# Kalman Filtering in R

Eli Gurarie

StatR 301 - Lecture 10
University of Washington - Seattle

June 6, 2013

## Kálmán Filter (in words)

The Kalman filter is an *algorithm* that optimally estimates a *linear*, *stochastic*, *discrete* process (with optional *control*) that is observed with *noise*.

## Kálmán Filter (in equations)

**Process** equation:

$$\mathbf{x}_k = \mathbf{F}_k \mathbf{x}_{k-1} + \mathbf{B}_k \mathbf{u}_k + \mathbf{w}_k$$

where

- $\mathbf{F}_k$ is the state transition model which is applied to the previous state $\mathbf{x}_{k-1}$
- $\mathbf{B}_k$ is some known control input model, applied to the control vector $\mathbf{u}_k$
- $\mathbf{w}_k$ is (multi-variate normal) the process noise $\mathcal{N}(0,\mathbf{Q}_k)$.

**Observation** equation:

$$\mathbf{z}_k = \mathbf{H}\mathbf{x}_k + \mathbf{v}_k$$

where

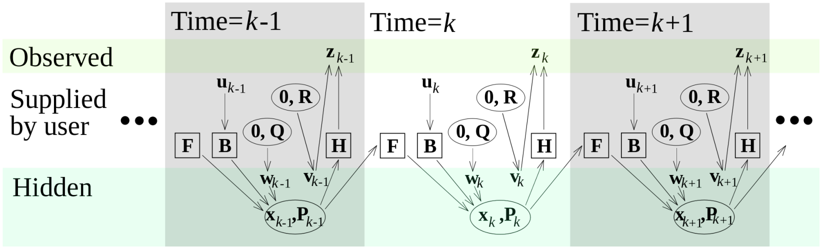
- $\mathbf{H}$ translates the state to the observation
- $\mathbf{v}_k$ is the observation noise $\mathcal{N}(0,\mathbf{R}_k)$.

(note: upper-case are all matrices, lower-case all vectors)

## Kalman Model: Schematic

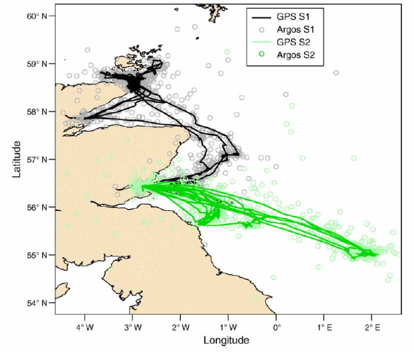
$$\mathbf{x}_k = \mathbf{F}_k \mathbf{x}_{k-1} + \mathbf{B}_k \mathbf{u}_k + \mathbf{w}_k$$

$$\mathbf{z}_k = \mathbf{H} \mathbf{x}_k + \mathbf{v}_k$$



If you can shoe-horn your problem into this fairly flexible framework, you can apply the filter.

## Parsed example with Animal Movement[1]



- GPS locations are "exact"
- ARGOS locations are "lousy" (but much cheaper / longer lived)
- GOAL: to see if the GPS process can be obtained from the ARGOS observations

**Process:**

$$\mathbf{x}_t = \mathbf{x}_{t-1} + \mathbf{c}(\Delta t) + \mathbf{w}$$

where $\mathbf{x}_t = \{x_t, y_t\}$ - seal position (2-D), **c** is a random walk movement step.
so:
$\mathbf{F} = \mathbf{I}$; $\mathbf{Bu} = \mathbf{c\Delta t}$
$\mathbf{w} \sim \mathcal{N}\left(0, \Sigma = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}\right)$

