# ATS Coursework - week 4

Federico Grazi, Federico Podrecca, Guido Verosimile

May 10, 2024

## 1 Fitting LLM to Nile Dataset

The first task required us to reproduce a table of fitted values for a Local Level Model on the `Nile` in R. The table[1] we had to refer to had the output of the iterations of the ML estimation of the Signal to Noise Ratio using a quasi-Newton scheme BFGS optimisation procedure. Given the high degree of difficulty of the problem, we decided to try two solution for obtain similar table.

1. We opted for a simpler optimizer using the EM algorithm in order to obtain the estimates of the variances and the SNR.
2. We used the package `optimg` to implement an automatic maximization of the log-likelihood.

In general, the problem was essentially to fit the parameters to which DK referred as

$$\psi = \begin{pmatrix} \sigma_\varepsilon^2 \\ \sigma_\eta^2 \end{pmatrix},$$

and rather than focusing on maximizing with respect to $q$, we decided to estimate both parameters.

### 1.1 Numerical Optimization via EM algorithm

DK develop the ML estimation for the iteration while estimating a local level model through a Kalman Filter. The model specified in (2.3) of DK has 2 parameters, thus DK re-parameterise the model such that $\sigma_\eta^2 = q\sigma_\varepsilon^2$ and $\eta_t \sim \mathcal{N}(0, q\sigma_\varepsilon^2)$ allowing to reduce the parameter to estimate to just $\sigma_\varepsilon^2$ and then independently $q$.
By stating again the model, DK then computed the *diffuse* likelihood as

$$\begin{aligned} \log L_d &= -\frac{n}{2}\log(2\pi) - \frac{1}{2}\sum_{t=2}^{n}\big(\log F_t + \frac{v_t^2}{F_t}\big) = \\ &= -\frac{n}{2}\log(2\pi) - \frac{n-1}{2}\log\sigma_\varepsilon^2 - \frac{1}{2}\sum_{t=2}^{n}\big(\log F_t^* + \frac{v_t^2}{\sigma_\varepsilon^2 F_t^*}\big). \end{aligned} \tag{1}$$

Section 7.3.4 leaves us with a recursive formula obtained by EM on the above log likelihood with which we can estimate the values of the parameters:

$$\sigma_\varepsilon^{2(\text{new})} = \sigma_\varepsilon^{2(\text{old})} + \frac{1}{n}\sigma_\varepsilon^{4(\text{old})}\sum_{t=1}^{n}(u_t^2 - D_t) \tag{2}$$

$$\sigma_\eta^{2(\text{new})} = \sigma_\eta^{2(\text{old})} + \frac{1}{n-1}\sigma_\eta^{4(\text{old})}\sum_{t=2}^{n}(r_t^2 - N_t) \tag{3}$$

where $r_t$ and $N_t$, respectively a weighted sum of innovations after $t$ and its variance, are obtained in the state smoothing procedure, while $u_t$ and $D_t$ in the disturbance smoothing. As DK state: *"The disturbance smoothing values $u_t$, $D_t$, $r_t$ and $N_t$ are based on $\sigma_\varepsilon^{2(old)}$ and $\sigma_\eta^{2(old)}$. The new values $\sigma_\eta^{2(new)}$ and $\sigma_\varepsilon^{2(new)}$ replace the old ones and the procedure is repeated until convergence has been attained".*

[1] Table 2.1 from Durbin-Koopman, henceforth abbreviated with DK

Thus, in order to obtain the EM updates of the parameter we shall fit the Kalman Filter, the State Smoothing and the Disturbance Smoothing. For these tasks, we have constructed 3 function which allow us to obtain all the needed values for the Equations 2 (which are reported in the Appendix A). With a simple `for` loop we were able to put everything together to obtain the recursion for the EM algorithm. The stopping criterion used was a significantly small change in the log likelihood, hence:

$$|\log L_d^{(\text{new})} - \log L_d^{(\text{old})}| < \epsilon.$$

We initialized the values for the variance at 1000 such that $q = 1$ and used $\epsilon = 0.01$: the following code was the recursive iterations for the EM parameters

```
theta_init <- c(1000,1000)
matrixTheta <- matrix(c(theta_init), ncol = 2, byrow = T)
crit <- 1000
loglik <- extract_loglik(theta_init)
i = 0
while(crit>0.001){
  i = i+1
  theta_old <- as.numeric(matrixTheta[i,])

  theta_new <- EM_LLM(theta_old)
  loglik <- c(loglik,extract_loglik(theta_new))

  matrixTheta <- rbind(matrixTheta, theta_new)
  crit <- abs(loglik[i+1]-loglik[i])
}
```

The criterion was met at iteration 158, even though, as it can be seen in Figure 1, the log likelihood converged pretty early, but took some time to reach the actual criterion.

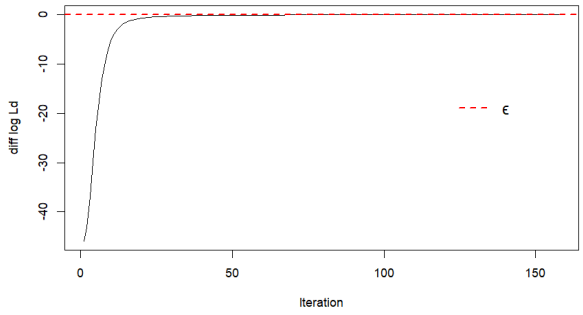



**Figure 1:** Plot of $\log L_d^{(\text{new})} - \log L_d^{(\text{old})}$ against $\epsilon = 0.001$

Our value for the iterations are presented in Table 1.

