MATH 8090: State-Space Models II

Whitney Huang, Clemson University

11/28-30/2023

Contents

Local level model

Simulate data from local level models

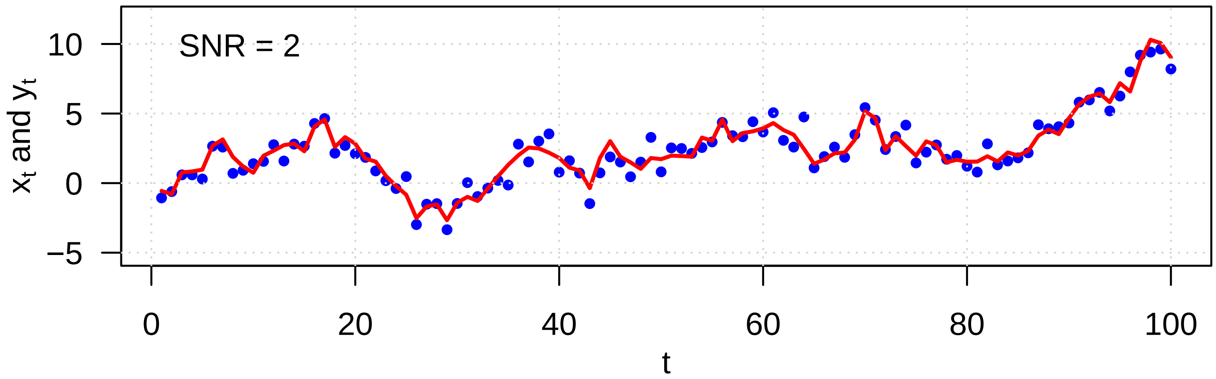
$$
\begin{aligned}
Y_t &= X_t + W_t, & W_t &\sim \mathrm{N}(0, \sigma_W^2), \\
X_t &= X_{t-1} + V_t, & V_t &\sim \mathrm{N}(0, \sigma_v^2).
\end{aligned}
$$

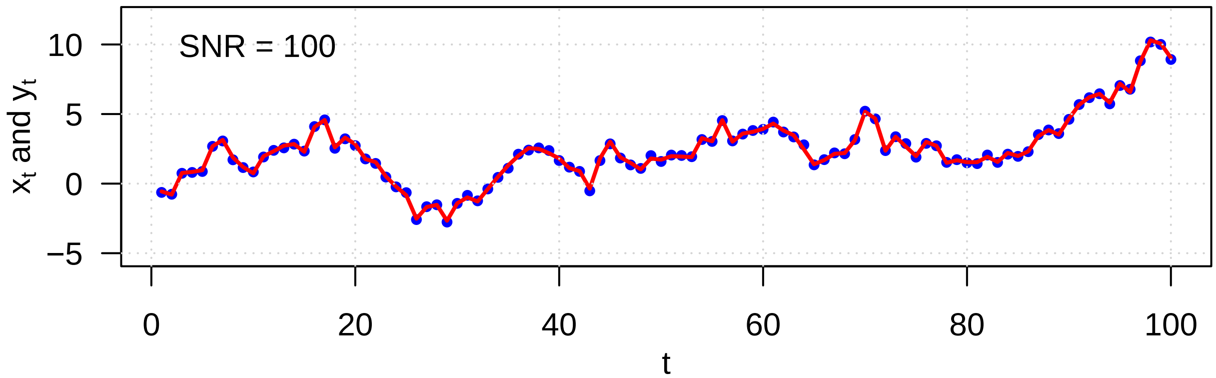
Here $\mu_0 = 0$, $\sigma_0^2 = 1$, $\sigma_V^2 = 1$, $\sigma_W^2 = \sigma_V^2/\mathrm{SNR}$.

```r
set.seed(123)
mu0 <- 0; sig0 <- 1; sig2.V <- 1
X.0 <- rnorm(1, mean = mu0, sd = sqrt(sig0))
X <- cumsum(c(X.0, rnorm(99, sd = sqrt(sig2.V))))
W <- rnorm(100)
SNR <- 2; Y.2 <- X + W * sqrt(sig2.V / SNR)
SNR <- 100; Y.100 <- X + W * sqrt(sig2.V / SNR)
```

Plot state vectors and observation vectors

```r
par(las = 1, mgp = c(2, 1, 0), mar = c(3.5, 3.5, 1, 0.6), mfrow = c(2, 1))
plot(Y.2, col = "blue", pch = 16, cex = 0.75, xlab = "t",
     ylab = expression(paste(x[t], " and ", y[t])), main = "", ylim = c(-5.25, 12))
lines(X, col = "red", lwd = 2)
grid()
legend("topleft", legend = "SNR = 2", bty = "n")
plot(Y.100, col = "blue", pch = 16, cex = 0.75, xlab = "t",
     ylab = expression(paste(x[t], " and ", y[t])), main = "", ylim = c(-5.25, 12))
lines(X, col = "red", lwd = 2)
grid()
legend("topleft", legend = "SNR = 100", bty = "n")
```





Carrying out Kalman filter

The function below is adapted from Dr. `Donald B. Percival`'s UW Stat 519 R codes.

1. Compute innovation:

$$U_t = Y_t - Y_t^{t-1} = Y_t - \mu_t^f.$$

2. Compute `MSE` for Y_t^{t-1} :

$$\Sigma_t^f + \sigma_W^2 = F_t.$$

3. Compute the new filtered value:

$$\mu_t^a = \mu_t^f + K_t U_t,$$

where $K_t = \Sigma_t^f / F_t$ is the `Kalman gain`.

4. Compute MSE for new filtered value:

$$\Sigma_t^a = \Sigma_t^f (1 - K_t).$$

5. Compute new forecast:

$$\mu_{t+1}^f = \mu_t^f + K_t U_t = \mu_t^a.$$

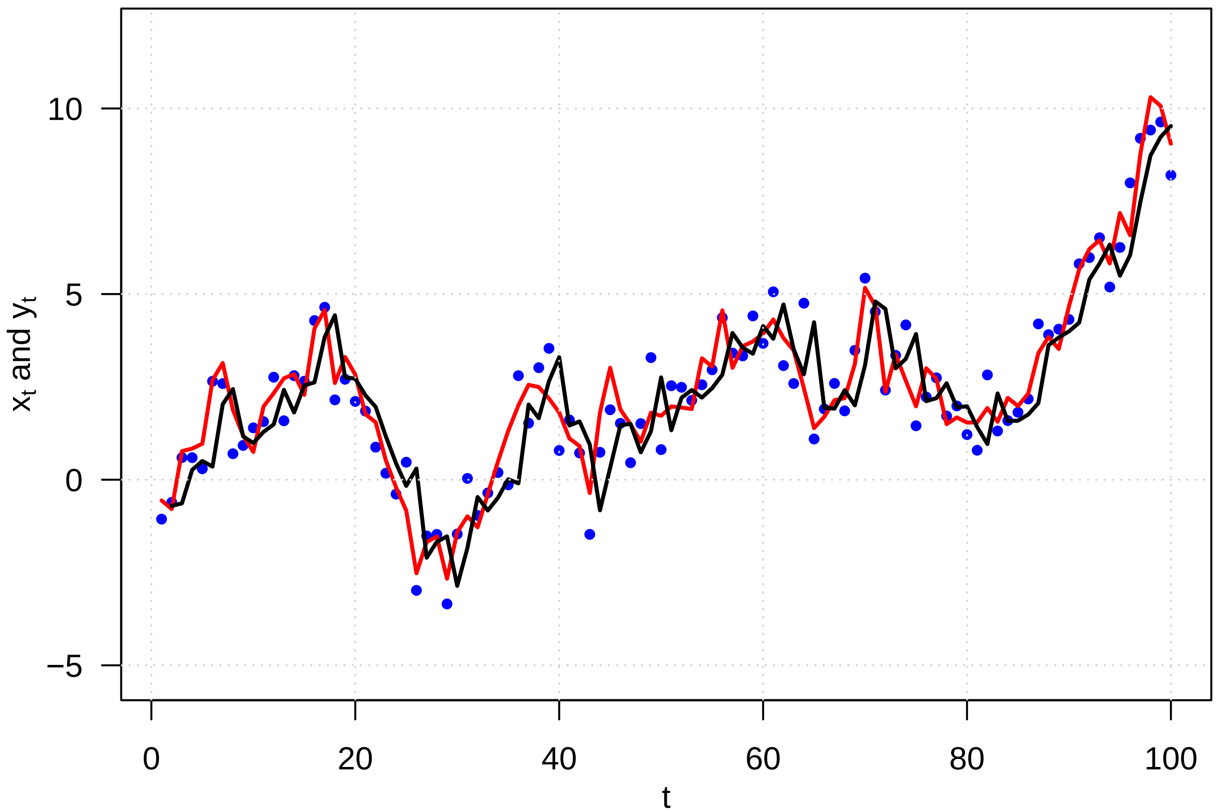
6. Compute MSE for new forecast:

