Nile Data

Andrea Bianco Egle Caudullo Giovanni Contento

Contents

1 Libraries

```
library(TSA)
library(dlm)
library(statespacer)
library(graphics)
library(stats)
library(kableExtra)
library(dplyr)
library(ggplot2)
library(hrbrthemes)
library(float)
library(tidyverse)
```

2 The Local level model

The Local Level Model is an Unobserved Component Model that is already cast in SSF.

It has a constantly changing mean following a random walk model: which models the observed time series as a mean plus a random measurement error or disturbance . The mean evolves over time as a random walk driven by innovations.

The well known model formulation is:

$$\begin{cases} y_t = \alpha_t + \varepsilon_t, \quad \varepsilon_t \sim \mathcal{NID}(0, \sigma_\varepsilon^2) \\ \alpha_{t+1} = \alpha_t + \eta_t, \quad \eta_t \sim \mathcal{NID}(0, \sigma_\eta^2) \end{cases}$$

Where t is the time index and n is time series length.

The first equation ($y_t = \alpha_t + \varepsilon_t$) is called ***observation/measurement equation*** and allows us to specify assumptions about the behavior of the observed data relative to the unobserved components contained in the state vector α_t

While the second one ($\alpha_t = \alpha_{t+1} + \eta_t$) is known as the ***state/transition equation*** and it describes the temporal evolution of the unobserved state vector α_{t+1}

The state disturbances η_t , and the observations disturbances ε_t are IID Gaussian random variables both with zero means and respectively constant variances σ_η^2, σ_ε^2. They are mutually independent for each $t, s \in \mathbb{Z}$ and independent of the initial state α_1.

2.1 Model construction

The model construction requires an assumption regarding the initial state α_1:

- The distribution of the initial state could be assumed as known and Gaussian with a_1 , P_1 known and finite: $\alpha_1 \sim \mathcal{N}(a_1, P_1)$;
- The distribution of the initial state α_1 could be unknown. In this case the choice is to represent α_1 as having a ***diffuse prior density***, which means that a_1 is fixed at an arbitrary value and $P_1 \to \infty$.

Fitting the model to the data requires specifying the constant variances of the ***state disturbance*** σ_η^2 and the ***observation disturbance*** σ_ε^2 , which are estimated via maximum likelihood.

Although the diffuse initialization $P_1 \to \infty$ leads to a different form of the *log -likelihood* function, the maximum likelihood estimates of σ_ε^2 and σ_η^2 obtained under this specification converge to the same values as those derived from the formulation assuming a known initial distribution for α_1, as will be shown below. (lo mostriamo davvero?)

3 The Data Set

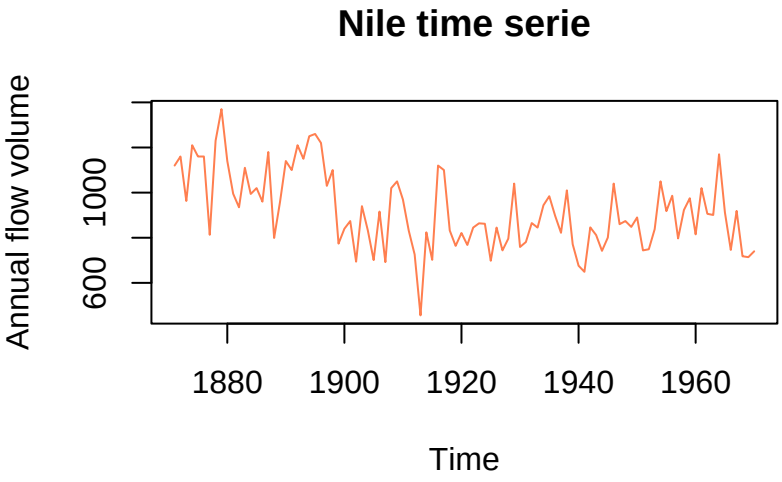
The data set under analysis collects 100 measurements of the ***annual flow volume of the Nile river***, at the Aswan city, from the year ***1871*** to **1970**.

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   456.0   798.5   893.5   919.4  1032.5  1370.0
```

