MAS8382 Summative Practical Report

Iro Chalastani Patsioura

09/12/2021

Question 1

a)

Using the arma function from the formative exercises we can write a function which simulates from an ARIMA(p,d,q) process is written below. The code for bot functions can be found below:

```
my_arma = function(n, sigsq=1, mu=0, phi=numeric(0), theta=numeric(0), inits=NULL) {
  p = length(phi)
  q = length(theta)
  # Check that the process is stationary
  if(p>0) {
    roots = polyroot(c(1, -phi))
    if(all(Mod(roots)>1)) message("Process is stationary.")
    else message("Process is not stationary.")
  }
  # Check that the process is invertible
  if(q>0) {
    roots = polyroot(c(1, theta))
    if(all(Mod(roots)>1)) message("Process is invertible.")
    else message("Process is not invertible.")
  }
  # Establish initial values
  maxpq = max(c(p, q))
  if(!is.null(inits)) {
    if(length(inits)!=maxpq) stop("If initial values are supplied,
there must be max(p, q) of them.")
  } else {
    # Note: sets x[1], ..., x[maxpq] = 0
    inits = rep(0, maxpq)
  }
  # Simulate from model
  x = numeric(n)
  x[1:maxpq] = inits
  eps = rnorm(n, 0, sqrt(sigsq))
  for(t in (maxpq+1):n) {
    x[t] = sum(phi * x[(t-1):(t-p)]) + sum(c(1, theta) * eps[t:(t-q)])
  }
  return(x+mu)
}
```

```
my_arima = function(n, sigsq=1, mu=0, phi=numeric(0), theta=numeric(0), d){
  x = my_arma(n, sigsq, mu, phi, theta)
  y = numeric(length(x))
  for (i in 1:d){
  y[1] = 0
  for(t in 2:length(x)){
    y[t] = x[t] + y[t-1]
  }
  x=y
  }
  return(y)
}
```

b

In order to simulate from an ARIMA(2,1,1) process, we first need to set some initial values as follows:

```
n = 1000
sigsq = 0.5
mu = 0
phi = c(0.8, 0.1)
theta = -0.6
```

Now we can sample three times from the process as follows:

```
set.seed(3)
output1 = my_arima(n, sigsq=1, mu=0, phi, theta, 1)
```

```
## Process is stationary.

## Process is invertible.
```

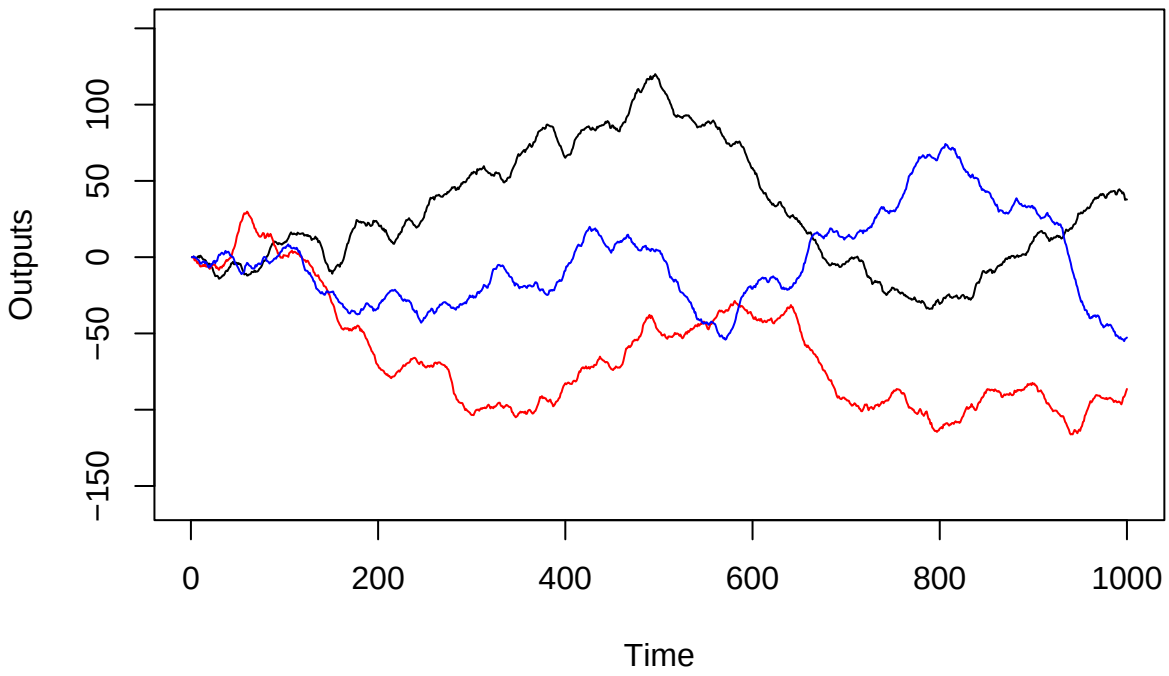
```
output2 = my_arima(n, sigsq=1, mu=0, phi, theta, 1)
```

```
## Process is stationary.
## Process is invertible.
```

