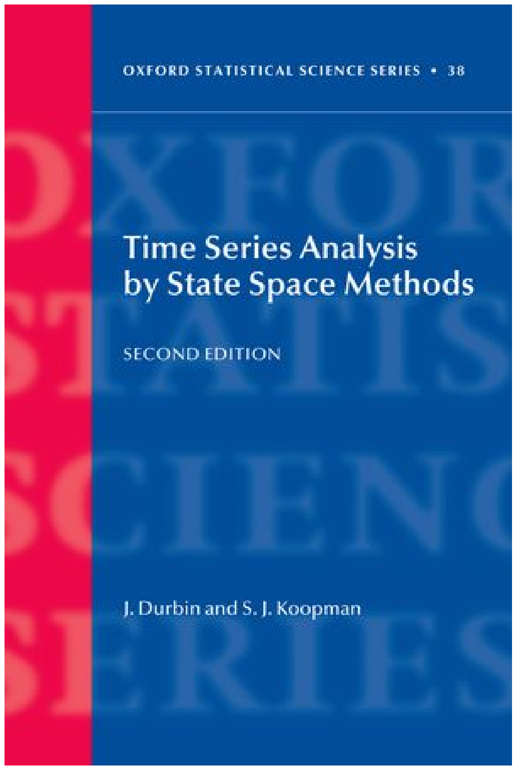
Time Series Analysis by State Space Methods

RK

May 5, 2014

Abstract

The purpose of this document is to summarize Part I of the book "Time Series Analysis by State Space Methods ", written by James Durbin and Siem Jan Koopman.


OXFORD STATISTICAL SCIENCE SERIES • 38
OXFORD STATISTICAL SCIENCE SERIES
Time Series Analysis by State Space Methods
SECOND EDITION
J. Durbin and S. J. Koopman

Contents

Preface

The distinguishing feature of state space time series models is that observations are regarded as made up of distinct components such as trend, seasonal, regression elements and disturbance terms, each of which is modeled separately. These models for the components are put together to form a single model called a state space model which provides the basis for analysis. The book is primarily aimed at applied statisticians and econometricians. Not much of math background is needed to go through the book. State space time series analysis began with the path breaking paper of Kalman and early developments of the subject took place in the field of engineering. The term "state space" comes form engineering. Statisticians and econometricians tend to stick to the same terminology.

Part I of the book deals with linear Gaussian models and Part II deals with the extensions to non linear, non Gaussian world. Extensions like exponentially distributed observations , allowing non-linearities in the model , allowing heavy-tailed densities to deal with outliers in the observations and structural shifts in the state are dealt in the latter part of the book. The treatment given in the book is a simulation based approach, as excepting for a few models, there are no closed form solutions available. Instead of following the MCMC treatment, the book shows the use of importance sampling and antithetic variables in analyzing state space models. The book provides analysis from the classical treatment as well as the Bayesian treatment.

In this document I will summarize Part I of the book that take up 175 out of 240 odd pages in the book. Part II of the book is all about nonlinear non Gaussian models and the use of importance sampling to estimate the filtering and smoothing estimates. In contrast to Part II of the book, the math in the first part of the book is definitely manageable. In fact one can go through the entire Part I of the book by knowing just one Lemma.In any case, let me summarize the contents of the book and also provide `R` code to replicate some of the visuals and numbers from the book.

1 Introduction

The developments of the system under study are guided by unobserved series of state vectors $\alpha_1, \alpha_2, \ldots$ with which are associated a series of observations $y_1, y_2, \ldots, y_n$, the relation between the α's and y_t's are specified by the state space model. The purpose of the state space analysis is to infer the relevant properties of the α's from a knowledge of observations $y_1, \ldots, y_n$. The introductory chapter of the book gives the road map to the entire book by explaining a bit about each of the 14 chapters in the book.

2 Local Level Model

This chapter is like a trailer to the first part of the book. Instead of bombarding the reader with the math right from the start, the authors take a more tempered approach and present the simplest of all state space models, *Local Level Model.* This model takes the following form

$$\begin{aligned} y_t =& \alpha_t + \epsilon_t, \quad \epsilon_t \sim N(0, \sigma^2_\epsilon) \\ \alpha_{t+1} =& \alpha_t + \eta_t, \quad \eta_t \sim N(0, \sigma^2_\eta), \quad \alpha_1 \sim N(a_1, P_1) \end{aligned}$$

Even though the form looks very simple, it is the basis for exponentially weighted moving average method, one of the most popular methods for smoothing a time series. Given the above model, one can write down the distribution in a multivariate form and do whatever stats one needs to do. However the routine computations become very cumbersome. This is where Kalman Filter comes in. By filtering and smoothing the series, KF offers an efficient way of obtaining results that arise from Multivariate theory. The chapter gives extensive visuals for each of the concepts that are discussed in the context of local level model. I guess this is the chapter with the maximum visuals out of all the fourteen chapters. That also means that this book is filled with math and one has to patiently work through the details. The authors are not in a mood to dumb down the math stuff and one has to slog through the book to understand the concepts. No easy path here. At least that's what I felt while going over the book. Through the use of local level model, the authors touch upon the following principles of state space analysis

- Filtering
- Smoothing
- Disturbance Smoothing
- Simulation Smoothing
- How to deal with missing observations ?
- How to forecast the state vector n steps ahead ?
- How to initialize the recursive system ?
- How to estimate the meta parameters of the local level model ?
- Diagnostics to check model validity

All the visuals in this chapter can be replicated via `KFAS` package in `R`. My goto package for state space modeling till now has been `dlm`. However for this book, I have found that `KFAS` closely mirrors all the algorithms from the book.

The first question that comes up for anyone just beginning to understand local level model is that,"Why do we need a recursive updating of various estimates"? Isn't the local level model after all a multivariate realization?.

$$y \sim N(1a_1, \Omega), \text{ with } \Omega = 11'P_1 + \Sigma$$

where the (i, j)th element of the $n \times n$ matrix Σ is given by

$$\Sigma_{ij} = \begin{cases} (i-1)\sigma_\eta^2, & i < j \\ \sigma_\epsilon^2 + (i-1)\sigma_\eta^2, & i = j \\ (j-1)\sigma_\eta^2, & i > j \end{cases}$$

Given the above equations, one can compute away all the conditional means, variances and covariances. However there is problem here. As soon as n increases, the computations become very cumbersome. That's why you need a recursive framework so that you don't carry much luggage as you chug along. By using filtering and smoothing, the estimation can be improved considerably.

2.1 Filtering

This section derives the filtering recursive equations for the local level model. They are

$$\begin{aligned}
\nu_t &= y_t - a_t \\
F_t &= P_t + \sigma_\epsilon^2 \\
K_t &= P_t / F_t \\
a_{t+1} &= a_t + K_t \nu_t \\
P_{t+1} &= P_t(1 - K_t) + \sigma_\eta^2
\end{aligned}$$

where $a_t = E(\alpha_t|Y_{t-1})$ and $P_t = Var(\alpha_t|Y_{t-1})$
The above relations constitute the celebrated *Kalman Filter* for this problem.

I will use the samedata set that the authors use in the book and replicate the figures via `R` packages.

```
data(Nile)
model    <- SSModel( Nile ~ SSMtrend(1, Q = list(matrix(NA))) , H = matrix(NA))
model    <- fitSSM( inits = c(log(var(Nile)) , log(var(Nile) ) ) ,
                    model = model, method  ='BFGS' )$model
out      <- KFS(model, filtering='state', smoothing=c('state','disturbance'),simplify=F)
```

```
par(mfrow=c(2,2))
conf.bands <- cbind(out$a, as.vector(out$a) +
                      (sqrt(cbind(out$P))* qnorm(0.95))%*%  (t(c(-1,1))))
temp       <- cbind(Nile, conf.bands[-1,])
cols       <- c("grey","blue","red","red")
lwd        <- c(1,2,1,1)
lty       <- c(1,1,2,2)
plot.ts(temp, plot.type="single", col = cols,lwd = lwd,lty=lty,xlab="",ylab="",main="i")
legend("topright",
       legend = c("Observation Data","Filtered state","Confidence intervals"),
       col = c("grey","blue","red"), lty = c(1,1,2),
       lwd = c(1,2,1),bty="n", cex=0.9 )

temp       <- ts( c(out$P)[-1],start = 1871)
plot(temp, col = "blue",lwd = 2,xlab="",ylab="",main="ii")

temp       <- ts( c(out$v[-1]),start = 1871)
plot(temp, col = "blue",lwd = 2,xlab="",ylab="",main="iii")
abline(h=0,col="grey")

temp       <- ts( c(out$F)[-1],start = 1871)
plot(temp, col = "blue",lwd = 2,xlab="",ylab="", main = "iv")
```

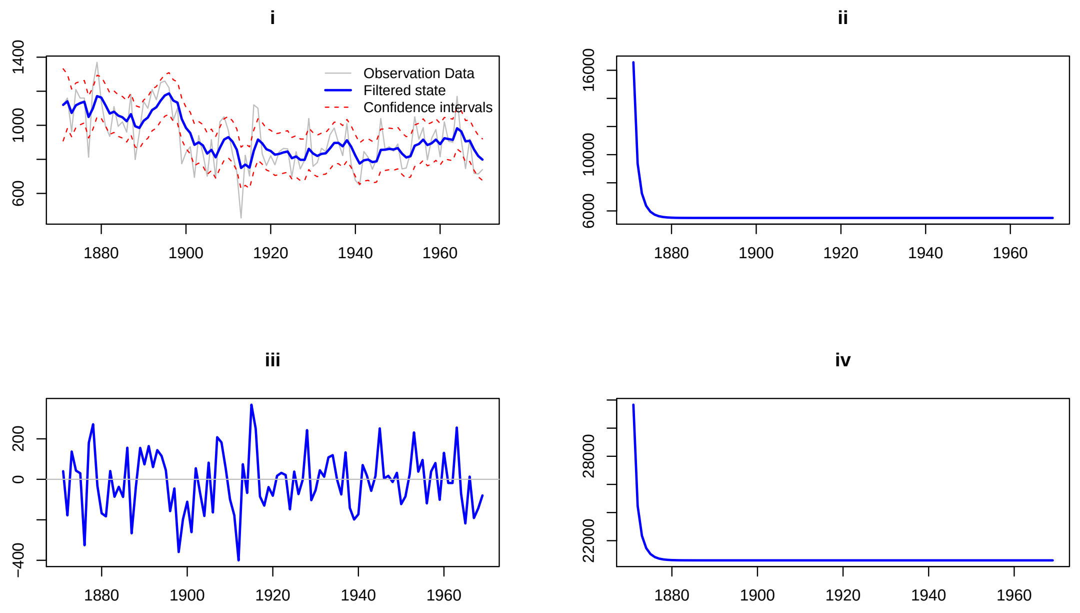


Figure 2.1: Nile data and output of Kalman filter : (i) filtered state and confidence intervals, (ii) filtered state variance, (iii) prediction error , (iv) prediction variance

The most obvious feature of the above illustrations is that F_t and P_t converge rapidly to constant values which confirms that the local level model has a steady state solution.

2.2 Forecast errors

This section shows that the relation between innovation terms in the model, i.e $\nu_t = y_t - a_t$ are observations

$$\nu = C(y - 1a_1)$$

where C is a lower triangular matrix.

$$\nu \sim N(0, C\Omega C'), \quad \nu \sim N(0, F)$$

The transformation of a symmetric positive matrix Ω in to a diagonal matrix F using a lower triangular matrix C is known as Cholesky decomposition. The Kalman filter can therefore be regarded as essentially a Cholesky decomposition of the variance matrix implied by the local level model.
One can also restate the state space model in its *innovation form*

$$\begin{aligned}\nu_t &= x_t + \epsilon_t, \quad \epsilon_t \sim N(0, \sigma_\epsilon^2)\\ x_{t+1} &= L_t x_t + \eta_t - K_t \epsilon_t\\ L_t &= 1 - K_t\end{aligned}$$

2.3 State Smoothing

Smoothing essentially comprises estimation of $\alpha_1, \dots, \alpha_n$ given the entire sample Y_n. The basic behind these computations is that the forecast errors $\nu_1, \dots, \nu_n$ are mutually independent and linear transformation of $y_1, \dots, y_n$, and $\nu_t, \dots, \mu_n$ are independent of $y_1, \dots, y_{t-1}$ with zero means. Using the standard multivariate theory, the smoothed state estimate and variance are derived. The following are the state smoothing recursions and state variance smoothing recursions:

$$\begin{aligned}
\hat{\alpha}_t &= a_t + P_t r_{t-1} \\
r_{t-1} &= \frac{\nu_t}{F_t} + L_t r_t \\
r_t &= \frac{\nu_{t+1}}{F_{t+1}} + L_{t+1}\frac{\nu_{t+2}}{F_{t+2}} + \ldots + L_{t+1} + \ldots + L_{n-1}\frac{\nu_n}{F_n} \\
r_n &= 0, \text{ for } t = n, n-1, \ldots, 1 \\
N_{t-1} &= F_t^{-1} + L_t^2 N_t \\
V_t &= P_t - P_t^2 N_{t-1} \\
N_t &= \frac{1}{F_{t+1}} + L_{t+1}^2 \frac{1}{F_{t+2}} + \ldots + L_{t+1}^2 \ldots L_{n-1}^2 \frac{1}{F_n} \\
N_n &= 0
\end{aligned}$$

```r
par(mfrow=c(2,2))
conf.bands <- cbind(out$alphahat, as.vector(out$alphahat) +
                     (sqrt(cbind(out$V))* qnorm(0.95))%*%  (t(c(-1,1))))
temp       <- cbind(Nile, conf.bands)
cols       <- c("grey","blue","red","red")
lwd        <- c(1,2,1,1)
lty       <- c(1,1,2,2)
plot.ts(temp, plot.type="single", col = cols,lwd = lwd,
        lty=lty,xlab="",ylab="",main="i")
legend("topright",
       legend = c("Observation Data","Smoothed state","Confidence intervals"),
       col = c("grey","blue","red"), lty = c(1,1,2),
       lwd = c(1,2,1),bty="n", cex=0.7 )
temp       <- ts( c(out$V),start = 1871)
plot(temp, col = "blue",lwd = 2,xlab="",ylab="",main="ii")
temp       <- ts( c(out$r)[-1],start = 1871)
plot(temp, col = "blue",lwd = 2,xlab="",ylab="",main="iii")
abline(h=0,col="grey")
temp       <- ts( c(out$N)[-1],start = 1871)
plot(temp, col = "blue",lwd = 2,xlab="",ylab="",main = "iv")
```

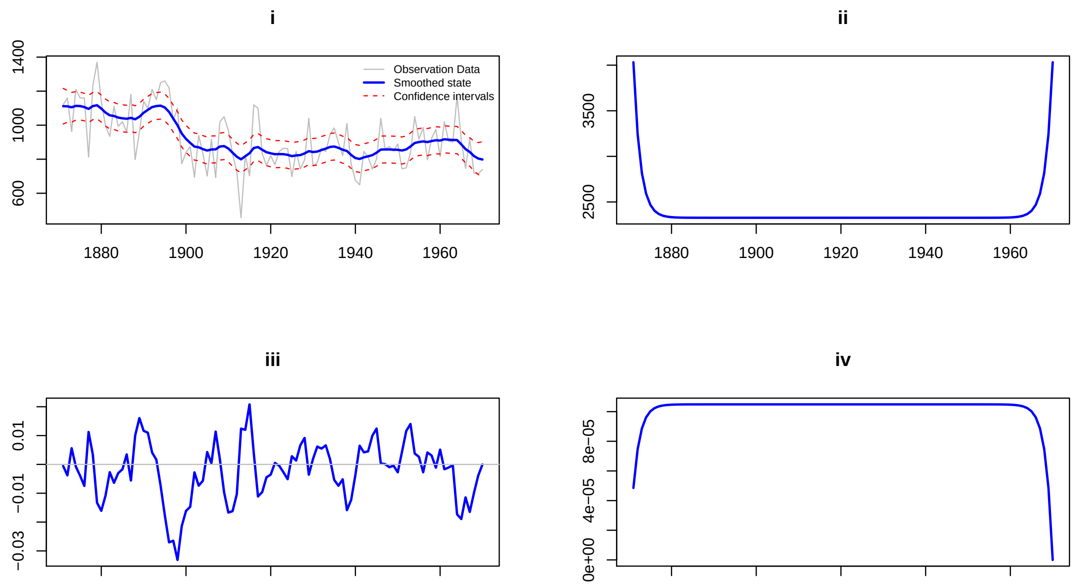


Figure 2.2: Nile data and output of state smoothing recursion: (i) smoothed state and confidence intervals, (ii) smoothed state variance, (iii) smoothing cumulant , (iv) smoothing variance cumulant