```
## [1] 169.23
```

Some descriptive statistics of our data: the series has a mean flow of 919.35 and a standard deviation of 169.23. Furthermore, the minimum recorded flow is 456, while the maximum reaches 1370.

Figure 1 presents a time series plot of the Nile data set.



A simple visual inspection of the series reveals several notable features. Most prominently, an unusually low flow is observed shortly after 1910, while an exceptionally high flow occurs around 1880.

4 Fitting a Local Level Model

Fitting a Local Level Model requires finding the best representation of the our data under the assumption that the underlying mean evolves gradually over time, while the observations are subject to measurement noise.
The steps needed to fit the model are:

- Estimating σ_ε^2 and σ_η^2
- Inferring the latent level component α_t using methods like ***Kalman Filter*** (or ***Kalman Smoothing***)
- Maximizing the likelihood of the observed data with respect to the model parameters

4.1 Kalman Filter algorithm

In order to fit the model it is needed the *filtered estimator* of the state α_t and α_{t+1}.

The filtering procedure,implemented via the Kalman algorithm, as described by *Durbin and Koopman* (Chapter 2, Section 2.2), requires:

- The assumption that the disturbances are normally distributed so that the conditional joint distribution of the states given a set of observations is normal.

- Y_{t-1} defined as the time series $(y_1, \ldots, y_{t-1})$ for $t = 2, 3, \ldots$
- The conditional distribution of the state at time t given the past observation is defined as follows: $\alpha_t | Y_{t-1} \sim \mathcal{N}(a_t, P_t)$ where a_t and P_t are assumed known.
- the one-step ahead prediction errors v_t of y_t, computed in this way: $v_t = y_t - a_t$.

The primary objective is to determine (real time) updated $a_{t|t}$ and the associated variance $P_{t|t}$, respectively, the recursive estimator of the state α_t and the state variance P_t.

To do so it is required the derivation of the distribution of α_t given the current observation rather than only the past ones. It is obtained after computation that

$$p(\alpha_t | Y_t) = \mathcal{N}(a_t + \frac{P_t v_t}{P_t + \sigma_\varepsilon^2} \quad , \frac{P_t \sigma_\varepsilon^2}{P_t + \sigma_\varepsilon^2})$$

Then, defining F_t (***Variance of the Prediction Error***)$v_t$and K_t (***Kalman Gain***) as follows:

$$F_t = \text{Var}(v_t \mid Y_{t-1}) = P_t + \sigma_\varepsilon^2$$

$$K_t = \frac{P_t}{F_t}$$

We can write the full set of relations for updating from time t to time $t + 1$.

$$v_t = y_t - a_t, \quad F_t = P_t + \sigma_\varepsilon^2$$

$$a_{t|t} = a_t + K_t v_t, \quad P_{t|t} = P_t(1 - K_t)$$

$$a_{t+1} = a_t + K_t, \quad P_{t+1} = P_t(1 - K_t) + \sigma_\eta^2$$

This set of relations constitute the ***Kalman Filter for the LLM***.

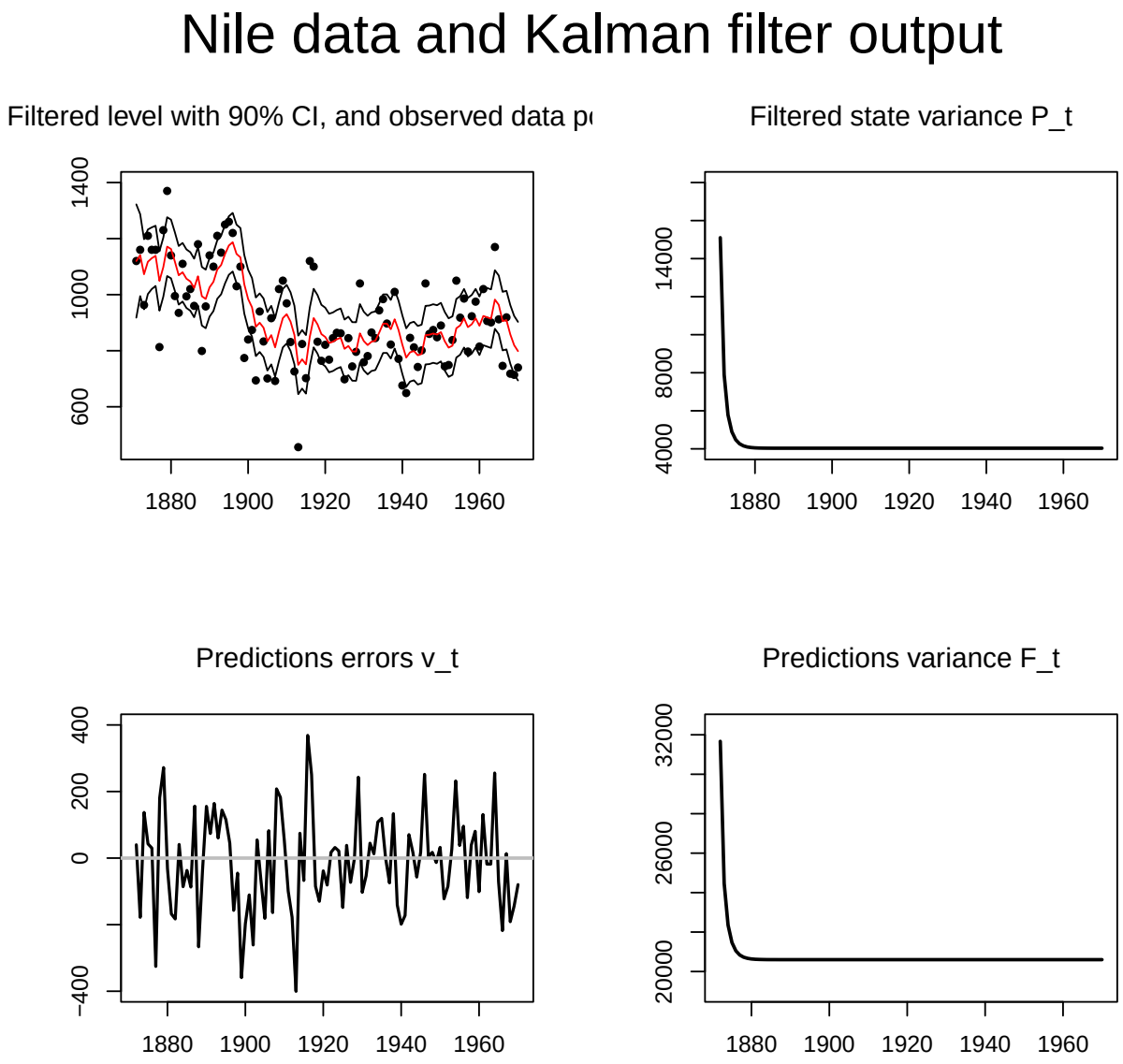
4.1.1 Fitting using statespacer

For the model fitting in R we chose to use the `statespacer()` function from the `statespacer` package.

