variance of the level disturbances σ^2_ξ

epsilon — variance of the observation disturbances σ^2_ϵ

Local linear trend model as implemented in StructTS

$$x_t = \mu_t + \epsilon_t, \qquad \epsilon_t \sim N(0, \sigma^2_\epsilon), \qquad \text{observation equation}$$

$$\mu_{t+1} = \mu_t + \nu_t + \xi_t, \qquad \xi_t \sim N(0, \sigma^2_\xi), \qquad \text{state equation, level}$$

$$\nu_{t+1} = \nu_t + \zeta_t, \qquad \zeta_t \sim N(0, \sigma^2_\zeta), \qquad \text{state equation, slope}$$

R Code: The `StructTS` function

```
> StructTS(GAS,type="trend")

Call:
StructTS(x = GAS, type = "trend")

Variances:
   level     slope   epsilon
0.006082  0.008082  0.000000
```

slope   variance of the slope disturbances σ^2_ζ

Basic structural model as implemented in StructTS

$$
\begin{aligned}
x_t &= \mu_t + \gamma_t + \epsilon_t, & \epsilon_t &\sim N(0, \sigma_\epsilon^2), && \text{observation equation} \\
\mu_{t+1} &= \mu_t + \nu_t + \xi_t, & \xi_t &\sim N(0, \sigma_\xi^2), && \text{state equation, level} \\
\nu_{t+1} &= \nu_t + \zeta_t, & \zeta_t &\sim N(0, \sigma_\zeta^2), && \text{state equation, slope} \\
\gamma_{t+1} &= -(\gamma_t + \gamma_{t-1} + \dots \\
&\qquad + \gamma_{t-S+2}) + \omega_t, & \omega_t &\sim N(0, \sigma_\omega^2), && \text{state eq. for S seasons}
\end{aligned}
$$

Basic structural model as implemented in StructTS

R Code: The `StructTS` function

```
> StructTS(GAS,type="BSM")

Call:
StructTS(x = GAS, type = "BSM")

Variances:
    level      slope       seas    epsilon
5.548e-03  5.300e-03  2.496e-06  0.000e+00
```

level	variance of the level disturbances σ_ξ^2
slope	variance of the slope disturbances σ_ζ^2
seas	variance of the seasonal disturbances σ_ω^2
epsilon	variance of the observation disturbances σ_ϵ^2

Outline

1. Introduction to state space models and the dlm package
2. DLM estimation and forecasting examples
3. Structural time series models and StructTS
4. Exponential smoothing models and the forecast package
5. Time series cross validation
6. Summary

Single source of error models

Exponential smoothing models arise from state space models with only a single source of error (SSOE). This type of model is also called an *innovations* state space model[†]:

$$y_t = \mathbf{w}'\mathbf{x}_{t-1} + \varepsilon_t, \qquad \varepsilon_t \sim N(0, \sigma_\varepsilon^2), \qquad \text{observation equation}$$

$$\mathbf{x}_t = \mathbf{F}\mathbf{x}_{t-1} + \mathbf{g}\varepsilon_t, \qquad \text{state equation}$$

where

$\mathbf{x}_t$ state vector (unobserved)

y_t observed time series

ε_t white noise series

[†] Hyndman, 2008

Time series decomposition

Time series components:

$$y = T + S + E$$

Trend (T)	long-term direction
Seasonal (S)	periodic pattern
Error (E)	random error component

Trend components:

None	$T_h = l$
Additive	$T_h = l + bh$
Additive Damped	$T_h = l + (\phi + \phi^2 + \ldots + \phi^h)b$
Multiplicative	$T_h = lb^h$
Multiplicative Damped	$T_h = lb(\phi + \phi^2 + \ldots + \phi^h)$

see Hyndman 2008

ETS model family

Trend Component		Seasonal Component: N None	Seasonal Component: A Additive	Seasonal Component: M Multiplicative
N	None	N,N	N,A	N,M
A	Additive	A,N	A,A	A,M
A_d	Additive damped	A_d,N	A_d,A	A_d,M
M	Multiplicative	M,N	M,A	M,M
M_d	Multiplicative damped	M_d,N	M_d,A	M_d,M

N,N	simple exponential smoothing
A,N	Holt linear method
A,A	additive Holt-Winters
A,M	multiplicative Holt-Winters
A_d,N	damped trend (additive errors)
A_d,M	damped trend (multiplicative errors)

Example of Holt's Linear Method (A,N)

Forecasting method:

Forecast	$\hat{y}_{t+h\|t} = l_t + hb_t$
Level	$l_t = \alpha y_t + (1-\alpha)(l_{t-1} + b_{t-1})$
Growth	$b_t = \beta^*(l_t - l_{t-1}) + (1-\beta^*)b_{t-1}, \qquad \beta = \alpha\beta^*$

Model:

Observation	$y_t = l_{t-1} + b_{t-1} + \varepsilon_t$
Level	$l_t = l_{t-1} + b_{t-1} + \alpha\varepsilon_t$
Growth	$b_t = b_{t-1} + \beta\varepsilon_t$

SSOE state space model:

$$y_t = \begin{bmatrix} 1 & 1 \end{bmatrix} \mathbf{x}_{t-1} + \varepsilon_t \qquad \varepsilon \sim NID(0, \sigma^2)$$

$$\mathbf{x}_t = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \mathbf{x}_{t-1} + \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \varepsilon_t \qquad \mathbf{x}_t = (l_t, b_t)'$$

see Hyndman 2008

The `forecast` package

Author

- Rob Hyndman, Professor of Statistics, Monash University
  - `http://robjhyndman.com/`

Journal of Statistical Software technical paper

- Automatic Time Series Forecasting: The forecast Package for R
- `http://www.jstatsoft.org/v27/i03`

Key functions

- ets - exponential smoothing state space models
- forecast - forecast n-steps ahead with confidence levels
- plot.forecast - create color-coded forecast probability cone
- auto.arima - automatically select terms for an arima model

The ets function

The ets function (**e**rror-**t**rend-**s**easonal) fits an exponential smoothing state space model

R Code: The ets function

```
> library(forecast)
> args(ets)

function (y, model = "ZZZ", damped = NULL, alpha = NULL, beta = NULL,
    gamma = NULL, phi = NULL, additive.only = FALSE, lambda = NULL,
    lower = c(rep(1e-04, 3), 0.8), upper = c(rep(0.9999, 3),
        0.98), opt.crit = c("lik", "amse", "mse", "sigma", "mae"),
    nmse = 3, bounds = c("both", "usual", "admissible"), ic = c("aic",
        "aicc", "bic"), restrict = TRUE)
NULL
```

Main arguments:

y	univariate time series (numeric vector or time series)
model	3-letter model identifying for the error, trend, season components
damped	TRUE for damped trend models

The forecast object

model	model object
mean	time series of mean forecast
lower	matrix of lower confidence bounds for prediction intervals
upper	matrix of upper confidence bounds for prediction intervals
level	confidence values associated with the prediction intervals
fitted	time series of fitted values
residuals	time series of residuals
x	original time series of data
method	name of the method used to fit the model

Outline

1. Introduction to state space models and the dlm package
2. DLM estimation and forecasting examples
3. Structural time series models and StructTS
4. Exponential smoothing models and the forecast package
5. Time series cross validation
6. Summary

Time series cross validation

Research tips, A blog by Rob J Hyndman

- Why every statistician should know about cross-validation (2010-10-04)

  `http://robjhyndman.com/researchtips/crossvalidation/`

Time series cross validation (from Hyndman)

1. Fit model to data $y_1, \ldots, y_t$
2. Generate 1-step ahead forecast $\hat{y}_{t+1}$
3. Compute forecast error $e^*_{t+1} = y_{t+1} - \hat{y}_{t+1}$
4. Repeat steps 1-3 for $t = m, \ldots, n-1$
   where m is minimum number of observations to fit model
5. Compute forecast MSE from $e^*_{m+1}, \ldots, e^*_n$

Time series cross validation

Research tips, A blog by Rob J Hyndman

- Time series cross-validation: an R example (2011-08-26)

  http://robjhyndman.com/researchtips/tscvexample/

Modern Toolmaking, A blog by Zach Mayer

- Functional and Parallel time series cross-validation (2011-11-21)

  http://moderntoolmaking.blogspot.com/2011/11/functional-and-parallel-time-series.html

- Additional wrapper functions (2011-11-22)

  http://moderntoolmaking.blogspot.com/2011/11/time-series-cross-validation-2.html

- Ability to include xregs (2011-12-12)

  http://moderntoolmaking.blogspot.com/2011/12/time-series-cross-validation-3.html

Cross validation fit/forecast functions

R Code: Cross validation fit/forecast functions

```
> library(tsfssm)
> etsForecast

function (x, h, lambda = NULL, ...)
{
    require(forecast)
    fit <- ets(x, lambda = lambda, ...)
    forecast(fit, h = h, lambda = lambda, fan = TRUE)
}
<environment: namespace:tsfssm>

> stsForecast

function (x, h, lambda = NULL, ...)
{
    require(forecast)
    if (!is.null(lambda))
        x <- BoxCox(x = x, lambda = lambda)
    fit <- StructTS(x, ...)
    forecast(fit, h = h, lambda = lambda, fan = TRUE)
}
<environment: namespace:tsfssm>
```