2.4 Disturbance smoothing

When I first learnt about disturbance smoothing, I did not understand the point in computing smoothed observation disturbance $E(\epsilon_t|y)$ and $E(\eta_t|y)$. Later I realized that they are immensely useful in making MLE computations quicker and in simulating state vectors. The section does not derive these equations and instead refers to the later chapters for the complete derivation. It merely states the results for the local level model. The following are the recursive relations for disturbance smoothing

$$\begin{aligned}
\hat{\epsilon}_t &= \sigma_\epsilon^2 u_t \\
u_t &= F_t^{-1}\nu_t - K_t r_t \\
\text{Var}(\epsilon_t|y) &= \sigma_\epsilon^2 - \sigma_\epsilon^4 D_t \\
D_t &= F_t^{-1} + K_t^2 N_t \\
\hat{\eta}_t &= \sigma_\eta^2 r_t \\
u_t &= F_t^{-1}\nu_t - K_t r_t \\
\text{Var}(\eta_t|y) &= \sigma_\eta^2 - \sigma_\eta^4 N_t
\end{aligned}$$

```r
par(mfrow=c(2,2))
temp       <- ts( c(out$epshat),start = 1871)
plot(temp, col = "blue",lwd = 2,xlab="",ylab="",main="i")
abline(h=0,col="grey")
temp       <- ts( c(out$V_eps),start = 1871)
plot(temp, col = "blue",lwd = 2,xlab="",ylab="",main="ii")
```

```
temp       <- ts( c(out$etahat),start = 1871)
plot(temp, col = "blue",lwd = 2,xlab="",ylab="",main="iii")
abline(h=0,col="grey")
temp       <- ts( c(out$V_eta),start = 1871)
plot(temp, col = "blue",lwd = 2,xlab="",ylab="",main="iv")
```

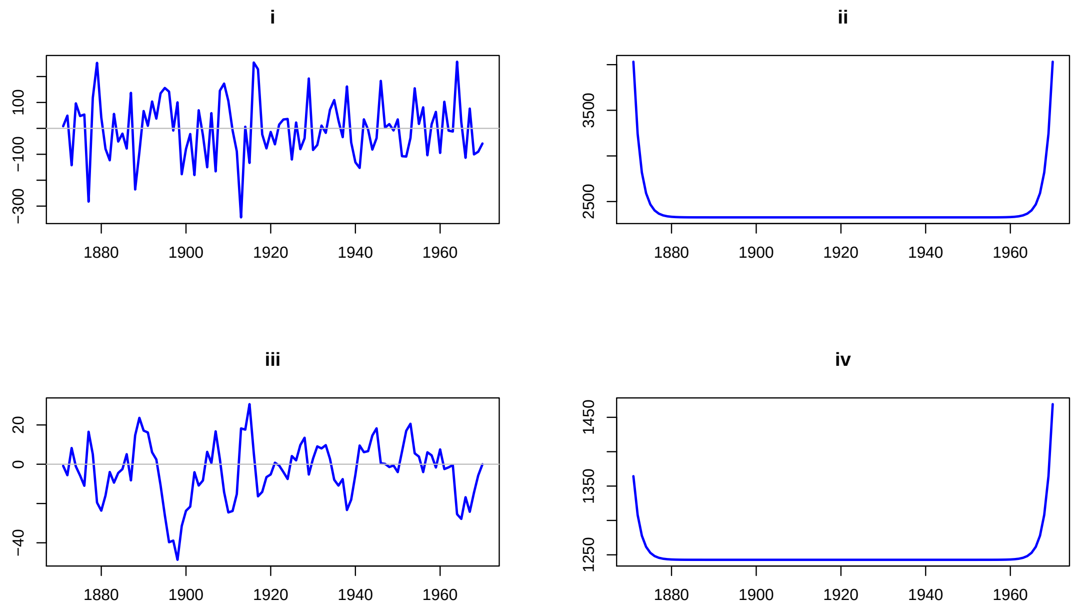


Figure 2.3: Outout of disturbance smoothing recursion : (i) observation error, (ii) observation error variance , (iii) state error , (iv) state error variance

2.5 Simulation Smoothing

The authors have come up with a better simulation smoothing algorithm than the one mentioned in this book after the book went for publishing. It is available here. `KFAS` implements this simulation smoother.

```
set.seed(1)
n             <- length(Nile)
theta.uncond <- numeric(n)
theta.uncond[1] <- Nile[1]
for(i in 2:n){
theta.uncond[i] <- theta.uncond[i-1] + rnorm(1,0,sqrt(out$model$Q))
}
theta.cond <- simulateSSM(model, type = c("states"))
eps.sim    <- simulateSSM(model, type = c("epsilon"))
eta.sim    <- simulateSSM(model, type = c("eta"))
```

```
par(mfrow=c(2,2))
ylim = c(400,max(c(out$alphahathat,theta.cond,theta.uncond))+50)

plot(1:100, Nile, type="l", col = "grey", ylim = ylim, main = "i")
points(1:100, out$alphahat , type="l", col = "green", lwd = 2)
points(1:100, c(theta.uncond), type="l", col = "red", lwd = 2)
leg <-c("actual data","smoothed estimate","unconditional sample")
legend("topleft",leg,col=c("grey","green","red"),lwd=c(1,2,2),cex=0.7,bty="n")


plot(1:100, Nile, type="l", col = "grey", ylim = ylim, main = "ii")
points(1:100, out$alphahat , type="l", col = "green", lwd = 2)
points(1:100, c(theta.cond), type="l", col = "blue", lwd = 2)
leg <-c("actual data","smoothed estimate","conditional sample")
legend("topleft",leg,col=c("grey","green","blue"),lwd=c(1,2,2),cex=0.7,bty="n")

temp       <- ts( c(out$epshat),start = 1871)
ylim       <- range(out$epshat,eps.sim)
plot(temp, type="l", col = "blue",ylim=c(-400,300),
     , main = "iii")
points(1871:1970,eps.sim,pch = 19,col="sienna",cex=0.5)

temp       <- ts( c(out$etahat),start = 1871)
ylim       <- range(out$etahat,eta.sim)
plot(temp, type="l", col = "blue",ylim=c(-400,300),
     , main = "iv")
points(1871:1970,eta.sim,pch = 19,col="sienna", cex=0.5)
```

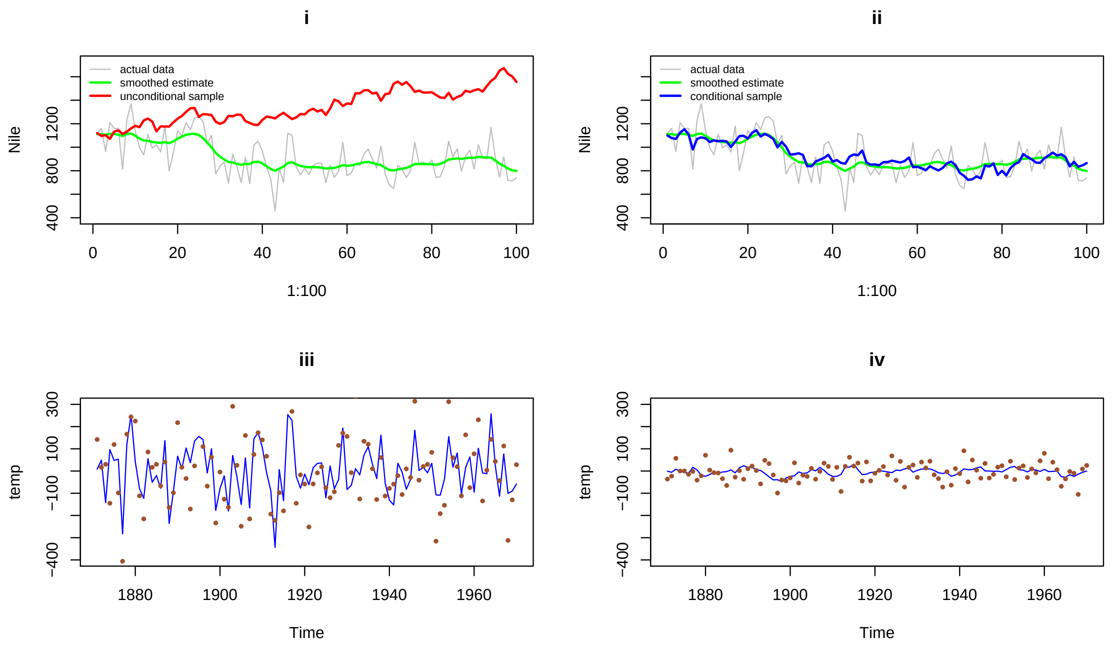


Figure 2.4: Simulation : (i) smoothed state and unconditional sample, (ii) smoothed state and conditional sample , (iii) smoothed observation error and conditional sample, (iv) smoothed state error and conditional sample

2.6 Missing observations

Missing observations are easily handled in the state space setting where all the recursive equations hold good with the respective innovation term and Kalman gain equated to 0 for the missing observations.

```
Nile.miss          <- Nile
Nile.miss[21:40] <- NA
Nile.miss[61:80] <- NA

model.miss <- SSModel( Nile.miss ~ SSMtrend(1, Q = c(model$Q)) , H = c(model$H))
out.miss    <- KFS(model.miss, filtering='state',
                  smoothing=c('state','signal','disturbance'), simplify=F)
```

```
par(mfrow=c(2,2))
temp        <- cbind(Nile.miss, c(out.miss$a)[-1])
cols        <- c("grey","blue")
plot.ts(temp, plot.type="single", col = cols,xlab="",ylab="",
        main="i", lwd=c(1,2,2))
legend("topright",
       legend = c("Observation Data","Filtered"),
       col = c("grey","blue"),
       lwd = c(1,2),bty="n", cex=0.8 )
```

```
temp       <- ts( c(out.miss$P)[-1],start = 1871)
plot(temp, col = "blue",lwd = 2,xlab="",ylab="",main ="ii")

temp       <- cbind(Nile.miss, c(out.miss$alphahat))
cols       <- c("grey","blue")
plot.ts(temp, plot.type="single", col = cols,xlab="",ylab="",
        main="iii", lwd=c(1,2,2))
legend("topright",
       legend = c("Observation Data","Smoothed"),
       col = c("grey","blue"),
       lwd = c(1,2),bty="n", cex=0.8 )


temp       <- ts( c(out.miss$V)[-1],start = 1871)
plot(temp, col = "blue",lwd = 2,xlab="",ylab="",main ="iv")
```

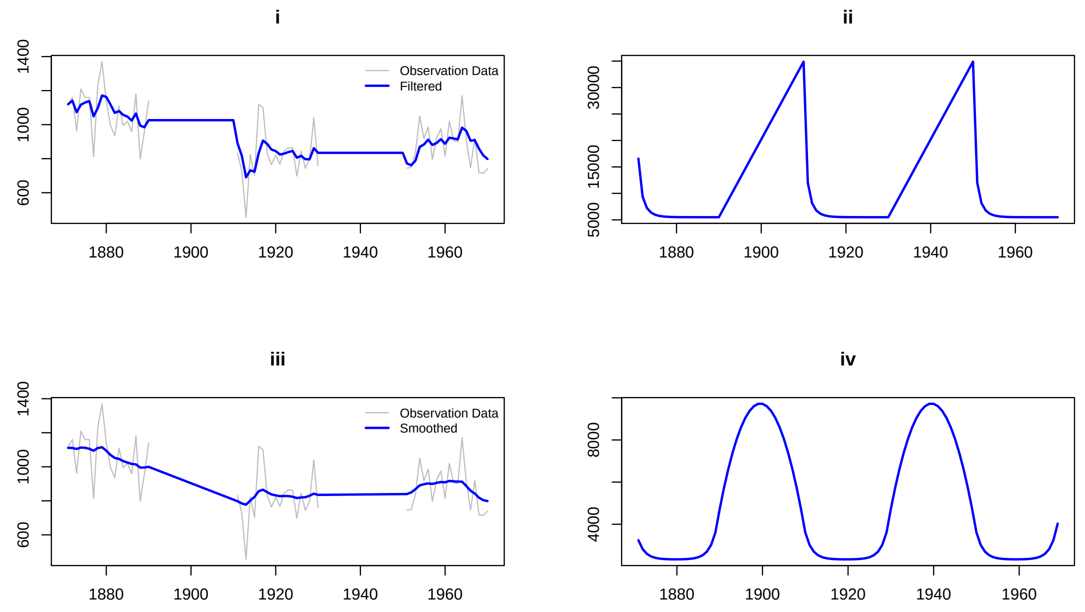


Figure 2.5: Filtered and smoothing output when observations are missing (i) filtered state , (ii) filtered state variance , (iii) smoothed state (iv) smoothed state variance

2.7 Forecasting

Forecasting in State space setting is done via simply assuming that the forecast period observations are missing and hence the Kalman gain and innovation for the period are assumed to 0. The chapter illustrates by extending the Nile dataset and placing `NA` for the observations

```
data(Nile)
Nile2    <- c(Nile,rep(NA,30))
model    <- SSModel( Nile2 ~ SSMtrend(1, Q = c(model$Q)) , H = c(model$H))
out      <- KFS(model, filtering='state',
                smoothing=c('state','signal','disturbance'),simplify=F)
```

```
par(mfrow=c(2,2))
conf.bands <- cbind(out$a, as.vector(out$a) +
                      (sqrt(cbind(out$P))* qnorm(0.95))%*%  (t(c(-1,1))))
temp       <- cbind(Nile2, conf.bands[-1,])
cols       <- c("grey","blue","red","red")
lwd        <- c(1,2,1,1)
lty       <- c(1,1,2,2)
plot.ts(temp, plot.type="single", col = cols,lwd = lwd,lty=lty,xlab="",ylab="",main="i")
legend("topright",
       legend = c("Observation Data","Forecasts","Confidence intervals"),
       col = c("grey","blue","red"), lty = c(1,1,2),
       lwd = c(1,2,1),bty="n", cex=0.8 )



temp       <- ts( c(out$P)[-1],start = 1871)
plot(temp, col = "blue",lwd = 2,xlab="",ylab="",main ="ii")

temp       <- ts( c(out$alphahat)[-1],start = 1871)
plot(temp, col = "blue",lwd = 2,xlab="",ylab="",
     ylim=c(700,1250),main = "iii")

temp       <- ts( c(out$F)[-1],start = 1871)
plot(temp, col = "blue",lwd = 2,
     xlab="",ylab="",main = "iv")
```

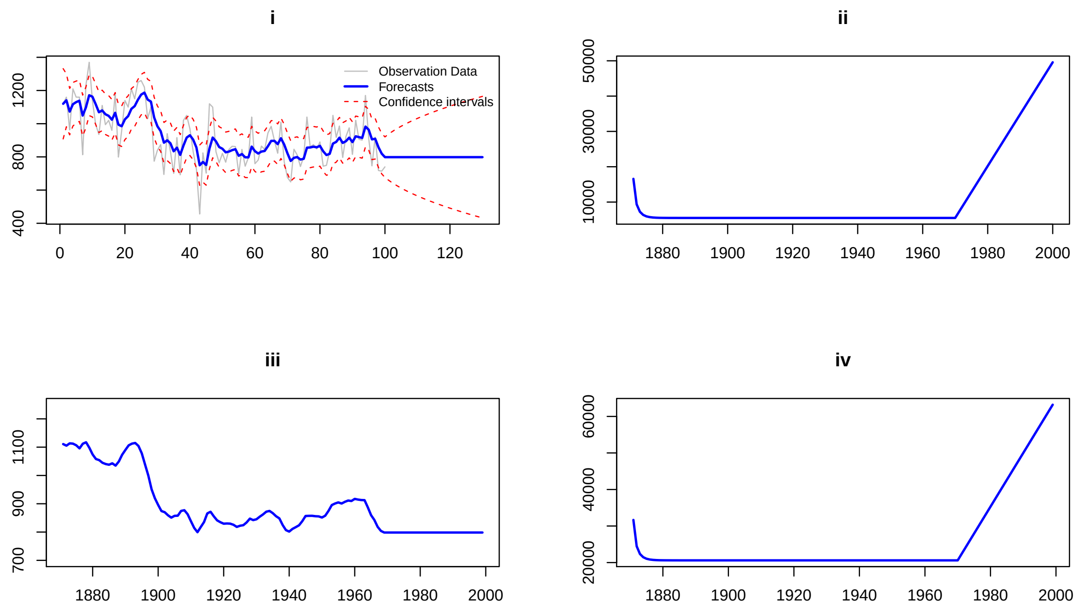


Figure 2.6: Nike data and output of forecasting (i) state forecast and confidence intervals, (ii) state variance , (iii) observation forecast (iv) observation forecast variance

2.8 Initialization

This section explains about initialization of KF. When one does not know anything about the prior density of the initial state, it is reasoanable to represent α_1 as having diffuse prior density.

$$\nu_1 = y_1 - a_1, \quad F_1 = P_1 + \sigma_\epsilon^2$$

By substituting the above values in to the equations for a_2 and P_2, and letting $P_1 \to \infty$, we get $a_2 = y_1, P_2 = \sigma_\epsilon^2 + \sigma_\eta^2$, and we can then proceed normally with KF for $t = 2, \ldots, n$. Based on the diffuse prior, the state and smoothing equations also have to be changed. They are not affected for $t = 2, \ldots, n$ but for the smoothed state for initial state and smoothed conditional variance of initial state change. So do smoothed mean and variance of the disturbances.

$$\hat{\alpha}_1 = y_1 + \sigma_\epsilon^2 r_1$$

$$V_1 = \sigma_\epsilon^2 - \sigma_\epsilon^4 N_1$$

Use of diffuse prior might find resistance from some people as they may regard the assumption of infinite variance as unnatural. An alternative approach is to assume α_1 is an unknown constant to be estimated from data. Using this approach one initializes $a_1 = y_1$ and $P_1 = \sigma_\epsilon^2$. It follows that we obtain the same initialization of KF by representing α_1 as a random variable with infinite variance as by assuming that it is fixed and unknown.