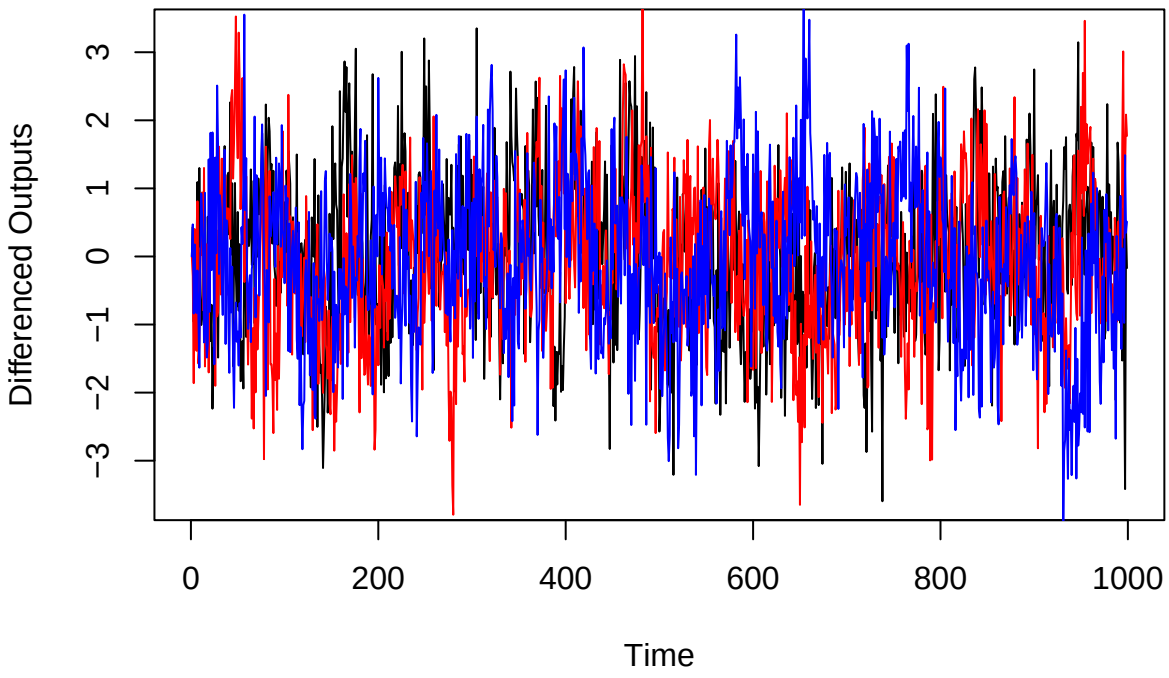
```
output3 = my_arima(n, sigsq=1, mu=0, phi, theta, 1)
```

```
## Process is stationary.
## Process is invertible.
```

The we can plot the three outputs:



By looking at the plot we observe that the mean for all realizations, seems to vary a bit over time, so the data cannot have arisen form a stationary process. Other than that, it seems like the data appear fairly homogeneous with time. Now we can difference the data and plot them again:



After differencing we observe that for all three realizations there is no evidence of non stationary. In this case we only need to difference once, to achieve stationarity.