The local level model can now be fitted using as initialisation values $a_1 = 0$ and $P_1 = 10^7$ as DK use in 2.2.5. The following Kalman Filter is plotted against the raw data in Figure 2

**Table 1:** Estimation of parameters of local level model by EM algorithm.

| Iteration | q | $\psi$ | Loglikelihood |
|---|---|---|---|
| 1 | 1 | 0 | -903.13 |
| 32 | 0.12952 | -2.043 | -650.23 |
| 64 | 0.10541 | -2.249 | -646.77 |
| 95 | 0.10010 | -2.301 | -646.04 |
| 127 | 0.09856 | -2.317 | -645.84 |
| 158 | 0.09813 | -2.321 | -645.78 |

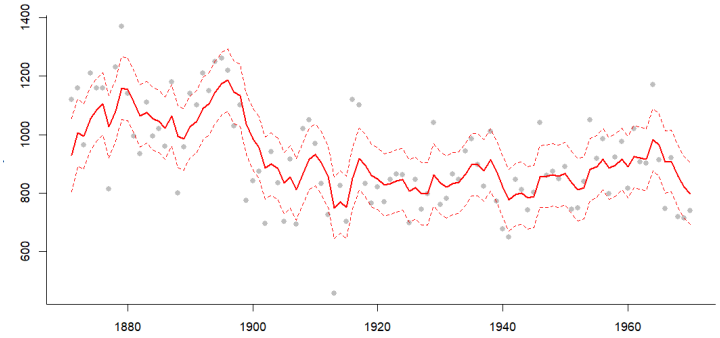



**Figure 2:** Nile data (dots) and filtered state $\alpha_{t|t-1}$ of the Kalman filter (solid line) with 95% confidence interval.

## 1.2 Package `optimg` for minimizing the log likelihood

We also implemented a more direct way of optimizing the log likelihood found in Equation 1. The broadly known function `optim` worked perfectly, but lacked in giving back the values at each iteration. We decided to use a lesser-known package that can comply with our need: `optimg`[1]. The package works with almost the same parameters of `optim`:

- **par**: initial values of the parameter (`c(1000,1000)`);
- **fn**: function to be minimized (`extract_loglik`, see Appendix A);
- **full**: Boolean argument for returning all results.

The following chunk of code has returned as output a convergence at iteration 26, thus way quicker than our manual EM implementation.

```
llik_optg <- optimg(
                par = c(1000,1000),
                fn = extract_loglik,
                method = "STGD",
                full = T)
```

Our results are summed up in Table 2. First of all, a quicker convergence was obtained (with only 26 iterations), then, it can be observed that also the value for the log likelihood is lower, suggesting that our estimate for the EM algorithm is still not optimal and does not return the best solution loglikelihood-wise. Another important aspect to put our attention on is the value of $q$, which has also improved and is now closer to the value obtained by Durbin and Koopman.

**Table 2:** Estimation of parameters of local level model by maximum likelihood.

| Iteration | q | $\psi$ | Loglikelihoood |
|---|---|---|---|
| 1 | 1 | 0 | -635.68 |
| 5 | 0.12740 | -2.060 | -633.53 |
| 10 | 0.11253 | -2.184 | -633.47 |
| 15 | 0.10054 | -2.297 | -633.46 |
| 20 | 0.09925 | -2.310 | -633.46 |
| 26 | 0.09779 | -2.324 | -633.46 |

Using `plotly`[4], as shown in Figure 3, we can see that the surface of the log likelihood is pretty flat around the values of interest, but drop significantly at the boundaries of the parameter space. This surface does creates an extra challenge for a sub-optimal algorithm, such as the EM we have tried to implement: in fact, the changes in $\log L_d$ may happen to be so small that the optimizer takes too long to reach the global maximum.

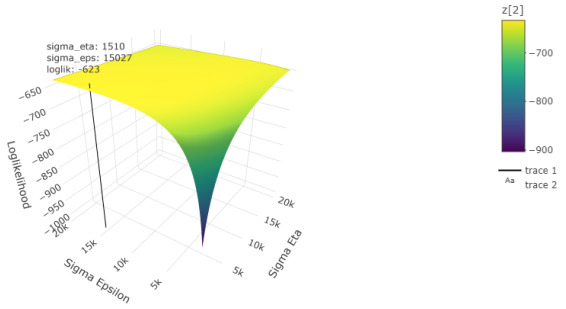



**Figure 3:** Surface plot of the loglikelihood for various inputs of the parameter space with line at the maximum of $\log L_d$.

## 1.3 Comparison of the parameters

The result can be compared with those obtain by DK, that can be replicated exactly with the function `StructTS` in the package `stats`[3].

```
y <- Nile
LLM <- stats::StructTS(y, "level")
```

The signal to noise ratio is kept by all three models, even if the EM has performed poorly on the actual parameter values.

| | `StructTS` | `optimg` | EM algorithm |
|---|---|---|---|
| $\sigma_\varepsilon^2$ | 15099 | 15085 | 8026 |
| $\sigma_\eta^2$ | 1469 | 1469 | 786 |
| $q$ | 0,0973 | 0,0977 | 0,0981 |

**Table 3:** Comparison between the results

At last, we also decided to check whether the Guassian assumption introduced when using a state space model (for which the local level model is a special case) was adequate. In Figure 4 we plotted a Quantile-Quantile plot: the general behaviour of the noise term is indeed similar to what we would expect with a Gaussian noise term. Other than some observation at the tails, the Gaussian assumption for the noise term can be considered as adequate.