2.9 Parameter estimation

Parameters in the state space models are often called *hyperparameters* to distinguish them from elements of state vector which can plausibly be thought of as random parameters.The loglikelihood function of the observation data is written using the conditional probability of innovation terms and a straightforward MLE estimation can be done. However a better way to estimate parameters is by using EM algorithm that is dealt in Chapter 7. To do MLE using `KFAS`, one can use `fitSSM` function.

```
model    <- SSModel( Nile ~ SSMtrend(1, Q = list(matrix(NA))) , H = matrix(NA))
model    <- fitSSM( inits = c(log(var(Nile)) , log(var(Nile) ) ) ,
                    model = model, method  ='BFGS' )$model
c(model$H)

## [1] 15099

c(model$Q)

## [1] 1469
```

The section also shows a way to re-parametrise the model to obtain a concentrated diffuse likelihood function. This is maximized to estimate the parameters.

2.10 Steady state

KF converging to a steady state is important thing to know as it makes the recursive computations even faster. The condition for steady state is given by the condition

$$x > 0, \text{ where } x = h + \sqrt{h^2 + 4 * h}/2, \quad h = \sigma_\eta^2/\sigma_\epsilon^2$$

This holds for all non-trivial local level models.

2.11 Diagnostic checking

The assumptions underlying the local level model are that the disturbances ϵ_t and η_t are normally distributed and serially independent with constant variances. On these assumptions the standardized one-step forecast errors are also normally distributed and serially independent with unit variance. `KFAS` has functions to retrieve the standardized residuals and one step ahead forecast errors

```
out      <- KFS(model, filtering='state', smoothing=c('state','signal','disturbance'),
                simplify=F)
stdres  <- rstandard(out)
```

```
par(mfrow=c(2,2))
temp       <- ts(stdres,start = 1871)
plot(temp, col = "blue",lwd = 2,xlab="",ylab="", main ="i")
abline(h=0, col = "grey")
```

```
hist(stdres, prob=T, col = "grey", main = "ii", xlab ="",ylab="")
lines(density(stdres[-1]), col = "blue",lwd = 2)

qqnorm(c(stdres), col="blue",main = "iii")
qqline(c(stdres),col="red")

acf(stdres[-1], ,main="iv")
```

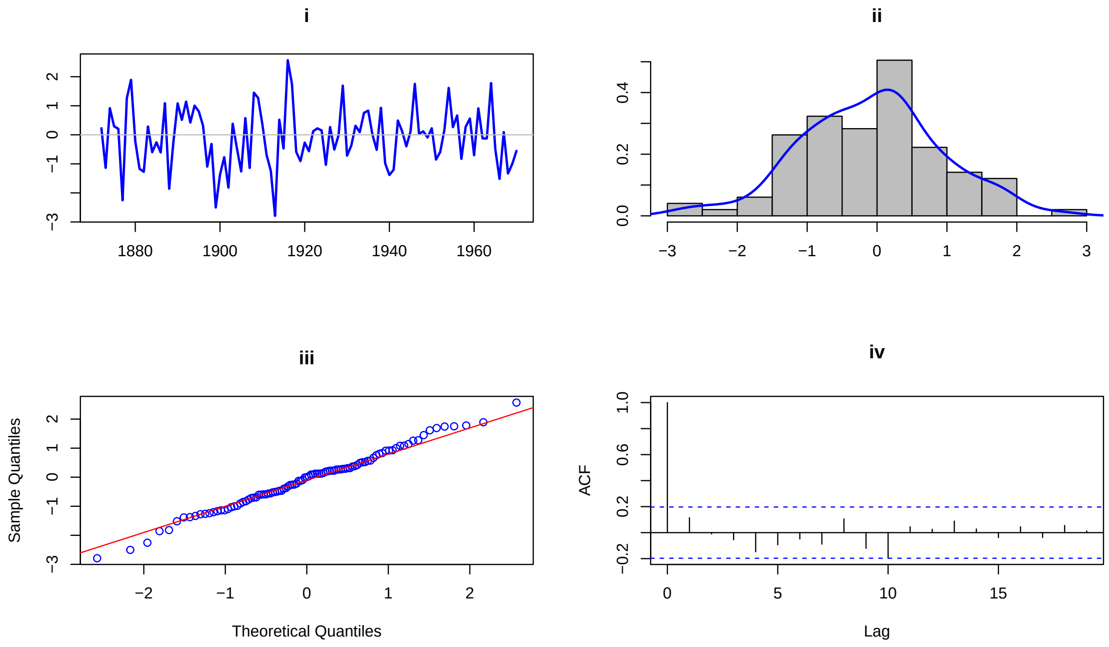


Figure 2.7: Diagnostic plots for standardised prediction errors : (i) standardised residual, (ii) histogram plus estimated density, (iii) ordered residuals (iv) correlogram

```
out       <- KFS(model, filtering='state', smoothing=c('state','signal','disturbance'),
                simplify=F)
temp1  <-  out$epshat/sqrt(c(out$V_eps))
temp2  <-  out$etahat/sqrt(c(out$V_eta))
```

```
par(mfrow=c(2,2))
temp        <- ts(temp1,start = 1871)
plot(temp, col = "blue",lwd = 2,xlab="",ylab="",
     main="i")
abline(h=0, col = "grey")

hist(temp, prob=T, col = "grey", ,ylim=c(0,0.3),
     main="ii",xlab="",ylab="")
lines(density(temp), col = "blue",lwd = 2)
```

```
temp        <- ts(temp2,start = 1871)
plot(temp, col = "blue",lwd = 2,xlab="",ylab="" ,main ="iii")
abline(h=0, col = "grey")

hist(temp, prob=T, col = "grey", main = "iv",
     ylim=c(0,1.2),xlab="",ylab="")
lines(density(temp), col = "blue",lwd = 2)
```

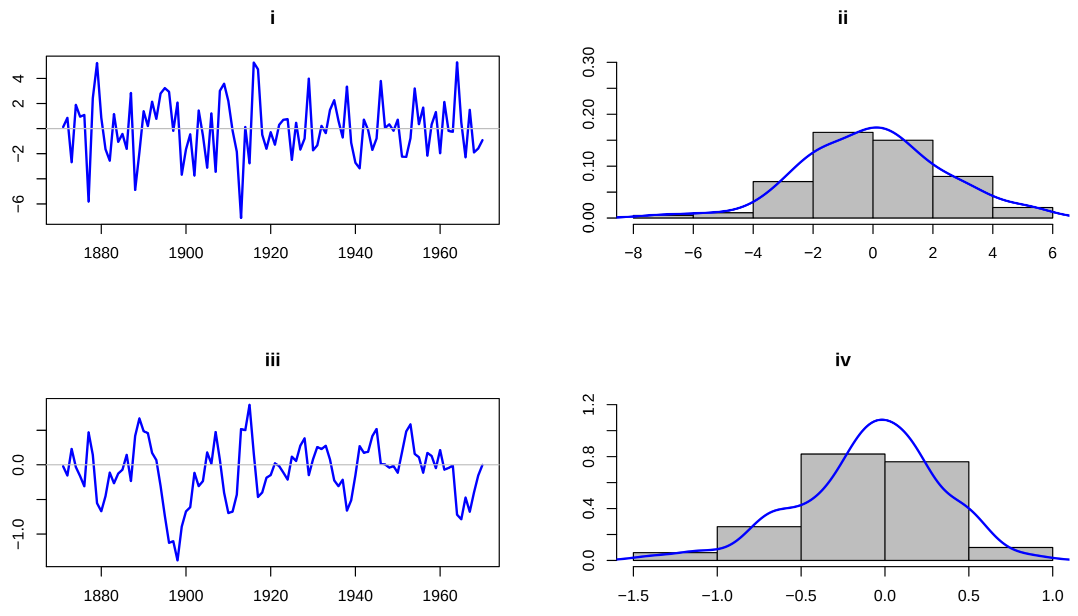


Figure 2.8: Auxilary Residual plots : (i) observation residual , (ii) histogram and estimated density for observation residual, (iii) state residual , (iv) histogram and estimated density for state residual

By sprinkling the chapter with adequate visuals, the authors provide the reader a nice trailer of all the principles that are relevant for state space modeling.

3 Linear Gaussian state space models

3.1 Introduction

The general linear Gaussian state space model can be written in a variety of ways. The version used in the book is

$$
\begin{aligned}
y_t &= Z_t \alpha_t + \epsilon_t, \quad \epsilon_t \sim N(0, H_t) \\
\alpha_{t+1} &= T_t \alpha_t + R_t, \quad \eta_t \sim N(0, Q_t)
\end{aligned}
$$

where y_t is a $p \times 1 vector$ of observations and α_t is an unobserved $m \times 1$ vector called the *state vector*. The first equations is called the *observation equation* and the second equation is called the *state equation*. The initial state vector $\alpha_1 \sim N(a_1, P_1)$. The matrices Z_t, T_t, R_t, H_t, Q_t are initially assumed to be known and the error terms ϵ_t and η_t are assumed to be serially independent and independent of each other at all time points. Matrices Z_t and T_{t-1} are permitted to depend on $y_1, y_2, \ldots y_{t-1}$. The first equation has the structure of a linear regression model where the coefficient vector α_t varies over time. The second equation represents a first order vector autoregressive process, the Markovian nature of which accounts for many of the elegant properties of the state space model.

3.2 Structural time series models

A *structural time series model* is one in which the trend, seasonal and error terms in the local level, plus other relevant components are modelled explicitly. This is in sharp contrast to the philosophy underlying the philosophy underlying Box-Jenkins ARIMA models where trend and seasonal are removed by differencing prior to the detailed analysis. This section shows the state space model version of trend , trend + seasonality , regression models, seemingly unrelated time series equations model.

3.3 ARMA models and ARIMA models

ARIMA models eliminate the trend and seasonality by differencing and then try to fit a stationary ARMA process to the differenced series. These models can be readily put in state space framework as follows :

Consider the ARMA(p,q) process defined as

$$
Y_t = \sum_{j=1}^{r} \phi_j Y_{t-j} + \sum_{j=1}^{r} \theta_j \zeta_{t-j} + \zeta_t, \quad r = \max(p, q+1), \phi_j = 0 \text{ for } p > q, \zeta_j = 0 \text{ for } j > q,
$$

$$
\begin{aligned}
Z_t &= [1\,0\,\ldots\,0] \\
T_t &= \begin{bmatrix} \phi_1 & 1 & 0 & \ldots & 0 \\ \phi_2 & 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \\ \phi_{r-1} & 0 & 0 & \ldots & 1 \\ \phi_r & 0 & 0 & \ldots & 0 \end{bmatrix} \\
R_t &= [1\,\theta_1\,\ldots\,\theta_{r-1}]'
\end{aligned}
$$

The basic idea of the above representation is to shift the ARMA dependence from Y_t to the state vector $Y_t = \alpha_{1,t}$.

The DLM representation of an ARIMA(p,d,q) model with $d > 0$ is : For a general d, set $Y_t^* \Delta_d Y_t$.

$$
\Delta^{d-j} Y_t = Y_t^* + \sum_{i=1}^{j} \Delta^{d-i} Y_{t-1}, \quad j = 1, \ldots, d
$$

$$
\alpha_t = \begin{bmatrix} Y_{t-1} \\ \Delta Y_{t-1} \\ \vdots \\ \Delta^{d-1} Y_{t-1} \\ Y_t^* \\ \phi_2 Y_{t-1}^* + \ldots \phi_r Y_{t-r+1}^* + \theta_1 \epsilon_t + \ldots + \theta_{r-1} \zeta_{t-r+2} \\ \phi_3 Y_{t-1}^* + \ldots \phi_r Y_{t-r+2}^* + \theta_2 \epsilon_t + \ldots + \theta_{r-1} \zeta_{t-r+3} \\ \vdots \\ \phi_r Y_{t-1}^* + \theta_{r-1} \zeta_t \end{bmatrix}
$$

$$
\begin{aligned}
Z_t &= [1\,1 \ldots 1\,0 \ldots 0] \\
T_t &= \begin{bmatrix} 1 & 1 & \ldots & 1 & 0 & \ldots & 0 & \\ 0 & 1 & \ldots & 1 & 0 & \ldots & 0 & \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \\ 0 & \ldots & 1 & 1 & 0 & \ldots & 0 & \\ 0 & \ldots & 0 & \phi_1 & 1 & 0 & \ldots & 0 \\ 0 & \ldots & 0 & \phi_2 & 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \\ \ldots & \ldots & \ldots & \phi_{r-1} & 0 & 0 & 0 & 1 \\ \ldots & \ldots & \ldots & \phi_r & 0 & 0 & 0 & 0 \end{bmatrix} \\
R_t &= [0 \ldots 0\,1\,\theta_1\,\ldots\,\theta_{r-1}]'
\end{aligned}
$$

ARMA estimation using dlm package

```
set.seed(1)
y     <- arima.sim(n = 100, list(ar = c(0.8897), ma = c(-0.2279)),sd = sqrt(0.1796))
arima(y,c(1,0,1),include.mean=T)

## Series: y
## ARIMA(1,0,1) with non-zero mean
##
## Coefficients:
##          ar1     ma1  intercept
##        0.932  -0.294     -0.015
## s.e.  0.041   0.099      0.360
##
## sigma^2 estimated as 0.15:  log likelihood=-47.7
## AIC=103.4   AICc=103.8   BIC=113.8

fn <- function(params){
    model <- dlmModARMA(ar = params[1],
                        ma = params[2],
                        sigma2 = exp(params[3]),
                        dV = 1e-7)
}
fit <- dlmMLE(y,rep(0,3),fn)
#ARMA parameters
print(c(fit$par[1:2]))

## [1]  0.9324 -0.2966

# error variance
exp(fit$par[3])

## [1] 0.1499
```

ARMA estimation using KFAS package

It took me sometime to figure out the way to write my own function that can be used in MLE.

```
model    <- SSModel(y ~ SSMarima(ar = c(0.5), ma = c(0.5), d = 0, Q = 1e-7),H=1e-7)
fn              <-function(pars,model,...){
  model$T[2,2,1]      <- pars[1]
  model$R[3,,1]       <- pars[2]
  model$Q[,,1]        <- exp(pars[3])
  model
}


model  <- fitSSM(inits=c(0.5,0.5,0.5),
            model=model,updatefn = fn,method='BFGS')$model
#ar component
```

```
model$T[2,2,1]

## [1] 0.9735

#ma component
model$R[3,,1]

##  arima2
## -0.3161

#Q
c(model$Q)

## [1] 0.1533
```

3.4 Exponential smoothing

EWMA started appearing in 1950s and became very popular method as it required little storage of past data and could be easily implemented on a computer. For a one step ahead forecasting of y_{t+1} given a univariate time series $y_t, y_{t-1}, \ldots$, it has the form

$$\hat{y}_{t+1} = (1-\lambda)y_t + \lambda\hat{y}$$

This can be shown to be an ARIMA model. EWMA forecasts produced by the recursion are minimum mean square error forecasts in the sense that they minimize $E(\hat{y}_{t+1} - y_{t+1})^2$ for the observations from a local level model. The EWMA model was extended by Holt(1957) and Winters(1960) to a series containing trend and seasonal. This extension is also a type of ARIMA model. It was shown later that the forecasts produced by these Holt-Winters recursions are minimum mean square error forecasts for the local linear trend model. This pattern continues with adding seasonal components too. The state space models and the Box-Jenkins ARIMA models which appear to be very different conceptually but both give minimum mean square error forecasts from EWMA recursions. The explanation is that when the time series has an underlying structure which is sufficiently simple, then the appropriate state space and ARIMA models are essentially equivalent. It is when we move towards more complex structures that the differences emerge.

