Danmarks Tekniske Universitet

02417
Time Series Analysis

A1 - Forecasting apartment prices

Manuel Montoya Catalá (s162706)

September 2016

Contents

1 Question 1.1 **2**

2 Question 1.2 **2**

3 Question 1.3 **3**

4 Question 1.4 **4**

5 Question 1.5 **5**

6 Question 1.6 **7**

7 Question 1.7 **8**

8 R code **8**

1 Question 1.1

Plot the apartment price as a function of time.

In order to do so, we first load the CSV, then we obtain its variables as numeric vectors for convenience and plot the graph with the "plot" function. The plotting is improved using the parameters of the function and it is stored into a .png file.
The result is the following.

Figure 1: Apartment price

2 Question 1.2

Does the global mean value and standard deviation of the prices give a reasonable representation of the data? (The answer should be elaborated.)

No, the global mean value and standard deviation do not give a reasonable representation. They only do if we assume that the data points follow a Gaussian Distribution and they are independent (i.i.d gaussian distribution), which crearly does not happen in the figure. The are several recognizable trends and there is a strong correlation between the prices around a given timeframe.

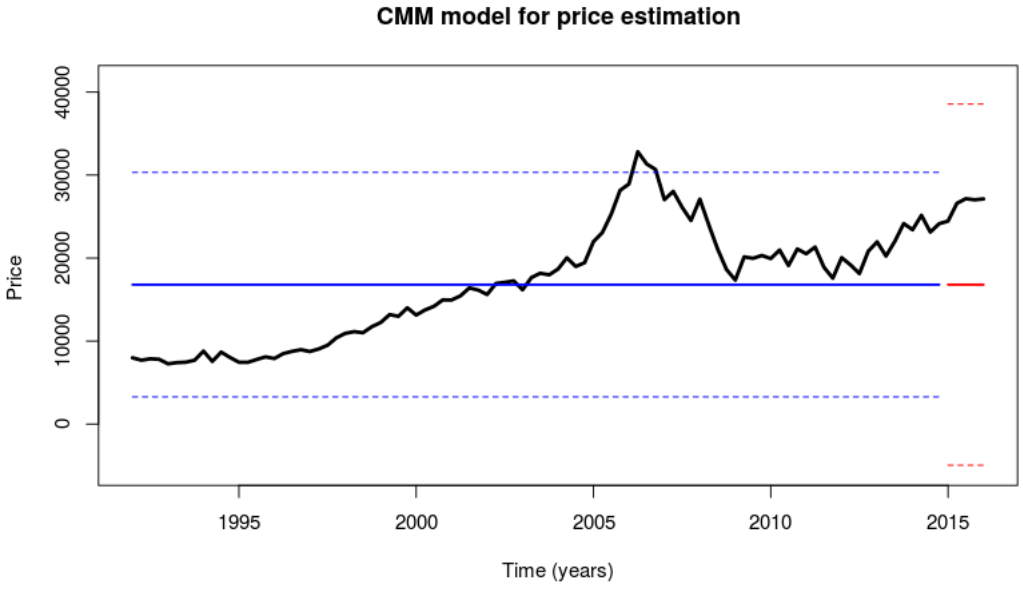

Figure 2: Constant Mean Model (CMM)

In the picture we can see the Maximum Likelihood (ML) estimation of the mean for the **training samples** (samples up to 2014) along with ± 2 times their unbiased theoretical standard deviation. As well as the

prediction for the **test samples** (samples happening after 2015) and their empirical standard deviation. Estimated training samples are in blue, and predicted test samples are in red.

Even though most samples fall inside the 2 standard deviation zone, this is just because the error is too high. Also the empirical test error is much higher than the train error, but since there are not many test samples, this is not very reliable.

Maybe the log-returns might follow a quasi-gaussian distribution but even then, we can observe that the variance changes over time, so we would need to use a heteroscedastic process to model it.

3 Question 1.3

Formulate a GLM model in form of a simple linear regression model for all the data. Estimate the model parameters and plot your fit. Would this model be useful for making predictions of future apartment prices?

The model we have chosen is a simple regression model where the price Y_t depends linearly on the time t plus a bias. We consider the noise of each sample indepent and identically distributed following a guassian distribution $\epsilon_t \sim \mathcal{N}(\mu, \sigma^2)$. The model follows the equation (3.2) from the reference book [1]:

$$\hat{Y_t} = \theta_0 + \theta_1 \cdot t + \epsilon_t$$

Of course we can also see the estimation of Y_t as a ML estimate of a proabilistic model where the samples have two dimensions, the bias (1) and time. Also we consider the samples independent and their error terms follow a guassian distribution with the same variance.

The parameters $\{\theta_0, \theta_1\}$ are obtained using the equation (3.5), that is, taking the Moore–Penrose pseudoinverse of the correlation between the dimensions of the samples and multiplying it by the desired outputs.

$$\hat{\theta} = (X \cdot X^T)^{-1} X^T \cdot Y$$

The next figure shows the estimations $\hat{Y_t}$ of the prices Y_t of the training samples along with ± 2 times their unbiased theoretical standard deviation in blue. As well as the prediction for the test samples and their empirical standard deviation in red.

