MATH 8090: State-Space Models

Whitney Huang, Clemson University

11/16-23/2021

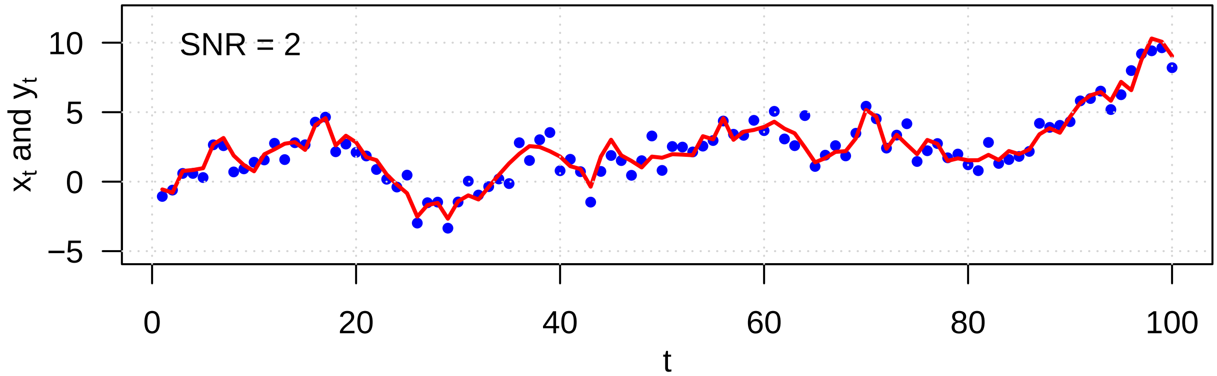
Contents

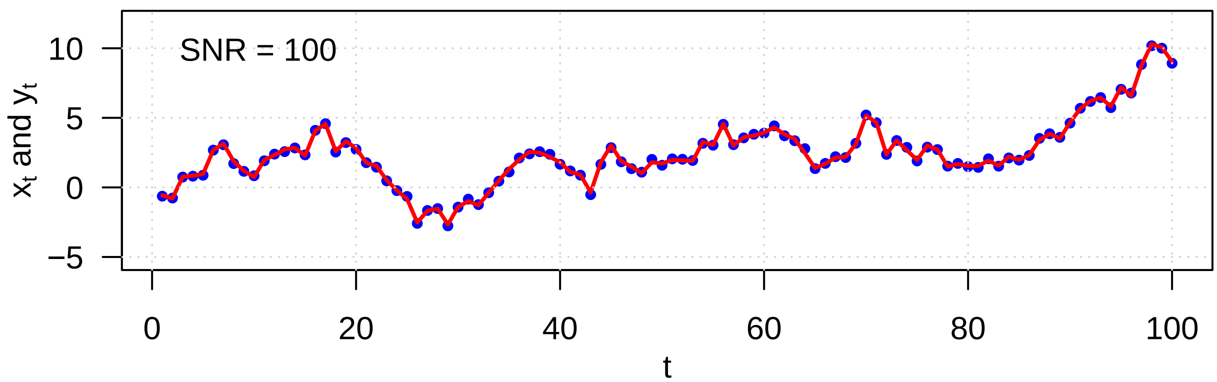
Simulate data from local level model 1
Function for carrying out Kalman filter (adapted from Dr. Donald B. Percival's UW Stat 519 R codes) 2
Kalman filter: forecasting 3
Kalman filter: missing values imputation (adapted from Dr. Donald B. Percival's UW Stat 519 R codes) 5
  Load the Nile rvier flows 5
  Impute the missing values 6
  Construct confidence interval 7
Kalman smoothing (adapted from Dr. Donald B. Percival's UW Stat 519 R codes) 8
Kalman smoothing: simulated example 9
R codes from Shumway & Stoffer 10
Parameter estimation 12
  Simulated example 12
  Global temperature example 13
EM algorithm example 17
References 19

Simulate data from local level model

```
set.seed(123)
m.1 <- 0; P.1 <- 1
sig2.V <- 1
X.1 <- rnorm(1, mean = m.1, sd = sqrt(P.1))
X <- cumsum(c(X.1, 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)

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")
```





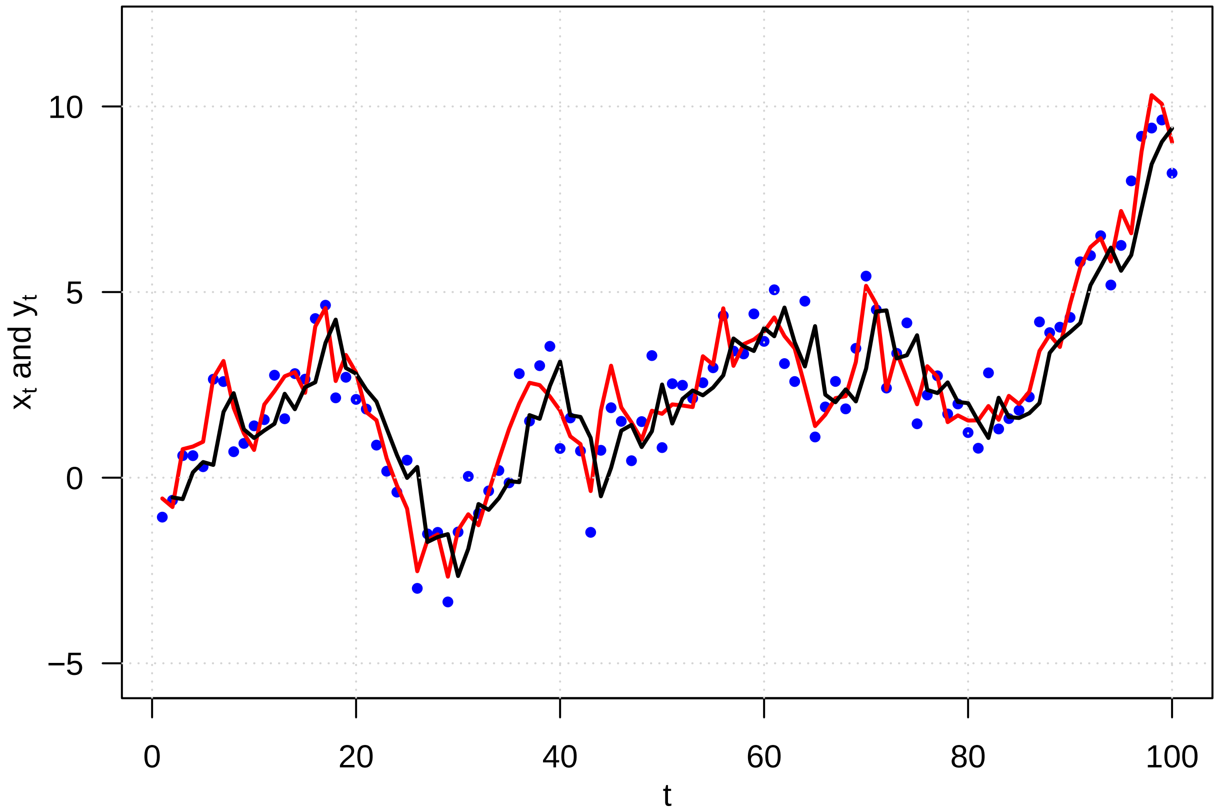
Function for carrying out Kalman filter (adapted from Dr. Donald B. Percival's UW Stat 519 R codes)

```
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))
}
```