Time series cross validation function

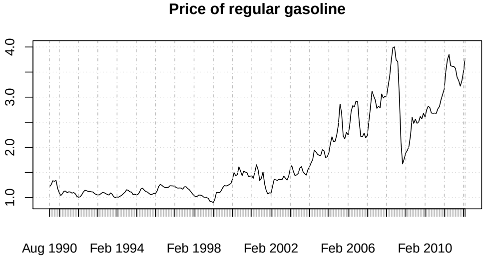
R Code: Time series cross validation function

```
> cv.ts <- function(x, FUN, tsControl, ...) {
   stepSize <- tsControl$stepSize
   maxHorizon <- tsControl$maxHorizon
   minObs <- tsControl$minObs
   fixedWindow <- tsControl$fixedWindow
   freq <- frequency(x)
   n <- length(x)
   st <- tsp(x)[1]+(minObs-2)/freq
   steps <- seq(1,(n-minObs),by=stepSize)
   cl <- makeCluster( detectCores()-1 )
   registerDoParallel(cl) # register foreach backend
   forcasts <- foreach(i=steps, .multicombine=FALSE) %dopar% {
     if (fixedWindow) {
       training.window <- window(x, start=st+(i-minObs+1)/freq, end=st+i/freq)
     } else {
       training.window <- window(x, end=st + i/freq)
     }
     return(FUN(training.window, h=maxHorizon, ...))
     }
   stopCluster(cl)
   return( forcasts )
 }
```

Energy price data from FRED

R Code: Energy price data from FRED

```
> head(energyPrices,3)

           OILPRICE GASREGCOVM MHOILNYH
1990-08-01   27.174      1.218    0.753
1990-09-01   33.687      1.258    0.888
1990-10-01   35.922      1.335    0.942

> GAS <- ts(coredata(energyPrices[,"GASREGCOVM"]),
   start=as.yearmon(time(energyPrices)[1]), frequency=12)
> plot(as.xts(GAS),main="Price of regular gasoline")
```

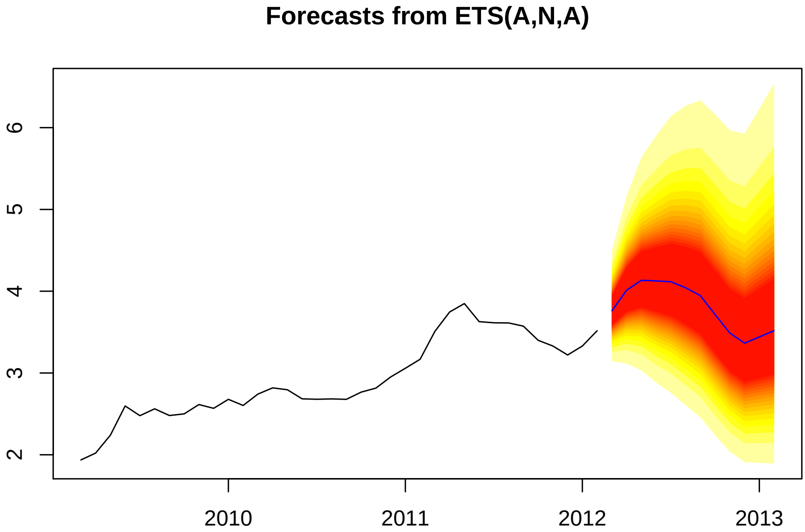


Running the time series cross validation

R Code: Run time series cross validation

```
> myControl <- list(minObs=6*12,   stepSize=1,
   maxHorizon=12, fixedWindow=TRUE)

> zzz.list <- cv.ts(x=GAS, FUN=etsForecast, tsControl=myControl, lambda=0)

> class(zzz.list)

[1] "list"

> class(zzz.list[[length(zzz.list)]])

[1] "forecast"

> names(zzz.list[[length(zzz.list)]])

[1] "model"     "mean"      "level"     "x"         "upper"     "lower"
[7] "fitted"    "method"    "residuals"

> names(zzz.list[[length(zzz.list)]]$model)

 [1] "loglik"     "aic"        "bic"        "aicc"       "mse"
 [6] "amse"       "fit"        "residuals"  "fitted"     "states"
[11] "par"        "m"          "method"     "components" "call"
[16] "initstate"  "sigma2"     "x"          "lambda"

> plot(zzz.list[[length(zzz.list)]],include=3*12)
```