```
fit <- statespacer(Nile_mat ,
                   local_level_ind = TRUE ,
                   diagnostics = TRUE ,
                   initial = 0.5*log(var(Nile_mat)))
```

- Setting `local_level_ind = TRUE` fit a LLM to the data.
- `diagnostic = TRUE` allows us to compute quantities related to the diagnostic analysis.
- The choice of setting `initial = 0.5 * log(var(Nile_mat))` is based on Chapter 7 of *Durbin and Koopman*.
  - There the variance parameters are reparameterized as $\psi = \frac{1}{2} log(\sigma^2)$ to enable unconstrained optimization. This transformation removes the need to enforce non-negativity constraints on variances.
  - The factor `0.5 *` reflects the assumption that, in a local level model with two variances to estimate (σ_ε^2 ,σ_η^2), it is sensible to begin the optimization by equally splitting the total log-variance between the two components.

4.1.2 Output of the Kalman Filter



Plot 1 and plot 3 shows the filtered states a_t compared to the observations y_t and the prediction errors v_t. plot 2 and plot 4 display the filtered states variances P_t and the prediction variances F_t, both rapidly converge to stable values. This suggests that the Kalman filter applied to our Local Level Model reaches a steady state as $n \to \infty$, with P_t converging to a positive constant $\bar{P} \approx 4031,915$, after about 10 iterations.

4.2 Smoothing

The **filtered** estimate uses **data up to time** t while the **smoothed** estimate uses **data up to time** n, the end of the time series.

4.2.1 Smoothed state

The smoothed estimate of the state uses data up to time n, the end of the time series.

And obtain

$$\alpha_t|Y_n \sim \mathcal{N}(\hat{\alpha}_t, V_t)$$

where

$$\hat{\alpha}_t = \mathbb{E}[\alpha_t|Y_n] = \mathbb{E}[\hat{\alpha}_t|Y_{t-1}, y_t, \ldots, y_n]$$

4.2.2 Smoothed disturbances

The computation of the smoothed observation and state disturbances can be done as follows:

$$\hat{\varepsilon}_t = \mathbb{E}(\varepsilon_t|Y_n) = \sigma_\varepsilon^2 \cdot u_t$$

$$\hat{\eta}_t = \mathbb{E}(\eta_t|Y_n) = \sigma_\eta^2 \cdot r_t$$