Question 2

a)

For the AR(p) model we can estimate it's parameters using the Yule-Walker equations. The function below does exactly that:

```
par_est_func = function(x, p){
  lag = p + 1
  rho = acf(x, lag.max = lag -1, plot = FALSE)
  #Remove the first element
  rho_minus_1 = rho$acf[-1]
  # remove the last element of rho
  rho_new = rho$acf[-lag]
  #Construct the capital rho matrix
  capital_rho = toeplitz(rho_new)
  #Find gammas
  gammas = acf(x, plot=FALSE, type="covariance", lag.max = p)
  # Find gamma zero
  gamma_zero = gammas$acf[1]
  # Find phi
  phi = solve(capital_rho)%*%rho_minus_1[1:p]
```

```
  # Find sigma
  sigma_sq = gamma_zero*(1 - t(rho_minus_1)%*%solve(capital_rho)%*%rho_minus_1)
  my_list = list("phi"= phi, "sigma squared" = sigma_sq)
  return(my_list)
}
```

b)

Now we can see the estimates. To do that we just sample some data from an AR(p) process using the function from question 1, and then checking if the function from part a gives estimates that are close to the ones used to simulate the process. Now to simulate an AR(3) model we set:

```
n = 1000
sigsq = 0.5
mu = 0
phi = c(0.8, 0.1, -0.4)
f = my_arma(n, sigsq, mu, phi)
```

```
## Process is stationary.
```

Now we can use the function from part a to estimate phi and sigma:

```
set.seed(75)
par_est_func(f,3)
```

```
## $phi
##             [,1]
## [1,]  0.7931879
## [2,]  0.1095956
## [3,] -0.3615144
##
## $`sigma squared`
##           [,1]
## [1,] 0.5084436
```

Looking at the values for phi and the value for sigma squared, we observe that the estimates are quite close to the values that we used to simulate the process, so it seems that the function works as expected. To check further we can simulate an AR(4) process, and then check if the estimates are accurate. Again we need to set some initial values:

```
n = 1000
sigsq = 1.3
mu = 0
phi = c(0.8, 0.1, -0.4, 0.3)
g = my_arma(n, sigsq, mu, phi)
```

```
## Process is stationary.
```

Now we can again use the function from part a to estimate phi and sigma:

```
par_est_func(g,4)
```

```
## $phi
##              [,1]
## [1,]  0.78437396
## [2,]  0.09328497
## [3,] -0.36280763
## [4,]  0.28536641
##
## $`sigma squared`
##          [,1]
## [1,] 1.23942
```

Just like before, the estimates for phi and sigma squared generated by the function are quite close to the values used to simulate the process, so the function works just fine.

Question 4

In order to simulate from a DLM function we need to simulate values from a multivariate normal, so we first need to install the MASS package and load it using the command below:

```
library(MASS)
```

The function that simulates form a DLM can be found below:

```
my_DLM = function(f, G, W, V, n, m_0, C_0){
  theta = matrix(ncol = 2, nrow = n+1, byrow = TRUE)
  theta[1,] = as.vector(mvrnorm(1, m_0, C_0))
  for (t in 2:(n+1)){
      theta[t,] = as.vector(G%*%theta[(t-1),]) + mvrnorm(1, m_0, W)
  }
  y = numeric(n)
  for(row in 2:(n+1)){
    y[row] = f%*%theta[row,] +  rnorm(1, 0, sqrt(V))
  }
    y = ts(y)
    plot(y)
}
```