3.5 State space versus Box-Jenkins approaches

I found this section brilliant as it brings out so many aspects that Box-Jenkins approach misses out. There are a dozen are so, differences mentioned that show that state space models are superior to ARIMA models. This is the nicest comparison between state space models and Box-Jenkins methods that I have come across till date. I think this sort of comparison must be mentioned by all the faculty who teach undergrad classic time series stuff, so that interested students can dig up the content on State space models and learn to model at a more generic level.

	State Space Models	Box-Jenkins approach
1	Wide class of problems can be encapsulated in a simple linear model	specific type of models only
2	Due to Markovian nature of the model, the calculations needed for practical application of the model could be set up in recursive form in a way that is particularly convenient on a computer	non recursive nature
3	The key advantage of the state space approach is that is based on structural analysis of the problem. The different components that make up the series such as trend, seasonal , cycle, and calendar variations, together with the effects of explanatory variables and interventions	more a black box approach where the model adopted depends purely on the data without prior analysis of the structure of the system over time.
4	Multivariate observations can be handled by straightforward extensions	not the case here
5	Explanatory variables can be easily incorporated	not the case here
6	Associated regression coefficients can be made to vary over time	not the case here
7	Because of Markovian form, the calculations can be put in a recursive form. This enables increasingly large models to be handled effectively without disproportionate increases in the computational burden.	not the case here
8	No dependency on differencing	Reliance on differencing before fitting an stationary series
9	No dependency on differencing	Artificial requirement that differenced series is stationary
10	Easy to handle missing data	not the case here
11	No reliance on auto correlation	Main tool is the sample auto correlation function which is notoriously imprecise due to high sampling variability
12	Highly transparent	not the case here

3.6 Regression with time-varying coefficient

If we want to the coefficient vector α to vary over time, state space models are flexible enough to incorporate this requirement. All one needs to do is that write a random walk equation for each of the coefficients.

3.7 Regression with ARMA error

Without any difficulty, one can easily set up the required matrices for a regression framework where the errors follow ARMA. These matrices are given in the section.

The chapter ends with a section on state space models in continuous time.

4 Filtering, smoothing and forecasting

This chapter is the core chapter from the book that shows the derivations behind all the main estimates from a state space model.

$$\begin{aligned} y_t &= Z_t\alpha_t + \epsilon_t, & \epsilon_t &\sim N(0, H_t) \\ \alpha_{t+1} &= T_t\alpha_t + R_t\eta_t, & \eta_t &\sim N(0, H_t) \\ & & \alpha_1 &\sim N(a_1, P_1) \end{aligned}$$

The key theme running behind all these derivations is recursion. All the recursive formula derivations flow from the matrix form of conditional normal moments.

4.1 Filtering

The key term that makes computations easier is the innovation term denoted by ν_t where $\nu_t = y_t - E(y_t|Y_{t-1})$, where $Y_t = \{y_1, \ldots, y_t\}$. Some useful relations and the recursive formula for Kalman Filtering.

$$\begin{aligned} \nu_t &= y_t - Z_t a_t \\ M_t &= \mathrm{Cov}(\alpha_t, \nu_t) = P_t Z_t' \\ F_t &= \mathrm{Var}(\nu_t) = Z_t P_t Z_t' + H_t \\ K_t &= T_t M_t F_t = T_t P_t Z_t' F_t^{-1} \\ L_t &= T_t - K_t Z_t \\ a_{t+1} &= T_t a_t + K_t \nu_t \\ P_{t+1} &= T_t P_t L_t' + R_t Q_t R_t' \end{aligned}$$

When dealing with a time-invariant state space model in which system matrices are constant over time, the Kalman recursion of P_{t+1} converges to a constant matrix $\overline{P}$ which is the solution to the matrix equation

$$\overline{P} = T\overline{P}T' - T\overline{P}Z'\overline{F}^{-1}Z\overline{P}T' + RQR', \quad \text{where } \overline{F} = Z\overline{P}Z' + H$$

State space model can be written in terms of state estimation error$x_t = \alpha_t - a_t$ and this form is called innovation form of state space model.

$$\begin{aligned} \nu_t &= Z_t x_t + \epsilon_t, \quad \epsilon_t \sim N(0, H_t) \\ x_{t+1} &= L_t x_t + R_t\eta_t - K_t\epsilon_t \end{aligned}$$

The advantage of writing an equation in terms of forecast errors is that these error are independent of each other and hence make computations easier.

4.2 State smoothing

This involves estimation of α_t given the entire series $y_1, \ldots, y_n$ and is denoted by $\hat{\alpha}_t$. For deriving smoothing recursion innovation form is used as the estimation can be split in to two independent set of variables

$\{y_1, \ldots, y_{t-1}\}$ and $\{\nu_t, \nu_{t+1}, \ldots \nu_n\}$

Smoothed state vector

$$\begin{aligned} r_{t-1} &= Z_t' F_t^{-1} \nu_t + L_t' r_t \\ \hat{\alpha}_t &= a_t + P_t r_{t-1} \\ r_n &= 0 \end{aligned}$$

$$r_t = Z_{t+1}' F_{t+1}' \nu_{t+1} + L_{t+1}' Z_{t+2}' F_{t+2}^{-1} \nu_{t+2} + \ldots + L_{t+1}' \ldots L_{n-1}' Z_n' F_n^{-1} \nu_n$$

As one can see the smoothed estimate is a linear combination of filtered mean and one step ahead forecast errors.

Smoothed state variance

$$\begin{aligned} N_{t-1} &= Z_t' F_t^{-1} Z_t + L_t' N_t L_t \\ \hat{V}_t &= P_t - P_t N_{t-1} P_t \\ N_n &= 0 \end{aligned}$$

$$N_t = Z_{t+1}' F_{t+1}' \nu_{t+1} + L_{t+1}' Z_{t+2}' F_{t+2}^{-1} Z_{t+2} L_{t+1} + \ldots + L_{t+1}' \ldots L_{n-1}' Z_n' F_n^{-1} Z_n L_{n-1} \ldots L_{t+1}$$

As one can see the smoothed estimate is a linear combination of filtered mean and one step ahead forecast errors. The above filtering and smoothing recursions enable us to update our knowledge of the system each time a new observations comes in. The key advantage of the recursion is that we do not have to invert a $(pt \times pt)$ matrix to fit the model each time the tth observation comes in for $t = 1, \ldots, n$,We have to invert the $(p \times p)$ matrix F_t and p is generally much smaller than n. For the multivariate case, an additional improvement can be made by introducing the elements of the new observation one at time rather than the whole observation.

4.3 Disturbance smoothing

This section derives the recursions for $\hat{\epsilon}_t = E(\epsilon_t|y)$ and $\hat{\eta}_t = E(\eta_t|y)$. The recursive equations are as follows:

$$\begin{aligned} \hat{\epsilon}_t &= H_t(F_t^{-1}\nu_t - K_t r_t') \\ \text{Var}(\epsilon_t|y) &= H_t - H_t(F_t^{-1} + K_t' N_t K_t) H_t \\ \hat{\eta}_t &= Q_t R_t' r_t \\ \text{Var}(\eta_t|y) &= Q_t - Q_t R_t' N_t R_t Q_t \\ r_{t-1} &= Z_t' F_t^{-1} \nu_t + L_t' r_t \\ N_{t-1} &= Z_t' F_t^{-1} Z_t + L_t' N_t L_t \\ r_n &= 0 \\ N_n &= 0 \end{aligned}$$

These estimates have a variety of uses, particularly for parameter estimation and diagnostic checking. The above equations are collectively referred to as *disturbance smoothing recursion*

4.4 Covariance matrices of smoothed observations

This section contains pretty laborious derivations of covariance matrices. pretty painful to go over. Thanks to `KFAS`, you get all the output without having to code recursive equations.

4.5 Weight functions

The conditional mean vectors and variance matrices from the filtering , smoothing and disturbance smoothing estimates are all weighted sums of $y_1, \dots, y_n$. It is of interest to study these weights to gain a better understanding of the properties of the estimators. Models which produce weight patterns for trend component which differ from what is regarded as appropriate should be investigated. In effect, the weights can be regarded as what are known as kernel functions in the field of nonparametric regression.

4.6 Simulation Smoothing

The drawing of samples of state or disturbance vectors conditional on the observations held fixed is called simulation smoothing. Such samples are useful for investigating the performance of techniques of analysis proposed for the linear Gaussian model and for Bayesian analysis based on this model. The primary purpose of simulation smoothing in the book is to serve as the basis for importance sampling techniques. The recursive algorithms in this section have been replaced by a simpler version, written by the same authors after this book was published. The paper is titled, "A simple and efficient simulation smoother for state space time series analysis".

I will take a brief detour here and summarize the new and efficient simulation smoother.

Simple and Efficient Smoother

For any model that you build, it is imperative that you are able to sample random realizations of the various parameters of the model. In a structural model, be it a linear Gaussian or nonlinear state space model, an additional requirement is that you have to sample the state and disturbance vector given the observations. To state it precisely, the problem we are dealing is to draw samples from the conditional distributions of $\epsilon = (\epsilon_1', \epsilon_2', \dots, \epsilon_n')'$, $\eta = (\eta_1', \eta_2', \dots, \eta_n')'$ and $\alpha = (\alpha_1', \alpha_2', \dots, \alpha_n')'$ given $y = (y_1', y_2', \dots, y_n')'$.

The past literature on this problem is :

- Fruhwirth-Schnatter(1994), Data augmentation and dynamic linear models. : The method comprised drawing samples of $\alpha|y$ recursively by first sampling $\alpha_n|y$, then sampling $\alpha_{n-1}|\alpha_n, y$, then $\alpha_{n-2}|\alpha_{n-1}, \alpha_n, y$.
- de Jong and Shephard(1995),The simulation smoother for time series models. : This paper was a significant advance as the authors first considered sampling the disturbances and subsequently sampling the states. This is more efficient than sampling the states directly when the dimension of η is smaller than the dimension of α

In this paper, the authors present a new simulation smoother which is simple and is computationally efficient relative to that of de Jong and Shephard(1995).

The dataset used in the paper is “Nile” dataset. The State space model used is *Local level model*

$$\begin{aligned} y_t &= \alpha_t + \epsilon_t, \quad \epsilon_t \sim N(0, \sigma^2_\epsilon) \\ \alpha_{t+1} &= \alpha_t + \eta_t, \quad \eta_t \sim N(0, \sigma^2_\eta) \end{aligned}$$

The New Simulation Smoother

Algorithm 1

1. Draw a random vector w^+ from density $p(w)$ and use it to generate y^+ by means of the recursive observation and system equation with the error terms related by w^+, where the recursion is initialized by the draw $\alpha_1^+ \sim N(a_1, P_1)$

$$p(w) = N(0, \Omega), \quad \Omega = \text{diag}(H_1, \ldots, H_n, Q_1, \ldots, Q_n)$$

2. Compute $\hat{w} = (\hat{\epsilon}', \hat{\eta}')' = E(w|y)$ and $\hat{w}^+ = (\hat{\epsilon^+}', \hat{\eta^+}')' = E(w^+|y^+)$ by means of standard Kalman filtering and disturbance smoothing using the following equations

$$\begin{aligned} \hat{\epsilon}_t &= H_t F_t^{-1} \nu_t - H_t K_t' r_t \\ \hat{\eta}_t &= Q_t R_t' r_t \\ r_{t-1} &= Z_t F_t^{-1} \nu_t + L_t' r_t \end{aligned}$$

3. Take $\tilde{w} = \hat{w} - \hat{w}^+ + w^+$

4.7 Simulation of State vector

Algorithm 2

1. Draw a random vector w^+ from density $p(w)$ and use it to generate α^+, y^+ by means of the recursive observation and system equation with the error terms related by w^+, where the recursion is initialized by the draw $\alpha_1^+ \sim N(a_1, P_1)$

$$p(w) = N(0, \Omega), \quad \Omega = \text{diag}(H_1, \ldots, H_n, Q_1, \ldots, Q_n)$$

2. Compute $\hat{\alpha} == E(\alpha|y)$ and $\hat{\alpha}^+ = E(\alpha^+|y^+)$ by means of standard Kalman filtering and smoothing equations
3. Take $\tilde{\alpha} = \hat{\alpha} - \hat{\alpha}^+ + \alpha^+$

R excursion

Now let’s use the above algorithms to generate samples from the state vector and disturbance vector. Let me use the same “Nile” dataset.