Where:

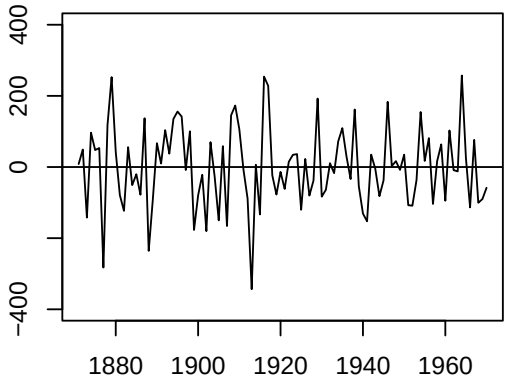
$$u_t = F_t^{-1} \cdot v_t - K_t \cdot r_t \qquad L_t = 1 - K_t = \sigma_\varepsilon^2 \cdot F_t^{-1}$$
$$\text{Var}[\varepsilon_t|Y_n] = \sigma_\varepsilon^2 - \sigma_\varepsilon^4 \cdot D_t \qquad \text{Var}[\eta_t|Y_n] = \sigma_\eta^2 - \sigma_\eta^4 \cdot N_t$$
$$r_{t-1} = v_t \cdot F_t^{-1} + L_t \cdot r_t \qquad D_t = F_t^{-1} + K_t^2 \cdot N_t$$

Although the output of `statespacer()` provides the smoothed estimates, it is preferable to demonstrate the computation of each quantity for illustrative purposes.

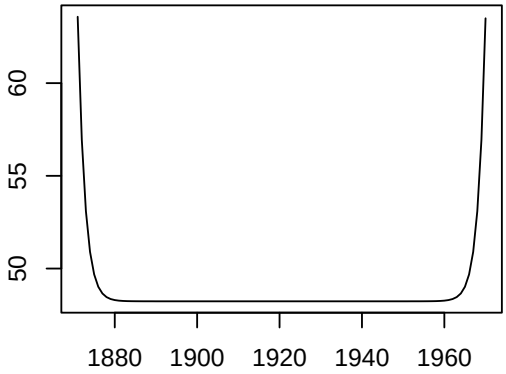
```
#Computation of Observation Error Standard Deviation
sigma2_eps <- as.vector(fit$system_matrices$H$H)
D_t <- as.vector(fit$diagnostics$D)
sigma2_eps_v <- rep(sigma2_eps, length(D_t))
df_sod <- data.frame(
  sigma2_eps = sigma2_eps_v,
  D_t = D_t
)
var_sod <- sigma2_eps_v - (sigma2_eps^2) * D_t
std_sod <- sqrt(var_sod)
#Computation of State Error Standard Deviation
sigma2_eta <- as.vector(fit$system_matrices$Q$full)
N_t <- as.vector(fit$diagnostics$N)
sigma2_eta_v <- rep(sigma2_eta, length(N_t))
df_ssd <- data.frame(
  sigma2_eta = sigma2_eta_v,
  N_t = N_t
)
var_ssd <- sigma2_eta_v - (sigma2_eta^2) * N_t
std_ssd <- sqrt(var_ssd)
```

Output of disturbance smoothing recursion

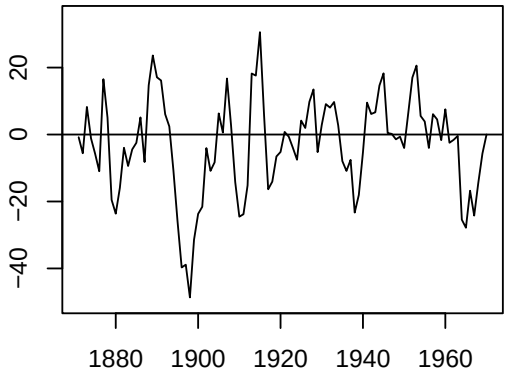
Smoothed Observation Disturbances



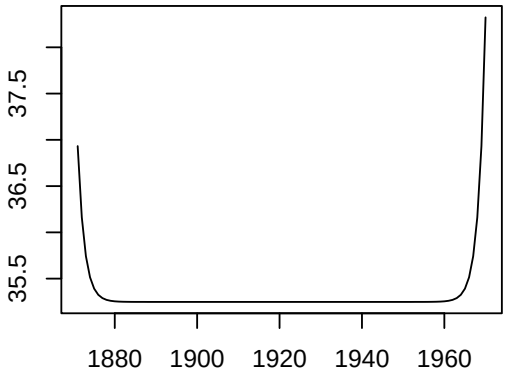
Observation Error Standard Deviation



Smoothed State Disturbances

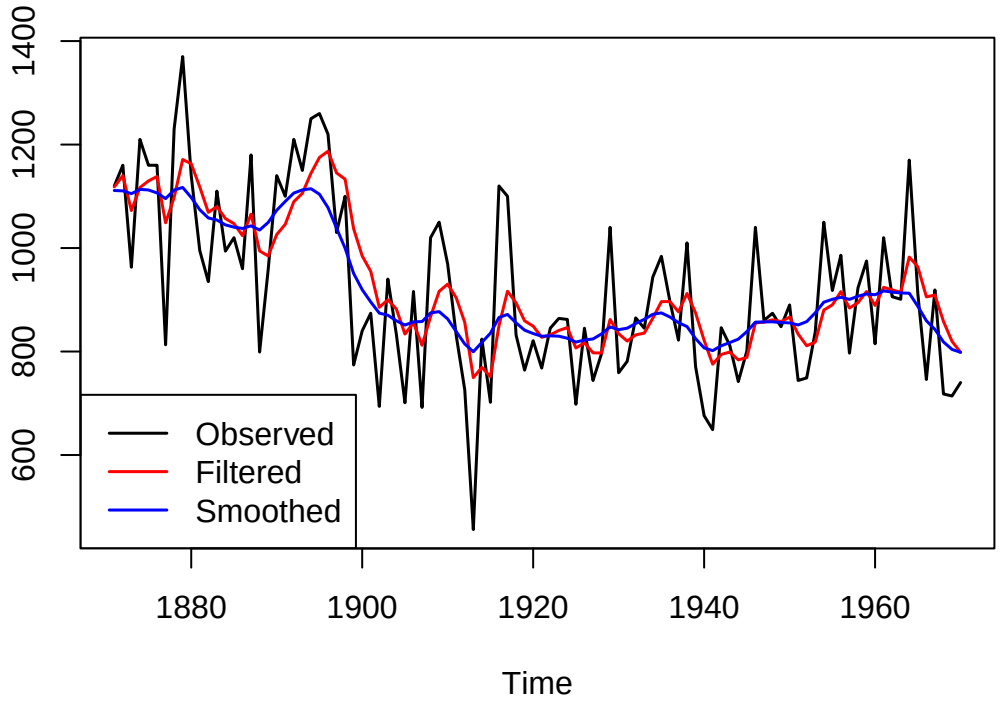


State Error Standard Deviation



4.3 Comparison between filtered and smoothed estimates of the state

The aim of this section is to visualize the differences between the obtained estimates of $\hat{\alpha}_t$, using the Kalman filter, and those obtained through smoothing procedure.



5 Simulation

Firstly, it is necessary to define σ_ε^2 , σ_η^2 , a_1 which are assumed to be known.

The MLE estimators of σ_ε^2 and σ_η^2 provided in the output of `statespacer()` are used here in order to compare the results with those reported in the book.