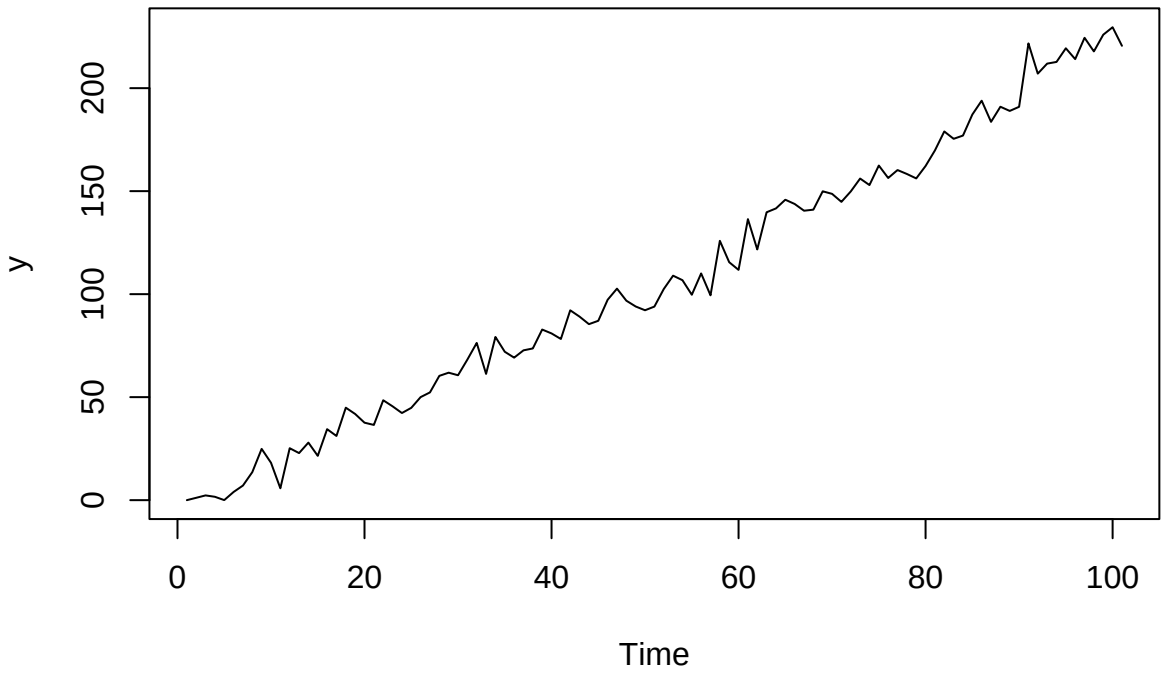
b)

Now in order to simulate a realization from a linear growth model, we first need to set some values:

```
m_0 = t(c(0,0))
C_0 = 10*diag(2)
W = diag(c(0.7, 0.05))
V = 35
n = 100
G = matrix(c(1,1,0,1),  nrow = 2, byrow = 2)
f = matrix(c(1,0), nrow = 1, ncol = 2)
```

Now calling the function and using the above values, generates the plot below:

```
my_DLM(f, G, W, V, n, m_0, C_0)
```