Algorithm 1 - Implementation

```
# MLE for estimating observational and state evolution variance.
fn               <- function(params){
                 dlmModPoly(order= 1, dV= exp(params[1]) , dW = exp(params[2]))
                 }
y                 <- c(Nile)
fit               <- dlmMLE(y, rep(0,2),fn)
mod               <- fn(fit$par)
(obs.error.var    <- V(mod))

##        [,1]
## [1,] 15100

(state.error.var <- W(mod))

##       [,1]
## [1,] 1468

filtered          <- dlmFilter(y,mod)
smoothed          <- dlmSmooth(filtered)
smoothed.state    <- dropFirst(smoothed$s)
w.hat             <- c(Nile- smoothed.state,c(diff(smoothed.state),0))
```

Step 1 :

```
set.seed(1)
n              <- length(Nile)

# Step 1
w.plus         <- c( rnorm(n,0,sqrt(obs.error.var)),rnorm(n,0,sqrt(state.error.var)))
alpha0         <- rnorm(1, mod$m0, sqrt(mod$C0))

y.temp         <- numeric(n)
alpha.temp     <- numeric(n)

#Step 2
for(i in 1:n){
  if(i==1){
    alpha.temp[i] <- alpha0 + w.plus[100+i]
  }else{
    alpha.temp[i] <- alpha.temp[i-1] + w.plus[100+i]
  }
  y.temp[i]       <- alpha.temp[i] + w.plus[i]
}
temp.smoothed     <- dlmSmooth(y,mod)
```

```
alpha.smoothed    <- dropFirst(temp.smoothed$s)
```

Step 2 :

```
w.hat.plus        <- c(Nile- alpha.smoothed,c(diff(alpha.smoothed),0))
```

Step 3 :

```
w.tilde        <- w.hat - w.hat.plus + w.plus
```

This generates one sample of the observation and state disturbance vector.
Let's simulate a few samples and overlay them on the actual observation and state disturbance vectors

```
set.seed(1)
n              <- length(Nile)
disturb.samp  <- matrix(data= 0,nrow = n*2, ncol = 25)

for(b in 1:25){
  # Step 1
  w.plus        <- c( rnorm(n,0,sqrt(obs.error.var)),rnorm(n,0,sqrt(state.error.var)))
  alpha0        <- rnorm(1, mod$m0, sqrt(mod$C0))

  y.temp        <- numeric(n)
  alpha.temp    <- numeric(n)

  #Step 2
  for(i in 1:n){
    if(i==1){
      alpha.temp[i] <- alpha0 + w.plus[100+i]
    }else{
      alpha.temp[i] <- alpha.temp[i-1] + w.plus[100+i]
    }
    y.temp[i]       <- alpha.temp[i] + w.plus[i]
  }
  temp.smoothed     <- dlmSmooth(y.temp,mod)
  alpha.smoothed    <- dropFirst(temp.smoothed$s)
  w.hat.plus        <- c(y.temp- alpha.smoothed,c(diff(alpha.smoothed),0))
  #Step 3
  w.tilde           <- w.hat - w.hat.plus + w.plus
  disturb.samp[,b]  <- w.tilde
}
```

```
temp  <- cbind(disturb.samp[101:200,],w.hat[101:200])
plot.ts(temp, plot.type="single",col =c(rep("grey",25),"blue"),
        ylim = c(-150,200),lwd=c(rep(0.5,25),3), ylab = "",xlab="")
leg <-c("realized state error","simulated state error")
legend("topright",leg,col=c("blue","grey"),lwd=c(3,1),cex=0.7,bty="n")
```

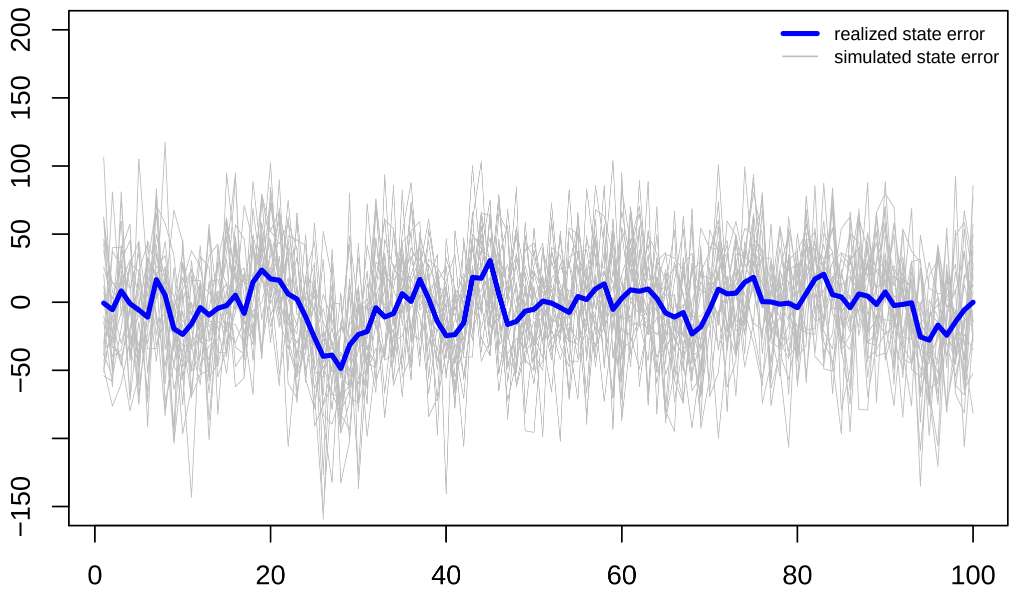


Figure 4.1: Comparison of Simulated and Realized State disturbance

```r
temp  <- cbind(disturb.samp[1:100,],w.hat[1:100])
plot.ts(temp, plot.type="single",col =c(rep("grey",25),"blue"),
        ,lwd=c(rep(0.5,25),3), ylab = "",xlab="")
leg <-c("realized observation error","simulated observation error")
legend("topright",leg,col=c("blue","grey"),lwd=c(3,1),cex=0.7,bty="n")
```

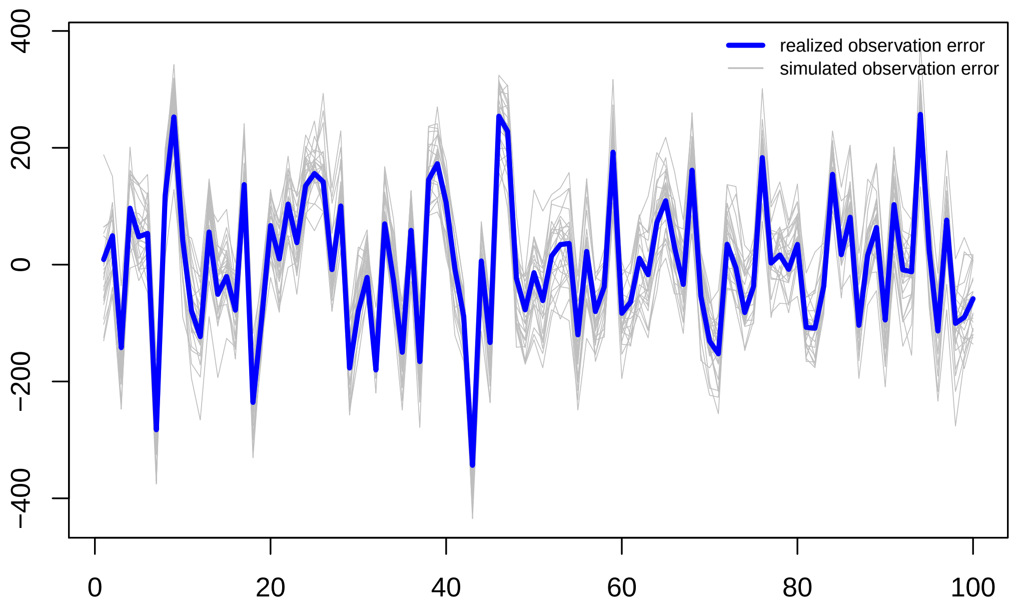


Figure 4.2: Comparison of Simulated and Realized Observation disturbance

Algorithm 2 - Implementation

```r
set.seed(1)
n              <- length(Nile)
alpha.samp     <- matrix(data= 0,nrow = n, ncol = 25)
for(b in 1:25){
  # Step 1
  w.plus        <- c( rnorm(n,0,sqrt(obs.error.var)),rnorm(n,0,sqrt(state.error.var)))
  alpha0        <- rnorm(1, mod$m0, sqrt(mod$C0))

  y.temp        <- numeric(n)
  alpha.temp    <- numeric(n)

  #Step 2
  for(i in 1:n){
    if(i==1){
      alpha.temp[i] <- alpha0 + w.plus[100+i]
    }else{
      alpha.temp[i] <- alpha.temp[i-1] + w.plus[100+i]
    }
    y.temp[i]       <- alpha.temp[i] + w.plus[i]
  }
  temp.smoothed     <- dlmSmooth(y.temp,mod)
  alpha.smoothed    <- dropFirst(temp.smoothed$s)
  alpha.hat.plus    <- alpha.smoothed

  #Step 3
  alpha.tilde      <- smoothed.state - alpha.hat.plus + alpha.temp
  alpha.samp[,b]   <- alpha.tilde
}
```

```
temp        <- cbind(alpha.samp,smoothed.state)
plot.ts(temp, plot.type="single",col =c(rep("grey",25),"blue"),
        ,lwd=c(rep(0.5,25),3), ylab = "",xlab="")
leg <-c("realized state vector","simulated state vector")
legend("topright",leg,col=c("blue","grey"),lwd=c(3,1),cex=0.7,bty="n")
```

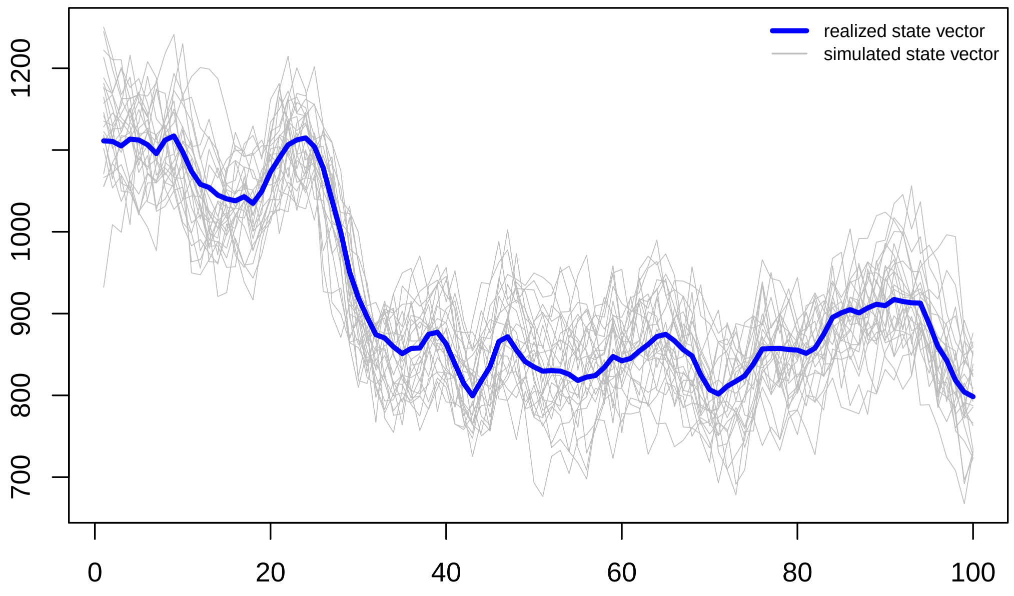


Figure 4.3: Comparison of Simulated and Realized State vector

Simulation via KFAS package

KFAS package closely follows the implementations suggested by the authors. One can simulate the state observations using simulateSSM() function. One can also turn on the antithetic sampling suggested in this paper via the function's argument.

```
modelNile   <-SSModel(Nile~SSMtrend(1,Q=list(matrix(NA))),H=matrix(NA))
modelNile   <-fitSSM(inits=c(log(var(Nile)),log(var(Nile))),
                      model=modelNile,
                       method='BFGS',
                     control=list(REPORT=1,trace=0))$model
out          <- KFS(modelNile,filtering='state',smoothing='state')
state.sample <- simulateSSM(modelNile, type = c("states"))
```

```
temp       <- cbind(state.sample,smoothed.state)
plot.ts(temp, plot.type="single",col =c(rep("grey",1),"blue"),
        ,lwd=c(rep(1,1),2), ylab = "",xlab="")
leg <-c("realized observation error","simulated observation error")
legend("topright",leg,col=c("blue","grey"),lwd=c(3,1),cex=0.7,bty="n")
```

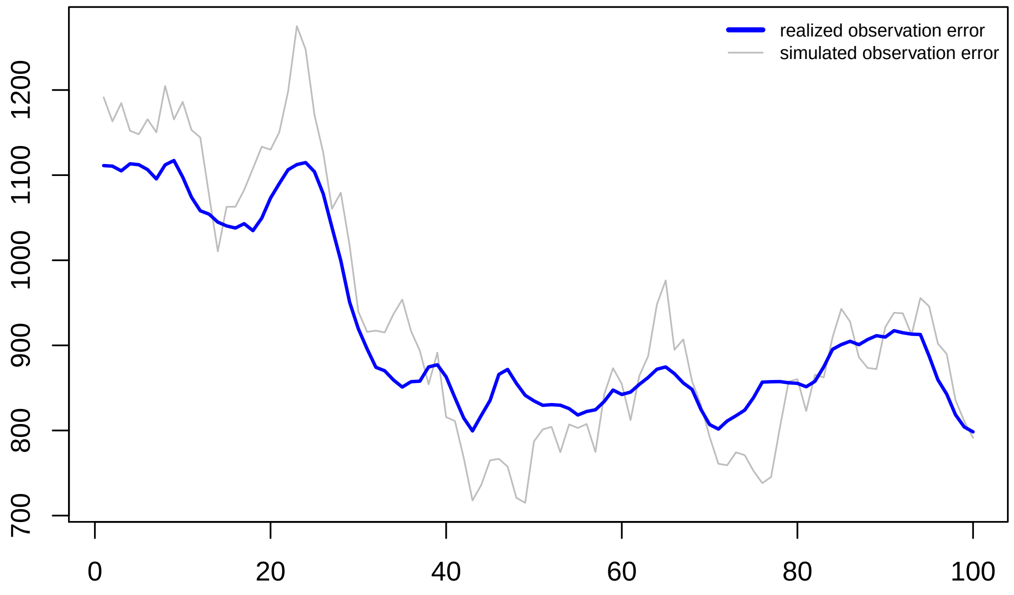


Figure 4.4: Comparison of Simulated and Realized State vector

Conclusion

The paper presents a simulation smoother for drawing samples from the conditional distribution of the disturbances given the observations. Subsequently the paper highlights the advantages of this simulation technique over the previous methods.

- derivation is simple
- the method requires only the generation of simulated observations from the model together with the Kalman Filter and standard smoothing algorithms
- no inversion of matrices are needed beyond those in the standard KF
- diffuse initialization of state vector is handled easily
- this approach solves problems arising from the singularity of the conditional variance matrix W

4.8 Missing observations

One of the advantages of dealing with state space models is that the missing observations can be easily integrated in to the framework. When the set of observations y_t for $t = \tau, \dots, \tau^* - 1$ is missing, the vectors ν_t and Kalman gain matrix K_t are set to 0. Thus the updates become

$$\begin{aligned} a_{t+1} &= T_t a_t \\ P_{t+1} &= T_t P_t T_t' + R_t Q_t R_t', \quad t = \tau, \dots, \tau^* - 1 \\ r_{t-1} &= T_t' r_t \\ N_{t-1} &= T_t' N_t T_t \end{aligned}$$

A similar treatment is done in the multivariate case When some but not all elements of the observation vector y_t are missing.Of course, the dimensionality of the observation vector varies over time, but this does not affect the validity of the formula.

4.9 Forecasting

Forecasting in State space framework is very easy. All you have to is to extend the observation series to whatever number of steps ahead you want to forecast and replace it with missing values. Why does this work ? In the case of missing observations, the filtered and smoothed recursions go through after replacing the ν's associated with the missing observations as 0 and replacing Kalman gain matrices as 0. In the case of forecasting, if you take the loss function as minimum mean squared error, then the conditional expectation given data turns out to have the same form as recursions based on missing values. The authors in fact derive the forecasts in the book and show them to be equal to those formed by assuming missing observations.
The last section of the chapter casts the state space model in a matrix format, which enables the reader to get a quicker derivation of smoothing estimates, state disturbance vectors etc.

5 Initialisation of filter and smoother

In all the previous chapters the starting value , the starting value came from a known distribution, $\alpha_1 \sim N(a_1, P_1)$. This chapter develops methods of starting up the series when at least some of the elements of a_1 or P_1 are unknown. The methods fall under the name, *initialisation.* A general model for the initial state vector α_1 is

$$\alpha_1 = a + A\delta + R_0\eta_0, \quad \sim N(0, Q_0)$$

where the $m \times 1$ vector a is known, δ is a $q \times 1$ vector of unknown quantities, the $m \times q$ matrix A, and the $m \times (m-q)$ matrix R_0 are selection matrices I_m, they are defined so that when taken together, their columns constitute a set of g columns of I_m with $g \leq m$ and $A'R_0 = 0$. The matrix Q_0 is assumed to be positive definite and known. There are two cases to be considered here, first case is where δ is fixed and unknown, the second case is where δ is random. In the second case, the authors assume a specific form

$$\delta \sim N(0, \kappa I_q)$$

The initial conditions are such such that $a_1 = E(\alpha_1)$ and $P_1 = \text{Var}(\alpha_1)$, where

$$P_1 = \kappa P_\infty + P_*, \qquad \text{where } P_\infty = AA' \text{ and } P_* = R_0 Q_0 R_0'$$

A vector δ with distribution $N(0, \kappa I_q)$ as $k \to \infty$ is said to be *diffuse.* Initialisation of the Kalman filter when some elements of α_1 are diffuse is called *diffuse initialisation* of the filter. One approximate technique is to replace κ by a large number and run the kalman filter. The authors use a different approach where they expand F_t^{-1} . The mean square error matrix P_t has the decomposition

$$P_t = \kappa P_{\infty,t} + P_{*,t} + O(\kappa^{-1})$$

where $P_{\infty,t}, P_{*,t}$ do not depend on κ. The chapter also shows that $P_{\infty,t} = 0, t > d$ where d is a positive integer, which is small relative to n. The consequence is that the usual Kalman filter applies without change for $t = d+1, \ldots, n$. The recursion for exact initial state smoothing, exact initial disturbance smoothing and exact initial simulation smoothing are derived. I felt very painful to go over these derivations where there is nothing new here exact mind numbing math. A major relief for me was going through an example of structural time series, ARMA model ,non stationary ARIMA models and Regression model with ARMA errors, for whom exact initial Kalman filter and smoother are given. I skipped the augmented Kalman filter part of the chapter as it was kind of overwhelming for me after the heavy dose of initialisation process in the previous sections. The documentation in `KFAS` mentions this process in precise words

> KFS uses exact diffuse initialization where the unknown initial states are set to have a zero mean and infinite variance, so
>
> $$P_1 = \kappa P_\infty + P_*, \qquad \text{where } P_\infty = AA' \text{ and } P_* = R_0 Q_0 R_0'$$
>
> with κ going to infinity and $P_{\infty,1}$ being diagonal matrix with ones on diagonal elements corresponding to unknown initial states. Diffuse phase is continued until rank of $P_{\infty,1}$ becomes zero. Rank of P_∞ decreases by 1, if $F_\infty > tol > 0$. Usually the number of diffuse time points equals the number unknown elements of initial state vector, but missing observations or time varying Z can affect this.