```
sigma_eta<-as.double(fit$system_matrices$Q$full)
sigma_epsilon<-as.double(fit$system_matrices$H$H)
a1 <- Nile[1]
```

Then we can draw the random normal deviates ε_t^+, η_t^+

```
# Simulate the model
n <- 100
set.seed(57)
eta_t_sim <- rnorm(n, 0, sqrt(sigma_eta)) # Simulate the state disturbance
epsilon_t_sim <- rnorm(n, 0, sqrt(sigma_epsilon))   # Simulate observation disturbance
```

Thirdly we generate observation using the local level recursion as follows:

$$y_t^+ = \alpha_t^+ + \varepsilon_t^+, \quad \alpha_{t+1}^+ = \alpha_t^+ + \eta_t^+, \quad t = 1, \ldots, n$$

```
y_sim <- a1 + cumsum(eta_t_sim) + epsilon_t_sim # Create the observed time series
alpha_t_sim <- y_sim-epsilon_t_sim

y_mat <- matrix(y_sim)
fit_sim <- statespacer(y_mat ,
                       local_level_ind = T,
                       initial = 0.5*log(var(y_mat)))
```

Then, a conditional draw for ε_t given Y_n is obtained as follows:

$$\tilde{\varepsilon}_t = \varepsilon_t^+ + \hat{\varepsilon}_t^+ + \hat{\varepsilon}_t \quad for \quad t = 1, \ldots, n$$

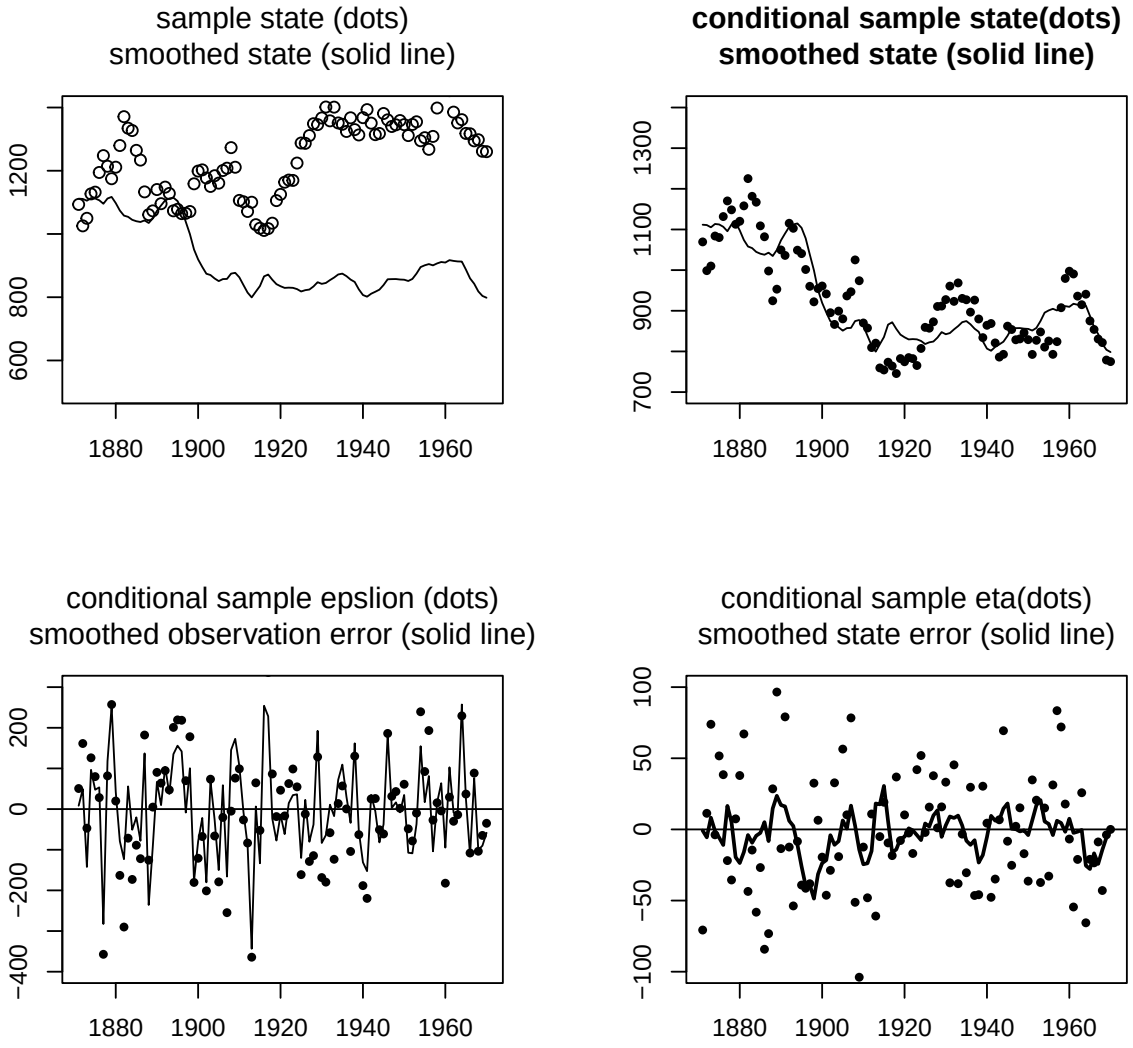
where $\hat{\varepsilon}_t = \mathbb{E}(\varepsilon_t | Y_n)$ and $\hat{\varepsilon}_t^+ = \mathbb{E}(\varepsilon_t | Y_n^+)$

Then given the sample $\tilde{\varepsilon}_1, \ldots, \tilde{\varepsilon}_n$ we obtain simulated samples for $\tilde{\alpha}_t$ and $\tilde{\eta}_t$ as follows:

$$\tilde{\alpha}_t = y_t - \tilde{\varepsilon}_t, \quad \tilde{\eta}_t = \tilde{\alpha}_{t+1} - \tilde{\alpha}_t, \quad for \quad t = 1, \ldots, n$$