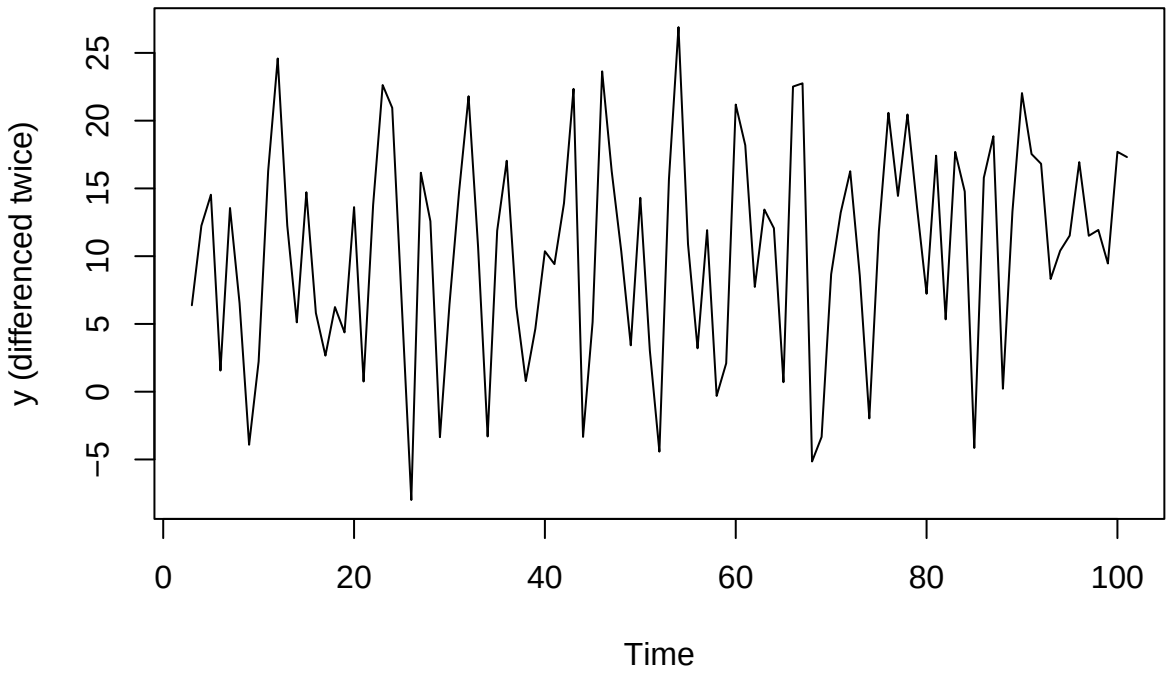


As expected we observe a non stationary process, that exhibits a clear linear trend. Now in order to get around this problem, we can difference that data twice and then generate the corresponding plot. In order to do that we just modify the function and call in again:

```
my_DLM = function(f, G, W, V, n, m_0, C_0){
  theta = matrix(ncol = 2, nrow = n+1, byrow = TRUE)
  theta[1,] = as.vector(mvrnorm(1, m_0, C_0))
  for (t in 2:(n+1)){
    theta[t,] = as.vector(G%*%theta[(t-1),]) + mvrnorm(1, m_0, W)
  }
  y = numeric(n)
  for(row in 2:(n+1)){
    y[row] = f%*%theta[row,] +  rnorm(1, 0, sqrt(V))
  }
  y = ts(y)
  plot(diff(y,2), ylab = "y (differenced twice)")
}
```

Now we just need to call the function:

```
my_DLM(f, G, W, V, n, m_0, C_0)
```



If we look at the plot now, we do not observe any evidence of non stationarity, as expected.

Finally we can find the estimates for the autocovariance and mean of the second differences. First we need to store the values for the second differences:

```
my_DLM_val = function(f, G, W, V, n, m_0, C_0){
  theta = matrix(ncol = 2, nrow = n+1, byrow = TRUE)
  theta[1,] = as.vector(mvrnorm(1, m_0, C_0))
  for (t in 2:(n+1)){
    theta[t,] = as.vector(G%*%theta[(t-1),]) + mvrnorm(1, m_0, W)
  }
  y = numeric(n)
  for(row in 2:(n+1)){
    y[row] = f%*%theta[row,] +  rnorm(1, 0, sqrt(V))
  }
  y = ts(y)
  return(diff(y,2))
}
second_dif = my_DLM_val(f, G, W, V, n, m_0, C_0)
```

For the autocovariance at lag k= 0,. . . , 5 we need the following comands:

```r
autocovs = acf(second_dif, lag.max = 5, type = "covariance", plot = FALSE)
autocovs_values = autocovs$acf
autocovs_values
```

```
## , , 1
##
##             [,1]
## [1,]  78.638589
## [2,]   4.403458
## [3,] -29.589378
## [4,]   4.111861
## [5,]  -9.470333
## [6,]  -6.221905
```

The mean can be also foud using the following command:

```r
mean(second_dif)
```

```
## [1] -0.03234417
```