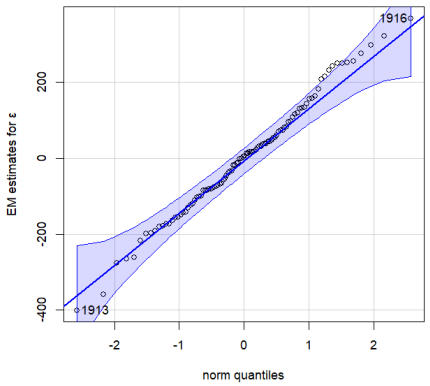



**Figure 4:** qqPlot for $\varepsilon_t$

## 2 Simulation of section 2.6

The request in this assignment was to reproduce a simulation study proposed in section 2.6 by DK. DK proposed to illustrate the difference between simulating a sample from a unconditional model and a conditional model. To show this they proceeded to sample data from a LLM with the parameter of those fitted from the Nile dataset, hence from Table 3. The smoothed state ($\hat{\alpha}_t$) obtain through disturbance smoothing is thus compared with the sample generated at random ($\alpha_t^+$) and a sample generated by conditioning on the observed values ($\tilde{\alpha}_t$).

We will focus first on the difference between the filtered state obtain with the Kalman filter and the smoothed state, which is computed by conditioning to the whole observation and can be evaluated with a backward recursion. The smoothed state is obtain by the equation

$$\hat{\alpha}_t = \alpha_{t|t-1} + P_t r_{t-1} \tag{4}$$

$$r_{t-1} = \frac{v_t}{F_t} + (1 - K_t) r_t. \tag{5}$$

This backward recursion was implemented in `R` with

```
rt <- Nt <- double(length(y))
a_hat_keep <- NULL
for(i in length(y):1){ # from n to 1

  a1 <- a_keep[i] # Use the filtered state from KF
  P1 <- P_keep[i]

  at <- a1
  Pt <- P1
  yt <- y[i]
  cur_rt <- rt[i]
  cur_Nt <- Nt[i]

  params <- c(yt, at, Pt,sigma_eps, sigma_eta, cur_rt,cur_Nt)
  upd_recursion <- SS_LLM(params) # apply State Smoothing

  rt[i-1] <- upd_recursion[1]
  Nt[i-1] <- upd_recursion[2]
  a_hat <- upd_recursion[3]

  a_hat_keep <- c(a_hat_keep, a_hat)
}
```

In Figure 5 the plot of the two time series are plotted together. The blue line is slightly smoother, but still quite rough with respect to the one obtain by DK in Figure 2.1 (i) of their book.

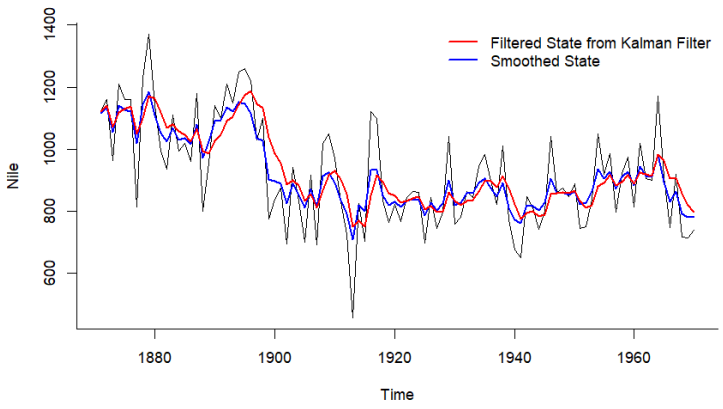



**Figure 5:** Comparison between filtered state $\alpha_{t|t-1}$ and the smoothed state $\hat{\alpha}_{t|t-1}$.

Instead, the disturbance smoothing error is obtain by

```
eps_hat <- eta_hat <- double(length(y))
for(i in 1:length(y)){
  a1 <- a_keep[i]
  P1 <- P_keep[i]

  at <- a1
  Pt <- P1
  yt <- y[i]
  cur_rt <- rt[i]
  cur_Nt <- Nt[i]

  params <- c(yt, at, Pt,sigma_eps, sigma_eta, cur_rt, cur_Nt)
  upd_smootherr <- DS_LLM(params)

  eps_hat[i] <- sigma_eps * upd_smootherr[1]
  eta_hat[i] <- sigma_eta * rt[i]
}
```

Now, we only need to obtain the smoothed state from the generated data from which we can draw $\varepsilon_t$ conditionally to $\mathcal{F}_t$. This is computationally done with the next formula.

$$\tilde{\varepsilon}_t = \varepsilon_t^+ - \hat{\varepsilon}_t^+ + \hat{\varepsilon}_t \tag{6}$$

To generate the data we used this chunck of R codes. We tried to keep the notation coherent with the one of DK.

```
a1_plus <- y[1]
eps_plus <- eta_plus <- y_plus <- a_plus <- double(length(y))
# Generate the data
for(i in 1:length(y)){
  eps_plus[i] <- rnorm(1,0,sqrt(sigma_eps))
  eta_plus[i] <- rnorm(1,0,sqrt(sigma_eta))

  a_plus[i] <- a1_plus
  y_plus[i] <- a_plus[i] + eps_plus[i]
```

```
  a1_plus <- a_plus[i] + eta_plus[i]
}
```

Once generated the new time series $y_t^+$ we can compute the state smoothing and the disturbance smoothing directly on $y_t^+$ and consequently obtain $\hat{\varepsilon}_t^+$ by simply repeating the backward recursion with the generated data.