$$\Sigma_{t+1}^f = \Sigma_t^f (1 - K_t) + \sigma_V^2 = \Sigma_t^a + \sigma_{V.}^2$$

```
KF.one.step.local.level <- function(X.t.tm1, P.t.tm1, Y.t, sig2.W, sig2.V){
  U.t <- if(is.na(Y.t)) NA else Y.t - X.t.tm1
  F.t <- P.t.tm1 + sig2.W
  K.t <- if(is.na(Y.t)) 0 else P.t.tm1 / F.t
  X.t.t <- X.t.tm1 + if(is.na(Y.t)) 0 else K.t * U.t
  P.t.t <- P.t.tm1 * (1 - K.t)
  X.tp1.t <- X.t.t
  P.tp1.t <- P.t.t + sig2.V
  structure(list(filter = X.t.t, forecast = X.tp1.t, filter.var = P.t.t,
                 forecast.var = P.tp1.t, innovation = U.t, innovation.var = F.t,
                 gain = K.t))
}

KF.n.steps.local.level <- function(ts, m.1 = 0, P.1 = 1, sig2.W = 1, sig2.V = 1){
  n <- length(ts)
  filter.ts <- forecast.ts <- filter.var.ts <- innovations.ts <- rep(0, n)
  forecast.var.ts <- innovations.var.ts <- gain.ts <- rep(0, n)
  X.forecast.in <- m.1; X.forecast.var.in <- P.1
  forecast.ts[1] <- X.forecast.in; forecast.var.ts[1] <- X.forecast.var.in
  Y.in <- ts[1]
  for(t in 1:n){
    temp <- KF.one.step.local.level(X.forecast.in, X.forecast.var.in, Y.in,
                                    sig2.W, sig2.V)
    filter.ts[t] <- temp$filter; filter.var.ts[t] <- temp$filter.var
    innovations.ts[t] <- temp$innovation; innovations.var.ts[t]  <- temp$innovation.var
    gain.ts[t] <- temp$gain
    if(t < n){
      forecast.ts[t + 1] <- temp$forecast
      forecast.var.ts[t + 1] <- temp$forecast.var
      X.forecast.in <- temp$forecast
      X.forecast.var.in <- temp$forecast.var
      Y.in <- ts[t + 1]
    }
  }
  structure(list(filter.ts = filter.ts, forecast.ts = forecast.ts,
                 filter.var.ts = filter.var.ts, forecast.var.ts = forecast.var.ts,
                 innovations.ts = innovations.ts, innovations.var.ts = innovations.var.ts,
                 gain.ts = gain.ts))
}
```

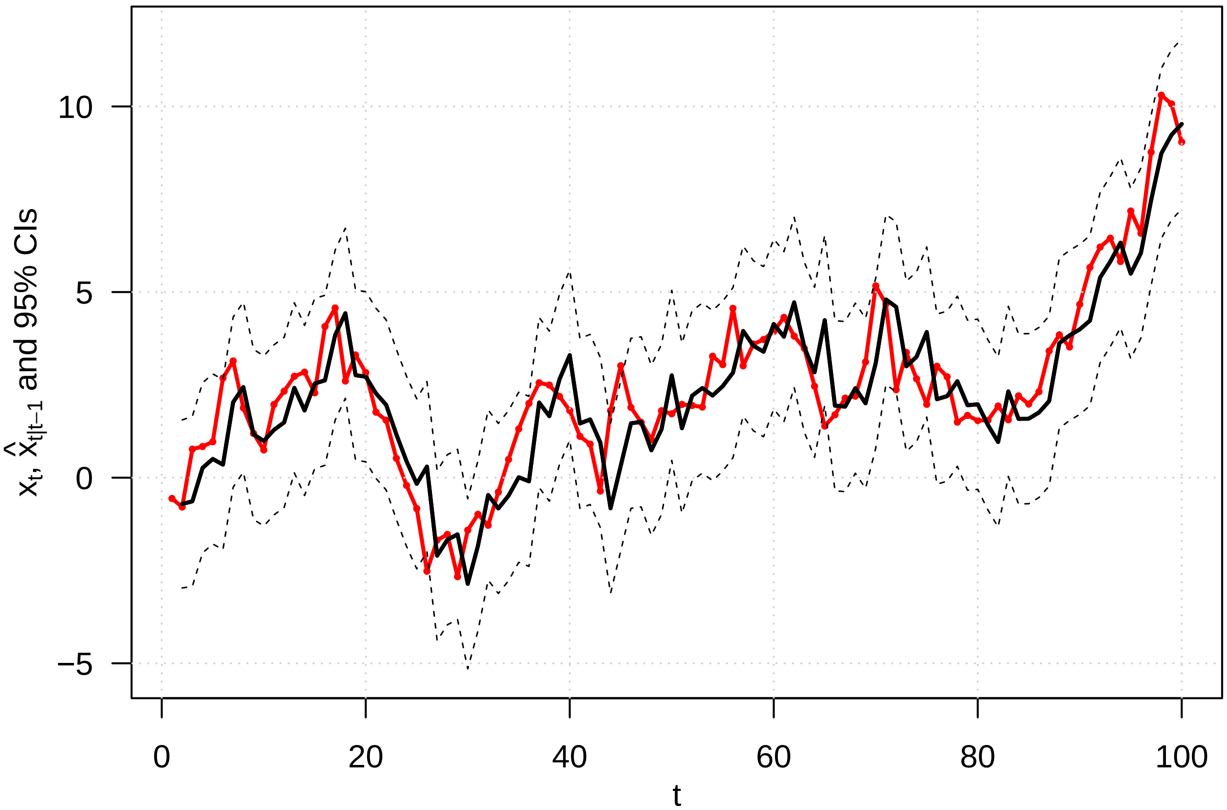
Kalman filter: forecasting

```
Y.2.KF <- KF.n.steps.local.level(Y.2, sig2.W = 0.5)

par(las = 1, mgp = c(2, 1, 0), mar = c(3.5, 3.5, 1, 0.6))
plot(Y.2, col = "blue", pch = 16, cex = 0.75, xlab = "t",
     ylab = expression(paste(x[t], " and ", y[t])), main = "", ylim = c(-5.25, 12))
lines(X, col = "red", lwd = 2)
lines(2:100, Y.2.KF$forecast.ts[-1], lwd = 2)
grid()
```