6 Further Computational Aspects

This chapter is very interesting as it talks about the implementation issues with the state space recursive framework. However it begins with a method to incorporate regression effects within the Kalman filter which I felt had nothing to do with the title of the chapter. One of the methods that I followed was the inclusion of independent variables as state vectors and write trivial state vector equations for those coefficients. The only reason that this has been mentioned I guess is to make a point that the enlargement of the state space model will not cause extra computing because of the sparse nature of the system matrices.

6.1 Square root filter and smoother

Because of rounding errors and matrices being to singularity, the possibility arises that the calculated value of P_t is negative definite, or close to this, giving rise to unacceptable rounding errors. Since P_t appears in all the recursive equations, it can happen that the calculated value of P_t becomes negative definite when, for example, erratic changes occur in the system matrices over time.The problem can be avoided by using a transformed version of the Kalman filter called the *square root filter.* However the amount of computation required is substantially larger than that required for the standard Kalman filter. First the usual recursive equations used

$$\begin{aligned} F_t &= Z_t P_t Z_t' + H_t \\ K_t &= T_t P_t Z_t' + F_t^{-1} \\ P_{t+1} &= T_t P_t T_t' + R_t Q_t R_t' - K_t' F_t K_t' \end{aligned}$$

Define the partitioned matrix U_t by

$$U_t = \begin{bmatrix} Z_t \tilde{P}_t & H_t & 0 \\ T_t \tilde{P}_t & 0 & R_t \tilde{Q}_t \end{bmatrix}$$

where

$$P_t = \tilde{P}_t \tilde{P}_t', \quad H_t = \tilde{H}_t \tilde{H}_t', \quad Q_t = \tilde{Q}_t \tilde{Q}_t'$$

$$U_t U_t' = \begin{bmatrix} F_t & Z_t P_t T_t' \\ T_t P_t Z_t' & T_r P_t T_t' + R_t Q_t R_t' \end{bmatrix}$$

The matrix U_t is transformed in to a lower triangular matrix using the orthogonal matrix G , $U_t G = U_t^*$. The lower triangular matrix can be written as

$$U_t^* = \begin{bmatrix} U_{1,t}^* & 0 \\ U_{2,t}^* & U_{3,t}^* \end{bmatrix}$$

By equating the $U_t^* U_t^{*\prime}$, we get that $P_{t+1} = U_{3,t}^* U_{3,t}^{*\prime}$. Thus one finds that $U_{3,t}^* = \tilde{P}_{t+1}$. The recursive equation becomes

$$a_{t+1} = T_a a_t + U_{2,t}^* U_{1,t}^{*-1} \nu_t$$

The above equation is numerically stable.Matrices $U_{1,t}^*, U_{2,t}^*, U_{3,t}^*$ are used for deriving numerically smoothing recursions. The square root formulation is not that useful for initialisation since it usually only requires a limited number of d updates, the numerical problems are not substantial.

6.2 Univariate treatment of multivariate series

This section converts a multivariate KF in to a set of univariate KF equations, the main motivation being computational efficiency. This approach avoids the inversion of F_t matrix and two matrix multiplications. The results from the section show that there is considerable computational savings for higher dimensional observation data.

6.3 The algorithms of SsfPack

This is a commerical software and the authors provide all the functions relevant to state space model analysis. However we are living in a world of open source. My goto packages for state space models are `DLM` and `KFAS`. In fact the latter package closely mirrors this book and all the analysis done in the book can be replicated using `KFAS`. There are certain functions though that are not in the open source packages such as

- `SimSmoWgt`
- `SsfLikConc`
- `SsfLikSco`

However one can write these functions from the `KFAS` output with ease.

7 Maximum Likelihood estimation

7.1 Likelihood function, Hessian and Score function

Nobody is going to hand down the parameters for a state space model for they need to be estimated for the data. There are generally two ways one can estimate the meta parameters of the model, one via frequentist a.k.a MLE approach, the second is via Bayesian approach. This chapter deals with the first approach of computing the log likelihood function and then using Newton Raphson method to compute the MLE. The method basically solves the equation

$$\partial_1(\psi) = \frac{\partial \log L(y|\psi)}{\partial \psi} = 0$$

using the Taylor series

$$\partial_1(\psi) \approx \tilde{\partial}_1(\psi) + \tilde{\partial}_2(\psi)(\psi - \tilde{\psi})$$

for some trial value $\tilde{\psi}$, where

$$\tilde{\partial}(\psi) = \partial_1(\psi)|_{\psi=\tilde{\psi}}, \quad \tilde{\partial}_2(\psi) = \partial_2(\psi)|_{\psi=\tilde{\psi}}$$

The revised equation for $\tilde{\psi}$

$$\overline{\psi} = \tilde{\psi} - \tilde{\partial}_2(\psi)^{-1}\tilde{\partial}_1(\psi)$$

The gradient $\partial_1(\psi)$ determines the direction of the step taken to the optimum and the Hessian modifies the size of the step. It is possible to overstep the maximum in the direction determined by the vector

$$\tilde{\pi}(\psi) = -\partial_2(\psi)^{-1}\tilde{\partial}_1(\psi)$$

and therefore it is common to include a line search along the gradient vector within the optimisation process.

The first object needed for MLE is the likelihood function. There are three forms of likelihood functions depending on the initial conditions.

1. Prediction error decomposition - When the initial conditions are known
2. Diffuse Loglikelihood - When some of the elements of α_1 are diffuse
3. Likelihood when elements of the initial state vector are fixed but unknown

All the three forms are derived in this chapter.

The second object that one needs to get going on the Newton Raphson is the Hessian and the authors suggest the use of BFGS method. This method approximates the Hessian matrix in such a way that the resulting matrix remains negative definite.

The third object needed is the gradient vector or *score vector*. The chapter gives the score vector in all the possible cases based on the initial distribution. The math is laid out in a crystal clear way. What's the score vector ?

$$\partial_1(\psi) = \frac{\partial \log L(y|\psi)}{\partial \psi}$$

For the first case, the initial vector has the distribution $\alpha_1 \sim N(a_1, P_1)$ where a_1, P_1 are known.
Let $p(\alpha, y|\psi)$ be the joint density of α and y, $p(\alpha|y, \psi)$ be the conditional density of α given y, α and let $p(y|\psi)$

be the marginal density of y for given ψ.

$$\log p(y|\psi) = \log p(\alpha, y|\psi) - \log p(\alpha|y, \psi)$$

Let $\tilde{E}$ be the expectation with respect to density $p(\alpha|y, \psi)$ and taking expectations on both sides

$$\log p(y|\psi) = \tilde{E}[\log p(\alpha, y|\psi)]$$

as $\tilde{E}[\log p(\alpha|y, \psi)] = 0$ and $\log p(y|\psi)$ is independent of α.
Thus the score vector becomes

$$\frac{\partial \log L(y|\psi)}{\partial \psi}|_{\psi=\tilde{\psi}} = -\frac{1}{2}\frac{\partial}{\partial \psi}\sum_{t=1}^{n} \log|H_t| + \log|Q_{t-1}| + \text{tr}[(\hat{\epsilon}_t\hat{\epsilon}_t' + \text{Var}(\epsilon_t|y))H_t^{-1}] + \text{tr}[(\hat{\eta}_{t-1}\hat{\eta}_{t-1}' + \text{Var}(\eta_{t-1}|y))Q_{t-1}^{-1}]$$

where $\hat{\eta}_{t-1}, \hat{\epsilon}_t, \text{Var}(\epsilon_t|y), \text{Var}(\eta_t|y)$, are evaluated for $\psi = \tilde{\psi}$. The score vector for the diffuse case is also derived in this section.

7.2 EM Algorithm

EM algorithm crops in variety of places. In contexts where you see that the log likelihood function that is difficult to maximize, it makes sense to introduce additional latent variables and then make it tractable. In the context of state space models, EM algorithm for many state space models has a particularly neat form. The EM consists of E step which involves the evaluation of the conditional expectation of $\tilde{E}(\log p(\alpha, y|\psi)$ and the M step that involves maximizing the expectation with respect to the element of ψ. From the previous section the conditional expectation can be written down as
The log likelihood function

$$\log p(\alpha, y|\psi) = -\frac{1}{2}\sum_{t=1}^{n} \log|H_t| + \log|Q_{t-1}| + \epsilon_t' H_t^{-1}\epsilon_t + \eta_{t-1}' Q_{t-1}^{-1}\eta_{t-1} +$$

E step - Evaluating the loglikelihood function with respect to density $p(\alpha|y, \psi)$

$$\tilde{E}(\log p(\alpha, y|\psi)) = -\frac{1}{2}\sum_{t=1}^{n} \log|H_t| + \log|Q_{t-1}| + \text{tr}[(\hat{\epsilon}_t\hat{\epsilon}_t' + \text{Var}(\epsilon_t|y))H_t^{-1}] + \text{tr}[(\hat{\eta}_{t-1}\hat{\eta}_{t-1}' + \text{Var}(\eta_{t-1}|y))Q_{t-1}^{-1}]$$

M step - Maximize Expectation by taking the derivative

$$\frac{\partial}{\partial \psi}\tilde{E}(\log p(\alpha, y)|\psi) = -\frac{1}{2}\frac{\partial}{\partial \psi}\sum_{t=1}^{n} \log|H_t| + \log|Q_{t-1}| + \text{tr}[(\hat{\epsilon}_t\hat{\epsilon}_t' + \text{Var}(\epsilon_t|y))H_t^{-1}] + \text{tr}[(\hat{\eta}_{t-1}\hat{\eta}_{t-1}' + \text{Var}(\eta_{t-1}|y))Q_{t-1}^{-1}]$$

The values in the next iteration are

$$\begin{aligned} H_t^{new} &= n^{-1}[(\epsilon_t^{old}\epsilon_t^{old'} + \text{Var}(\epsilon_t^{old}|y)] \\ Q_t^{new} &= n^{-1}[(\eta_{t-1}^{old}\eta_{t-1}^{old'} + \text{Var}(\eta_{t-1}^{old}|y)] \end{aligned}$$

More specific information on the EM algorithm can be found in the paper, "An approach to timeseries smoothing and forecasting " by Shumway and Stoffer. The only irritant in state space model literature is that every one follows his/her own notation. As of now, I have not found a convenient way to quickly grasp various notations. In anycase, that's the way it is, so better get used to it.

Here is the R code that implements EM. I will run the algo on a known dataset so that the estimates can be compared. The dataset is Nile and the model used is *local level model*. EM algo will be used to estimate the observation variance and state evolution variance.

```
new           <- c(var(Nile),var(Nile))
old           <- numeric(2)
n             <- length(Nile)

while(sum(abs(new - old)) > 1e-7){
 old         <- new
 modelNile   <- SSModel(Nile~SSMtrend(1,Q=old[1]),H=old[2])
 out         <- KFS(modelNile,filtering='state',smoothing=c('state','disturbance'))
 W           <- diag(n)
 diag(W)     <- c(out$V_eta)
 new[1]      <- sum(diag((out$etahat)%*% t(out$etahat) + W))/n

 V           <- diag(n)
 diag(V)     <- out$V_eps
 new[2]      <- sum(diag((out$epshat)%*% t(out$epshat) + V))/n
}
#Parameters - Observation error variance and State error variance
params      <- data.frame( H= new[2], Q = new[1])
params

##       H    Q
## 1 15099 1469
```

To compare the output of EM algo with the output from using `fitSSM`

```
modelNile   <-SSModel(Nile~SSMtrend(1,Q=list(matrix(NA))),H=matrix(NA))
modelNile   <-fitSSM(inits=c(log(var(Nile)),log(var(Nile))),
                      model=modelNile,
                       method='BFGS',
                     control=list(REPORT=1,trace=0))$model
c(modelNile$H) #15098.65

## [1] 15099

c(modelNile$Q) #1469.163

## [1] 1469
```

As one can see the EM algorithm concurs with the MLE values.

Given the MLE estimate, it can be shown that under reasonable assumptions about the stability of the model over time, the distribution of $\hat{\psi}$ for large n is approximately

$$\hat{\psi} \sim N(\psi, \Omega), \quad \Omega = \left[\frac{\partial^2 \log L}{\partial\psi\partial\psi'}\right]^{-1}$$

The section introduces the antithetic sampling to reduce the bias amongst various estimates, conditional on ψ, the metaparameters. The authors touch upon briefly the model selection criterion (based on AIC and BIC) and diagnostics. The diagnostic tests involve standardized residuals and auxiliary residuals, the former are used to check for normality, serial correlation etc, while the latter are used to identify structural breaks.

8 Bayesian Analysis

8.1 Posterior analysis of state vector

When the vector ψ is not fixed and known, it is treated as a random vector with prior density $p(\psi)$. The section introduces importance sampling as a technique to estimate

$$\overline{x} = E[x(\alpha)|y]$$

The basic idea of importance sampling is to generate random samples from a distribution that is as close as possible to $p(\psi|y)$ and then take a weighted average of the samples to obtain posterior mean $\overline{x}$. The book gives a chapter length treatment to "importance sampling" in Part II of the book.

8.2 MCMC

This basic idea here is to start with some initial parameter vector ψ^0 and generate a state sample $\alpha^{(i)}$ from $p(\alpha|y, \psi^0)$. The second step involves sampling a new ψ from $p(\psi|y, \alpha^{(i)})$. This is repeated until convergence. The first step of sampling given metaparameters can be accomplished via Simulation smoothing algorithm from `KFAS` or `dlmBSample`. However the speed from the former package is much faster than the latter. The algorithm used in the former is from the paper titled, "A simple and efficient simulation smoother for state space time series analysis". After generating samples from the posterior distribution of metaparameters, one has to apply the usual methods like "burning-in", "thinning" etc to estimate the parameters. The second step in the above algorithm where one has to sample from $p(\psi|y, \alpha)$ depends partly on the model for ψ and is usually only possible up to proportionality. To work around this problem, "accept-reject" methods have been developed in the MCMC literature. A convenient way is to assume appropriate conjugate priors for hyper parameters so that the posterior distributions have a closed form.

9 Illustrations of the use of the linear Gaussian model

This chapter gives a set of examples that illustrate various aspects of State space model analysis. I will try to replicate the analysis using R.