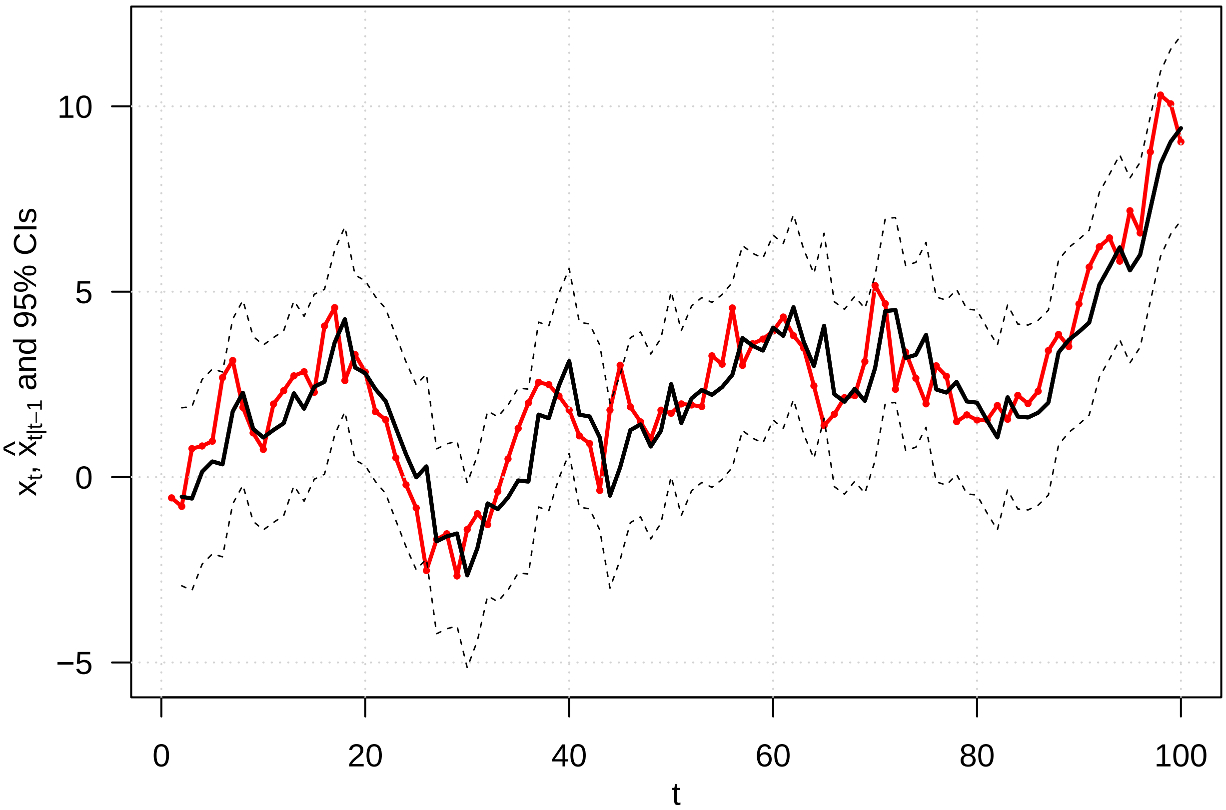
Kalman filter: forecasting

```
Y.2.KF <- KF.n.steps.local.level(Y.2)

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

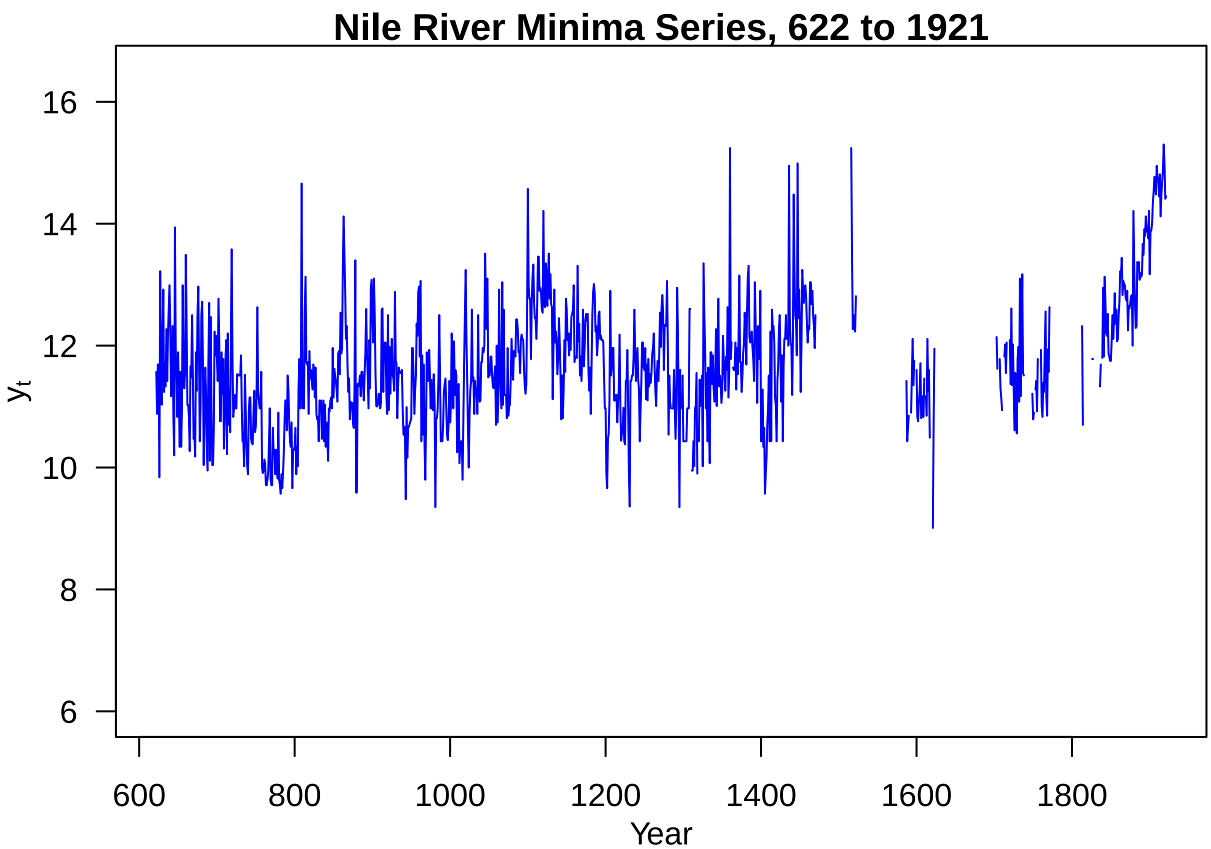
```
par(las = 1, mgp = c(2, 1, 0), mar = c(3.5, 3.5, 1, 0.6))
plot(1:100, X, col = "red", pch = 16, cex = 0.5, xlab = "t",
     ylab = expression(paste(x[t], ", ", hat(x)[paste(t, "|", t-1)]," and 95% CIs")),
     main = "", ylim = c(-5.25, 12))
lines(X, col = "red", lwd = 2)
lines(2:100, Y.2.KF$forecast.ts[-1],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