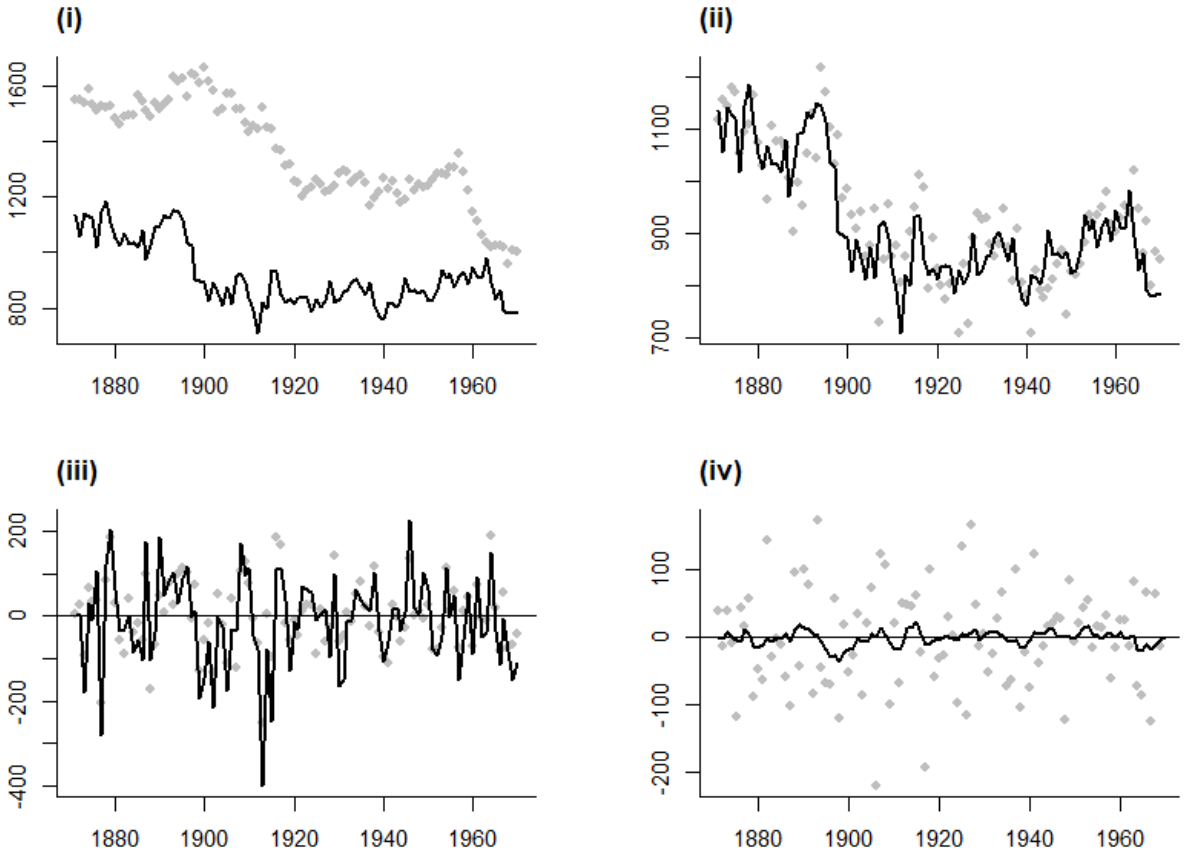



**Figure 6:** Simulation: (i) smoothed state $\hat{\alpha}_{t|t-1}$ (solid line) and sample $\alpha_t^+$ (dots);
(ii) smoothed state (solid line) and sample $\tilde{\alpha}_t$ (dots);
(iii) smoothed observation error $\hat{\varepsilon}_t$ (solid line) and sample $\tilde{\varepsilon}_t$ (dots);
(iv) smoothed state error $\hat{\eta}_t$ (solid line) and sample $\tilde{\varepsilon}_t$ (dots).

To check whether the results were consistent with DK, we plotted and compared the values for the filtered state variance $P_{t|t-1}$ and the smoothing variance cumulant $N_t$. Figure 7 show that the two values have the same behaviour, which is also coherent with what DK show in their illustrations.