```
epsilon_tilde<-epsilon_t_sim - fit_sim$smoothed$epsilon+fit$smoothed$epsilon
alpha_tilde <- Nile_df$flow - epsilon_tilde
alpha_tilde_1<-as.vector(alpha_tilde[2:length(alpha_tilde)])
alpha_tilde_1[100]<-774.9514
eta_tilde<-alpha_tilde_1-alpha_tilde
```

Simulation



What it is observe from these plots is the difference between simulating a sample from a local level model unconditionally and simulating a sample conditional on the observations. From Figures 1 and 2, we can see that the two series appear to have little in common, and that the sample generated conditionally on the data is much closer to the estimates of the latent states $\hat{\alpha}_t$

6 Parameter Estimation

6.1 The Theory Behind coding

The "Table 2.1" of the *Durbin and Koopman* book reports the number of iterations required for the Concentrated Diffuse LogLikelihood to converge to the maximum likelihood estimate and the associated values of: $q = \frac{\sigma_\eta^2}{\sigma_\varepsilon^2}$, $\psi = ln(q)$, score function and LogLikelihood.

The Diffuse LogLikelihood is used when the normality assumption is relaxed, with a_1 fixed at 0 and P_1 allowed to tend towards infinity. It thus seems reasonable to eliminate the influence of P_1 as $P_1 \to \infty$ by defining the $Diffuse\ LogLikelihood$ as

$$logL_d = \lim_{P_1 \to \infty} (logL + \frac{1}{2} logP_1)$$

Where L is the *likelihood* in the case where $\alpha_1 \sim \mathcal{N}(a_1, P_1)$ with a_1 , P_1 known and finite;

It simplify to:

$$logL_d = -\frac{n}{2} log(2\pi) - \frac{1}{2} \sum_{t=2}^{n} (logF_t \quad + \frac{v_t^2}{F_t})$$

The problem became easier if we re-parameterise the model's prior to maximization as follows, with the aim to reduce the dimensionality of the numerical search for the parameter estimation. .

$$\begin{cases} y_t = \alpha_t + \varepsilon_t, \quad \varepsilon_t \sim \mathcal{N}(0, \sigma_\varepsilon^2) \\ \alpha_{t+1} = \alpha_t + \eta_t, \quad \eta_t \sim \mathcal{N}(0, q\sigma_\varepsilon^2) \end{cases}$$

with $q = \frac{\sigma_\eta^2}{\sigma_\varepsilon^2}$ and estimate the pair σ_ε^2 , q .

The $log - likelihood$ become:

$$\log L_d = -\frac{n}{2} \log(2\pi) - \frac{n-1}{2} \log \sigma_\varepsilon^2 - \frac{1}{2} \sum_{t=2}^{n} (\log F_t^* \quad + \frac{v_t^2}{\sigma_\varepsilon^2 F_t^*})$$

Maximizing with respect to σ_ε^2, and substituing the MLE estimates $\hat{\sigma}_\varepsilon^2$ the $Concentrated\ Diffuse\ LogLikelihood$ is obtained:

$$\log L_{dc} = -\frac{n}{2} \log(2\pi) - \frac{n-1}{2} \log \hat{\sigma}_\varepsilon^2 - \frac{1}{2} \sum_{t=2}^{n} (\log F_t^*)$$

Where

$$\hat{\sigma}_\varepsilon^2 = \frac{1}{n-1} \sum_{t=2}^{n} (\frac{v_t^2}{F_t^*})$$

Therefore the dimensionality of the problem has been reduced, and now we have a one-dimensional function to minimize with respect to q. The "BFGS" procedure is then implemented, which, however, works only in minimization cases.