Kalman filter: missing values imputation (adapted from Dr. Donald B. Percival's UW Stat 519 R codes)

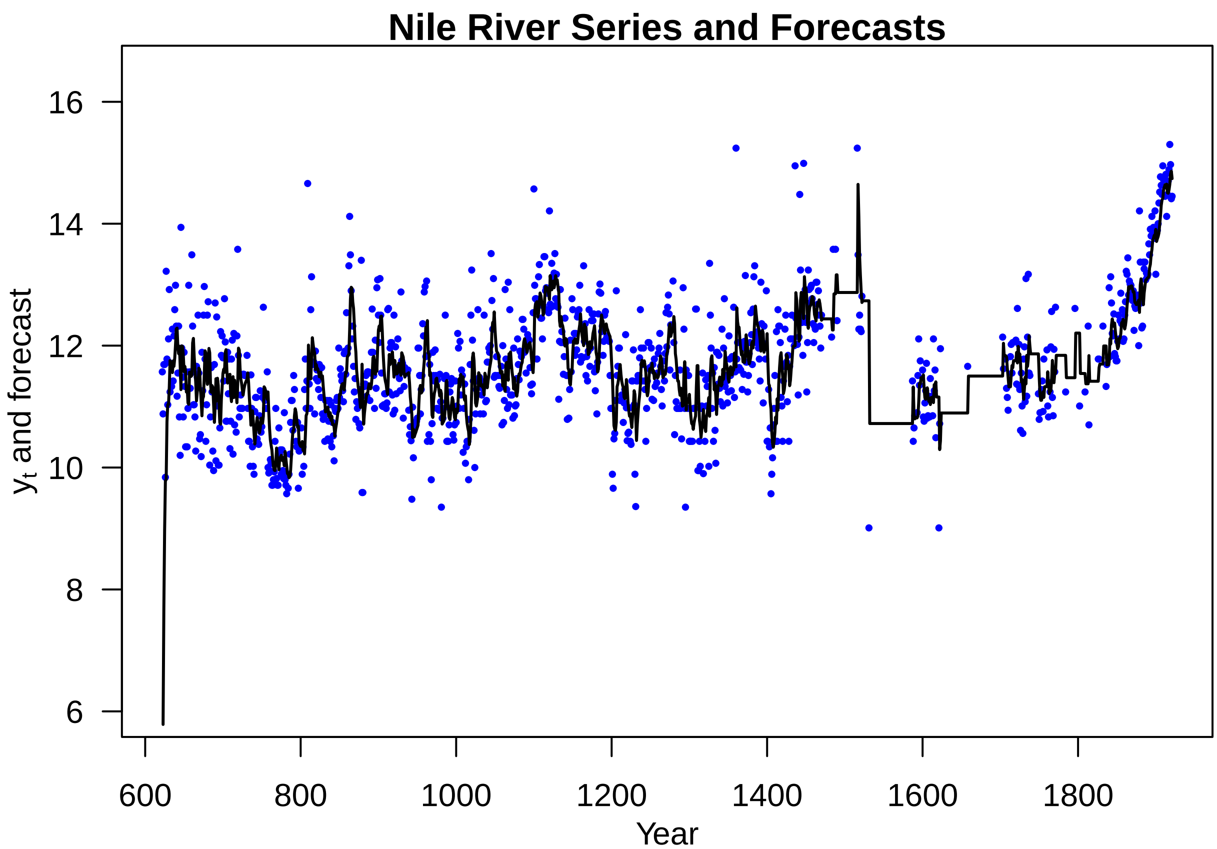
Load the Nile rvier flows

```
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")
```