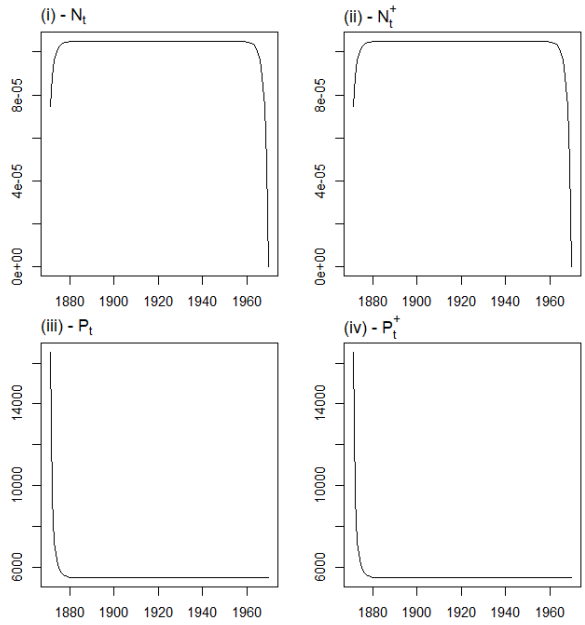



**Figure 7:** s

# 3 Real Gross Domestic Product

In order to meet the requirement of the third point, we have selected a time series from the FRED, the Federal Reserve Economic Data. In particular, we studied the Real Gross Domestic Product series, which is the inflation adjusted value of the goods and services produced by labor and property located in the United States[2], at 2017 prices[3].[**U.S. Bureau of Economic Analysis**, **Real Gross Domestic Product [GDPC1 retrieved from FRED**, **Federal Reserve Bank of St. Louis;** ]
Data are available at https://fred.stlouisfed.org/series/GDPC1.
Our analysis will cover the period from 1947-04-01 until 2020-01-01, studying all the series up to Covid, avoiding the shock caused by the pandemic. For this reason, we will use forecasting techniques to predict the progress from the second quarter of 2020 until 2024.

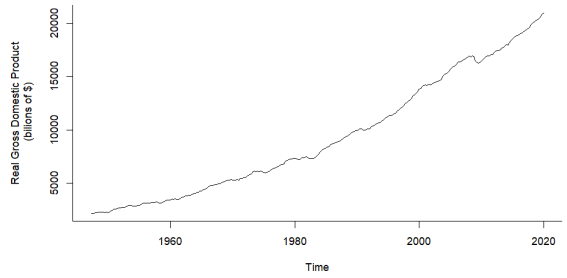



**Figure 8:** Real Gross Domestic Product, in Billions of Chaine 2017 dollars, Seasonally Adjusted Annual Rate

From the simple plot of the series, we can notice an evident positive trend, disturbed, as expected, by the 2008 shock, which should not interfere too much in the estimates.

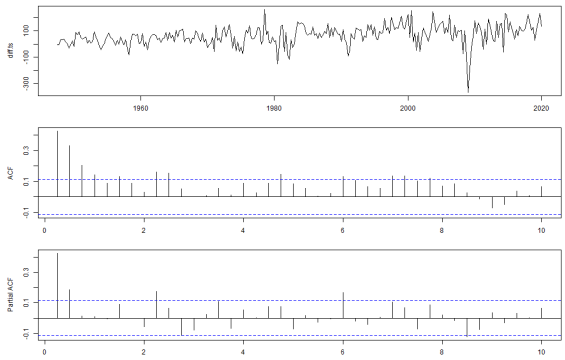
[2]https://fred.stlouisfed.org/series/GDPC1
[3]Source: U.S. Bureau of Economic Analysis http://www.bea.gov/

# 4 Analysis of the time series

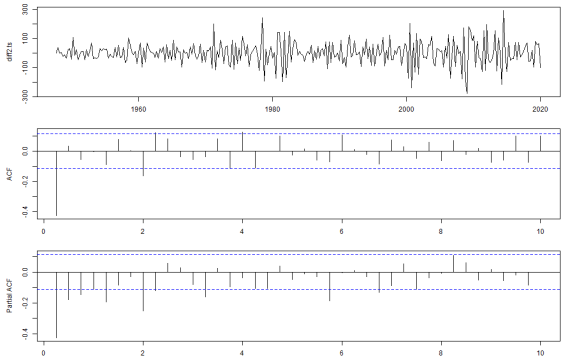
We tried two different approaches to modelling the time series. The first one, with ARIMA modelling, and the second one using the structural time series modelling

## 4.1 ARIMA approach

Given the evident dependence structure of the data, we tried to work on the first order differences $\Delta^1 y_t = (1-L)y_t = y_t - y_{t-1}$. What Figure 9a shows is a more appealing time series with stationarity in the mean even if the some could argue that the shock of 2008 has highly impacted the lagged series. Anyway, the ACF plot does not give a conclusive evidence towards a particular order of the MA part, while the PACF suggests a AR(2) could be adequate.




**(a)** Top: time series obtain with **first** order differences.
Center: ACF of the lagged time series.
Bottom: PACF of the lagged time series




**(b)** Top: time series obtain with **second** order differences.
Center: ACF of the lagged time series.
Bottom: PACF of the lagged time series

**Figure 9**

By further exploring the lag differences with the second order differences $\Delta^2 y_t = (1-L)^2 y_t$ the plots shown in Figure 9b gave us more satisfactory insights on the nature of the time series. First of all, the impact of the shock is less obvious, then the second order difference clearly suggest that a MA(1) term is adequate to describe the ACF and also a AR(2) component can be obtained from the study of the PACF plot. The package `forecast`[2] allows us to easily fit an ARIMA model through maximum likelihood. We will work on the first order difference, so we'll apply a ARIMA(2,1,1), rather than a ARMA(2,1) if we used the second order differences. The result were obtain with this code

