Tutorial 10: Kalman Filtering

We first introduce two packages in R for state space models; **dlm** (where the initials stand for *dynamic linear model*) and **MARSS** (the initials stand for *Multivariate Auto Regressive State Space*). We'll introduce **dlm** through the worked example of the 'Nile' data and **MARSS** through the 'Salmon' data.

dlm and the 'Nile' data

The `Nile` data is the historical data set consisting of readings of the annual flow volume of the Nile River at Aswan from 1871 to 1970. It exhibits *long range dependence* and was studied by Hurst, who introduced the parameter, now known as the *Hurst parameter* as a measure of this.

We now show to use **dlm** to fit a 'state space' model to the Nile data and use the model for prediction.

```
> install.packages("dlm")
> library("dlm")
```

Firstly, it is important to know what *notation* is being used; what does **dlm** mean by F, G, V, W, m, C. The notation for the **dlm** package is as follows: the equation is

$$\begin{cases} y_t = F_t\theta_t + v_t & v_t \sim N(0, V_t) \\ \theta_t = G_t\theta_{t-1} + w_t & w_t \sim N(0, W_t) \\ \theta_0 \sim N(m_0, C_0) \end{cases}$$

F_t is $m \times p$, V_t is $m \times m$, G_t is $p \times p$ and W_t is $p \times p$. In **dlm** these are referred to as `FF, V, GG, W` respectively.

There are several kinds of dynamic linear models that can be fitted, falling into categories *polynomial*, *Seasonal*, *ARMA*. Try to get information about these models using the help commands:

```
> ?dlmModPoly
> ?dlmModSeas
> ?dlmModARMA
```

We want to fit a *univariate* model here; the observed time series $\{Y_t\}$ is explained by a state $\{X_t\}$ plus noise. To specify the dimension of the model `mod`, type:

```
> mod <- dlmModPoly(1)
```

The *dimension* is the length of the *state* vector X_t. For the Nile data, we're taking a single state space variable.

Now we'll look at the 'Nile' data

```
> ?Nile
> nileFilt<-dlmFilter(Nile,mod) # Kalman filter
```

The filtering routine returns:

1. the series of filtering means $m_t = \mathbb{E}[X_t|\mathcal{F}_t^{(Y)}]$
2. the series of filtering variances $C_t = \mathbf{V}(X_t|\mathcal{F}_t^{(Y)})$
3. the series of one-step forecasts for the state $a_t = \mathbb{E}[X_t|\mathcal{F}_{t-1}^{(Y)}]$
4. the series of one-step forecasts for the variances for the state
$$R_t = \mathbf{V}(X_t|\mathcal{F}_{t-1}^{(Y)})$$
5. the series of one-step forecasts for the observation
$$f_t = \mathbb{E}[Y_t|\mathcal{F}_{t-1}^{(Y)}]$$

Variances are returned in terms of their singular value decomposition: $C_t = U_t D_t^2 U_t^t$ where D_t is diagonal and U^t is orthogonal. For one dimensional states, $C_t = D_t^2$. In general, the sequence of variance matrices can be recovered using:

```
> Ct <- with(nileFilt, dlmSvd2var(U.C, D.C))
> Rt <- with(nileFilt, dlmSvd2var(U.R, D.R))
```

Plotting the **ggplot2** together with **ggfortify** is very useful for plotting the output of dynamic linear model routines.

```
library(ggplot2)
library(ggfortify)
autoplot(nileFilt)
```

The black plot shows the observations, connected by line, and the red plot shows the filtered values.

Smoothing We now show how to do smoothing using **dlm**.

```
> nileSmooth <- dlmSmooth(Nile, mod) #Kalman smoother
> nileSmooth2 <- dlmSmooth(nileFilt) #the same
```

The smoothing routine returns

1. the series of smoothing means $m_t = \mathbb{E}[X_t|\mathcal{F}_T^{(Y)}]$
2. the series of smoothing variances $s_t = \mathbf{V}(X_t|\mathcal{F}_T^{(Y)})$.