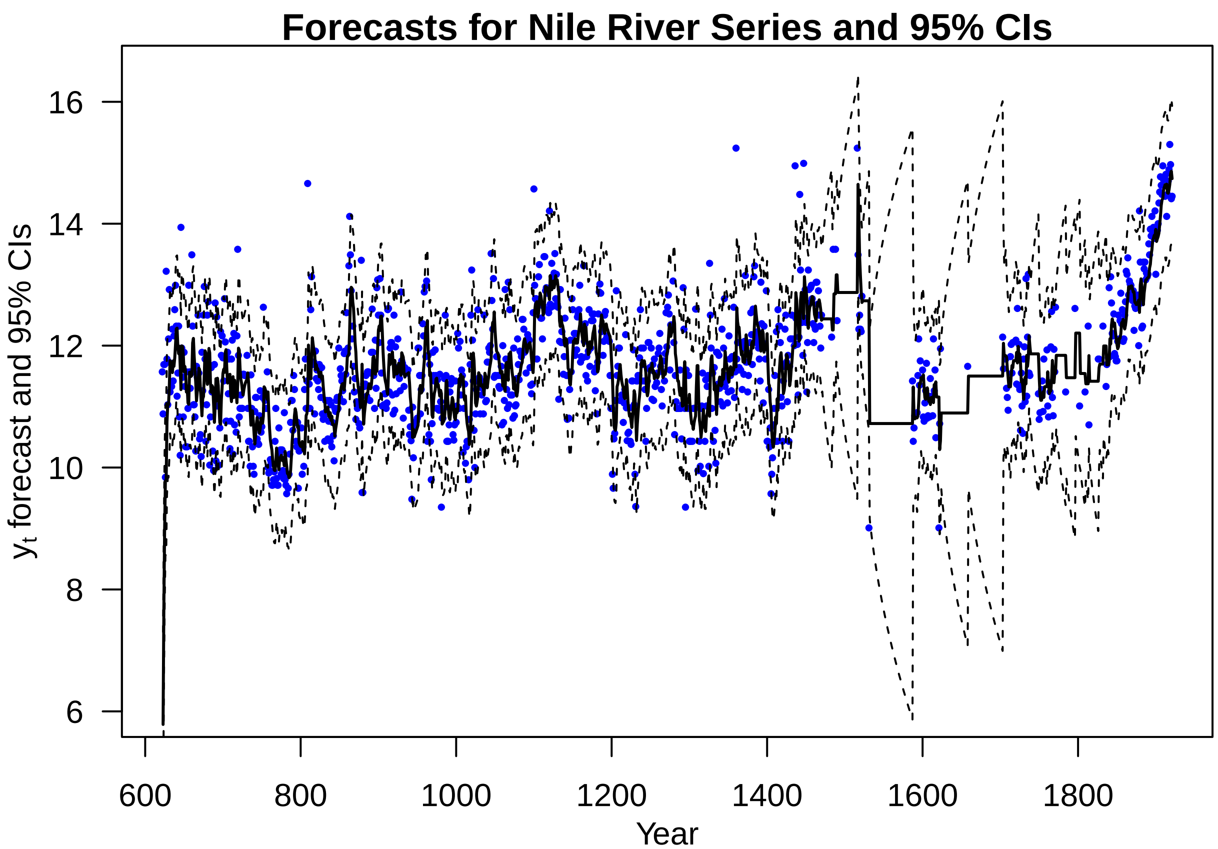
Impute the missing values

```
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)
```



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)
```

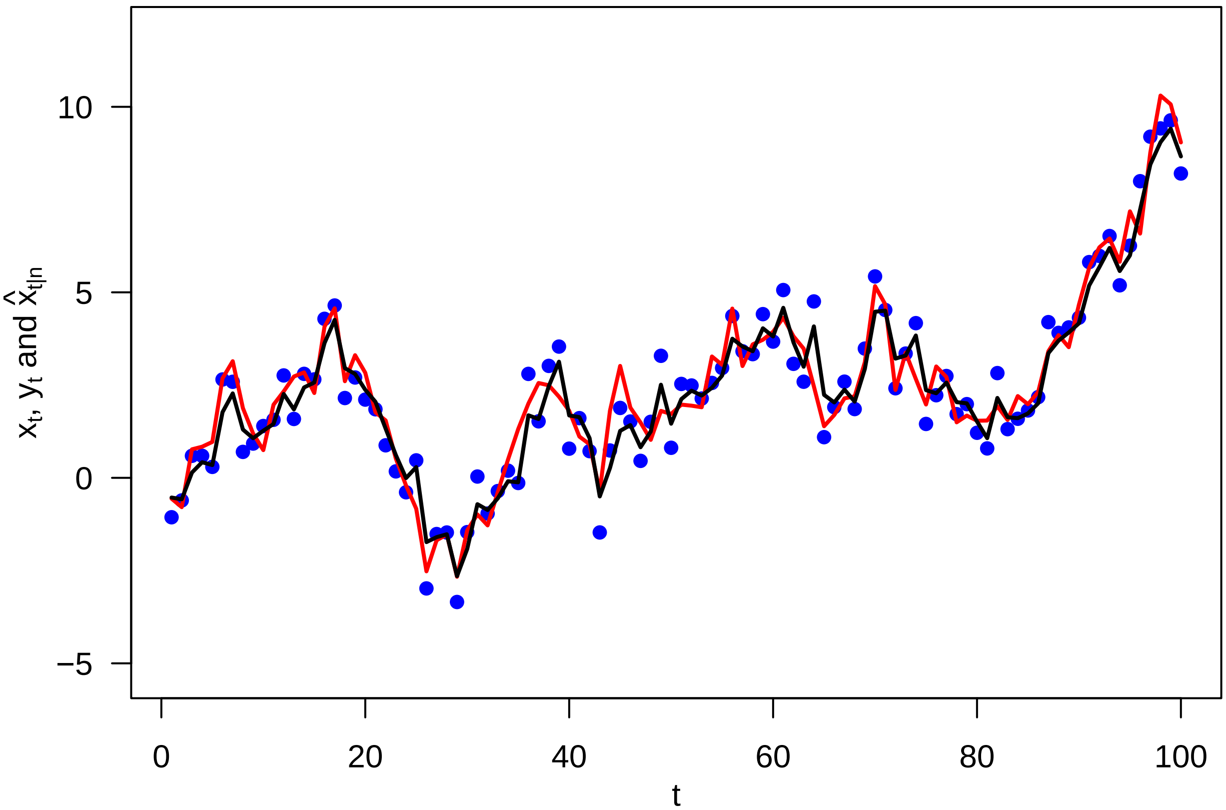


Kalman smoothing (adapted from Dr. Donald B. Percival's UW Stat 519 R codes)

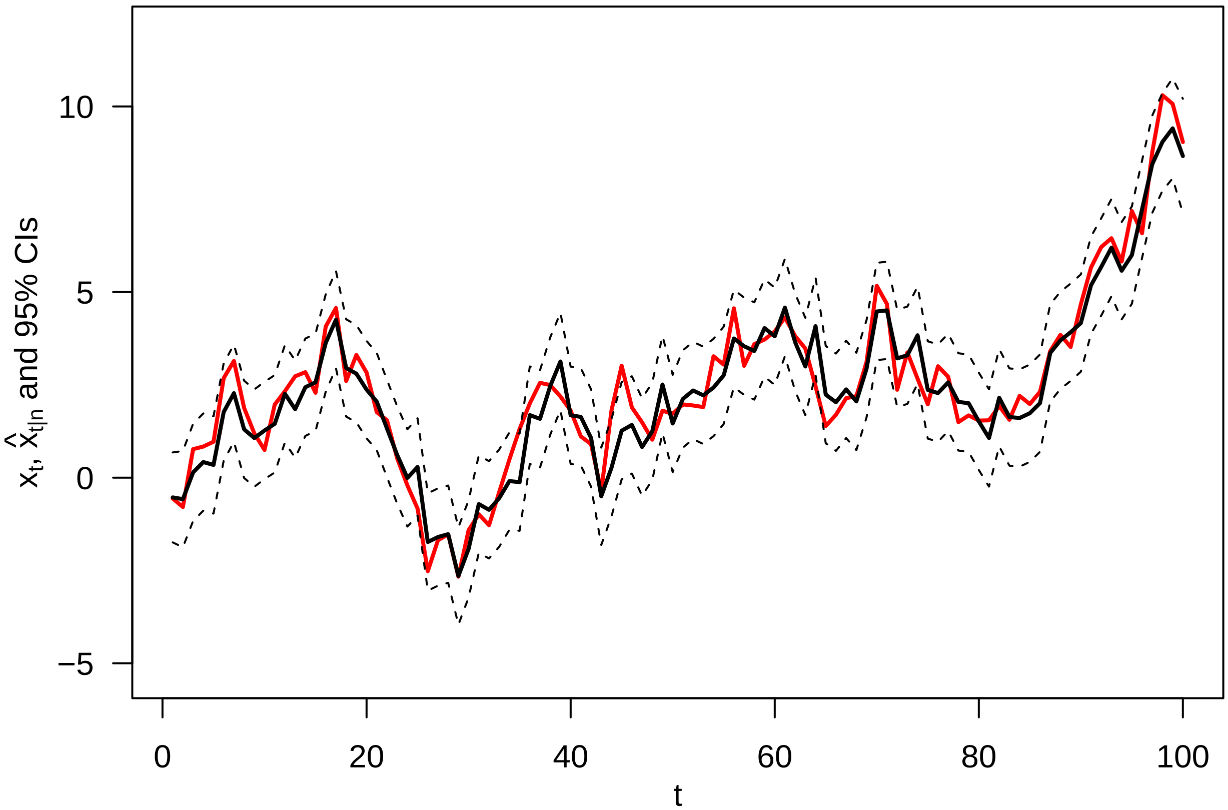
```
KS.local.level <- function(KF.results){
  n <- length(KF.results$filter.ts)
  L.t.ts <- 1 - KF.results$gain.ts
  r.ts <- rep(0, n + 1)
  bg <- is.na(KF.results$innovations.ts)
  innov.0.for.NA <- KF.results$innovations.ts
  innov.0.for.NA[bg] <- 0
  for(t in n:1) r.ts[t] <- innov.0.for.NA[t] / KF.results$innovations.var.ts[t]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