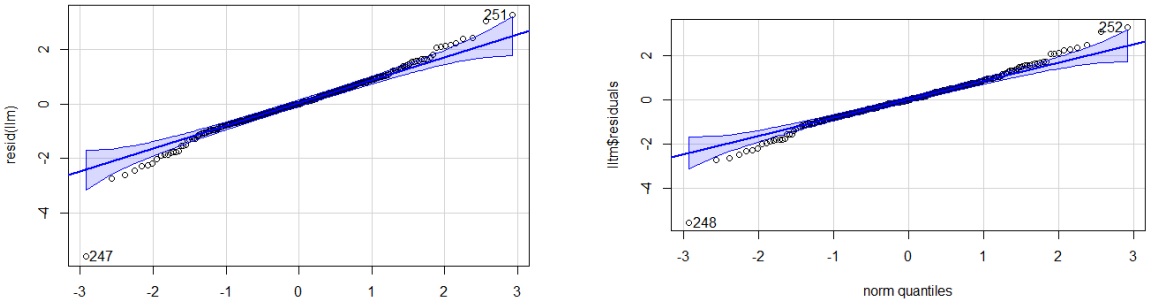
```
> arima211 <- arima(diff.ts,order = c(2,1,1))
> arima211
    Call:
    arima(x = diff.ts, order = c(2, 1, 1))

    Coefficients:
             ar1     ar2      ma1
          0.3026  0.1518  -0.9748
    s.e.  0.0594  0.0595   0.0128

    sigma^2 estimated as 4310:  log likelihood = -1625.97,  aic = 3257.94
```

## 4.2 Structural Modelling

The auto correlation analysis confirmed us that a MA(1) component was adequate to fit the data and the fact that the estimates for the parameter are significant is another confirm to this hypothesis. In general, the theory develop in DK's book tells us that the reduced form of the first order difference of a local level model is simply a MA(1), thus we could consider adequate also fitting a local level model and compare the two results. Firstly, we decided to check for the linear gaussianity assumption needed by both the llm and llt models. This has been done by comparing the distributions of the model residuals to the normal one. In both cases, as shown in Figure 10, we concluded that the normality assumption was adequate.




**(a)** qqplot for llm

**(b)** qqplot for llt

**Figure 10**

We eventually used again the `stats` package and fitted the LLM on the first order difference using the following code:

```
llm <- StructTS(diff.ts, type = "level")
```

Just for comparison and to practically confirm results from theory, we also fitted a ARIMA(0,1,1) and, indeed, the fitted values plotted in Figure 11 showed us that ARIMA(0,1,1) and LLM can be considered equivalent.

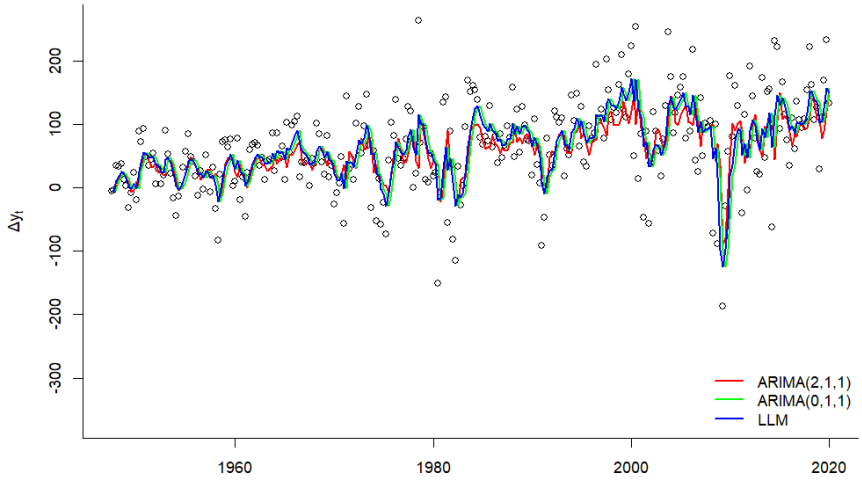



**Figure 11:** Comparison between the ARIMAs fitted values and the fitted values from a local level model

At last, given the obvious trend in the original time series, we decided to fit also a local linear trend model to the first order difference data. In fact, if we were to focus our attention on the local level

model, we would be left with a straight line (by construction of the forecasts of a local level model) as forecast, which would not be very meaningfully given the data structure.

```
>lltm <- StructTS(ts, type = "trend")
>lltm

Call:
StructTS(x = ts, type = "trend")

Variances:
  level    slope  epsilon
 3209.5    501.1      0.0
```

The model return a $\sigma_\varepsilon^2 = 0$, meaning that the lltm specified in (3.2) by DK is reduced to

$$\begin{cases} y_{t-1} = y_t + \beta_t + \eta_t, & \eta_t \sim \mathcal{N}(0, \sigma_\eta^2) \\ \beta_{t+1} = \beta_t + \zeta_t, & \zeta_t \sim \mathcal{N}(0, \sigma_\zeta^2) \end{cases} \tag{7}$$

which appears to be a random walk with another random walk as slope. It is interesting to note that if the local linear trend model were to be fitted to the first order difference we would obtain a local level model with a added constant $\beta$