Forecasting interval

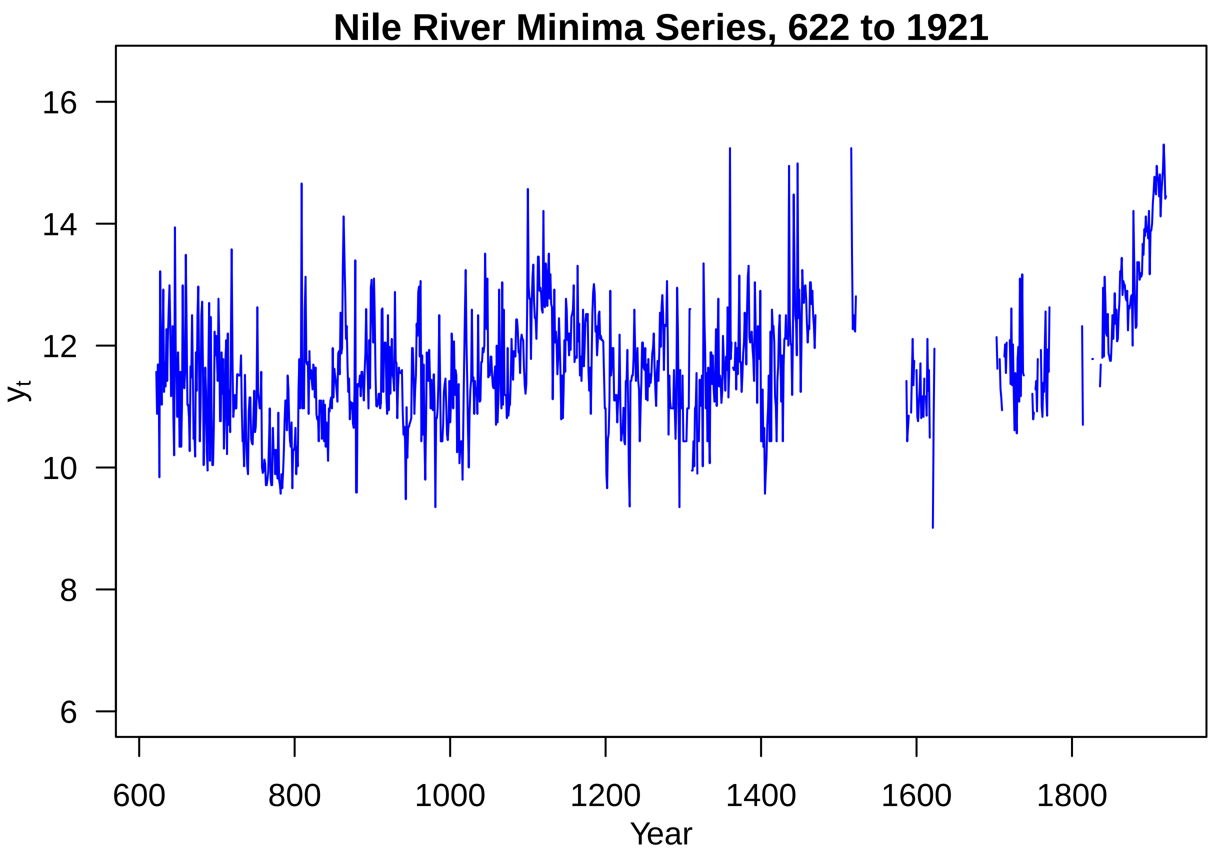
```
par(las = 1, mgp = c(2, 1, 0), mar = c(3.5, 3.5, 1, 0.6))
plot(1:100, X, col = "red", pch = 16, cex = 0.5, xlab = "t",
     ylab = expression(paste(x[t], ", ", hat(x)[paste(t, "|", t-1)]," and 95% CIs")),
     main = "", ylim = c(-5.25, 12))
lines(X, col = "red", lwd = 2)
lines(2:100, Y.2.KF$forecast.ts[-1], lwd = 2)
grid()
lines(2:100, Y.2.KF$forecast.ts[-1], lwd = 2)
ME <- qnorm(0.975) * sqrt(Y.2.KF$forecast.var[-1])
lines(2:100, Y.2.KF$forecast.ts[-1] - ME, lwd = .75, lty = 2)
lines(2:100, Y.2.KF$forecast.ts[-1] + ME, lwd = .75, lty = 2)
```



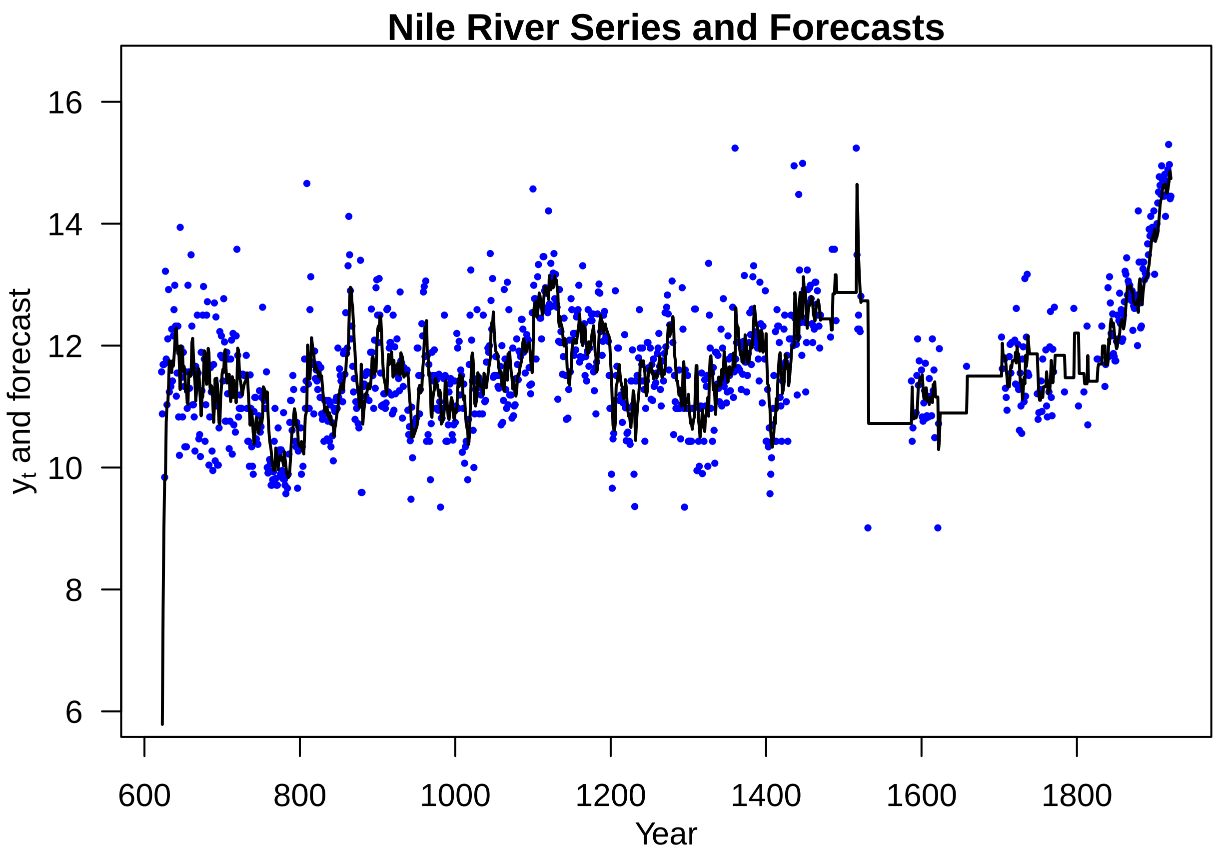
Nile rvier flows missing values imputation

The analysis below is adapted from Dr. `Donald B. Percival`'s UW Stat 519 R codes.