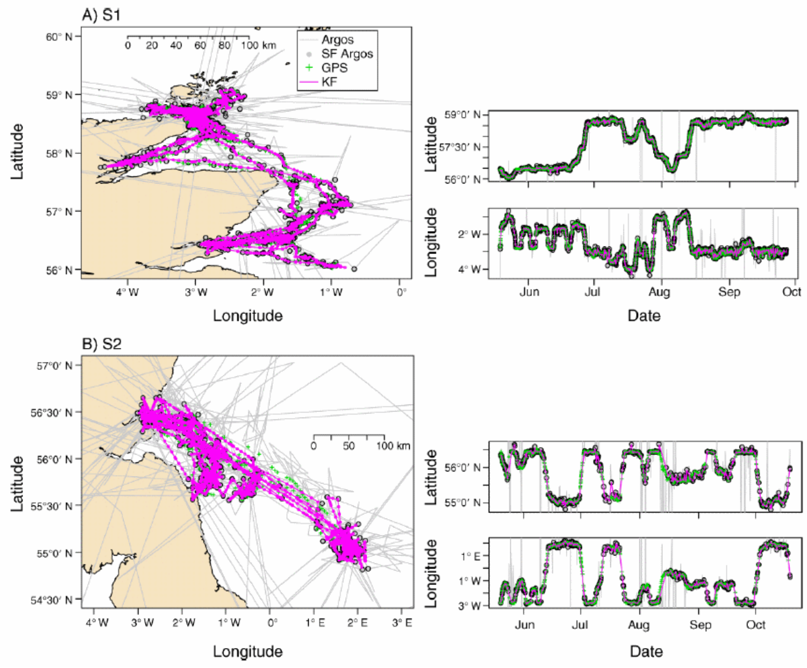
**Observation:**

$$\mathbf{z}_k = \mathbf{H}\mathbf{x}_k + \mathbf{v}_k$$

$\mathbf{z}_t$ - ARGOS observation; $\mathbf{H} = \mathbf{I}$
$\mathbf{v} \sim \mathcal{N}\left(0, \Sigma = \begin{bmatrix} \sigma_{x,f}^2 & 0 \\ 0 & \sigma_{y,f}^2 \end{bmatrix}\right)$
where $f$ the reported quality of satellite "fix" (3,2,1,A,B,C,Z) with known standard errors

[1]Patterson et al., 2010 *Ecology*, 91(1): 273-285

## Seal Movement: after applying the Kalman filter



## Ok, so how does the filtering work?

The goal is to estimate the state $\hat{\mathbf{x}}_{k|k}$ and the error-covariance matrix $\mathbf{P}_{k|k}$ (a measure of the estimated accuracy of the state estimate). The filter is an iterative, two-step process:

**Step 1. Prediction**

| | |
|---|---|
| Predicted (*a priori*) state estimate | $\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k \hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_{k-1}\mathbf{u}_{k-1}$ |
| Predicted (*a priori*) estimate covariance | $\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^{\mathsf{T}} + \mathbf{Q}_k$ |

**Step 2. Innovation / Updating**

| | |
|---|---|
| Innovation or measurement residual | $\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k|k-1}$ |
| Innovation (or residual) covariance | $\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^{\mathsf{T}} + \mathbf{R}_k$ |
| Optimal Kalman gain | $\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^{\mathsf{T}} \mathbf{S}_k^{-1}$ |
| Updated (*a posteriori*) state estimate | $\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k$ |
| Updated (*a posteriori*) estimate covariance | $\mathbf{P}_{k|k} = (I - \mathbf{K}_k \mathbf{H}_k)\mathbf{P}_{k|k-1}$ |

Note:

- it is *iterative* because your are always plugging in new estimates of $\hat{\mathbf{x}}$ and $\hat{\mathbf{P}}$
- The key piece is the Kalman gain - which combines the variance-covariance of the process with the variance-covariance of the observation.
- It is sort of important to have a good idea of $x_0$ and $P_0$ - if these are correct (and all independence + Gaussian assumptions hold + the covariance of the noise is perfect), then theory predicts that these estimates will be optimal and unbiased.

## Simple case

1-D random walk + unbiased observation error If $\mathbf{H} = \mathbf{F} = 1$ and all matrices are constants:

**Step 1. Prediction**

| | |
|---|---|
| Predicted state | $\hat{x}_{k\|k-1} = \hat{x}_{k-1\|k-1} + B_{k-1} u_{k-1}$ |
| Predicted variance | $P_{k\|k-1} = P_{k-1\|k-1} + Q_k$ |

**Step 2. Innovation / Updating**

| | |
|---|---|
| Measurement residual | $\tilde{y}_k = z_k - \hat{x}_{k\|k-1}$ |
| Kalman gain | $K_k = P_{k\|k-1} / (P_{k\|k-1} + R_k)$ |
| Updated state | $\hat{x}_{k\|k} = \hat{x}_{k\|k-1} + K_k \tilde{y}_k$ |
| Updated variance | $P_{k\|k} = (1 - K_k) P_{k\|k-1} = \frac{R}{R+P} P$ |

Note that the variance estimate $P$ becomes smaller with each update.