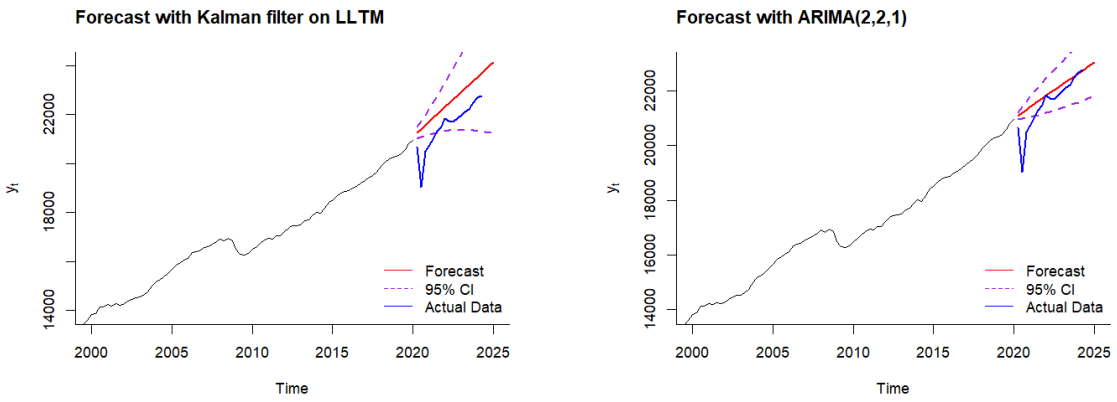
```
> lltm.diff <- StructTS(diff.ts, type = "trend")
> lltm.diff

Call:
StructTS(x = diff.ts, type = "trend")

Variances:
  level    slope  epsilon
  449.1      0.0   3279.1
```

## 4.3 Forecasting

As stated before, we will use a small portion of the last year of the dataset to compare the forecasting behaviour. We used the model stated in Equation 7 to obtain the forecast values with the Kalman filter, then we also extracted the forecast from the ARIMA(2,2,1) directly applied to the original time series. The code for the full recursive procedure of the Kalman filter was reported in Appendix B. Figure 12 shows the different behaviour of the two chosen models. Even if Covid brought a big shock




(a) (b)

**Figure 12**

in the time series, we can see that the values have approximately returned on track of the time series,

hence matching the values of 2024 is still relevant for the forecast. It is possible to see that the Kalman filter overshoots the data and also has a larger confidence interval: it looks like the local linear trend model tends to emphasise more the trend component and enhances its contribution in the forecast. In constrast, the ARIMA(2,2,1) model is more conservative, and forecasts values which actually match the data post-Covid.

# References


[1] Vithor Rosa Franco. *optimg: General-Purpose Gradient-Based Optimization*. R package version 0.1.2. 2021. URL: https://CRAN.R-project.org/package=optimg.

[2] Rob J Hyndman and Yeasmin Khandakar. "Automatic time series forecasting: the forecast package for R". In: *Journal of Statistical Software* 27.3 (2008), pp. 1–22. DOI: 10.18637/jss.v027.i03.

[3] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2024. URL: https://www.R-project.org/.

[4] Carson Sievert. *Interactive Web-Based Data Visualization with R, plotly, and shiny*. Chapman and Hall/CRC, 2020. ISBN: 9781138331457. URL: https://plotly-r.com.