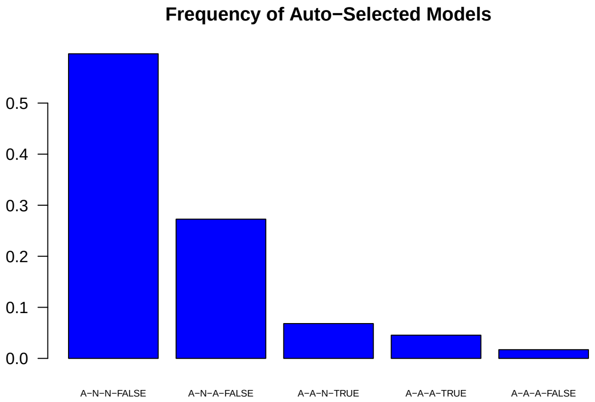
12-month ahead gas price forecast



Automatic model selection

R Code: Plot bar graph of model type selections

```
> ets.models <- sapply(zzz.list,function(x) paste(x$model$components,collapse="-"))
> barplot(sort(table(ets.models)/length(ets.models),dec=TRUE),
   las=1,col=4,cex.names=0.6)
> title("Frequency of Auto-Selected Models")
```



Extracting forecast from list of forecast objects

R Code: Function to extract forecasts from list of forecasts

```
> unpackForecasts <- function(model.list)
 {
   ts.list <- lapply(model.list, function(x) {x$mean})
   num.models <- length(model.list)
   ts.st <- tsp(ts.list[[1]])[1]
   ts.end <- tsp(ts.list[[num.models]])[2]
   date.seq <- seq(ts.st,ts.end,by=1/12)
   forecast.ts <- ts(matrix(NA,nrow=length(date.seq),ncol=12,
     dimnames=list(NULL,1:12)),start=ts.st,frequency=12)
   for(l in 1:num.models)
   {
     f <- ts.list[[l]]
     for(j in 1:12)
     {
       time.stamp <- tsp(f)[1]+(j-1)/12
       window(forecast.ts[,j],start=time.stamp,
         end=time.stamp) <- window(f,start=time.stamp,end=time.stamp)
     }
   }
   return( forecast.ts )
 }
```

Time series for forecasts

R Code: Extract forecasts

```
> forecast.zzz <- unpackForecasts(zzz.list)
> round(window(forecast.zzz,start=tsp(forecast.zzz)[2]-15/12,
   end=tsp(forecast.zzz)[2]),2)

            1    2    3    4    5    6    7    8    9   10   11   12
Nov 2011 3.18 3.57 3.61 3.61 3.63 3.85 3.75 3.51 3.62 3.06 2.95 3.06
Dec 2011 3.20 3.03 3.57 3.61 3.61 3.63 3.85 3.75 3.51 3.64 3.06 2.95
Jan 2012 3.34 3.25 3.08 3.57 3.61 3.61 3.63 3.85 3.75 3.51 3.65 3.06
Feb 2012 3.33 3.37 3.27 3.11 3.57 3.61 3.61 3.63 3.85 3.75 3.51 3.66
Mar 2012 3.76 3.33 3.63 3.50 3.32 3.57 3.61 3.61 3.63 3.85 3.75 3.51
Apr 2012   NA 4.01 3.33 3.86 3.75 3.54 3.57 3.61 3.61 3.63 3.85 3.75
May 2012   NA   NA 4.13 3.33 3.98 3.86 3.66 3.57 3.61 3.61 3.63 3.85
Jun 2012   NA   NA   NA 4.13 3.33 3.99 3.89 3.70 3.57 3.61 3.61 3.63
Jul 2012   NA   NA   NA   NA 4.12 3.33 4.01 3.88 3.70 3.57 3.61 3.61
Aug 2012   NA   NA   NA   NA   NA 4.04 3.33 4.00 3.85 3.67 3.57 3.61
Sep 2012   NA   NA   NA   NA   NA   NA 3.95 3.33 3.89 3.79 3.62 3.57
Oct 2012   NA   NA   NA   NA   NA   NA   NA 3.71 3.33 3.58 3.52 3.40
Nov 2012   NA   NA   NA   NA   NA   NA   NA   NA 3.49 3.33 3.32 3.33
Dec 2012   NA   NA   NA   NA   NA   NA   NA   NA   NA 3.36 3.33 3.22
Jan 2013   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA 3.44 3.33
Feb 2013   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA 3.52
```