Question 6

a)

The function that can be used to simulate from a VARMA(p,q) process, is written below:

```r
my_varma = function(n, sigma=1, mu=0, A=numeric(0), M=numeric(0)) {
  p = length(A)
  q = length(M)
  m = length(mu)
  # Establish initial values
  maxpq = max(c(p, q))
  inits = rep(0, maxpq)
  # Simulate from model
  x = matrix(nrow = n, ncol = m)
  x[1:maxpq,] = inits
  # simulate from a multivariate normal to get the errors
  e = mvrnorm(n,rep(0, m),sigma)
  for(t in (maxpq +1):n) {
    ar = rep(0,m)
    for(i in 1:p){
      ar = ar + A[[i]]%*%x[t-i,]
    }
    ma = rep(0, m)
    for(j in 1:q){
      ma = ma + M[[j]]%*%e[t-j,]
    }
    x[t,] = ar + ma + e[t,]
  }
  x = ts(x)
  return(x)
}
```

b)

Now in order to simulate a realization of the process, we first need to set some initial values. These are set as:

```
phi1 = matrix(c(0.8, -0.1, 0.1,0.85), nrow = 2, byrow = TRUE)
A = list(phi1)
theta1 = matrix(c(0.6,0.1,0.15,0.5), nrow = 2, byrow = TRUE)
theta2 = matrix(c(3, 0, -0.1, 0.2), nrow = 2, byrow = TRUE)
M = list(theta1, theta2)
sigma = matrix(c(1, 0.4, 0.4, 1.2), nrow = 2, byrow = TRUE)
mu = c(0,0)
n = 100
```

Now we can just call the function using the following command:

```
my_varma(n, sigma,mu, A, M)
```

Now we can find the sample mean vector, variance and covariance between Xt and Xt-1. The sample mean vector can be found as:

```
apply(my_varma(n, sigma,mu, A, M), 2, mean)
```

```
## Series 1 Series 2
## 1.899571 1.373296
```

The variance can be found using:

```
var((my_varma(n, sigma,mu, A, M)))
```

```
##          Series 1 Series 2
## Series 1 55.80224 14.88514
## Series 2 14.88514 14.10014
```

Finally, in order to find the covariance we need to remove the first and last column of the time series matrix and then we can find the covariance. This can be done with the following command:

```
varma_matrix = as.matrix(my_varma(n, sigma,mu, A, M))
cov(varma_matrix[-1,], varma_matrix[-100,])
```

```
##          Series 1 Series 2
## Series 1 51.53832 13.34700
## Series 2 20.32396 12.71403
```

UAS8382 ~ Summative report

③ $X_t - 2X_{t-1} + X_{t-2} = \phi_1 (X_{t-1} - 2X_{t-2} + X_{t-3}) + \varepsilon_t$

and $\phi_1 = 0.7$, $\sigma^2 = 0.09$, $X_n = 17.2$, $X_{n-1} = 16.5$
$X_{n-2} = 15.1$

a) $X_t = 2X_{t-1} - X_{t-2} + \phi_1 (X_{t-1} - 2X_{t-2} + X_{t-3}) + \varepsilon_t$

$\Leftrightarrow X_t = (2 + \phi_1) X_{t-1} - (1 + 2\phi_1) X_{t-2} + \phi_1 X_{t-3} + \varepsilon_t$

$\Leftrightarrow X_t = 2.7 X_{t-1} - 2.4 X_{t-2} + 0.7 X_{t-3} + \varepsilon_t$

Now: $X_{n+1} = 2.7 X_n - 2.4 X_{n-1} + 0.7 X_{n-2} + \varepsilon_{n+1}$
$= 2.7 \times 17.2 - 2.4 \times 16.5 + 0.7 \times 15.1 + \varepsilon_{n+1}$
$= 17.41 - \varepsilon_{n+1} = \hat{X}_n(1) + \varepsilon_{n+1}$