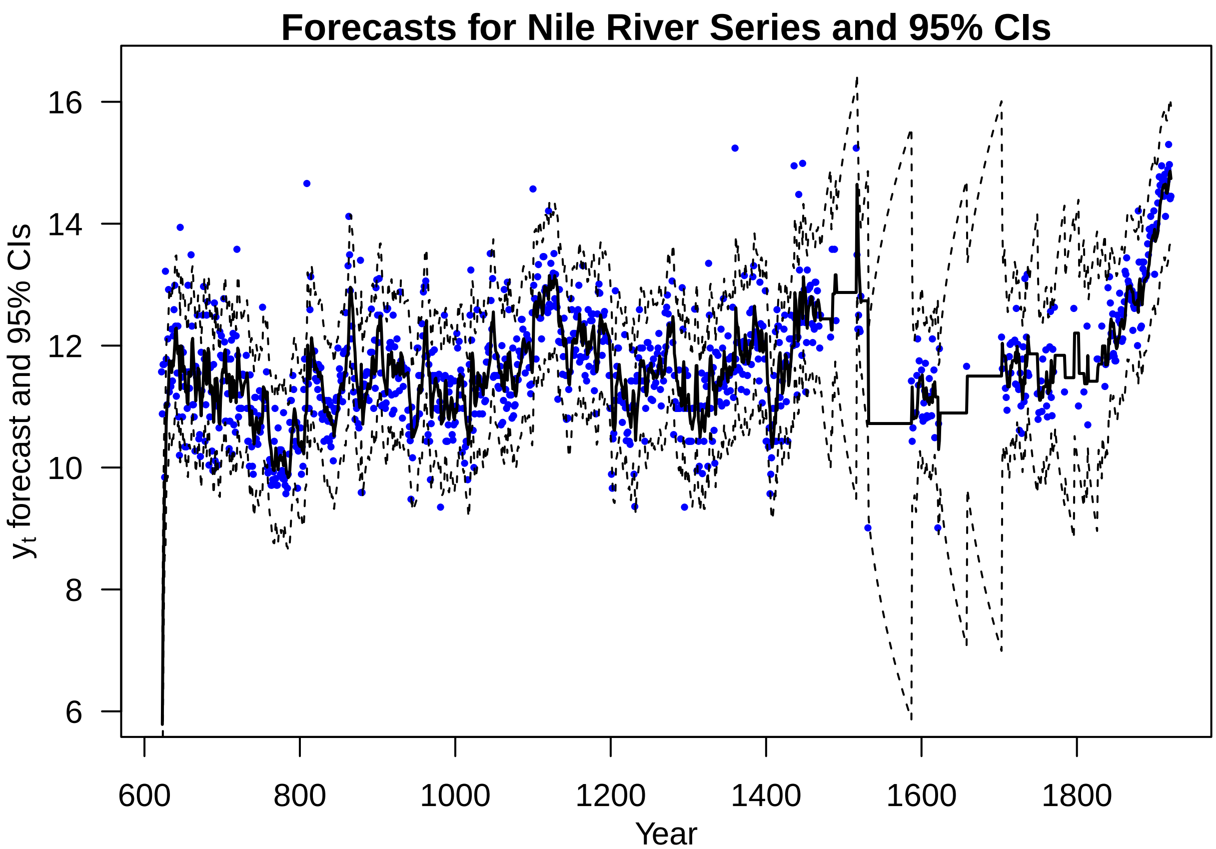
```
nile <- scan("http://faculty.washington.edu/dbp/s519/Data/Nile-622-1921.txt")
nile.years <- 622:1921

par(las = 1, mgp = c(2, 1, 0), mar = c(3.5, 3.5, 1, 0.6))
plot(nile.years, nile, ylim = c(6, 16.5), type = "l", col = "blue",
     xlab = "Year", ylab = expression(y[t]),
     main = "Nile River Minima Series, 622 to 1921")
```




```r
#### Impute the missing values
nile.KF <- KF.n.steps.local.level(nile, P.1 = 1, sig2.W = 1, sig2.V = 0.1)
par(las = 1, mgp = c(2, 1, 0), mar = c(3.5, 3.5, 1, 0.6))
plot(nile.years, nile, pch = 16, cex = 0.5, col = "blue", ylim = c(6, 16.5),
     xlab = "Year", ylab = expression(paste(y[t]," and forecast")),
     main = "Nile River Series and Forecasts")
lines(nile.years[-1], nile.KF$forecast.ts[-1], lwd = 1.5)
```




```r
#### Construct confidence interval
par(las = 1, mgp = c(2, 1, 0), mar = c(3.5, 3.5, 1, 0.6))
plot(nile.years, nile, pch = 16, cex = 0.5, col = "blue", ylim = c(6, 16.5),
     xlab = "Year", ylab = expression(paste(y[t]," forecast and 95% CIs")),
     main = "Forecasts for Nile River Series and 95% CIs")
ME <- qnorm(0.975) * sqrt(nile.KF$forecast.var.ts[-1])
lines(nile.years[-1], nile.KF$forecast.ts[-1], lwd = 1.5)
lines(nile.years[-1], nile.KF$forecast.ts[-1] - ME, lty = 2)
lines(nile.years[-1], nile.KF$forecast.ts[-1] + ME, lty = 2)
```



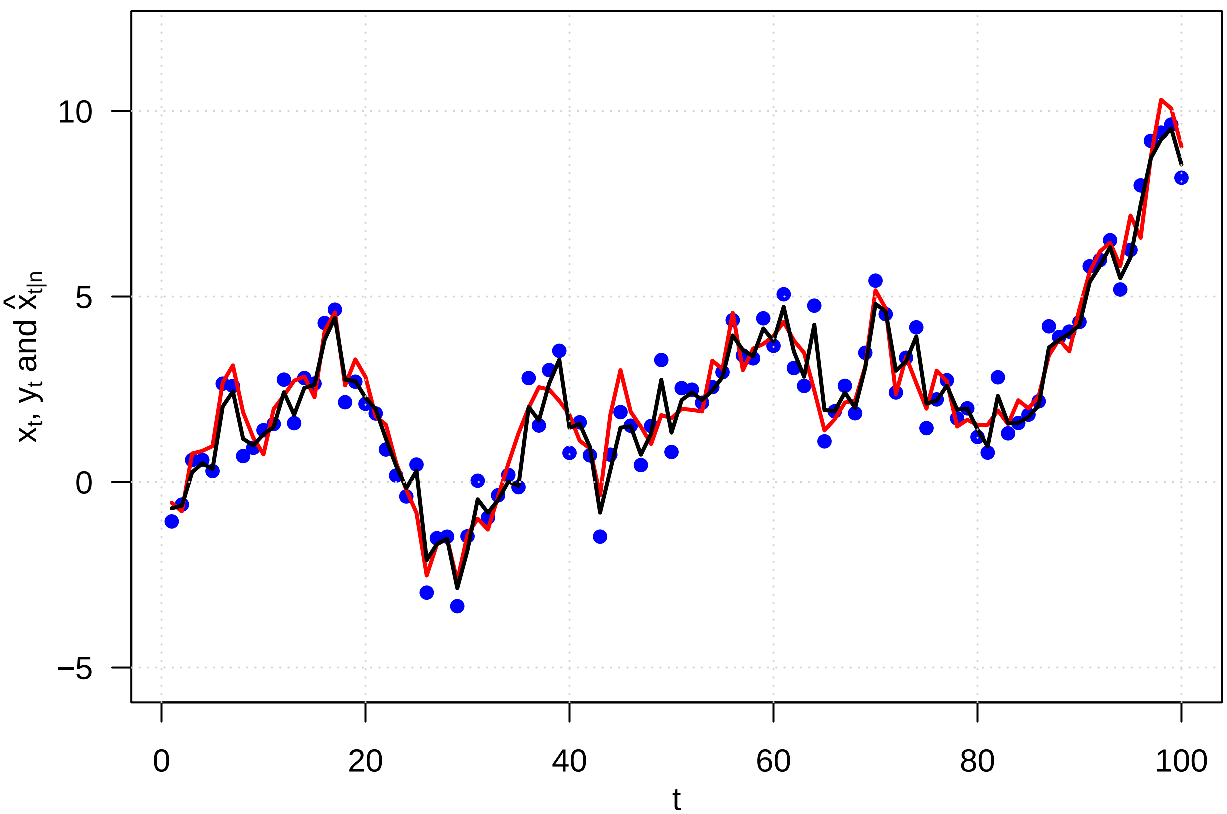
Kalman smoothing

The function below is adapted from Dr. `Donald B. Percival`'s UW Stat 519 R codes.