Function to compute forecast accuracy by horizon

R Code: Function to compute forecast accuracy by horizon

```
> modelAccuracy <- function(forecast.ts,actual.ts)
 {
   cnames <- c("RMSE","MAE","MAPE","MdAPE","Min-Err","Max-Err","Max-APE")
   stat.tab <- matrix(data=NA,nrow=length(cnames),ncol=12,
     dimnames=list(cnames,1:12))
   res <- forecast.ts-actual.ts
   pe <- log(forecast.ts)-log(actual.ts)
   stat.tab["RMSE",] <- round(sqrt(apply(res^2,2,mean,na.rm=T)),2)
   stat.tab["MAE",] <- round(apply(abs(res),2,mean,na.rm=T),2)
   stat.tab["MAPE",] <- round(100*apply(abs(pe),2,mean,na.rm=T), 2)
   stat.tab["MdAPE",] <- round(100*apply(abs(pe),2,median,na.rm=T), 2)
   stat.tab["Min-Err",] <- round(apply(res,2,min,na.rm=T),2)
   stat.tab["Max-Err",] <- round(apply(res,2,max,na.rm=T),2)
   stat.tab["Max-APE",] <- round(100*apply(abs(pe),2,max,na.rm=T),1)
   return( stat.tab )
 }
```

Forecast accuracy metrics by forecast horizon

R Code: Compute forecast accuracy

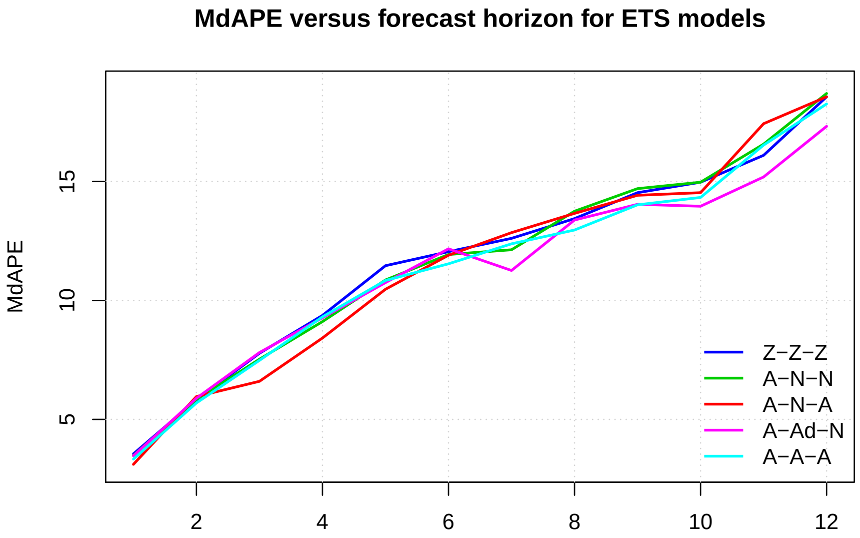
```
> stat.tab <- modelAccuracy(forecast.zzz,GAS)
> stat.tab

            1     2     3     4      5      6      7      8      9     10     11
RMSE     0.16  0.27  0.35  0.42   0.47   0.50   0.52   0.53   0.54   0.55   0.56
MAE      0.10  0.18  0.23  0.28   0.31   0.34   0.35   0.37   0.38   0.39   0.40
MAPE     4.98  8.55 10.87 12.93  14.49  15.66  16.63  17.34  18.30  18.88  19.47
MdAPE    3.55  5.82  7.78  9.37  11.46  12.05  12.61  13.44  14.53  14.97  16.10
Min-Err -0.47 -0.80 -1.00 -1.19  -1.48  -1.89  -1.82  -1.93  -1.87  -1.91  -2.04
Max-Err  0.83  1.27  1.64  2.07   2.33   2.23   2.11   2.06   1.98   1.88   1.79
Max-APE 33.20 54.70 72.10 88.30 108.20 130.70 132.10 140.40 141.10 145.00 152.00
             12
RMSE       0.58
MAE        0.44
MAPE      21.12
MdAPE     18.56
Min-Err   -2.01
Max-Err    1.52
Max-APE  152.20
```