# Appendix A

```
# KALMAN FILTER FOR LLM
KF_LLM <- function(params){
  yt <- params[1]
  at <- params[2]
  Pt <- params[3]
  sigma_eps <- params[4]
  sigma_eta <- params[5]

  vt <- yt-at
  Ft <- Pt + sigma_eps
  Kt <- Pt / Ft

  at_upd <- at + Kt * vt
  Pt_upd <- Pt * (1 - Kt) + sigma_eta

  names(at_upd) <- names(Pt_upd) <- NULL
  upd_params <- c("at" = at_upd, "Pt" = Pt_upd)

  return(upd_params)
}

# STATE SMOOTHING FOR LLM
SS_LLM <- function(params){

  yt <- params[1]
  at <- params[2]
  Pt <- params[3]
  sigma_eps <- params[4]
  sigma_eta <- params[5]
  rt <- params[6]
  Nt <- params[7]

  vt <- yt-at
  Ft <- Pt + sigma_eps
  Kt <- Pt / Ft
  Lt <- 1-Kt

  rt_1 <- vt / Ft + Lt * rt
  Nt_1 <- Ft^-1 + Lt^2 * Nt
  a_hat <- at + Pt * rt_1

  names(rt_1) <- names(Nt_1) <- NULL
  upd_params <- c("rt" = rt_1,"Nt" = Nt_1, "at" = a_hat)

  return(upd_params)
}

# DISTURBANCE SMOOTHING FOR LLM
DS_LLM <- function(params){
  yt <- params[1]
  at <- params[2]
  Pt <- params[3]
  sigma_eps <- params[4]
  sigma_eta <- params[5]
  rt <- params[6]
```

```
  Nt <- params[7]

  vt <- yt-at
  Ft <- Pt + sigma_eps
  Kt <- Pt / Ft


  Dt <- Ft^-1 + Kt^2 * Nt
  ut <- Ft^-1 * vt - Kt * rt
  names(Dt) <- names(ut) <- NULL
  upd_params <- c("ut" = ut, "Dt" = Dt)

  return(upd_params)
}

# LOGLIK FOR LLM AS IN (2.60) OF DK
extract_loglik <- function(theta){
  n <- length(y)
  a1 <- 0
  P1 <- 1e7
  sigma_eps <- theta[1]
  sigma_eta <- theta[2]
  vt_keep <- Ft_keep <- NULL
  for(i in 1:length(y)){

    at <- a1
    Pt <- P1
    yt <- y[i]

    vt <- yt-at
    Ft <- Pt + sigma_eps
    Kt <- Pt / Ft

    a1 <- at + Kt * vt
    P1 <- Pt * (1 - Kt) + sigma_eta

    vt_keep <- c(vt_keep, vt)
    Ft_keep <- c(Ft_keep, Ft)
  }

  constant <- -n/2 * log(2*pi)
  logsum <- -.5 * sum( log(Ft_keep[2:n]) + vt_keep[2:n]^2/Ft_keep[2:n])

  loglik <- -( constant  + logsum)
  return(loglik)
}

EM_LLM <- function(theta){

  a1 <- 0
  P1 <- 1e7
  n <- length(y)

  sigma_eps <- theta[1]
  sigma_eta <- theta[2]
```

```
# Obtain filtered state space
a_keep <- P_keep <- NULL
for(i in 1:length(y)){

  at <- a1
  Pt <- P1
  yt <- y[i]

  par_KF <- c(yt, at, Pt, sigma_eps,sigma_eta)
  upd_KF <- KF_LLM(par_KF)

  a1 <- upd_KF[1]
  P1 <- upd_KF[2]

  a_keep <- c(a_keep, a1)
  P_keep <- c(P_keep, P1)
}
names(a_keep) <- names(P_keep) <- NULL

# Obtain smoothed state space
rt <- Nt <- double(length(y))
for(i in length(y):1){

  a1 <- a_keep[i]
  P1 <- P_keep[i]

  at <- a1
  Pt <- P1
  yt <- y[i]
  cur_rt <- rt[i]
  cur_Nt <- Nt[i]

  params <- c(yt, at, Pt,sigma_eps, sigma_eta, cur_rt, cur_Nt)
  upd_recursion <- SS_LLM(params)

  rt[i-1] <- upd_recursion[1]
  Nt[i-1] <- upd_recursion[2]
}


# Obtain state space from disturbance smoothing
ut <- Dt <- double(length(y))
for(i in 1:length(y)){
  a1 <- a_keep[i]
  P1 <- P_keep[i]

  at <- a1
  Pt <- P1
  yt <- y[i]
  cur_rt <- rt[i]
  cur_Nt <- Nt[i]

  params <- c(yt, at, Pt,sigma_eps, sigma_eta, cur_rt, cur_Nt)
  upd_smootherr <- DS_LLM(params)

  ut[i] <- upd_smootherr[1]
```

```
    Dt[i] <- upd_smootherr[2]
  }

  em_vect <- c(
    "upt eps" = sigma_eps + 1/n * sigma_eps^2 * sum(ut^2 - Dt),
    "upd eta" = sigma_eta + 1/(n-1) * sigma_eta^2 * sum(rt^2 - Nt)
  )

  return(em_vect)
}
```

# Appendix B

Forecasting using the Kalman filter on the local linear trend model

```
a1 <- matrix(0,2,1)
P1 <- matrix(c(1e2,0,0,1e2), ncol = 2, nrow = 2, byrow = T)
Zt <- matrix(c(1,0), ncol = 2, nrow = 1)
Tt <- matrix(c(1,1,0,1), ncol = 2, nrow = 2, byrow = T)
sigma_eps <- lltm$coef[3]
sigma_eta <- lltm$coef[1]
sigma_zeta <- lltm$coef[2]
Qt <- matrix(c(sigma_eta,0,0,sigma_zeta), ncol = 2, byrow = T)

# KALMAN FILTER
mu_keep <- beta_keep <- NULL
for( i in 1:length(ts)){
  yt <- ts[i]

  Pt <- P1
  at <- a1

  vt <- yt-Zt %*% at

  Ft <- Zt %*% Pt %*% t(Zt) + sigma_eps
  Kt <- Tt %*% Pt %*% t(Zt) %*% solve(Ft)

  P1 <- Tt %*% Pt %*% t(Tt) + Qt - Kt %*% Ft %*% t(Kt)
  a1 <- Tt %*% at + Kt %*% vt

  mu_keep <- c(mu_keep, a1[1,1])
  beta_keep <- c(beta_keep, a1[2,1])

}

mu_keep <- ts(mu_keep,start = c(1947,2), frequency = 4)
beta_keep <- ts(beta_keep,start = c(1947,2), frequency = 4)

# Kalman Filter for forecasting
y_pred <- P_pred <-  NULL
ynew <- mu_keep[length(mu_keep)-1]
j = 0
Kt <- matrix(c(0,0),nrow = 2)
while(j < 20){
  j = j+1
  yt1ahead <- ynew
```

```
  Pt <- P1
  at <- a1

  vt <- yt1ahead - Zt %*% at

  Ft <- Zt %*% Pt %*% t(Zt) + 200


  P1 <- Tt %*% Pt %*% t(Tt) + Qt - Kt %*% Ft %*% t(Kt)
  a1 <- Tt %*% at + Kt %*% vt

  ynew <- Zt %*% a1
  y_pred <- c(y_pred, ynew)
  P_pred <- c(P_pred, P1[1,1])

}
```