To show the data and the smoothing means on the same plot, try:

```
nile2 = as.data.frame(Nile)
nile2$smooth = nileSmooth$s[-101]
nile2ts = as.ts(nile2)
autoplot(nile2ts,facets=FALSE)
```

I removed the final year (1971) for the smoother, because the series have to be the same length. I put it as a data frame so that the new column could be added. Unfortunately `autoplot` does not support `data.frame`, but with **ggplot2** and **ggfortify** activated (**ggfortify** is essential here) it does support `ts`.

Smoothing variances are returned in terms of the decomposition. To recover covariance matrices:

```
> St <- with(nileSmooth, dlmSvd2var(U.S, D.S)) #Smoothing variance
> level <- dropFirst(nileFilt$m) #filtered level
> sdC <- abs(dropFirst(nileFilt$D.C)) #filtering standard deviations
> ci <- drop(sdC%o% qnorm(c(0.05,0.95))) + as.vector(level) #90 percent interval
> ci2 <- ts(ci, start = start(Nile))
```

Forecasting The function `dlmForecast` returns, for $k = 1, 2, \ldots$

1. the series of k step ahead forecasts for the *states* $a_k = \mathbb{E}[X_{t+k}|\mathcal{F}_t^{(Y)}]$
2. the series of forecast variances for the states (covariance matrices if there is a vector of states) $R_k = \mathbf{V}(X_{t+k}|\mathcal{F}_t^{(Y)})$
3. the series of k step ahead forecasts of the observations
$$f_k = \mathbb{E}[Y_{t+k}|\mathcal{F}_t^{(Y)}]$$
4. the series of forecast variances for hte observations
$$Q_{t,k} = \mathbf{V}(Y_{t+k}|\mathcal{F}_t^{(Y)})$$

   Forecast variances are returned as matrices.

```
> nileForecast <- dlmForecast(nileFilt, n=20)
> nileForecast <- dlmForecast(nileFilt, n=20, sampleNew = 50)
> str(nileForecast,1)
> nodataForecast <-dlmForecast(mod, n=100, sampleNew = 100)
```

Model checking The following routines are standard for checking whether the residuals are normal i.i.d.

```
> nileRes <- residuals(nileFilt, sd = FALSE) # standardised innovations
> qqnorm(nileRes) #graphical normality assessment
> qqline(nileRes)
> tsdiag(nileFilt) #standard diagnostics
```

The MARSS Package

The **MARSS** (Multivariate Analysis and State Space Models) package is very useful. The *state* process takes the form

$$x_t = Bx_{t-1} + u + w_t \qquad \{w_t\} \sim WNN(0, Q)$$

where x_t is an m-vector, Q is an $m \times m$ covariance matrix. The initial state vector is specified at $t = 0$ (or $t = 1$) as:

$$x_0 \sim N(\pi, \Lambda) \qquad \text{or} \qquad x_1 \sim N(\pi, \Lambda).$$

The multivariate *observation* component is

$$y_t = Zx_t + a + v_t \qquad \{v_t\} \sim WNN(0, R)$$

where y_t is an n-vector and R is a $n \times n$ covariance matrix.

Example: Salmon Survival

The example is taken from Sheuerell and Williams (2005), using a DLM (dynamic linear model) to examine the relationship between marine survival of Chinook salmon and an index of *ocean upwelling strength* along the west coast of the USA. *Upwelling* is a process that brings cool, nutrient-rich waters from the deep ocean to shallower coastal areas. Sheuerell and Williams hypothesised that stronger upwelling in April should create better growing conditions for the plankton, so that juvenile salmon ('smolts') entering the ocean in May and June should find better feeding opportunities.