Run cross validation on various ETS models

R Code: Run cross validations and plot results

```
>   ann.list <- cv.ts(x=GAS, FUN=etsForecast, tsControl=myControl,
     model="ANN", damped=FALSE, lambda=0)
>   aan.list <- cv.ts(x=GAS, FUN=etsForecast, tsControl=myControl,
     model="AAN", damped=TRUE, lambda=0)
>   ana.list <- cv.ts(x=GAS, FUN=etsForecast, tsControl=myControl,
     model="ANA", damped=FALSE, lambda=0)
>   aaa.list <- cv.ts(x=GAS, FUN=etsForecast, tsControl=myControl,
     model="AAA", damped=TRUE, lambda=0)

> plot(0,type="n",xlim=c(1,12),ylim=c(3,19),xlab="",ylab="MdAPE")
> grid()
> lines(modelAccuracy(unpackForecasts(zzz.list),GAS)["MdAPE",],lwd=2,col=4)
> lines(modelAccuracy(unpackForecasts(ann.list),GAS)["MdAPE",],lwd=2,col=3)
> lines(modelAccuracy(unpackForecasts(ana.list),GAS)["MdAPE",],lwd=2,col=2)
> lines(modelAccuracy(unpackForecasts(aan.list),GAS)["MdAPE",],lwd=2,col=6)
> lines(modelAccuracy(unpackForecasts(aaa.list),GAS)["MdAPE",],lwd=2,col=5)
> legend(x="bottomright",legend=c("Z-Z-Z","A-N-N","A-N-A","A-Ad-N","A-A-A"),
   lty=1,lwd=2,col=c(4,3,2,6,5),bty="n")
> title("MdAPE versus forecast horizon for ETS models")
```

ETS forecast accuracy results



Run cross validation on various STS models

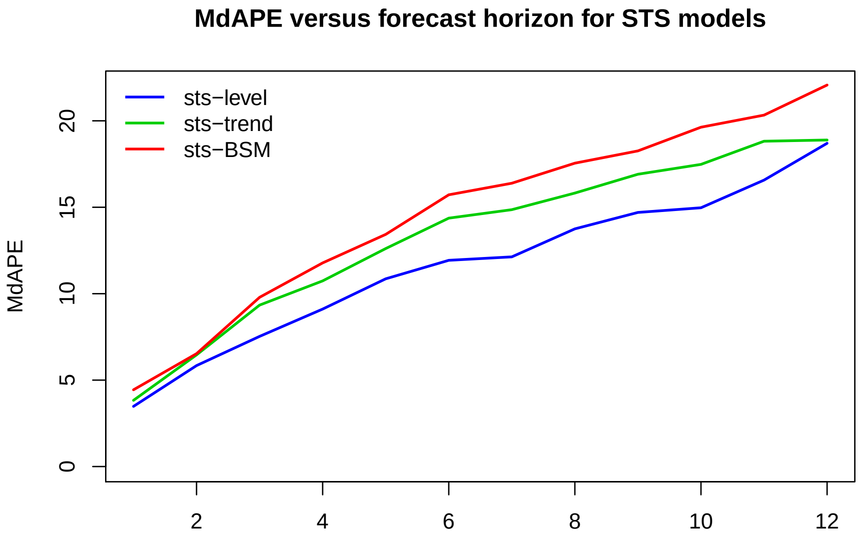
Run cross validation:

- STS local level model
- STS local linear trend model
- STS Basic Structural Model (BSM)

R Code: Run cross validations and plot results

```
> stsLevel.list <- cv.ts(x=GAS, FUN=stsForecast, tsControl=myControl,
   lambda=0, type="level")
> stsTrend.list <- cv.ts(x=GAS, FUN=stsForecast, tsControl=myControl,
   lambda=0, type="trend")
> stsBSM.list <- cv.ts(x=GAS, FUN=stsForecast, tsControl=myControl,
   lambda=0, type="BSM")

> plot(0,type="n",xlim=c(1,12),ylim=c(0,22),xlab="",ylab="MdAPE")
> lines(modelAccuracy(unpackForecasts(stsLevel.list),GAS)["MdAPE",],lwd=2,col=4)
> lines(modelAccuracy(unpackForecasts(stsTrend.list),GAS)["MdAPE",],lwd=2,col=3)
> lines(modelAccuracy(unpackForecasts(stsBSM.list),GAS)["MdAPE",],lwd=2,col=2)
> legend(x="topleft",legend=c("sts-level","sts-trend","sts-BSM"),lty=1,
   lwd=2,col=c(4,3,2),bty="n")
> title("MdAPE versus forecast horizon for STS models")
```