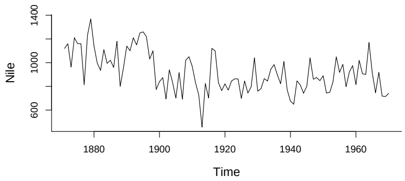
## Kalman filter in R

A review of R packages that perform Kalman filtering is here:
`http://www.jstatsoft.org/v39/i02/paper`

We will use the oldest package called `dse` (Dynamic Systems Estimation) which has many robust methods for fitting and analyzing time series, including multi-variate, state-space, and ARMA models (see the vignette).

## Using dse

Let's model the flow of the Nile at Ansha dam (an R dataset)



Very simple single state model:

$$\begin{aligned} x_t &= x_{t-1} + w_t \\ y_t &= x_t + v_t \end{aligned}$$

Initialize model using the SS (State space) function:

```r
require(dse)
m1.dse <- SS(F = matrix(1, 1, 1), Q = matrix(40, 1, 1), H = matrix(1, 1, 1),
    R = matrix(130, 1, 1), z0 = matrix(0, 1, 1), P0 = matrix(10^5, 1, 1))
```

Note: any number that is not a 0 or 1 (i.e., only process and observation variances Q and R) get estimated.

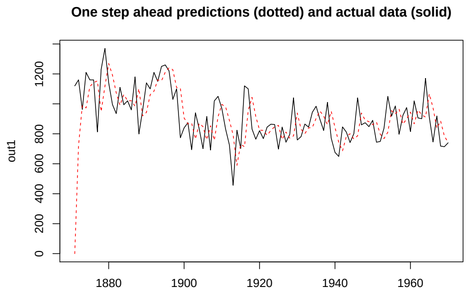
## Using dse

To fit the model:

```
m1.dse.est <- estMaxLik(m1.dse, TSdata(output = Nile))
```

Note that the `TSdata()` function creates a `TSmodel` object that the Maximum Likelihood estimator processes.

```
tfplot(m1.dse.est)
```



```
m1.dse.est

## neg. log likelihood= 666.9
##
##  F =
##      [,1]
## [1,]    1
##
##  H =
##      [,1]
## [1,]    1
##
##  Q =
##      [,1]
## [1,] 96.9
##
##  R =
##       [,1]
## [1,] 101.8
##
##  initial state =
## [1] 0
##
##  initial state tracking error =
##       [,1]
## [1,] 1e+05
```

## Dam construction

Less trivially, suppose we want to take into account a jump in the flow of the river following the construction of Ashwan dam in 1898. Here is a fitting of this model using Mdlm (Dynamic Linear modeling: a newer, mostly Bayesian, package). This example is from the vignette:

First, you have to specify a model in terms of a generic vector of parameter, here x is the initial version . The key function is dlmModPoly:

```r
require(dlm)
buildFun <- function(x) {
    m <- dlmModPoly(1, dV = exp(x[1]))
    m$JW <- matrix(1)
    m$X <- matrix(exp(x[2]), nc = 1, nr = length(Nile))
    m$X[which(time(Nile) == 1899), 1] <- m$X[which(time(Nile) == 1899), 1] *
        (1 + exp(x[3]))
    return(m)
}
```

m is a dlm object, which contains all of the process + observation errors and controls (see the vignette to keep the letters straight). m$X is an optional matrix that lets us inflate the system variance at that time. That the three parameters in X are related to the observation variance, and the process variance in general and during the switch, respectively.

## Fitting the dlm

As with so much in R, most of the work is in specifying the model. Fitting is fast:

```
fit <- dlmMLE(Nile, parm = c(0, 0, 0), build = buildFun)
fit$par

## [1]  9.699 -3.578 14.588
```

The following builds a process out of the results of the fit:

```
dlmNile <- buildFun(fit$par)
V(dlmNile)

##       [,1]
## [1,] 16300
```

Here, we can see how the variance was estimated around the contruction of the dam:

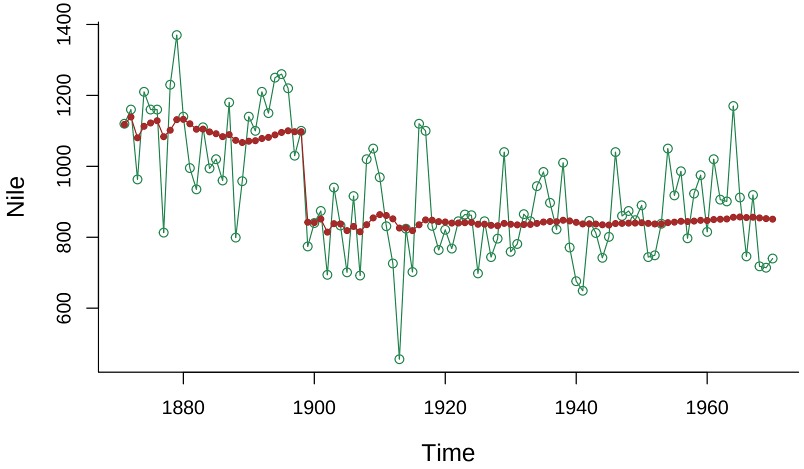
```
dlmNile$X[match(1898:1900, time(Nile))]

## [1] 2.792e-02 6.048e+04 2.792e-02
```

## Applying the Kalman filter

Now that we have estimated all of the variancea and covariances, to produce the actual Kalman filtered estimates of the process, pass your estimated dlm to the `dlmFilter` command

```
nileJumpFilt <- dlmFilter(Nile, dlmNile)
plot(Nile, type = "o", col = "seagreen")
lines(dropFirst(nileJumpFilt$m), type = "o", pch = 20, col = "brown")
```



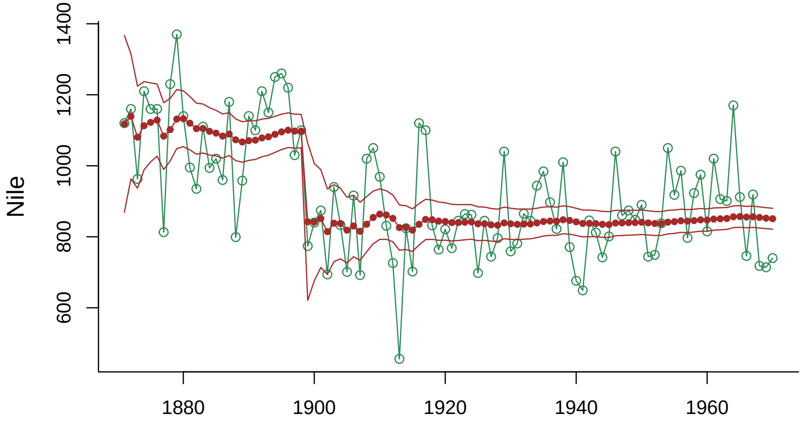
## Applying the Kalman filter

Finally, you can extract the variance of the process from the Kalman fit using:

```
v <- unlist(dlmSvd2var(nileJumpFilt$U.C, nileJumpFilt$D.C))
```

`U.C` and `D.C` are the single value decompositions (see your linear algebra book!) of the observation errors. The function `dlmSvd2var` converts those to an estimate of the variance at each observation.

```
lines(dropFirst(nileJumpFilt$m - 1.95 * sqrt(v)), col = "brown")
lines(dropFirst(nileJumpFilt$m + 1.95 * sqrt(v)), col = "brown")
```



**See also:**

- Review of R packages for Kalman Filtering: http://www.jstatsoft.org/v39/i02/paper
- Vignettes of `dlm` and `dsm`

## Final thoughts (both on KF and parametetric modeling in general)

- R's strength is the bewildering number of packages... more specifically, the incredible number of different kinds of models that you can fit to dependent, multivariate, state-changing, observed-with-error, etc. type processes, fitted with optimized likelihoods and Bayesian methods and classifications and on and on and on - often incredibly efficiently.
- ... the flipside of that is that there is very little consistency in syntax, specifying, etc. (This is partially a consequence of most packages being created by academics).
- Your job as sophisticated R users is to:
    1. translate a question (or process) into a model that you really understand,
    2. write it out symbolically,
    3. translate the symbolic model to the specifications of a built-in package (*often the most difficult part*) ... or
    4. translate the symbolic model into a hand-coded implementation in R (*even more difficult, but most satisfying*)
    5. be able to interpret the output of the model with respect to your research question