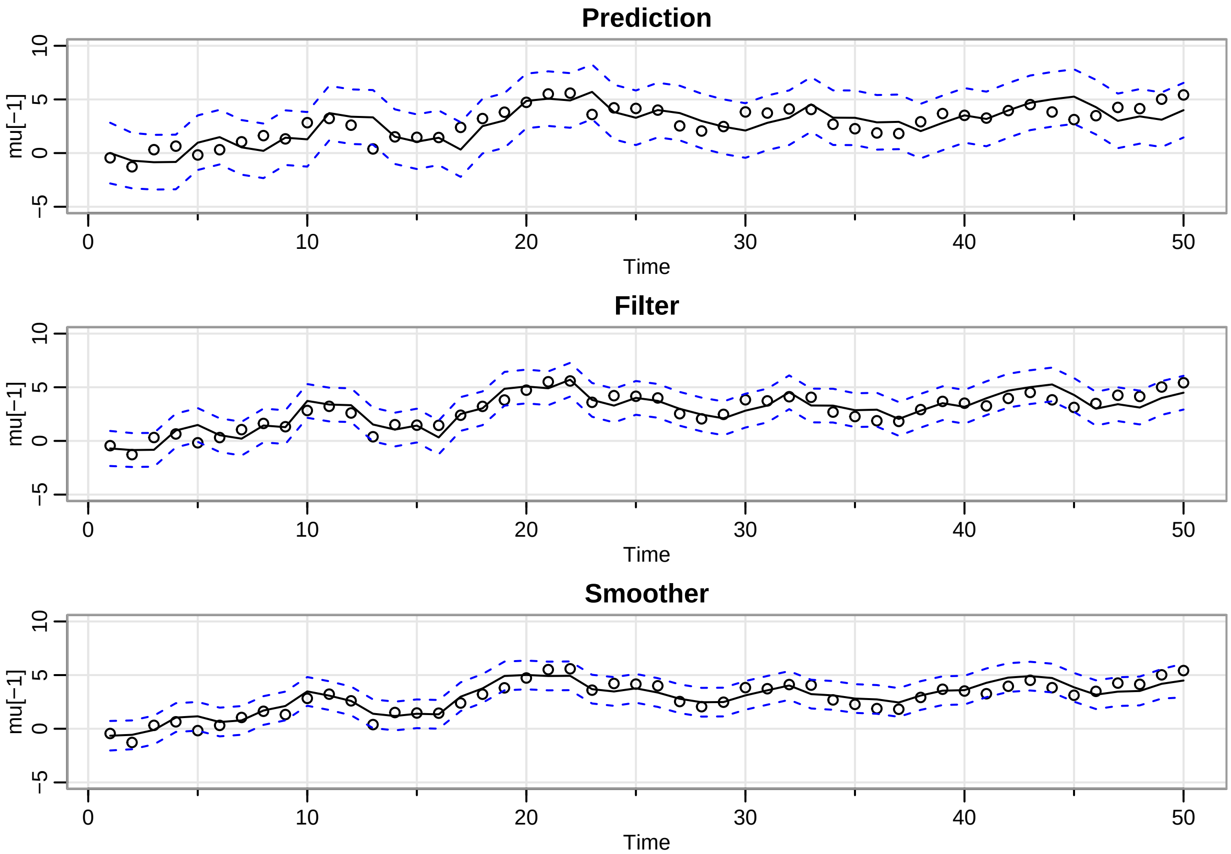

```
par(las = 1, mgp = c(2, 1, 0), mar = c(3.5, 3.5, 1, 0.6))
plot(X, col = "red", typ = "l", lwd = 2, xlab = "t",
     ylab = expression(paste(x[t],", ",hat(x)[paste(t, "|", n)]," and 95% CIs")),
     main = "", ylim = c(-5.25, 12))
lines(Y.2.KS$smooth.ts, lwd = 2)
lines(Y.2.KS$smooth.ts - 1.96 * sqrt(Y.2.KS$smooth.var.ts),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



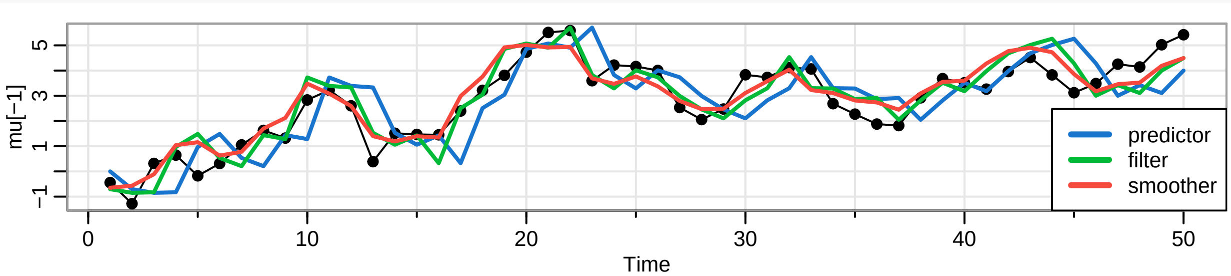
R codes from Shumway & Stoffer