STS forecast accuracy results

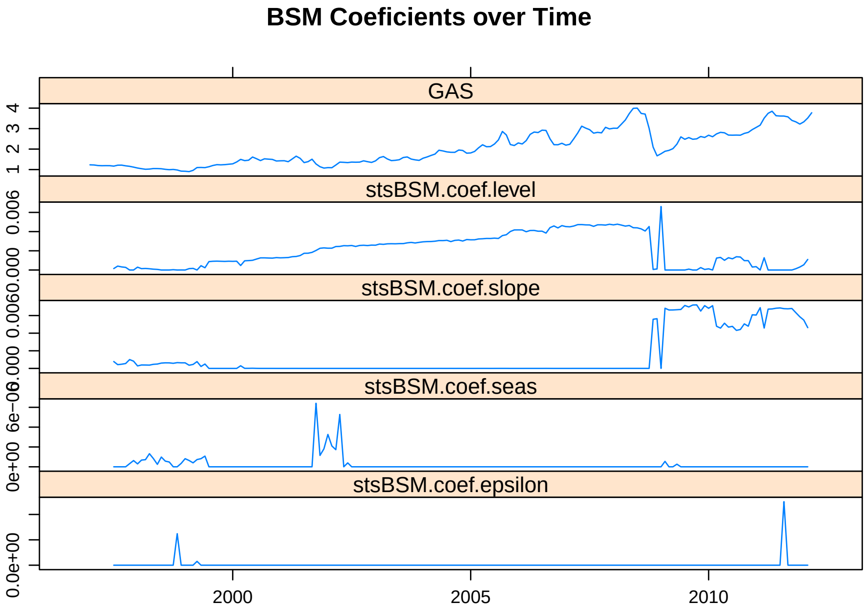


Analyze parameter stability

R Code: Extract model parameters and plot

```
> fit.st <- end(stsBSM.list[[1]]$model$data)
> fit.ed <- end(stsBSM.list[[length(stsBSM.list)]]$model$data)
> stsBSM.coef <- ts(t(sapply(stsBSM.list,function(x) x$model$coef)),
   start=fit.st,end=fit.ed,frequency=12)
> window(stsBSM.coef,start=c(2011,3))

                level       slope seas      epsilon
Mar 2011 0.0012795424 0.004583774    0 0.000000e+00
Apr 2011 0.0000000000 0.006732920    0 0.000000e+00
May 2011 0.0000000000 0.006753488    0 0.000000e+00
Jun 2011 0.0000000000 0.006834976    0 0.000000e+00
Jul 2011 0.0000000000 0.006868827    0 0.000000e+00
Aug 2011 0.0000000000 0.006784171    0 1.252526e-05
Sep 2011 0.0000000000 0.006767691    0 0.000000e+00
Oct 2011 0.0000000000 0.006802020    0 0.000000e+00
Nov 2011 0.0001338175 0.006327774    0 0.000000e+00
Dec 2011 0.0003103924 0.005863235    0 0.000000e+00
Jan 2012 0.0005438685 0.005487918    0 0.000000e+00
Feb 2012 0.0011015653 0.004624774    0 0.000000e+00

> library(lattice)
> xyplot(window(cbind(GAS,stsBSM.coef),start=c(1997,1)),xlab="",
   main="BSM Coeficients over Time")
```

Model parameters over time



dlm fitting functions mimicking StructTS

R Code: Time series cross validation function

```
> dlmSTSForecast <- function(x, h, lambda=NULL, ...) {
   require(forecast)
   require(dlm)
   if( !is.null(lambda) )
     x <- BoxCox(x=x, lambda=lambda)
   fit <- dlmStructTS(x, ...)
   forecast(object=fit, data=x, h=h, lambda=lambda)
 }
> dlmStructTS <- function(x, type = c("level", "trend", "BSM"), init = NULL,
   fixed = NULL, optim.control = NULL) {
   type <- if (missing(type)) "level" else match.arg(type)
   FUN <- switch(type, level = dlmLLM, trend = dlmLTM, BSM = dlmBSM)
   do.call(FUN,args=list(x,init,fixed,optim.control))
 }
> dlmLLM <- function(x,init=NULL,fixed=NULL,optim.control=NULL) {
   if( is.null(init) )
     init <- rep(log(0.1),2)
   buildFun <- function(theta) { dlmModPoly(1, dV = exp(theta[2]), dW = exp(theta[1
   fit <- dlmMLE(x, parm = init, build = buildFun)
   dlm.mod <- buildFun(fit$par)
 }
```