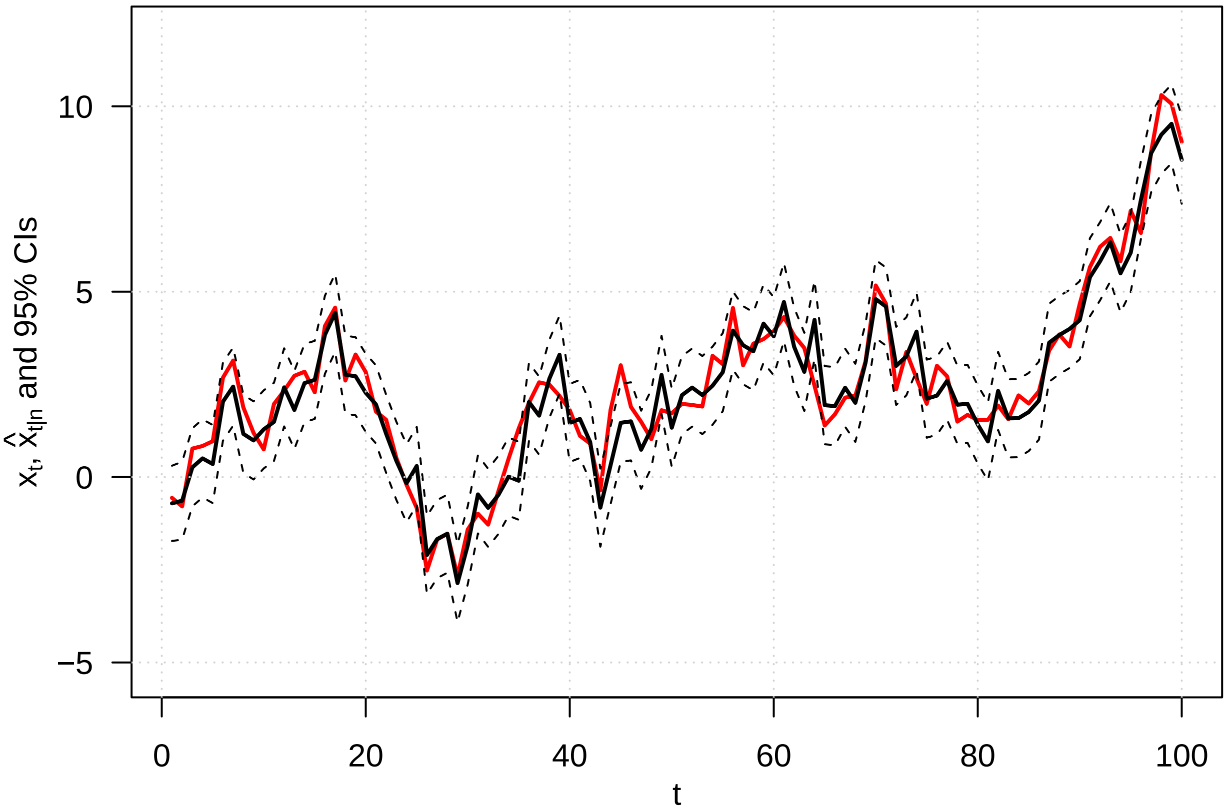
```
KS.local.level <- function(KF.results){
  n <- length(KF.results$filter.ts)
  L.t.ts <- 1 - KF.results$gain.ts
  r.ts <- rep(0, n + 1)
  bg <- is.na(KF.results$innovations.ts)
  innov.0.for.NA <- KF.results$innovations.ts
  innov.0.for.NA[bg] <- 0
  for(t in n:1) r.ts[t] <- innov.0.for.NA[t] / KF.results$innovations.var.ts[t]
  + L.t.ts[t] * r.ts[t+1]
  smooth.ts <- KF.results$forecast.ts + KF.results$forecast.var.ts * r.ts[-(n + 1)]
  N.t.ts <- rep(0, n + 1)
  for(t in n:1) N.t.ts[t] <- 1 / KF.results$innovations.var.ts[t] +
    (L.t.ts[t])^2 * N.t.ts[ t + 1]
  smooth.var.ts <- KF.results$forecast.var.ts -
    (KF.results$forecast.var.ts)^2 * N.t.ts[-(n + 1)]
  structure(list(L.t.ts = L.t.ts, r.ts = r.ts, smooth.ts = smooth.ts,
                 N.t.ts = N.t.ts, smooth.var.ts = smooth.var.ts))
}
```

Kalman smoothing: local level model example

```
Y.2.KS <- KS.local.level(Y.2.KF)

par(las = 1, mgp = c(2, 1, 0), mar = c(3.5, 3.5, 1, 0.6))
plot(Y.2, col = "blue", pch = 16, xlab = "t",
     ylab = expression(paste(x[t],", ",y[t]," and ", hat(x)[paste(t, "|", n)])), main = "",
     ylim = c(-5.25, 12))
lines(X, col = "red", lwd = 2)
lines(Y.2.KS$smooth.ts, lwd = 2)
grid()
```




```
par(las = 1, mgp = c(2, 1, 0), mar = c(3.5, 3.5, 1, 0.6))
plot(X, col = "red", typ = "l", lwd = 2, xlab = "t",
     ylab = expression(paste(x[t],", ",hat(x)[paste(t, "|", n)]," and 95% CIs")),
     main = "", ylim = c(-5.25, 12))
lines(Y.2.KS$smooth.ts, lwd = 2)
lines(Y.2.KS$smooth.ts - 1.96 * sqrt(Y.2.KS$smooth.var.ts), lty = 2)
lines(Y.2.KS$smooth.ts + 1.96 * sqrt(Y.2.KS$smooth.var.ts), lty = 2)
grid()
```



Parameter estimation

This example is taken from Shumway and Stoffer (2017) example 6.6

Generate data

```
library(astsa)
set.seed(123)
num = 100
N = num + 1
x <- sarima.sim(n = N, ar = .8)
y <- ts(x[-1] + rnorm(num, 0, 1))
```

Initial estimates

$\phi^{(0)} = \frac{\hat{\rho}_Y(2)}{\hat{\rho}_Y(1)}$.

$\sigma_W^{2^{(0)}} = (1 - \phi^{2^{(0)}})\hat{\gamma}_Y(1)/\phi^{(0)}$.

$\sigma_V^{2^{(0)}} = \hat{\gamma}_Y(0) - \left[\frac{\sigma_W^{2^{(0)}}}{1-\phi^{2^{(0)}}}\right]$

```
u = ts.intersect(y, lag(y, -1), lag(y, -2))
varu = var(u)
coru = cor(u)
```

```r
phi = coru[1, 3] / coru[1, 2]
q = (1 - phi^2) * varu[1, 2] / phi
r = varu[1, 1] - q / (1 - phi^2)
(init.par = c(phi, sqrt(q), sqrt(r)))
```

```
## [1] 0.7614651 1.0020091 0.8744762
```

Function to evaluate the likelihood

```r
Linn <- function(para){
  phi <- para[1]; sigw <- para[2]; sigv <- para[3]
  Sigma0 <- (sigw^2) / (1 - phi^2); Sigma0[Sigma0 < 0] = 0
  kf = Kfilter(y, 1, mu0 = 0, Sigma0, phi, sigw, sigv)
  return(kf$like)
}
```

Estimation

```r
(est = optim(init.par, Linn, gr = NULL, method = "BFGS", hessian = TRUE,
             control = list(trace = 1, REPORT = 1)))
```

```
## initial  value 84.170842
## iter   2 value 84.102702
## iter   3 value 83.916203
## iter   4 value 83.915653
## iter   5 value 83.889723
## iter   6 value 83.885783
## iter   7 value 83.885762
## iter   7 value 83.885762
## iter   7 value 83.885762
## final  value 83.885762
## converged
```

```
## $par
## [1] 0.8213276 0.8308274 0.9691287
##
## $value
## [1] 83.88576
##
## $counts
## function gradient
##       29        7
##
## $convergence
## [1] 0
##
## $message
## NULL
##
```

```
## $hessian
##             [,1]      [,2]       [,3]
## [1,] 263.738652 74.14214 -9.936399
## [2,]  74.142142 69.77014 44.355806
## [3,]  -9.936399 44.35581 85.616367
```