```
# generate data
library(astsa)
set.seed(1)
num = 50
w = rnorm(num + 1, 0, 1)
v = rnorm(num, 0, 1)

mu = cumsum(w)
y = mu[-1] + v

# filter and smooth (Ksmooth0 does both)
mu0 = 0; sigma0 = 1;  phi = 1; cQ = 1; cR = 1
ks = Ksmooth0(num, y, 1, mu0, sigma0, phi, cQ, cR)

# pictures
par(mfrow = c(3, 1), mgp = c(2, 1, 0), mar = c(3.5, 3.5, 1, 0.5))
Time = 1:num

tsplot(Time, mu[-1], type = 'p', main = "Prediction", ylim=c(-5, 10))
lines(ks$xp)
lines(ks$xp + 2 * sqrt(ks$Pp), lty = "dashed", col = "blue")
lines(ks$xp - 2 * sqrt(ks$Pp), lty = "dashed", col = "blue")

tsplot(Time, mu[-1], type = 'p', main = "Filter", ylim=c(-5, 10))
```

```
lines(ks$xf)
lines(ks$xf + 2 * sqrt(ks$Pf), lty = "dashed", col =" blue")
lines(ks$xf - 2 * sqrt(ks$Pf), lty = "dashed", col = "blue")

tsplot(Time, mu[-1], type = 'p',  main = "Smoother", ylim = c(-5, 10))
lines(ks$xs)
lines(ks$xs + 2 * sqrt(ks$Ps), lty = "dashed", col = "blue")
lines(ks$xs - 2 * sqrt(ks$Ps), lty = "dashed", col = "blue")
```




```
tsplot(Time, mu[-1], type = 'o', pch = 19, cex = 1)
lines(ks$xp, col = 4, lwd = 2)
lines(ks$xf, col = 3, lwd = 2)
lines(ks$xs, col = 2, lwd = 2)
names = c("predictor","filter","smoother")
legend("bottomright", names, col = 4:2, lwd = 3, lty = 1, bg = "white")
```



Parameter estimation

Simulated example

```r
# Generate Data
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
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)
(init.par = c(phi, sqrt(q), sqrt(r)))
```

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

```r
# Function to evaluate the likelihood
Linn <- function(para){
  phi <- para[1]; sigw <- para[2]; sigv <- para[3]
  Sigma0 <- (sigw^2) / (1 - phi^2); Sigma0[Sigma0 < 0] = 0
  kf = Kfilter0(num, y, 1, mu0 = 0, Sigma0, phi, sigw, sigv)
  return(kf$like)
  }

# Estimation
(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
```

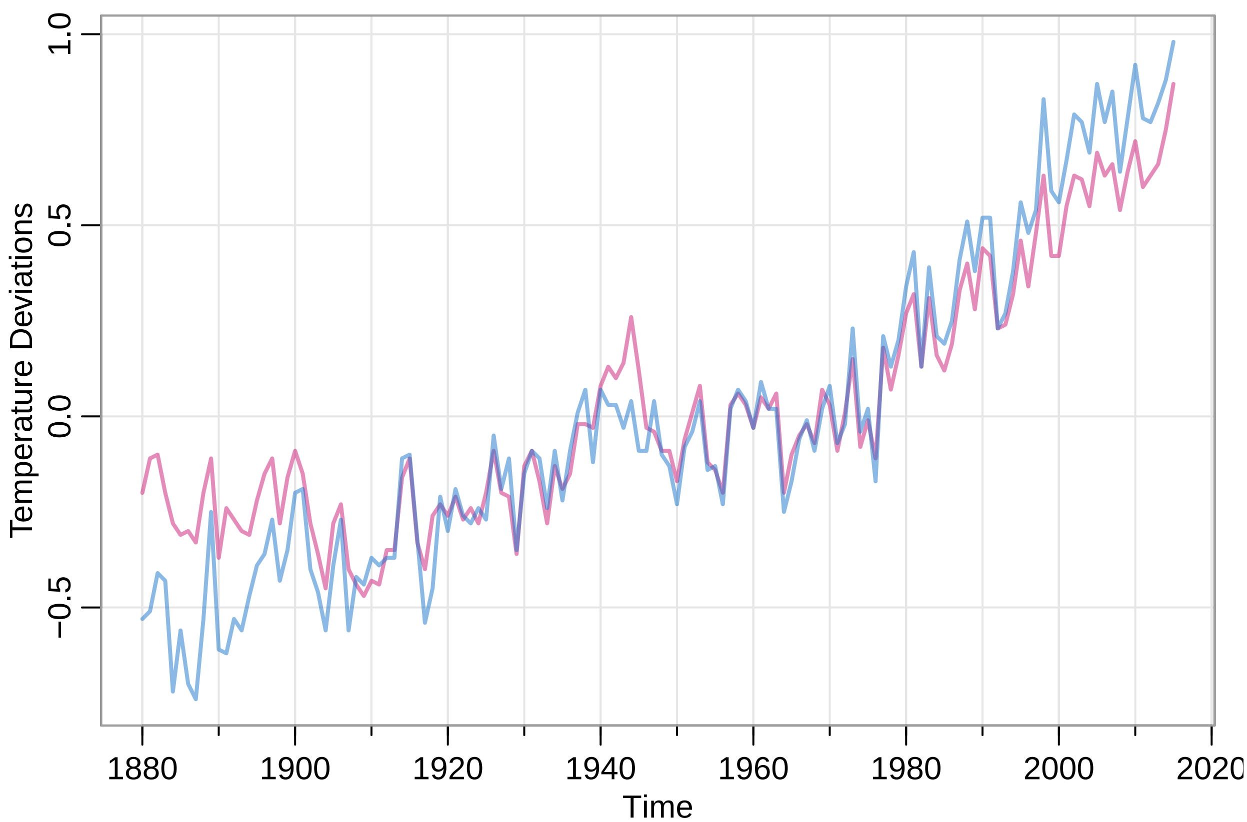
```
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## $hessian
##             [,1]     [,2]      [,3]
## [1,] 263.738652 74.14214 -9.936399
## [2,]  74.142142 69.77014 44.355806
## [3,]  -9.936399 44.35581 85.616367
```