9.1 Structural time series models

In this section, a level and seasonal model is used to fit the monthly numbers(logged) of drivers who were killed or seriously injured in road accidents in cars in Great Britain in a certain time period.

```
data.1             <- log(read.table("data/UKdriversKSI.txt",skip=1))
colnames(data.1)  <- "logKSI"
data.1             <- ts(data.1, start = c(1969),frequency=12)
```

```
par(mfrow=c(1,1))
plot(data.1,  col = "darkgrey", xlab="",ylab = "log KSI",pch=3,cex=0.5,
     cex.lab=0.8,cex.axis=0.7)
abline(v=1969:2003, lty= "dotted",col="sienna")
```

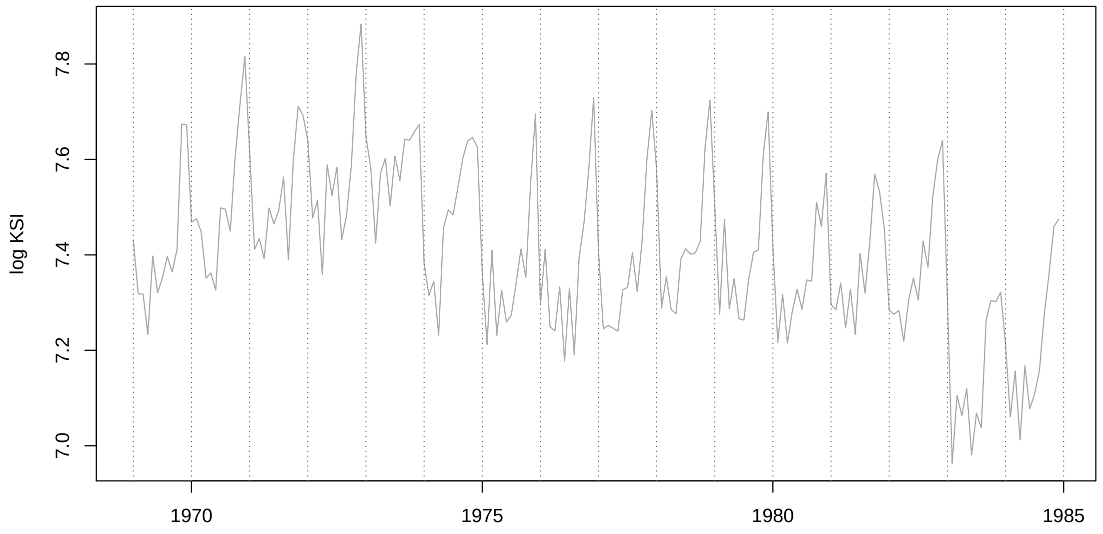


Figure 9.1: Monthly numbers(logged) of drivers who were killed or seriously injured in road accidents in cars in Great Britain in a certain time period.

Using dlm package

```
fn                    <- function(params){
  mod                 <- dlmModPoly(order = 1 ) + dlmModSeas(frequency =12)
  V(mod)             <- exp(params[1])
  diag(W(mod))[1:12] <- exp(c(params[2],rep(params[3],11)))
  return(mod)
}
fit                   <- dlmMLE(data.1, c(var(data.1),rep(0,12)),fn)
mod                   <- fn(fit$par)
(obs.error.var        <- V(mod))

##          [,1]
## [1,] 0.003371

(seas.var             <- (diag(W(mod))[2]))

## [1] 2.378e-06

(level.var            <- (diag(W(mod))[1]))

## [1] 0.0009313

res                 <- data.frame(V = obs.error.var,
                        W = level.var,
                        Ws = seas.var)
res

##          V         W        Ws
## 1 0.003371 0.0009313 2.378e-06
```

Using KFAS package

```
model <-SSModel(log(drivers)~SSMtrend(1,Q=list(NA))+
        SSMseasonal(period=12,sea.type='trigonometric',Q=NA),data=Seatbelts,H=NA)

ownupdatefn      <-  function(pars,model,...){
model$H[]          <- exp(pars[1])
diag(model$Q[,,1])<- exp(c(pars[2],rep(pars[3],11)))
model
}


fit                 <-fitSSM(inits=
                              log(c(var(log(Seatbelts[,'drivers'])),0.001,0.0001)),
                              model=model,
                              updatefn=ownupdatefn,method='BFGS')


res                 <- data.frame(V = fit$model$H[,,1],
                        W = diag(fit$model$Q[,,1])[1],
                        Ws = diag(fit$model$Q[,,1])[2])
res
```

```
##          V        W        Ws
## 1 0.003416 0.000936 5.004e-07
```

```
out        <- KFS(fit$model,filtering=c('state'), smoothing=c('state','disturbance','mean'))
level.sm   <- out$alphahat[,1]
seas.sm    <- rowSums(out$alphahat[,2:12])
epshat     <- out$epshat
```

```
par(mfrow=c(3,1))
temp        <- ts( cbind(data.1,level.sm) ,start = 1969,frequency =12)
plot.ts(temp,plot.type="single", ylim=c(7,8),
        xlab="",ylab = "log KSI",
        col = c("darkgrey","blue"),lwd=c(1,2),cex.lab=1.3,main = "i")

temp       <- ts( seas.sm,start = 1969,frequency = 12)
plot(temp,xlab="",ylab = "log KSI",  col = "blue",lwd=1,main="ii")
abline(h=0,col="grey")

temp       <- ts( epshat ,start = 1969,frequency  =12)
plot(temp,xlab="",ylab = "log KSI",  col = "blue",lwd=1,main="iii")
abline(h=0,col="grey")
```

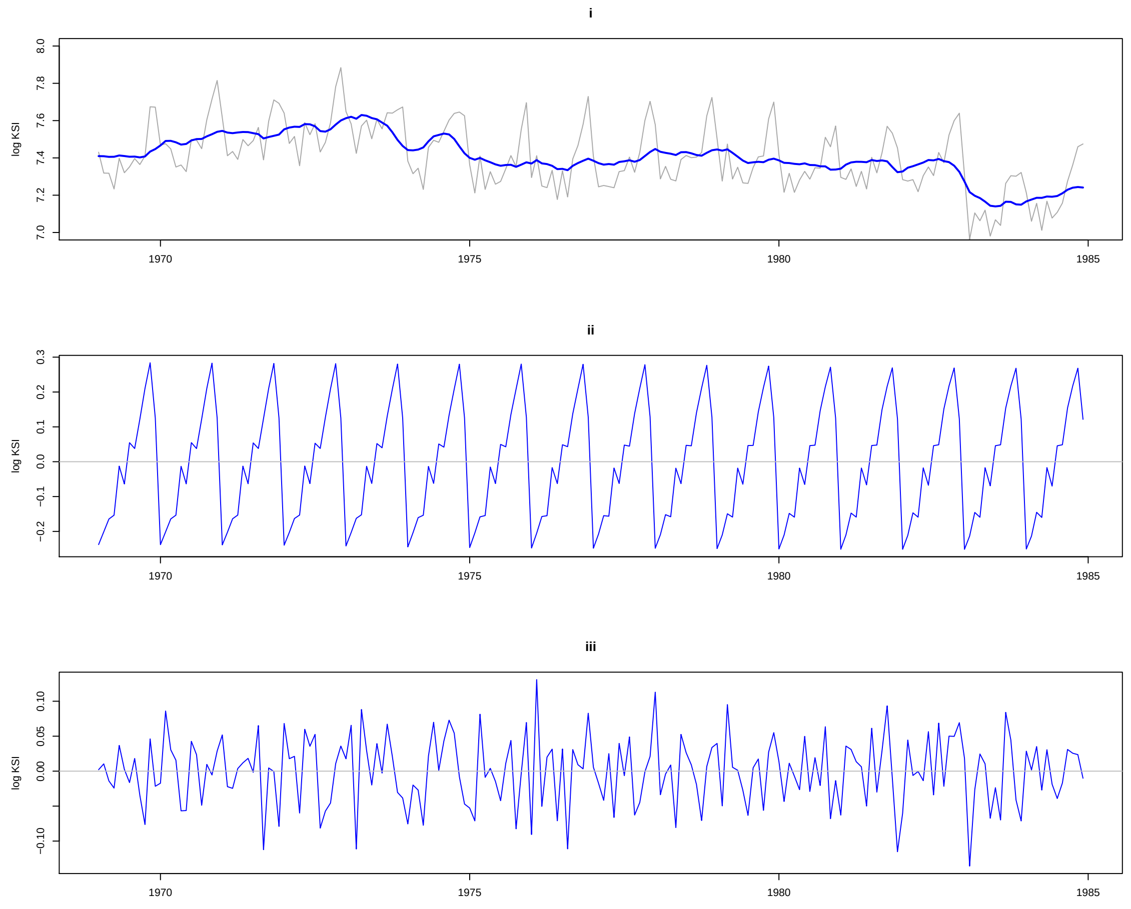


Figure 9.2: Estimated components : (i) level, (ii) seasonal, (iii) irregular

```
par(mfrow=c(1,1))
temp        <- ts( cbind(data.1,out$a[,1],out$alphahat[,1])[c(-1:-12),] ,
                   start = c(1970,2),frequency =12)
plot.ts(temp,plot.type="single", ylim=c(7,8),
        xlab="",ylab = "log KSI",
        col = c("darkgrey","blue","green"),lwd=c(1,2,2))
leg <-c("actual data","filtered level","smoothed level")
legend("topright",leg,col=c("grey","blue","red"),lwd=c(1,2,2),cex=0.7,bty="n")
```

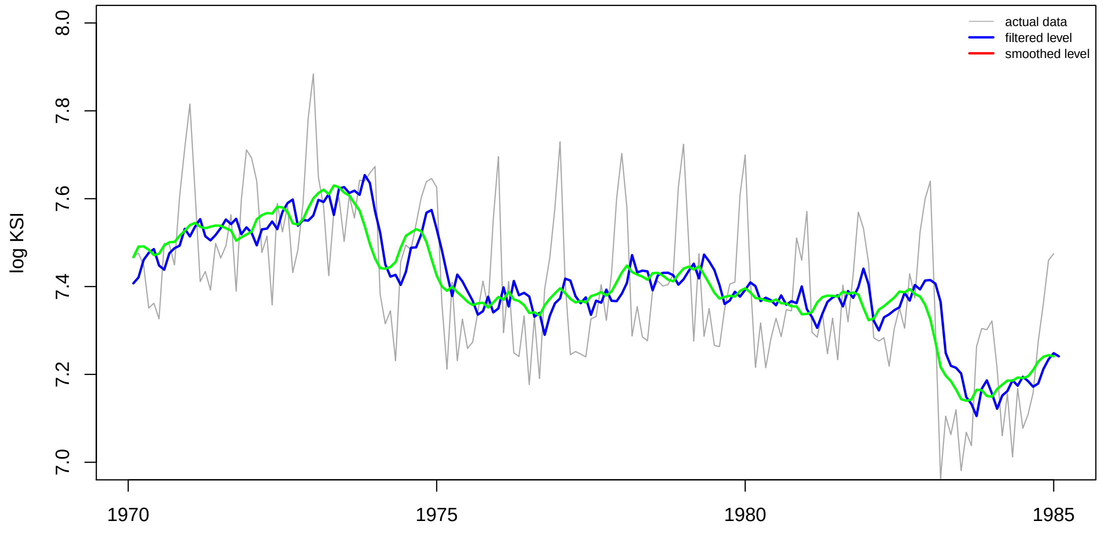


Figure 9.3: Filtered and Smoothed level estimate

```
par(mfrow=c(3,1))
one.step.ahead <- residuals(out, "recursive")
temp       <- ts( one.step.ahead,start = 1969,frequency = 12)
plot(temp,xlab="",ylab = "log KSI",  col = "blue",lwd=1,main = "i")
abline(h=0,col="grey")

temp       <- residuals(out, type="response")
temp       <- ts( temp,start = 1969,frequency = 12)
plot(temp,xlab="",ylab = "log KSI",  col = "blue",lwd=1, main = "ii")
abline(h=0,col="grey")

temp       <- residuals(out, type="state")[,1]
temp       <- ts( temp,start = 1969,frequency = 12)
plot(temp,xlab="",ylab = "log KSI",  col = "blue",lwd=1, main = "iii")
abline(h=0,col="grey")
```

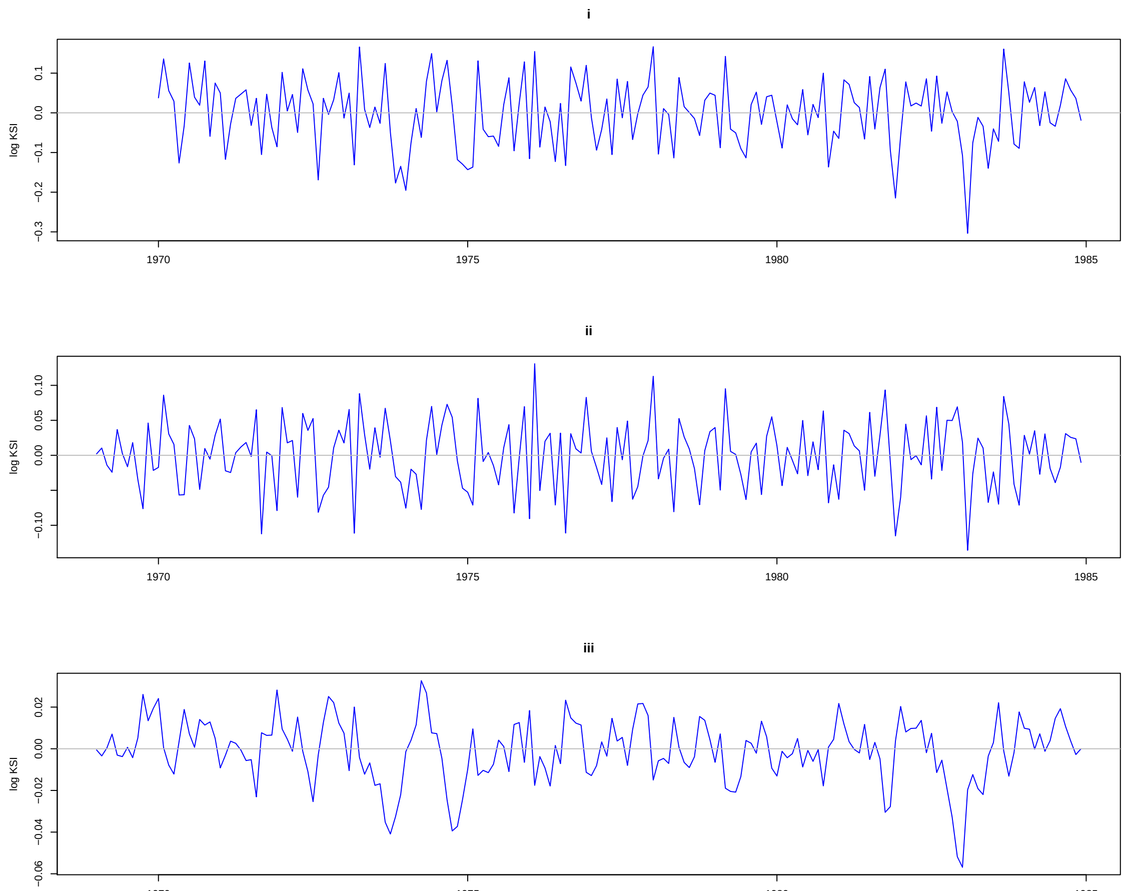


Figure 9.4: (i)One step ahead residuals,(ii) auxiliary irregular residuals, (iii) auxiliar level residuals

The residuals show that there is a large negative value for Feb 1983. This suggests a need to incorporate an intervention variable to measure the level shift. The analysis is redone using the intervention variable which equals one from Feb 1983 and is zero prior to that.

```
n           <- dim(Seatbelts)[1]
model        <- SSModel(log(drivers)~SSMtrend(1,Q=list(NA))+
                     SSMseasonal(period=12,sea.type='trigonometric',Q=NA)+
                     log(PetrolPrice)+law,data=Seatbelts,H=NA)

ownupdatefn  <-function(pars,model,...){
model$H[]    <-exp(pars[1])
diag(model$Q[,,1]) <-exp(c(pars[2],rep(pars[3],11)))
model
}

fit          <- fitSSM(inits=log(c(var(log(Seatbelts[,'drivers'])),0.001,0.0001)),
                model=model,updatefn=ownupdatefn,method='BFGS')

out          <- KFS(fit$model,smoothing=c('state','mean','disturbance'))
(out$alphahat[n,1])

## log(PetrolPrice)
##          -0.2914

(out$alphahat[n,2])

##     law
## -0.2377
```

```
par(mfrow=c(3,1))
level.sm    <- out$alphahat[,1]*log(as.data.frame(Seatbelts)$PetrolPrice) +
  out$alphahat[,2]*(as.data.frame(Seatbelts)$law) + out$alphahat[,3]
seas.sm     <- rowSums(out$alphahat[,2:14])
epshat      <- out$epshat

temp        <- ts( cbind(data.1,level.sm) ,start = 1969,frequency =12)
plot.ts(temp,plot.type="single",
        xlab="",ylab = "log KSI",
        col = c("darkgrey","blue"),lwd=c(1,2),cex.lab=1.3,main = "i")

temp       <- ts( seas.sm,start = 1969,frequency = 12)
plot(temp,xlab="",ylab = "log KSI",  col = "blue",lwd=1,main="ii")
temp       <- ts( epshat ,start = 1969,frequency  =12)
plot(temp,xlab="",ylab = "log KSI",  col = "blue",lwd=1,main="iii")
abline(h=0,col="grey")
```