Model Specification Suppose (for example) that we have a model

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}_t = \begin{pmatrix} b & 0 \\ 0 & b \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}_{t-1} + \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}_t$$

$$\begin{pmatrix} w_1 \\ w_2 \end{pmatrix}_t \sim N\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} q_{11} & q_{12} \\ q_{12} & q_{22} \end{pmatrix} \right)$$

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}_0 \sim N\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix} \right)$$

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}_t = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}_t + \begin{pmatrix} a_1 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}$$

$$\begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}_t \sim N\left(\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} r_1 & 0 & 0 \\ 0 & r_2 & 0 \\ 0 & 0 & r_3 \end{pmatrix} \right)$$

We first have to *specify the model*. To incorporate all the parameters,

```
> B1 =  matrix(list("b", 0, 0, "b"), 2, 2)
> U1 = matrix(0, 2, 1)
> Q1 = matrix(c("q11", "q12", "q12", "q22"), 2, 2)
> Z1 = matrix(c(1, 0, 1, 0, 1, 0), 3, 2)
> A1 =  matrix(list("a1", 0, 0), 3, 1)
> R1 = matrix(list("r1", 0, 0, 0, "r2", 0,
+ 0, 0, "r2"), 3, 3)
> pi1 = matrix(0, 2, 1)
> V1 = diag(1, 2)
> model.list = list(B = B1, U = U1, Q = Q1, Z = Z1,
+ A = A1, R = R1, x0 = pi1, V0 = V1)
```

MARSS has some shortcuts for some of the matrix specification (look it up).

10.1.6 Example: Salmon Survival Data

We begin by loading the data set (which is in the MARSS package). The data set has 3 columns:

1. the year the salmon smolts migrated to the ocean (year),
2. logit-transformed survival (logit.s), and
3. the coastal upwelling index, abbreviated CUI. For April, this is named `CUI.apr`. There are 42 years of data (1964–2005).

```
# load the data
data(SalmonSurvCUI, package = "MARSS")
# get time indices and call it years
years <- SalmonSurvCUI[, 1]
# count the number of years of data
TT <- length(years)
# get the response variable, which is logit(survival)
dat <- matrix(SalmonSurvCUI[, 2], nrow = 1)
```

Standardising the covariates to have zero mean and unit variance can be useful in model fitting and interpretation. In this case, the variance of `CUI.apr` is orders of magnitude larger than `survival`.

```
# get the predictor variable; we're using CUI (Coastal Upwelling
Index CUI)
CUI <- SalmonSurvCUI[, 3]
## z-score the CUI (i.e. centred and standardised)
CUI.z <- matrix((CUI - mean(CUI))/sqrt(var(CUI)), nrow = 1)
```

```
# number of regr params (slope + intercept)
m <- dim(CUI.z)[1] + 1
#note: we only have two parameters; CUI.z is a single column.
```

Now set up the matrices and vectors for MARSS. Reminder: the syntax here is

$$\begin{cases} x_t = Bx_{t-1} + u_t + w_t & w_t \sim N(0, Q_t) \\ y_t = Z_t x_t + a_t + v_t & v_t \sim N(0, R_t) \end{cases}$$

```
# for the process (or state space) equation, we need:
B <- diag(m)  ## 2x2; Identity
U <- matrix(0, nrow = m, ncol = 1)  ## 2x1; both elements = 0
Q <- matrix(list(0), m, m)  ## 2x2; all 0 for now
diag(Q) <- c("q.alpha", "q.beta")  ## 2x2; diag = (q1,q2)
```

Now let us define the correct form for the observation model.

```
# for observation eqn
Z <- array(NA, c(1, m, TT))  ## NxMxT; empty for now
Z[1, 1, ] <- rep(1, TT)  ## Nx1; 1's for intercept
Z[1, 2, ] <- CUI.z  ## Nx1; predictor variable
A <- matrix(0)  ## 1x1; scalar = 0
R <- matrix("r")  ## 1x1; scalar = r
```

Lastly, we need to define our lists of initial starting values and model matrices/vectors.

```
# only need starting values for regr parameters
inits.list <- list(x0 = matrix(c(0, 0), nrow = m))
# list of model matrices & vectors
mod.list <- list(B = B, U = U, Q = Q, Z = Z, A = A, R = R)
```