```
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
```

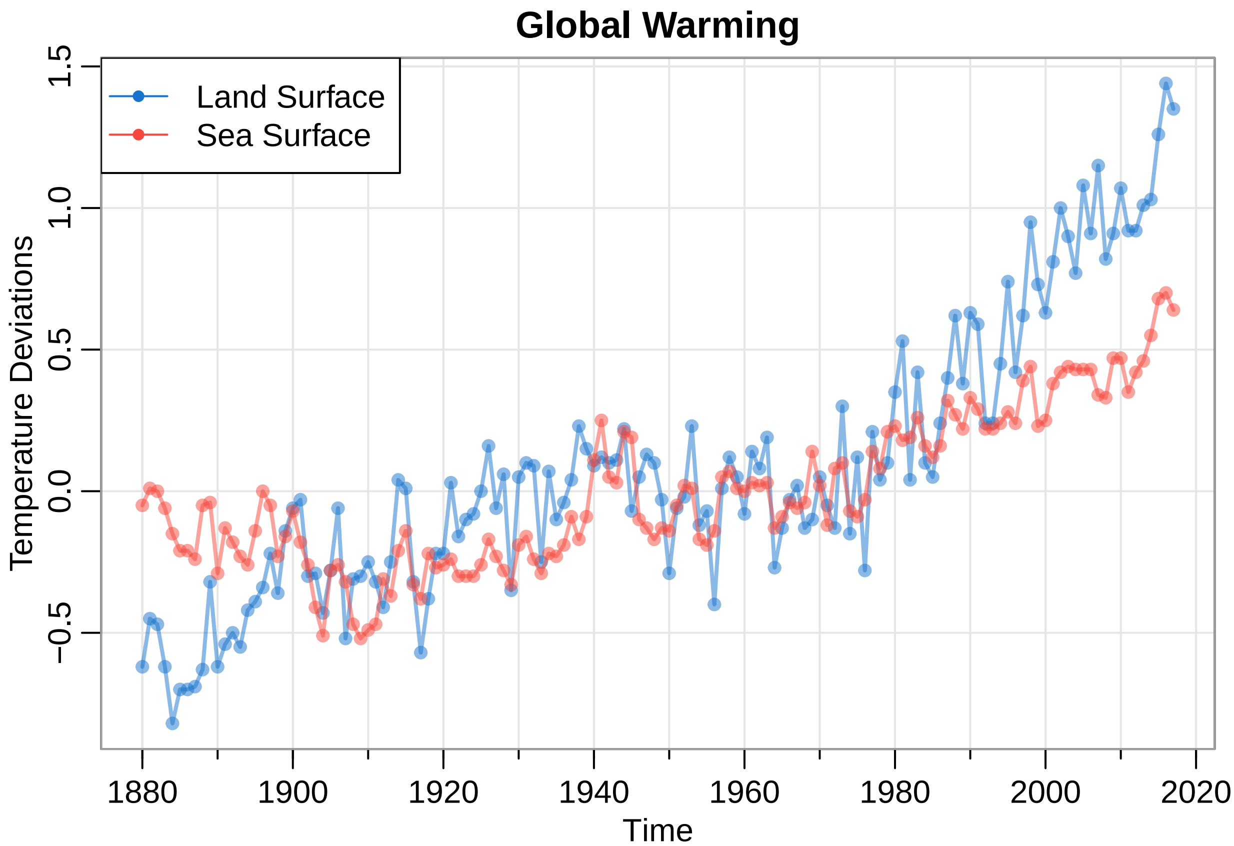
Global temperature example

```
tsplot(cbind(globtemp, globtempl), spag = TRUE, lwd = 2, col = astsa.col(c(6, 4), .5),
       ylab = "Temperature Deviations")
```




```
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")
```




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

# Function to Calculate Likelihood
Linn <- function(para){
  cQ = para[1]     # sigma_w
  cR1 = para[2]    # 11 element of chol(R)
  cR2 = para[3]    # 22 element of chol(R)
  cR12 = para[4]   # 12 element of chol(R)
 cR = matrix(c(cR1, 0, cR12, cR2), 2)  # put the matrix together
 drift = para[5]
 kf = Kfilter1(num, y, A, mu0, Sigma0, Phi, drift, 0, cQ, cR, input)
 return(kf$like)
 }

# 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
```