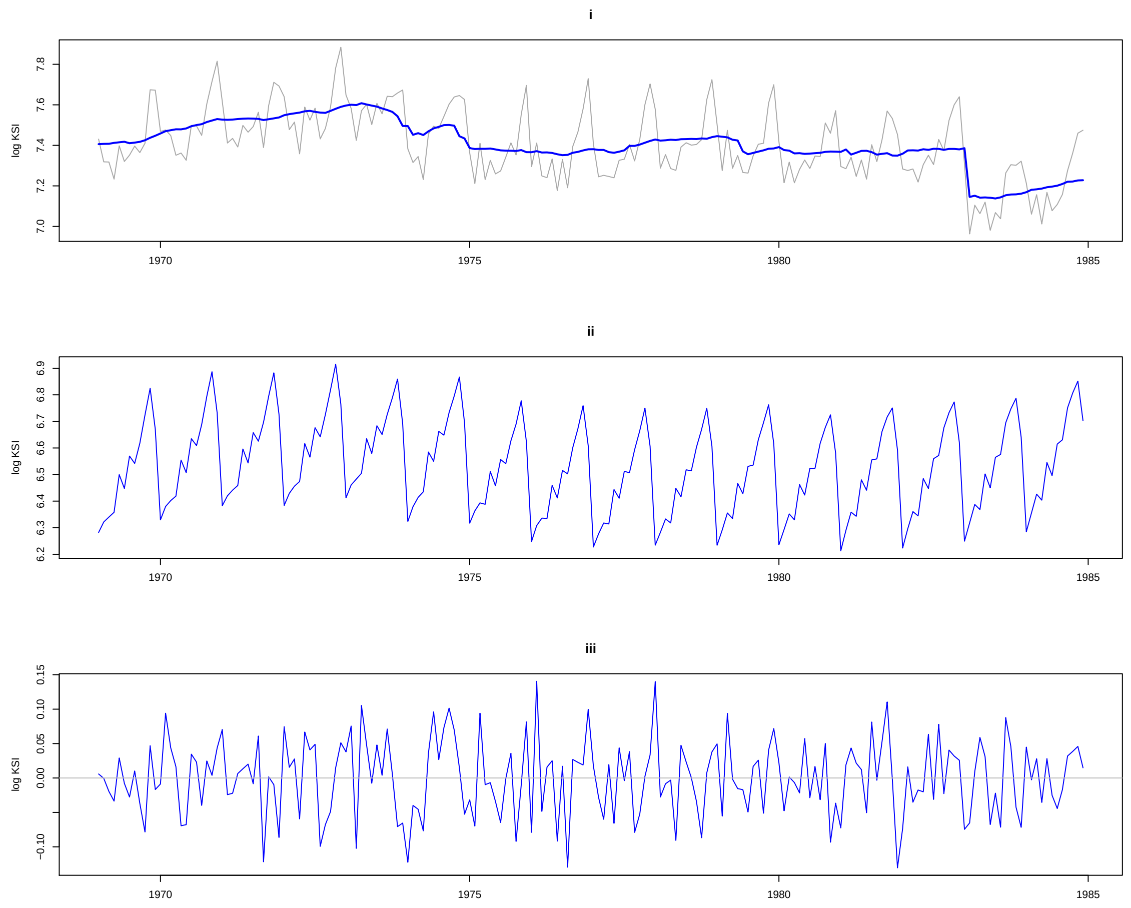
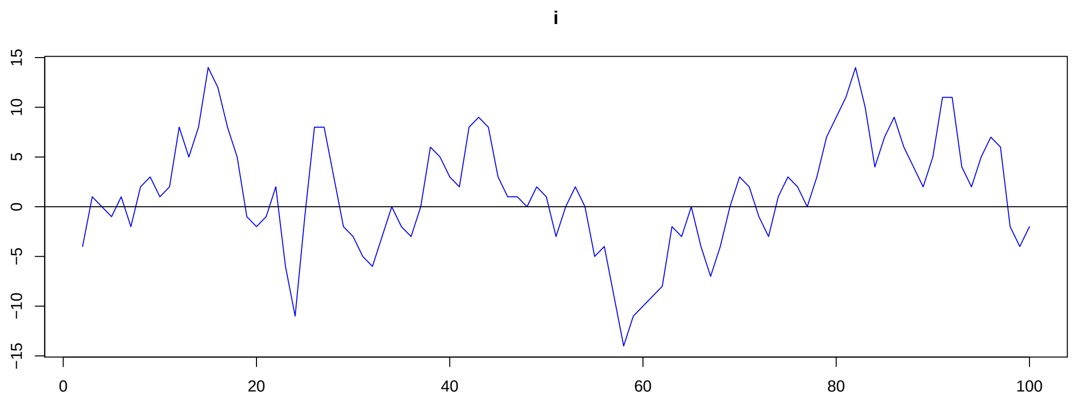


Figure 9.5: Estimated components for model with intervention and regression effects : (i) level, (ii) seasonal, (iii) irregular

9.2 Box-Jenkins analysis

```
data(internet)
internet.miss <- internet
internet.miss[c(6,16,26,36,46,56,66,72,73,74,75,76,86,90)] <- NA
d1            <- diff(internet)
d2            <- diff(internet.miss)
```

```
par(mfrow=c(2,1))
plot(diff(internet),xlab="",ylab = "",  col = "blue",lwd=1,main="i")
abline(h=0)
plot(diff(internet.miss),xlab="",ylab = "",  col = "blue",lwd=1,main="ii")
abline(h=0)
```



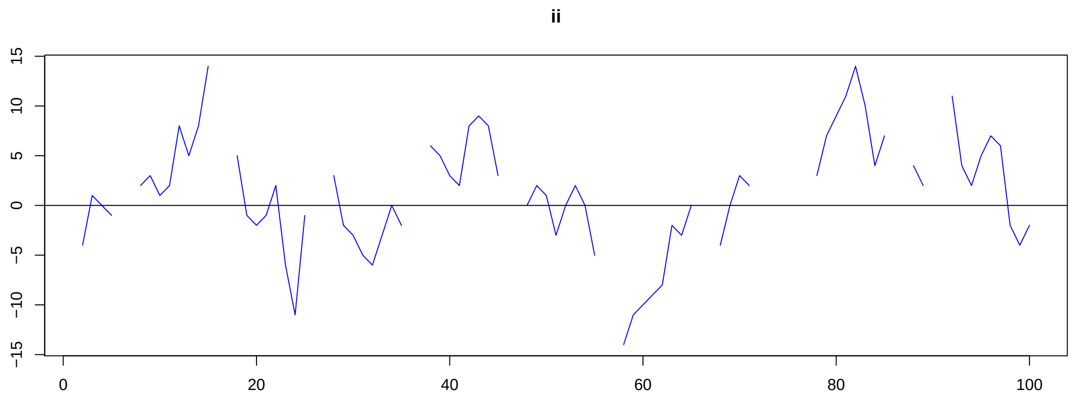


Figure 9.6: (i) First difference of number of users logged on to Internet server each minute (ii) The same data with 14 observations deleted

```
ps      <- 0:5
qs      <- 0:5
results <- matrix(0, nrow = length(ps), ncol= length(qs))
for( p in ps){
  for(q in qs ){
    if(p==0 & q==0) next
    if(q > 0 ){
    ar        <- rep(-0.05,p)
    ma        <- rep(0.001,q)
    model     <- SSModel(d1 ~ SSMarima(ar = ar, ma = ma,
                                  d = 0, Q = 1e-7),H=1e-7)

    fn        <- function(pars,model,...){
      model$T[2:(2+p-1),2,1]     <- (pars[1:p])
      model$R[3:(3+q-1),,1]      <- pars[(p+1):(p+q)]
      model$Q[,,1]               <- exp(pars[p+q+1])
      model
      }

    model     <- fitSSM(inits=c(ar,ma,1e-7),model=model,updatefn = fn,
                  method='BFGS', control=list(REPORT=1,trace=0))$model
    }else{

    ar       <- rep(-0.05,p)
    model    <- SSModel(d1 ~ SSMarima(ar = ar,
                                d = 0, Q = 1e-7),H=1e-7)

    fn       <- function(pars,model,...){
      model$T[2:(2+p-1),2,1]    <- (pars[1:p])
      model$Q[,,1]              <- exp(pars[p+q+1])
      model
      }

    model     <- fitSSM(inits=c(ar,1e-7),model=model,updatefn = fn,
                  method='BFGS', control=list(REPORT=1,trace=0))$model

    }

    aic.model <- -2*logLik(model) +2*(p+q)
    results[p+1,q+1] <- aic.model
  }
}

results <- results/length(d1)
rownames(results) <- paste("AR",0:5,sep="")
colnames(results) <- paste("MA",0:5,sep="")
results[results>100] <- 0
```

	MA0	MA1	MA2	MA3	MA4	MA5
AR0	0.0000	5.2210	5.2115	5.1785	5.1980	5.2182
AR1	5.3331	5.2155	5.2072	5.1813	9.8745	3.5363
AR2	5.2947	43.6969	5.3608	42.0075	4.4523	4.7491
AR3	5.1541	5.1741	0.0000	5.9370	6.8736	10.2940
AR4	5.1741	5.1943	0.0000	0.0000	5.1484	9.4143
AR5	5.1942	5.2144	5.2231	5.2043	5.0350	5.0954

Table 1: AIC for different ARMA Models

```
ps      <- 0:5
qs      <- 0:5
results2 <- matrix(0, nrow = length(ps), ncol= length(qs))
for( p in ps){
  for(q in qs ){
    if(p==0 & q==0) next
    if(q > 0 ){
    ar        <- rep(-0.05,p)
    ma        <- rep(0.001,q)
    model     <- SSModel(d1 ~ SSMarima(ar = ar, ma = ma,
                                 d = 0, Q = 1e-7),H=1e-7)

    fn        <- function(pars,model,...){
      model$T[2:(2+p-1),2,1]    <- (pars[1:p])
      model$R[3:(3+q-1),,1]     <- pars[(p+1):(p+q)]
      model$Q[,,1]              <- exp(pars[p+q+1])
      model
      }

    model     <- fitSSM(inits=c(ar,ma,1e-7),model=model,updatefn = fn,
                  method='BFGS', control=list(REPORT=1,trace=0))$model
    }else{

    ar        <- rep(-0.05,p)
    model     <- SSModel(d2 ~ SSMarima(ar = ar,
                                 d = 0, Q = 1e-7),H=1e-7)

    fn        <- function(pars,model,...){
      model$T[2:(2+p-1),2,1]    <- (pars[1:p])
      model$Q[,,1]              <- exp(pars[p+q+1])
      model
      }

    model     <- fitSSM(inits=c(ar,1e-7),model=model,updatefn = fn,
                  method='BFGS', control=list(REPORT=1,trace=0))$model

    }

    aic.model <- -2*logLik(model) +2*(p+q)
    results2[p+1,q+1] <- aic.model
  }
}
```

```
results2 <- results2/length(d1)
rownames(results2) <- paste("AR",0:5,sep="")
colnames(results2) <- paste("MA",0:5,sep="")
results2[results2>100] <- 0
```

	MA0	MA1	MA2	MA3	MA4	MA5
AR0	0.0000	5.2210	5.2115	5.1785	5.1980	5.2182
AR1	4.1606	5.2155	5.2072	5.1813	9.8745	3.5363
AR2	13.7568	43.6969	5.3608	42.0075	4.4523	4.7491
AR3	14.4141	5.1741	0.0000	5.9370	6.8736	10.2940
AR4	14.9953	5.1943	0.0000	0.0000	5.1484	9.4143
AR5	10.6752	5.2144	5.2231	5.2043	5.0350	5.0954

Table 2: AIC for different ARMA Models with missing observations

Choosing ARMA(1,1) model,

```
    d1.miss  <- c(d1,rep(NA,5))

    p        <- 1
    q        <- 1
    ar       <- rep(-0.05,p)
    ma       <- rep(0.001,q)
    model    <- SSModel(d1.miss ~ SSMarima(ar = ar, ma = ma,
                                 d = 0, Q = 1e-7),H=1e-7)

    fn       <- function(pars,model,...){
      model$T[2:(2+p-1),2,1]    <- (pars[1:p])
      model$R[3:(3+q-1),,1]     <- pars[(p+1):(p+q)]
      model$Q[,,1]              <- exp(pars[p+q+1])
      model
      }

    model     <- fitSSM(inits=c(ar,ma,1e-7),model=model,updatefn = fn,
                 method='BFGS', control=list(REPORT=1,trace=0))$model

## Warning:  Not a valid SSModel object.  Returning -Inf

  out          <- KFS(model,filtering=c('state'), smoothing=c('state','disturbance','mean'))
  d.smoothed   <-rowSums(as.matrix(out$alphahat))
```

```
par(mfrow=c(1,1))
plot.ts(d.smoothed, col="blue",xlab="",ylab="",main="",lwd =2)
points(d1.miss, pch = 19, col ="red",cex=0.7)
```

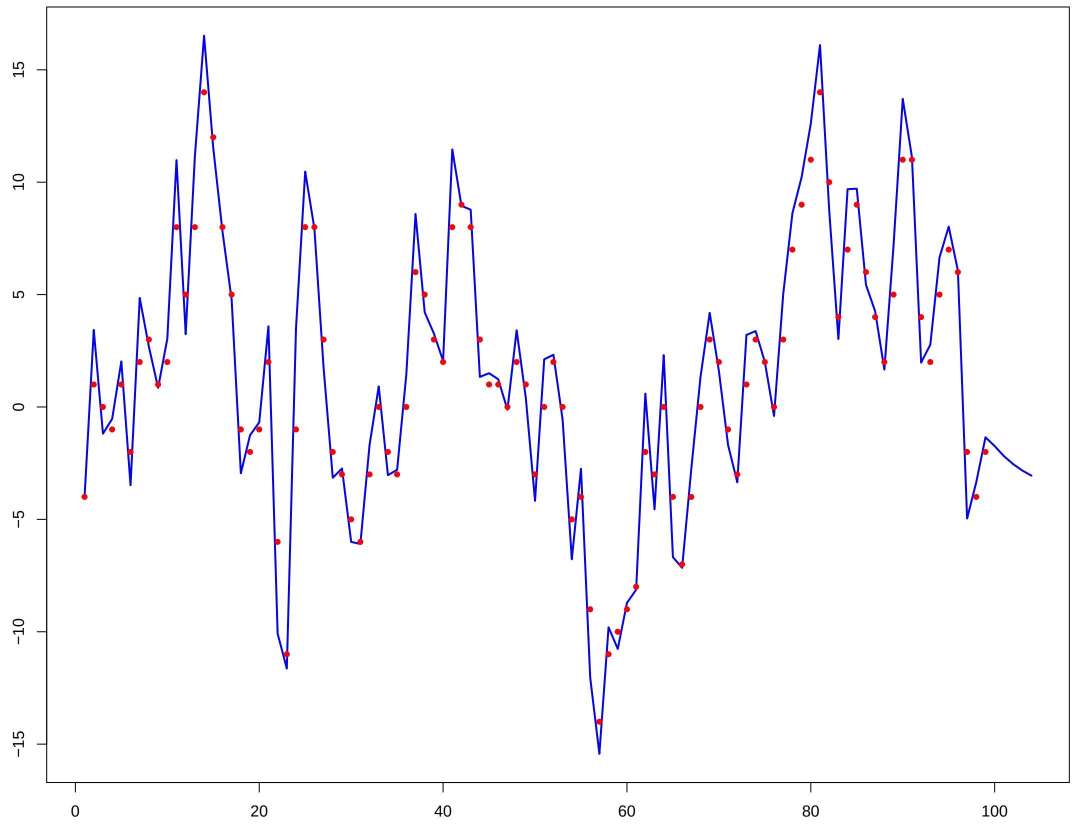


Figure 9.7: Internet series with in-sample one step forecasts and out of sample corecasts(points represent the actual data)