forecast function for dlm object

R Code: Time series cross validation function

```
> forecast.dlm <- function(object,data,h=12,lambda=NULL)
 {
   dlm.sm <- dlmSmooth(data, object)
   dlm.filt <- dlmFilter(data, object)
   dlm.for <- dlmForecast(dlm.filt, nAhead = h)
   hwidth <- qnorm(0.25, lower = FALSE) * sqrt(unlist(dlm.for$Q))
   if( is.null(lambda ) ) {
     fore <- dlm.for$f
     lower <- dlm.for$f - hwidth
     upper <- dlm.for$f + hwidth
   } else {
     fore <- InvBoxCox(dlm.for$f,lambda)
     lower <- InvBoxCox(dlm.for$f - hwidth,lambda)
     upper <- InvBoxCox(dlm.for$f + hwidth,lambda)
     data <- InvBoxCox(data,lambda)
   }
   ans <- structure(list(method = "dlm",model = object,level = 50,
     mean = fore,lower = as.matrix(lower),upper = as.matrix(upper),
     x = data,fitted = dlm.sm$s,residuals = residuals(dlm.filt,
     sd = FALSE)), class = "forecast")
   return( ans )
 }
```

Run cross validation on DLM models

Run cross validation:

- DLM local level model, DLM local linear trend

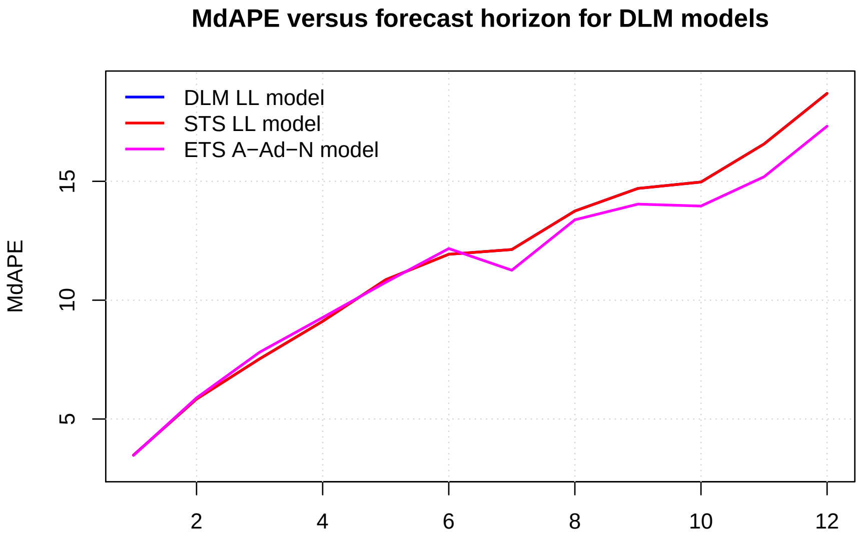
Compare model results:

- DLM local level model, STS local level model, ETS damped trend model

R Code: Run cross validations and plot results

```
> dlmLL.list <- cv.ts(x=GAS, FUN=dlmSTSForecast, tsControl=myControl, lambda=0,
   type="level")
> dlmLT.list <- cv.ts(x=GAS, FUN=dlmSTSForecast, tsControl=myControl, lambda=0,
   type="trend", init=rep(log(1e-4),3))

> plot(0,type="n",xlim=c(1,12),ylim=c(3,19),xlab="",ylab="MdAPE")
> grid()
> lines(modelAccuracy(unpackForecasts(dlmLL.list),GAS)["MdAPE",],lwd=2,col=4)
> lines(modelAccuracy(unpackForecasts(stsLevel.list),GAS)["MdAPE",],lwd=2,col=2)
> lines(modelAccuracy(unpackForecasts(aan.list),GAS)["MdAPE",],lwd=2,col=6)
> legend(x="topleft",legend=c("DLM LL model","STS LL model","ETS A-Ad-N model"),
   lty=1,lwd=2,col=c(4,2,6),bty="n")
> title("MdAPE versus forecast horizon for DLM models")
```

Compare DLM, STS, and ETS models



Outline

Summary

- dlm package gives R a fully-featured general state space capability
- StructTS provides easy, reliable basic structural models capabilities
- Time series cross validation can be used for model selection and out-of-sample forecast analysis
  - ets models
  - StructTS models
  - dlm models

COMPUTATIONAL FINANCE & RISK MANAGEMENT
UNIVERSITY of WASHINGTON
Department of Applied Mathematics

http://depts.washington.edu/compfin