In the code, it was necessary to modify the functional form of the log-likelihood by omitting terms that do not depend on q,in order to compute the first derivative of the log-likelihood with respect to ψ , which will be explained later.

$$\log L_{dc} \propto -\frac{n-1}{2} \log \hat{\sigma}_\varepsilon^2 - \frac{1}{2} \sum_{t=2}^{n} (\log F_t^*)$$

6.2 The BFGS Method

We would like to briefly discuss the algorithm used to complete the analysis, namely BFGS. It is currently the most widely used quasi-Newton method, particularly renowned for its efficiency in solving unconstrained nonlinear optimization problems. This iterative method approximates the inverse Hessian matrix and uses it to update the search direction, making it more computationally feasible than direct methods. Due to its robustness and relatively low memory requirements, BFGS is frequently employed where efficient optimization of complex models is required.

6.3 The code

The first step in our procedure was the construction of the `CDLL` function, which allows us to compute the relevant components needed to compute Concentrated Diffuse Log-Likelihood.

Starting from the initial conditions $a_2 = y_1$ and $P_2^* = 1 + e^{\psi}$ the function proceeds in the computation of the following quantities:

$$\begin{aligned} v_t &= y_t - a_t & a_{t+1} &= a_t + K_t v_t \\ F_t^* &= P_t^* + 1 & P_{t+1}^* &= P_t^*(1 - K_t) + e^{\psi} \\ K_t &= \frac{P_t}{F_t} = \frac{P_t^*}{F_t^*} \end{aligned}$$

These are then used to estimate $\hat{\sigma}_{\varepsilon}^2$ and the derive $\log L_{dc}$.

Eventually, the function returns these estimates in a list.

```
y <- as.numeric(Nile)
n <- length(y)

loglist <- list()

# Diffuse log-likelihood
CDLL <- function(y, psi) {
  a2 <- y[1]
  P2 <- 1 + exp(psi)
  n <- length(y)

  a_pred <- numeric(n)
  P <- numeric(n)
  v <- numeric(n)
  K <- numeric(n)
  F <- numeric(n)
  dllk <- numeric(n)

  a_pred[2] <- a2
  P[2] <- P2

  for (t in 2:(n - 1)) {
    v[t] <- y[t] - a_pred[t]
    F[t] <- P[t] + 1
    K[t] <- P[t] / F[t]
    P[t + 1] <- P[t] * (1 - K[t]) + exp(psi)
    a_pred[t + 1] <- a_pred[t] + K[t] * v[t]
    dllk[t] <- -0.5 * log(F[t])
  }

  F[n] <- P[n] + 1
```

```
  K[n] <- P[n] / F[n]
  dllk[n] <- -0.5 * log(F[n])
  v[n] <- y[n] - a_pred[n]

  sigma_e_hat <- sum(v[2:n]^2 / F[2:n]) / (n - 1)
  llk <- -0.5 * (n - 1) * log(sigma_e_hat) + sum(dllk[2:n])

  return(list(a_pred = a_pred, llk = llk, v = v, F = F, sigma_e_hat = sigma_e_hat))
}
```

The second step is the definition of the `loglikelihood`. This function computes the negative log-likelihood for a given parameter value.
When `record = TRUE`, it stores the current iteration number, the parameter ψ, the log-likelihood, and the numerical score (an approximation of the gradient using the formal definition of the first derivatives) into the global object `loglist`. This logging mechanism is bypassed when `record = FALSE`, allowing intermediate evaluations to proceed without recording unnecessary information.