And now we can fit our *dynamic linear model* (DLM) with **MARSS**.

```
# fit univariate DLM
dlm1 <- MARSS(dat, inits = inits.list, model = mod.list)
```

The *Kalman filter function* in MARSS is `MARSSkfss()`. We'll use the notation

$$\begin{cases} y_t = GX_t + w_t & \{w_t\} \sim IIDN(0, R) \\ X_t = FX_{t-1} + V_t & \{V_t\} \sim IIDN(0, Q) \\ \{w_t\} \perp \{V_t\} \end{cases}$$

and we'll consider $\{y_t\}$ a univariate process. We take $X_0 \sim N(\pi_0, \Lambda_0)$, a multivariate normal random vector. Since this involves *linear* transformations of normals, we have

$$X_{t-1}|\{Y_1,\ldots,Y_{t-1}\} \sim N(\pi_{t-1},\Lambda_{t-1})$$

and the one-step-ahead predictive distribution for X_t given $\{y_1,\ldots,y_{t-1}\}$ is:

$$\begin{cases} X_t|\{y_1,\ldots,y_{t-1}\} \sim N(\eta_t,\Omega_t) \\ \eta_t = F\eta_{t-1} \\ \Omega_t = F\Omega_{t-1}F' - \Theta_t\Delta_t^{-1}\Theta_t' + Q \\ \Delta_t = G\Omega_t G' + R \\ \Theta_t = F\Omega_t G' + S \end{cases}$$

so that the one-step-ahead predictive distribution for the observation y_t given $\{y_1,\ldots,y_{t-1}\}$ is:

$$\begin{cases} y_t|\{y_1,\ldots,y_{t-1}\} \sim N(\zeta_t,\Psi_t) \\ \zeta_t = G\eta_t \\ \Psi_t = G\Omega_t G' + R \end{cases}$$

The one-step-ahead forecasts at time t, which we call η_t, are termed $\hat{x}_t^{t-1}$ in MARSS and are stored in `xtt1` in the list produced by `MARSSkfss`. To get the forecast, simply write:

```
# get list of Kalman filter output
kf.out <- MARSSkfss(dlm1)
## forecasts of regr parameters; 2xT matrix
eta <- kf.out$xtt1
## ts of E(forecasts)
fore.mean <- vector()
for (t in 1:TT) {
    fore.mean[t] <- Z[, , t] %*% eta[, t, drop = FALSE]
}
```

Note that, in MARSS, Z is what we call G. The forecast variance is obtained as follows:

```
# variance of regr parameters; 1x2xT array
Phi <- kf.out$Vtt1
## obs variance; 1x1 matrix
R.est <- coef(dlm1, type = "matrix")$R
## ts of Var(forecasts)
fore.var <- vector()
for (t in 1:TT) {
    tZ <- matrix(Z[, , t], m, 1)  ## transpose of Z
    fore.var[t] <- Z[, , t] %*% Phi[, , t] %*% tZ + R.est
}
```

For forecast diagnostics: we can find the residuals as follows:

```
# forecast errors
innov <- kf.out$Innov
```

and we can analyse the residuals to see if they look like i.i.d. $N(0, \sigma^2)$.

Exercises

1. Consider the 'Nile' data. Work through the preceding and then:

   (a) Plot the data, the filtered level and the lower and upper 90% confidence bounds.

   (b) Plot the data, the smoothed level and the 90% probability interval for the smoothed data.

   (c) Plot the forecasts, with lower and upper confidence bounds.

   (d) Try Holt Winters filtering on the Nile data and compare the results.