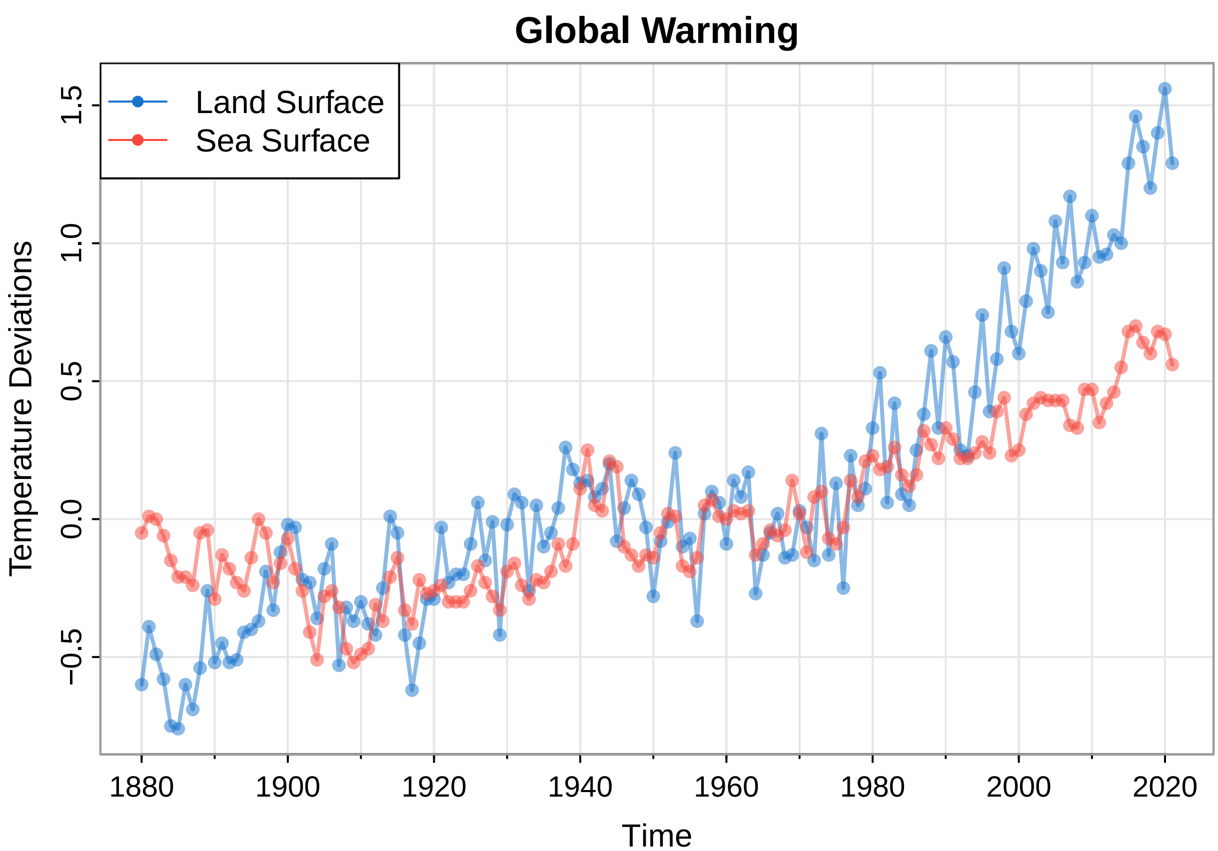
```
SE = sqrt(diag(solve(est$hessian)))
cbind(estimate = c(phi = est$par[1], sigw = est$par[2], sigv = est$par[3]), SE)
```

```
##       estimate         SE
## phi  0.8213276 0.08831157
## sigw 0.8308274 0.20920610
## sigv 0.9691287 0.15849779
```

Global temperature example

This example is taken from Shumway and Stoffer (2017) example 6.7

```
tsplot(cbind(gtemp_land, gtemp_ocean), spaghetti = TRUE,
       lwd = 2, pch = 20, type = "o", col=astsa.col(c(4,2),.5),
       ylab = "Temperature Deviations", main = "Global Warming")
legend("topleft", legend = c("Land Surface", "Sea Surface"), lty = 1,
       pch = 20, col = c(4, 2), bg = "white")
```



```r
#### Setup
y = cbind(globtemp/sd(globtemp), globtempl/sd(globtempl))
num = nrow(y)
input = rep(1, num)
A = matrix(c(1, 1), nrow = 2)
mu0 = -.35; Sigma0 = 1; Phi = 1
```

```r
#### Function to Calculate Likelihood
Linn=function(para){
  sQ = para[1]      # sigma_w
  sR1 = para[2]     # 11 element of  sR
  sR2 = para[3]     # 22 element of sR
  sR21 = para[4]    # 21 element of sR
  sR = matrix(c(sR1, sR21, 0, sR2), 2)  # put the matrix together
  drift = para[5]
  kf = Kfilter(y, A, mu0, Sigma0, Phi, sQ, sR,
               Ups = drift, Gam = NULL, input)  # NOTE Gamma is set to NULL now (instead of 0)
  return(kf$like)
}
```

```r
#### Estimation
init.par = c(.1, .1, .1, 0, .05)  # initial values of parameters
(est = optim(init.par, Linn, NULL, method = "BFGS", hessian = TRUE,
             control = list(trace = 1, REPORT = 1)))
```

```
## initial  value 66.388539
## iter   2 value -168.023751
## iter   3 value -176.435356
## iter   4 value -177.391799
## iter   5 value -179.269359
## iter   6 value -188.964297
## iter   7 value -198.772440
## iter   8 value -202.788250
## iter   9 value -203.540106
## iter  10 value -204.946439
## iter  11 value -205.940174
## iter  12 value -206.647210
## iter  13 value -206.670493
## iter  14 value -206.684192
## iter  15 value -206.694809
## iter  16 value -206.695776
## iter  17 value -206.695794
## iter  18 value -206.695801
## iter  18 value -206.695802
## iter  18 value -206.695805
## final  value -206.695805
## converged
```

```
## $par
## [1] 0.09461713 0.32401331 0.20283345 0.14761763 0.02472785
##
## $value
## [1] -206.6958
```

```
##
## $counts
## function gradient
##       86       18
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## $hessian
##             [,1]       [,2]      [,3]       [,4]        [,5]
## [1,] 2285.12223   494.6403  252.9334   715.3184    44.60378
## [2,]  494.64029  2075.6433 1505.5115 -1460.9779  -146.63890
## [3,]  252.93339  1505.5115 4001.0586  -665.6213   214.95949
## [4,]  715.31842 -1460.9779 -665.6213  2791.7353    31.79560
## [5,]   44.60378  -146.6389  214.9595    31.7956 14613.01275
```

```
SE = sqrt(diag(solve(est$hessian)))
```

```
#### Summary of estimation
estimate = est$par; u = cbind(estimate, SE)
rownames(u)=c("sigw","sR11", "sR22", "sR21", "drift"); u
```

```
##         estimate          SE
## sigw  0.09461713 0.025974347
## sR11  0.32401331 0.038005698
## sR22  0.20283345 0.019122980
## sR21  0.14761763 0.029219285
## drift 0.02472785 0.008292723
```