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

# Summary of estimation
estimate <- est$par; u <- cbind(estimate, SE)
rownames(u) <- c("sigw","cR11", "cR22", "cR12", "drift")
u
```

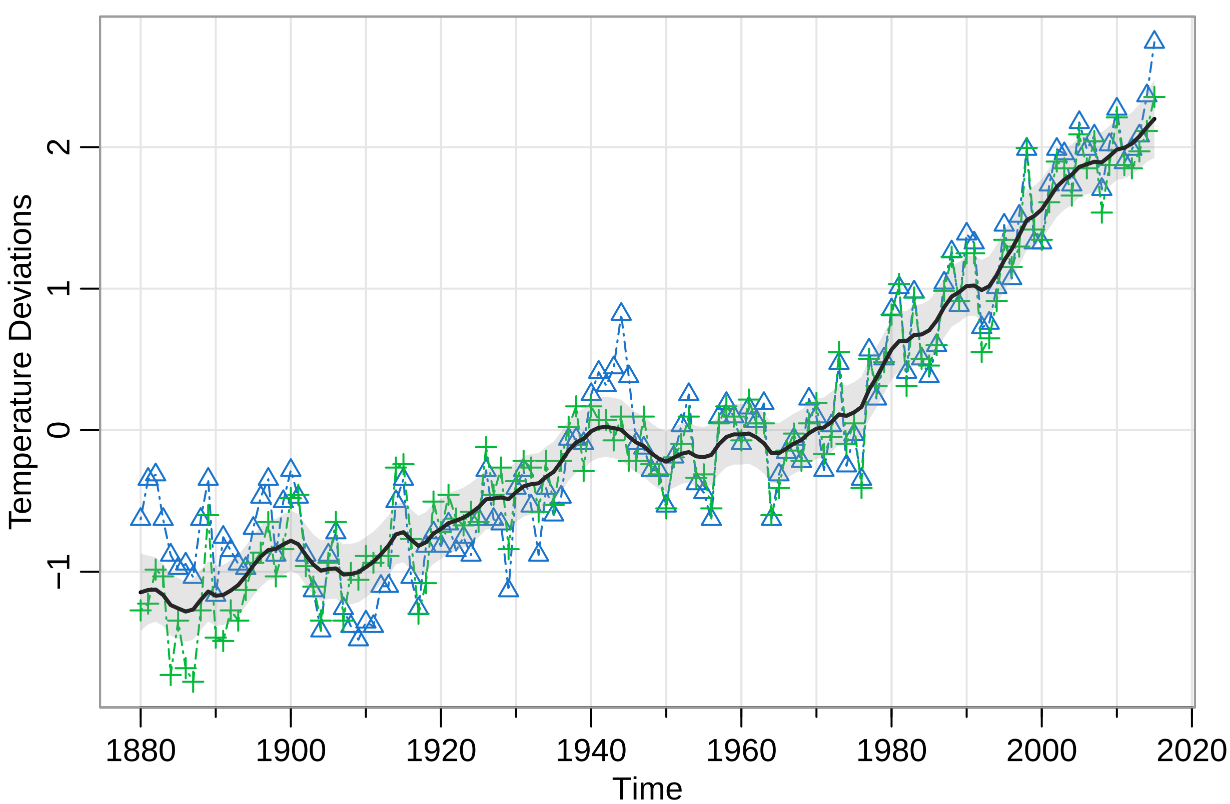
```
##          estimate          SE
```

```
## sigw  0.09461713 0.025974347
## cR11  0.32401331 0.038005698
## cR22  0.20283345 0.019122980
## cR12  0.14761763 0.029219285
## drift 0.02472785 0.008292723
```

```
# Smooth (first set parameters to their final estimates)
cQ = est$par[1]
cR1 = est$par[2]
cR2 = est$par[3]
cR12 = est$par[4]
cR = matrix(c(cR1, 0, cR12, cR2), 2)
(R = t(cR) %*% cR)     #  to view the estimated R matrix
```

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

```
drift = est$par[5]
ks = Ksmooth1(num, y, A, mu0, Sigma0, Phi, drift, 0, cQ, cR, input)

# Plot
tsplot(y, spag = TRUE, margins = .5, type = 'o', pch = 2:3, col = 4:3, lty = 6,
       ylab='Temperature Deviations')
xsm  = ts(as.vector(ks$xs), start = 1880)
rmse = ts(sqrt(as.vector(ks$Ps)), start = 1880)
lines(xsm, lwd = 2)
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

```
library(nlme)
# Generate data (same as Example 6.6)
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
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 = EM0(num, y, 1, mu0, Sigma0, phi, cq, cr, 75, .00001))
```

```
## iteration    -loglikelihood
##     1          84.20423
##     2          83.9686
##     3          83.8639
##     4          83.80029
##     5          83.75467
##     6          83.71866
##     7          83.68861
##     8          83.66273
##     9          83.64001
##     10          83.61983
##     11          83.60178
##     12          83.58556
##     13          83.57094
##     14          83.55774
##     15          83.54581
##     16          83.53501
##     17          83.52525
##     18          83.5164
##     19          83.5084
##     20          83.50116
##     21          83.4946
##     22          83.48867
##     23          83.4833
##     24          83.47843
##     25          83.47403
##     26          83.47005
##     27          83.46644
##     28          83.46317
##     29          83.4602
##     30          83.45752
##     31          83.45508
##     32          83.45287
##     33          83.45086
##     34          83.44904
##     35          83.44738
##     36          83.44587
```

```
##      37           83.44449
##      38           83.44324
##      39           83.44209
##      40           83.44105
##      41           83.44009
##      42           83.43921
##      43           83.43841
```

```
## $Phi
##            [,1]
## [1,] 0.8277147
##
## $Q
##            [,1]
## [1,] 0.6888479
##
## $R
##            [,1]
## [1,] 0.9347841
##
## $mu0
##            [,1]
## [1,] 0.8163222
##
## $Sigma0
##             [,1]
## [1,] 0.04999319
##
## $like
##  [1] 84.20423 83.96860 83.86390 83.80029 83.75467 83.71866 83.68861 83.66273
##  [9] 83.64001 83.61983 83.60178 83.58556 83.57094 83.55774 83.54581 83.53501
## [17] 83.52525 83.51640 83.50840 83.50116 83.49460 83.48867 83.48330 83.47843
## [25] 83.47403 83.47005 83.46644 83.46317 83.46020 83.45752 83.45508 83.45287
## [33] 83.45086 83.44904 83.44738 83.44587 83.44449 83.44324 83.44209 83.44105
## [41] 83.44009 83.43921 83.43841
##
## $niter
## [1] 43
##
## $cvg
## [1] 9.650115e-06
```

```r
# Standard Errors  (this uses nlme)
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 = Kfilter0(num, 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)))
```

```r
# 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    0.82771470 0.08935588
## sigw   0.82996860 0.21178833
## sigv   0.96684231 0.15874062
## mu0    0.81632224         NA
## Sigma0 0.04999319         NA
```

References