```
loglikelihood <- function(par, y, record = TRUE) {
  psi <- par[1]
  if (exp(psi) <= 0) return(1e10)
  dll_out <- CDLL(y, psi)
  llk <- dll_out$llk

  if (record) {
    h <- 1e-6
    l_plus <- loglikelihood(psi + h, y, record = FALSE)
    l_minus <- loglikelihood(psi , y, record = FALSE)
    score <- (l_plus - l_minus) / ( h)

    loglist[[length(loglist) + 1]] <<- list(
      Iteration = length(loglist),
      psi = psi,
      logLik = llk,
      score = score
    )
  }
  return(-llk)
}
```

Then, we defined the `optimizer` function which performs the optimization procedure. Convergence is controlled by the `reltol` parameter, which determines weather the gradient is sufficiently close to zero, indicating a local minimum.
We set `reltol = 1e-12` to ensure high numerical precision.

```
optimizer<- function(y, par, method = "BFGS") {
  loglist <<- list()
  optim(par = par, fn = loglikelihood, y = y, method = method,
        control = list(reltol = 1e-12, maxit = 50))
}

q_0 <- 1
psi_0<-log(q_0)
res <- optimizer(y, psi_0, method = "BFGS")

# Estrazione tabella delle iterazioni
```

```r
results_df <- data.frame(
  Iteration = sapply(loglist, function(x) x$Iteration),
  q = sapply(loglist, function(x) exp(round(x$psi, 4))),
  psi = sapply(loglist, function(x) round(x$psi, 2)),
  score = sapply(loglist, function(x) round(x$score, 5)),
  logLikelihood = sapply(loglist, function(x) round(x$logLik, 2))
)
```

We choose as starting point for the iteration procedure $\psi = 0$, and the results are stored in the `results_df`.

Table 1: Iteration table

Iteration	q	ψ	score	logLikelihood
0	1.000000	0.00	3.32307	-495.69
1	1.001001	0.00	3.32437	-495.69
2	0.999000	0.00	3.32177	-495.68
3	0.036041	-3.32	-0.92992	-492.53
4	0.036077	-3.32	-0.92889	-492.53
5	0.036005	-3.32	-0.93095	-492.53
6	0.074534	-2.60	-0.25035	-492.10
7	0.074609	-2.60	-0.24944	-492.10
8	0.074459	-2.60	-0.25126	-492.10
9	0.097413	-2.33	0.00106	-492.07
10	0.097510	-2.33	0.00204	-492.07
11	0.097315	-2.33	0.00009	-492.07
12	0.097305	-2.33	0.00003	-492.07
13	0.097403	-2.33	0.00100	-492.07
14	0.097208	-2.33	-0.00095	-492.07
15	0.097305	-2.33	0.00000	-492.07
16	0.097305	-2.33	-0.00002	-492.07
17	0.097305	-2.33	0.00000	-492.07
18	0.097305	-2.33	0.00000	-492.07
19	0.097305	-2.33	0.00000	-492.07

We can observe that the values obtained in the table closely match those reported in Table 2.1 of the book. The only difference is the number of iterations required for convergence. Durbin and Koopman achieve convergence in 4 iterations, while the procedure we used takes 19. This indicates that our function is less efficient than theirs, but equally accurate.

7 Bibliography

- Durbin, James, and Siem Jan Koopman, *Time Series Analysis by State Space Methods*, 2nd edn, Oxford Statistical Science Series (Oxford, 2012; online edn, Oxford Academic, 17 Dec. 2013),
- Tommaso Proietti, Alessandra Luati, *Maximum likelihood estimation of the time series models: the Kalman filter and beyond,* The University of Sydney (May 2012)

8 Website

- https://statisticssu.github.io/STM/tutorial/statespace/statespace.html
- https://www.linkedin.com/pulse/time-series-analysis-nile-dataset-alan-dennis/
- https://medium.com/data-science/bfgs-in-a-nutshell-an-introduction-to-quasi-newton-methods-21b0e13ee504
- https://bookdown.org/yihui/rmarkdown/bookdown-output.html#latexpdf
- https://github.com/cran/statespacer/blob/master/DESCRIPTION