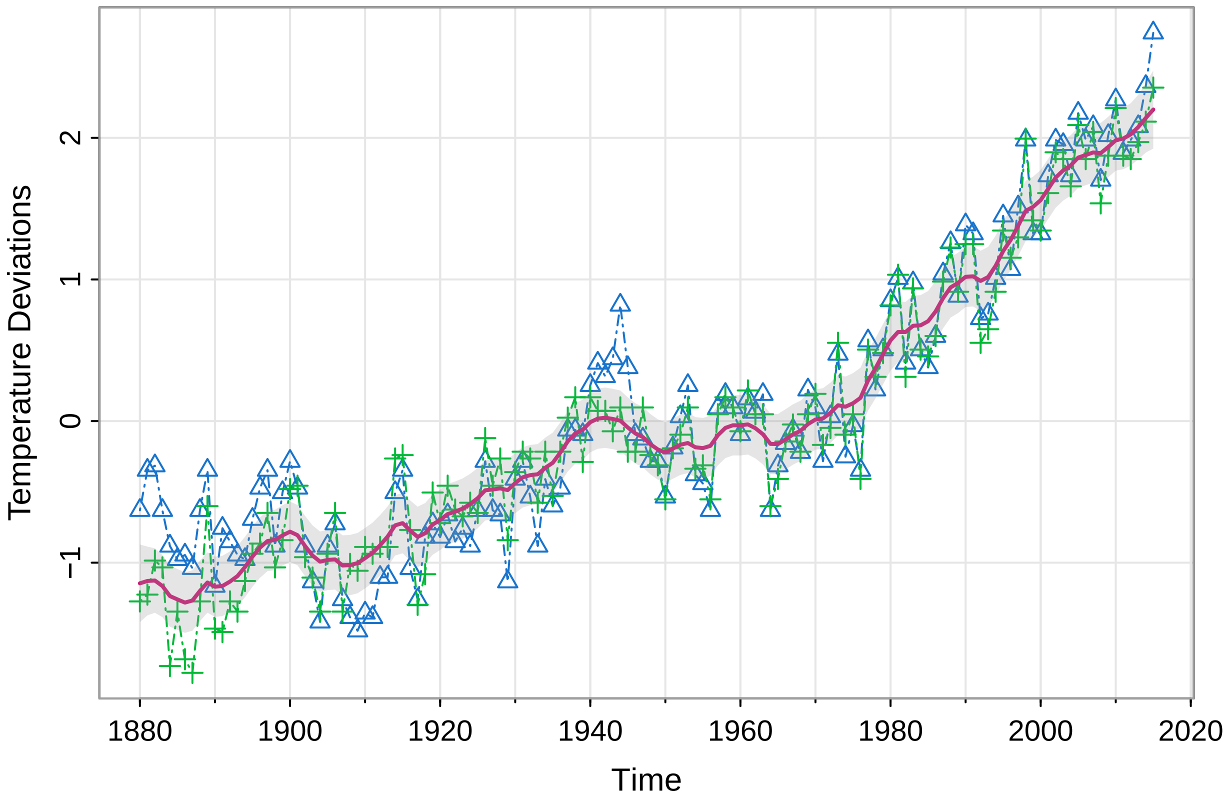
```
#### Smooth
sQ = est$par[1]
sR1 = est$par[2]
sR2 = est$par[3]
sR21 = est$par[4]
sR = matrix(c(sR1, sR21, 0, sR2), 2)
(R = sR %*% t(sR))   #  to view the estimated R matrix
```

```
##             [,1]       [,2]
## [1,] 0.10498463 0.04783008
## [2,] 0.04783008 0.06293237
```

```
drift = est$par[5]
ks = Ksmooth(y, A, mu0, Sigma0, Phi, sQ, sR, Ups = drift,
             Gam = NULL, input)  # NOTE Gamma is set to NULL now (instead of 0)
```

```
#### Plot the results
tsplot(y, spag = TRUE, margins = .5, type = 'o', pch = 2:3, col = 4:3,
       lty = 6, ylab = 'Temperature Deviations')
xsm  = ts(as.vector(ks$Xs), start = 1880)
```

```r
rmse = ts(sqrt(as.vector(ks$Ps)), start = 1880)
lines(xsm, lwd = 2, col = 6)
  xx = c(time(xsm), rev(time(xsm)))
  yy = c(xsm - 2 * rmse, rev(xsm + 2 * rmse))
polygon(xx, yy, border = NA, col = gray(.6, alpha = .25))
```



EM algorithm example

Generate data (same as Example 6.6)

```r
library(nlme)
set.seed(123); num = 100; N = num+1
x = sarima.sim(ar = .8, n = N)
y = ts(x[-1] + rnorm(num, 0, 1))
```

Initial Estimates

```r
u = ts.intersect(y,lag(y, -1), lag(y, -2))
varu = var(u); coru = cor(u)
phi = coru[1,3] / coru[1,2]
q = (1 - phi^2) * varu[1, 2] / phi
r = varu[1, 1] - q/(1 - phi^2)
cr = sqrt(r); cq = sqrt(q); mu0 = 0; Sigma0 = 2.8
(em = EM(y, 1, mu0, Sigma0, phi, cq, cr, 75, .00001))
```

```
## iteration    -loglikelihood
##     1          84.36778
##     2          83.97942
##     3          83.82139
##     4          83.74255
##     5          83.69475
##     6          83.66085
##     7          83.63427
##     8          83.61222
##     9          83.59335
##     10          83.57691
##     11          83.56242
##     12          83.54955
##     13          83.53808
##     14          83.52781
##     15          83.51859
##     16          83.5103
```

```
## $Phi
## [1] 0.8106963
##
## $Q
## [1] 0.7752158
##
## $R
##           [,1]
## [1,] 0.8704274
##
## $mu0
##           [,1]
## [1,] 0.7842457
##
## $Sigma0
##           [,1]
## [1,] 0.1469216
##
## $like
##  [1] 84.36778 83.97942 83.82139 83.74255 83.69475 83.66085 83.63427 83.61222
##  [9] 83.59335 83.57691 83.56242 83.54955 83.53808 83.52781 83.51859 83.51030
##
## $niter
## [1] 16
##
## $cvg
## [1] 9.921766e-05
```

Standard Errors

```r
phi = em$Phi; cq = chol(em$Q); cr = chol(em$R)
mu0 = em$mu0; Sigma0 = em$Sigma0
para = c(phi, cq, cr)
```

Evaluate likelihood at estimates

```
Linn = function(para){
  kf = Kfilter(y, 1, mu0, Sigma0, para[1], para[2], para[3])
  return(kf$like)
  }
emhess = fdHess(para, function(para) Linn(para))
SE = sqrt(diag(solve(emhess$Hessian)))
```

Display summary of estimation

```
estimate = c(para, em$mu0, em$Sigma0); SE = c(SE, NA, NA)
u = cbind(estimate, SE)
rownames(u) = c("phi", "sigw", "sigv", "mu0", "Sigma0")
u
```