2. (a) The file `m-ppiaco4709.txt` contains year, month, day and U.S. producer price index (PPI) from January 1947 to November 2009. The index is for all commodities and is not seasonally adjusted. Let

   $$z_t = \ln(Z_t) - \ln(Z_{t-1})$$

   where Z_t is the observed monthly PPI. It turns out that an AR(3) model is adequate for z_t if the minor seasonal dependence is ignored. Let y_t be the sample mean-corrected series of z_t.

   i. Fit an AR(3) model to y_t. Write down the fitted model and check that it is suitable. (hint: look at pacf).

   ii. Suppose that y_t has independent measurement errors so that $y_t = x_t + \epsilon_t$, where x_t is a zero mean AR(3) process. Use a state space model to estimate the parameters, including the variances of the innovations for the states and for ϵ_t. Write down the fitted model and obtain a time plot of the smoothed estimate of x_t. Also, show the time plot of filtered response residuals of the fitted state-space model.

   The command `dlmModARMA` may useful.

3. **Simulated Stock Market Data** Consider the following model for a stock price y_t:

   $$\begin{cases} y_t = \theta_t + v_t \\ \theta_t = \theta_{t-1} + w_t \end{cases}$$

   where $\theta_0 \sim N(25, 10)$, $v_t \sim N(0, 2)$, $w_t \sim N(0, 0.1)$.

   Install the `sspir` package and check the `SS` function. Check the `kfilter` function and `smooth`. Try the following:

```
> library(sspir)
> set.seed(1)
> Plummet.dat <- 20 + 2*rnorm(20) + c(rep(0,10), rep(-10,10))
> n <- length(Plummet.dat)
> Plummet.mat <- matrix(Plummet.dat, nrow = n, ncol = 1)
> m1 <- SS(y = Plummet.mat,
     Fmat = function(tt,x,phi) return( matrix(1) ),
```

```
     Gmat = function(tt,x,phi) return( matrix(1) ),
     Wmat = function(tt,x,phi) return( matrix(0.1)),
     Vmat = function(tt,x,phi) return( matrix(2) ),
     m0 = matrix(25), C0 = matrix(10)
     )
> plot(m1$y, ylab = "Closing price", main = "Simulated")
> m1.f <- kfilter(m1)
> m1.s <- smoother(m1.f)
> lines(m1.f$m, lty = 2)
> lines(m1.s$m, lty = 3)
```

and

```
> plot(m1$y, ylab = "Closing price", main = "Simulated")
> m1.f <- kfilter(m1)
> lines(m1.f$m, lty = 2)
> m2 <- m1
> Wmat(m2) <- function(tt, x, phi) {
  if (tt == 12) return(matrix(10)) else return(matrix(0.1))
  }
> m2.f <- kfilter(m2)
> lines(m2.f$m,lty=4)
```

(a) Change the variance of w_t from 0.1 to 10, and comment on the change in the filter and smoother paths.

(b) Change the variance of w_t from 0.1 to 10 and the variance of v_t from 2 to 200, and comment on the change in the filter and smoother paths.

4. **Global Temperatures** The data for this assignment is taken from the package **astsa**. Start by loading this package:

```
> install.packages("astsa")
```

The data for the exercise is found in the data file `gtemp` in this package. Let y_t represent the global temperature series.

(a) Plot the series.

(b) Fit a smoothing spline (try `?smooth.spline`) using GCV (generalised cross validation, which is the default method) and plot the result, superimposed on the data. Repeat using `spar=0.5` and `spar = 0.7`.

(c) Write the model $y_t = x_t + v_t$ with $\nabla^2 x_t t = w_t$, in state-space form. [Hint: The state will be a 2×1 vector, say, $x_t = (x_t; x_{t-1})'$.] Assume w_t and v_t are independent Gaussian white noise processes, both independent of x_0. Fit this state-space model to y_t, and produce a time plot the estimated smoother, $\widehat{x}_t^n$ and the corresponding error limits, $\widehat{x}_t^n \pm 2\sqrt{\widehat{P}_t^n}$ superimposed on the data.

(d) Superimpose all fits on the data and comment on the results.

(e) For one-step prediction, compare the performance of the Kalman filter with Holt Winters filtering.