So: $E[X_{n+1}] = E[17.41 - \varepsilon_{n+1}]$
$= 17.41 - E[\varepsilon_{n+1}]$ → 0
$= 17.41$

$\text{Var}[X_{n+1}] = \text{Var}[17.41 - \varepsilon_{n+1}] = \text{Var}(-\varepsilon_{n+1})$
$= \text{Var}(\varepsilon_{n+1}) = \sigma^2 = 0.09$

Having found X_{n+1} we can find:

$X_{n+2} = 2.7 X_{n+1} - 2.4 X_n + 0.7 X_{n-1} + \varepsilon_{n+2}$
$= 2.7 (\hat{X}_n(1) + \varepsilon_{n+2}) - 2.4 X_n + 0.7 X_{n-1} + \varepsilon_{n+2}$
$= 2.7 \times 17.41 + 2.7 \varepsilon_{n+1} - 2.4 \times 17.2 + 0.7 \times 16.5 + \varepsilon_{n+2}$
$= 17.27 + 2.7 \varepsilon_{n+1} + \varepsilon_{n+2}$

So: $E[X_{n+2}] = E[17.27 + 2.7\varepsilon_{n+1} + \varepsilon_{n+2}]$

$= 17.27$

and $Var[X_{n+2}] = Var[17.27 + 2.7\varepsilon_{n+1} + \varepsilon_{n+2}]$

$= 2.7^2 Var(\varepsilon_{n+1}) + Var(\varepsilon_{n+2})$

$= (2.7^2 + 1)\sigma^2$

$= 0.7461$

So $\hat{x}_n(2) = 17.27 + 2.7\varepsilon_{n+1} + \varepsilon_{n+2}$

Now: $X_{n+3} = 2.7X_{n+2} - 2.4X_{n+1} + 0.7X_n + \varepsilon_{n+3}$

$= 2.7(17.27 + 2.7\varepsilon_{n+1} + \varepsilon_{n+2})$

$-2.4(17.41 + \varepsilon_{n+1}) + 0.7X_n + \varepsilon_{n+3}$

$= 2.7 \times 17.27 + 2.7^2\varepsilon_{n+1} + 2.7\varepsilon_{n+2} - 2.4 \times 17.41$

$- 2.4 \times \varepsilon_{n+1} + 0.7 \times 17.2 + \varepsilon_{n+3}$

$= 16.885 + 7.29\varepsilon_{n+1} + 2.7\varepsilon_{n+2} - 2.4\varepsilon_{n+1} + \varepsilon_{n+3}$

$= 16.885 + 4.89\varepsilon_{n+1} + 2.7\varepsilon_{n+2} + \varepsilon_{n+3}$

So $E[X_{n+3}] = E[16.885 + 4.89\varepsilon_{n+1} + 2.7\varepsilon_{n+2} + \varepsilon_{n+3}]$

$= 16.885$

and $Var[X_{n+3}] = Var[16.885 + 4.89\varepsilon_{n+1} + 2.7\varepsilon_{n+2} + \varepsilon_{n+3}]$

$= (4.89^2 + 2.7^2 + 1) \times \sigma^2$

$= (4.89^2 + 2.7^2 + 1) \times 0.09$

$= 2.898$

b) For $\hat{x}_n(1)$ the 95% C.I is:

$$E(\hat{X}_n(1)) \pm 1.96 \times \sqrt{Var(\hat{X}_n(1))}$$

$$= 17.41 \pm 1.96 \times \sqrt{0.09}$$

$$= (16.822, 17.998)$$

For $\hat{x}_n(2)$ the 95% C.I is:

$$E(\hat{X}_n(2)) \pm 1.96 \times \sqrt{Var(\hat{X}_n(2))}$$

$$= 17.27 \pm 1.96 \times \sqrt{0.7461}$$

$$= (15.577, 18.961)$$

For $\hat{x}_n(3)$ the 95% C.I is:

$$E(\hat{X}_n(3)) \pm 1.96 \times \sqrt{Var(\hat{X}_n(3))}$$

$$= 16.885 \pm 1.96 \times \sqrt{2.898}$$

$$= (13.548, 20.222)$$

⑤ $Y_t = F\theta_t + v_t$ where $F(1,0)$ $v_t \sim N(0, \sigma^2_V)$

$\theta_t = G\theta_{t-1} + \omega_t$ where $G = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ $\omega_t = \begin{pmatrix} \omega_{t1} \\ \omega_{t2} \end{pmatrix} \sim N(0, W)$

in which $W = \begin{pmatrix} \sigma^2_{\omega,1} & 0 \\ 0 & \sigma^2_{\omega,2} \end{pmatrix}$

and $m_n = (m_{n1}, m_{n2})^T$ and $C_n = \begin{pmatrix} C_{n11} & C_{n12} \\ C_{n12} & C_{n22} \end{pmatrix}$

a) Prediction step

We have:

$a_{n+1} = E(\theta_{n+1} | y_{1:n}) = Gm_n$
$R_{n+1} = Var(\theta_{n+1} | y_{1:n}) = GC_nG^T + W$

Now: $f_{n+1} = E(Y_{n+1} | y_{1:n})$
$= Fa_{n+1}$
$= FGm_n$
$= (1,0)\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} m_{n1} \\ m_{n2} \end{pmatrix}$
$= (1,1)\begin{pmatrix} m_{n1} \\ m_{n2} \end{pmatrix}$
$= m_{n1} + m_{n2}$ ∎

Also:

$$\begin{aligned}
C_{n+1} &= \mathrm{Var}(Y_{n+1} \mid y_{1:n}) \\
&= F R_{n+1} F^T + V \\
&= F(G C_n G^T + W) F^T + V \\
&= (1\ 0)\left[\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} C_{n11} & C_{n12} \\ C_{n12} & C_{n22} \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}\right]\begin{pmatrix} 1 \\ 0 \end{pmatrix} + V \\
&= (1\ 0)\left[\begin{pmatrix} C_{n11}+C_{n12} & C_{n12}+C_{n22} \\ C_{n12} & C_{n22} \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}\right]\begin{pmatrix} 1 \\ 0 \end{pmatrix} + V \\
&= (1\ 0)\begin{pmatrix} C_{n11}+C_{n22}+2C_{n12} & C_{n11}+C_{n12} \\ C_{n12}+C_{n22} & C_{n22} \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} + V \\
&= (C_{n11}+C_{n22}+2C_{n12},\ C_{n11}+C_{n12})\begin{pmatrix} 1 \\ 0 \end{pmatrix} + V \\
&= C_{n11}+C_{n22}+2C_{n12}+V \quad \blacksquare
\end{aligned}$$

b) We have that:

$$f_n(k) = F a_n(k),\quad a_n(0) = m_n,\quad a_n(k) = G a_n(k-1)$$

Now:

$$\begin{aligned}
a_n(1) &= G a_n(1-1) = G a_n(0) = G m_n = G(m_{n1}, m_{n2})^T \\
a_n(2) &= G a_n(2-1) = G a_n(1) = G \cdot G \cdot (m_{n1}, m_{n2})^T \\
&= G^2 (m_{n1}, m_{n2})^T \\
&\vdots \\
a_n(k-1) &= G a_n(k-1-1) = G a_n(k-2) = G^{k-1}(m_{n1}, m_{n2})^T \\
a_n(k) &= G a_n(k-1) = G^k (m_{n1}, m_{n2})^T
\end{aligned}$$

So: $f_n(k) = F \times G^k (m_{n1}, m_{n2})^T$

$= (1\ 0) \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}^k (m_{n1}, m_{n2})^T$

$= (1\ 0) \begin{pmatrix} 1 & k \\ 0 & 1 \end{pmatrix} (m_{n1}, m_{n2})^T$

$= (1\ k)\ (m_{n1}, m_{n2})^T$

$= m_{n1} + k m_{n2}$

We observe that the expectation of the k-step ahead forecast is a straight like with gradient m_{n2}.