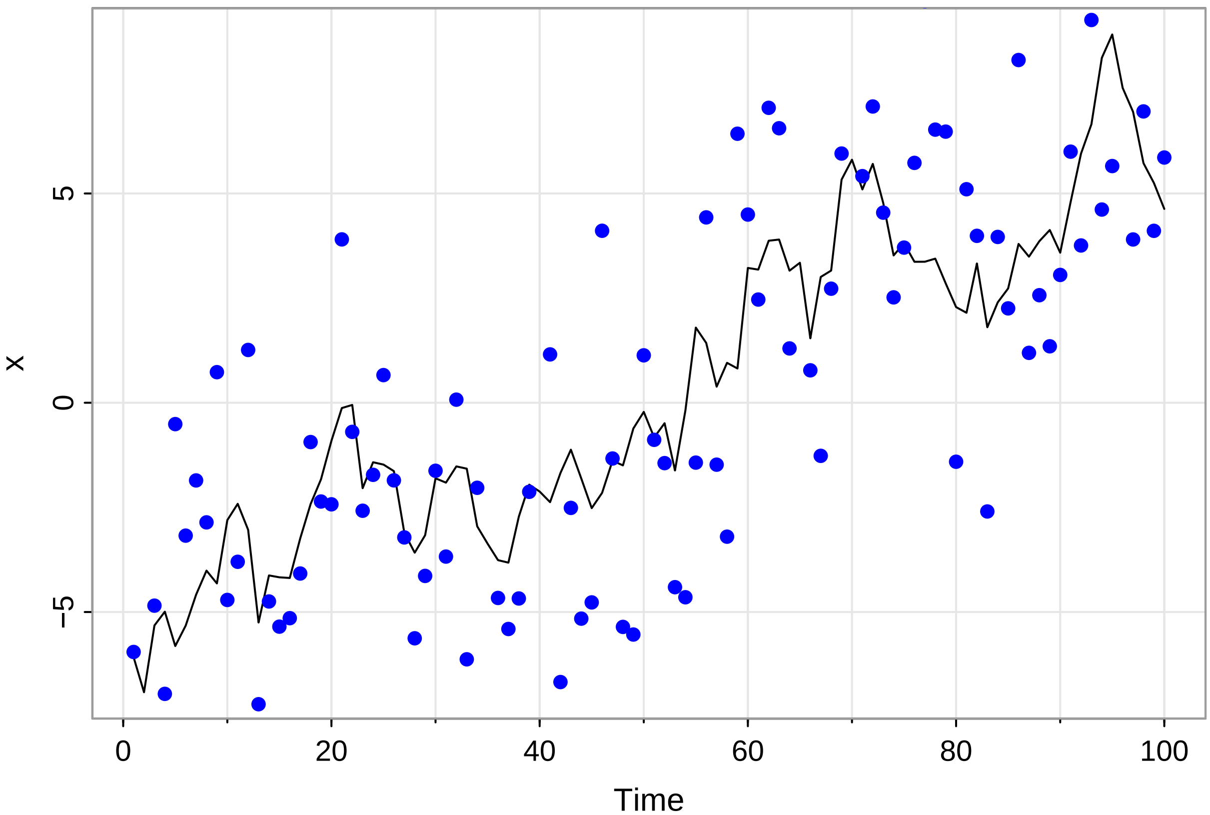
```
##         estimate         SE
## phi    0.8106963 0.09836856
## sigw   0.8804634 0.23235380
## sigv   0.9329670 0.17421057
## mu0    0.7842457         NA
## Sigma0 0.1469216         NA
```

Bayesian Estimation Local Level Model

This example is taken from Shumway and Stoffer (2017) example 6.7

Generate data

```
set.seed(1)
sQ = 1; sR = 3; n = 100
mu0 = 0; Sigma0 = 10; x0 = rnorm(1, mu0, Sigma0)
w = rnorm(n); v = rnorm(n)
x = c(x0 + sQ * w[1])  # initialize states
y = c(x[1] + sR * v[1])  # initialize obs
for (t in 2:n){
  x[t] = x[t - 1] + sQ*w[t]
  y[t] = x[t] + sR * v[t]
}

tsplot(x, pch = 16)
points(1:100, y, pch = 16, col = "blue")
```



Set up the Gibbs sampler

```
burn = 50; n.iter = 1000
niter = burn + n.iter
draws = c()
# priors for R (a,b) and Q (c,d) IG distributions
a = 2; b = 2; c = 2; d = 1
# (1) initialize - sample sQ and sR
sR = sqrt(1/rgamma(1, a, b)); sQ = sqrt(1/rgamma(1, c, d))
```

Progress bar

```
pb = txtProgressBar(min = 0, max = niter, initial = 0, style = 3)
```

```
##   |                                                                              |
```

```
# run it
for (iter in 1:niter){
  ## (2)  sample the states
  run = ffbs(y, 1, 0, 10, 1, sQ, sR)   # ffbs(y,A,mu0,Sigma0,Phi,Ups,Gam,sQ,sR,input)
  ## (1)  sample the parameters
  Xs = as.matrix(run$Xs)
  R = 1/rgamma(1, a + n/2, b + sum((y - Xs)^2)/2)
```

```
  sR = sqrt(R)
  Q = 1/rgamma(1,c + (n - 1)/2, d + sum(diff(Xs)^2)/2)
  sQ = sqrt(Q)
  ## store everything
  draws = rbind(draws, c(sQ, sR, Xs))
  setTxtProgressBar(pb,iter)
}
```

```
##   |                                                                              |
```

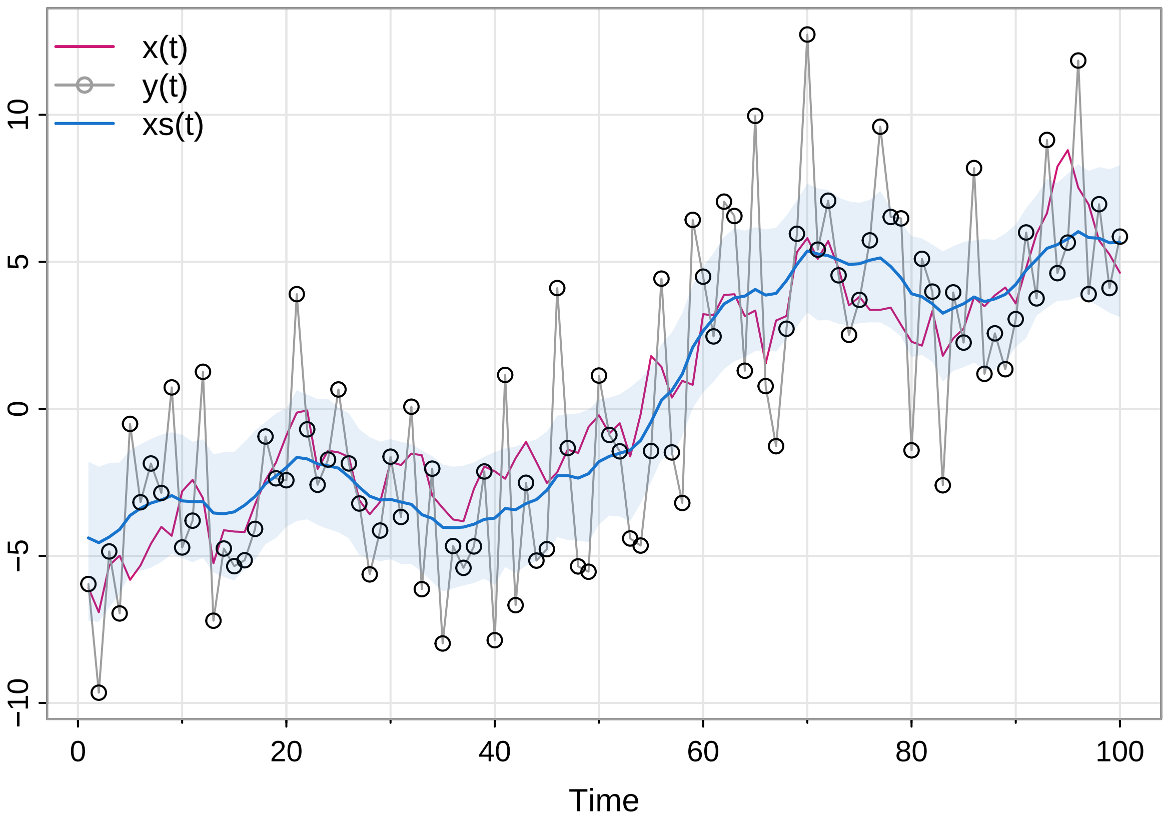
```
close(pb)
```

Pull out the results for easy plotting

```
draws = draws[(burn + 1):(niter),]
q025 = function(x){quantile(x, 0.025)}
q975 = function(x){quantile(x, 0.975)}
xs = draws[, 3:(n + 2)]
lx = apply(xs, 2, q025)
mx = apply(xs, 2, mean)
ux = apply(xs, 2, q975)
```

Plot the results

```
tsplot(cbind(x, y, mx), spag = TRUE, ylab='', col = c(6, 8, 4),
       lwd = c(1, 1, 1.5), type = 'o', pch = c(NA, 1, NA))
legend('topleft', legend = c("x(t)", "y(t)", "xs(t)"),
       lty = 1, col = c(6, 8, 4), lwd = 1.5, bty = "n", pch = c(NA, 1, NA))
points(y)
 xx = c(1:100, 100:1)
 yy = c(lx, rev(ux))
polygon(xx, yy, border = NA, col = astsa.col(4, .1))
```



References

Shumway, Robert H, and David S Stoffer. 2017. *Time Series Analysis and Its Applications.* 4th ed. Springer.