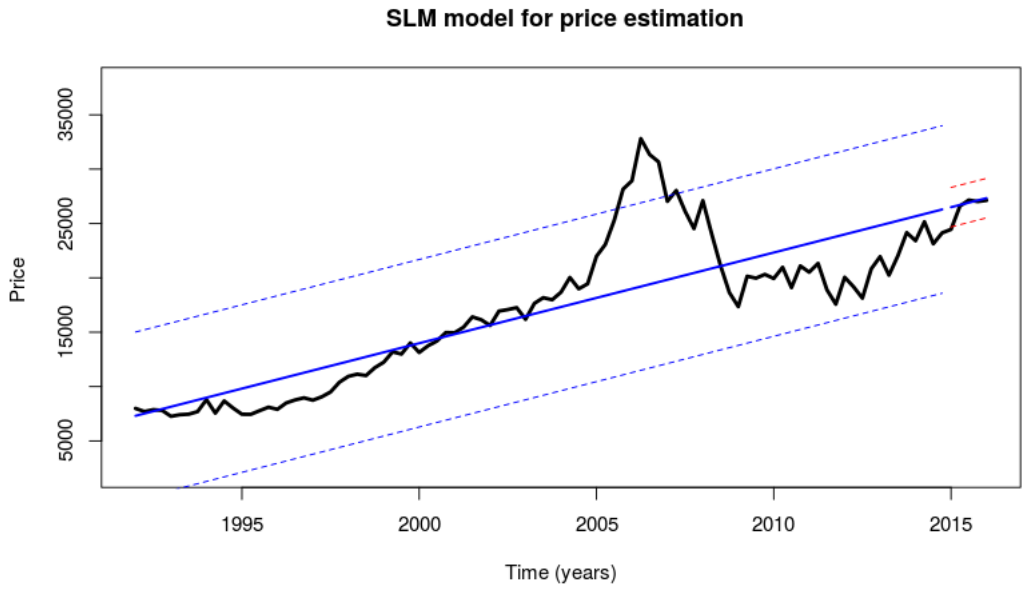

Figure 3: Simple Linear Model (SLM)

We can see that this model captures the linear trend of the data but it fails to capture higher order polinomial trends or local trends. It performs well in areas where the data follows the main linear trend and poorly otherwise.

As in the previous exercise, most training samples fall into the $\pm 2std$ interval but the variance of the error is still too high (but lower than in the previous model), maybe a more complex model could fit the distribution more precisely. The empirical test error is quite small, but again, since we do not have many samples, it is not realiable, it is just that the test samples happen close to the main trend.

4 Question 1.4

Now we will consider methods which considers data more locally. Use simple exponential smoothing with $\lambda = 0.8$ to predict the apartment prices for all of 2015 and 2016Q1. Plot the estimates along the entire series and make a table with the predictions. Comment on the results.

In this question, we will estimate the price for time the instance $N+1$, $\hat{Y}_{N+1}$, using a linear combination (weighted sum) of the price of all previous samples $\{Y_N, Y_{N-1}, ..., Y_1\}$. The weights will follow a decreasing exponential sequence, being $\lambda = 0.8$ the decreasing factor. The prediction of $\hat{Y}_{N+1}$ given all the past N samples can be written as in equation (3.70). Of course in the implementation, the efficient stepwise updating equation (3.75) is used.

$$\hat{Y}_{N+l|N} = c \sum_{j=0}^{N-1} \lambda^j Y_{N-j} = c[Y_N + \lambda Y_{N-1} + ... + \lambda^{N-1} Y_1]$$

The following image shows the smoothing prediction for all the time series for $\lambda = 0.8$. It can be observed that whenever a constant trend is stablished, this model performs a good prediction, but when the trend changes, it cannot follow the change inmediately, due to the memmory (lag) and then the error increases. Maybe a smaller value of λ would be better so that the prediction can follow the trend quicker.

In blue we can observe the estimation for the training samples and in red we can see the prediction for the test samples. For the test samples, the value of the predicted price $\hat{Y}_{N+l}$ also uses the previously predicted values $\{\hat{Y}_{N+1}, \hat{Y}_{N+2}, ..., \hat{Y}_{N+l-1}\}$ and that is why in the figure, the value of the prediction keeps growing after the first prediction. So in the case where we do not have Y_{N+1}, we calculate $\hat{Y}_{N+2}$ using its estimate, it is modification of equation (3.74).

$$\hat{Y}_{N+2|N+1} = (1-\lambda)\hat{Y}_{N+1} + \lambda \hat{Y}_{N+1|N}$$

This differs from the strict definition in (3.70) but I thought it would be a better approximation since the "trend" that it captures can evolve, instead of just give the same value for all test samples.
The estimation is a smoothed lagged representation of the original signal.

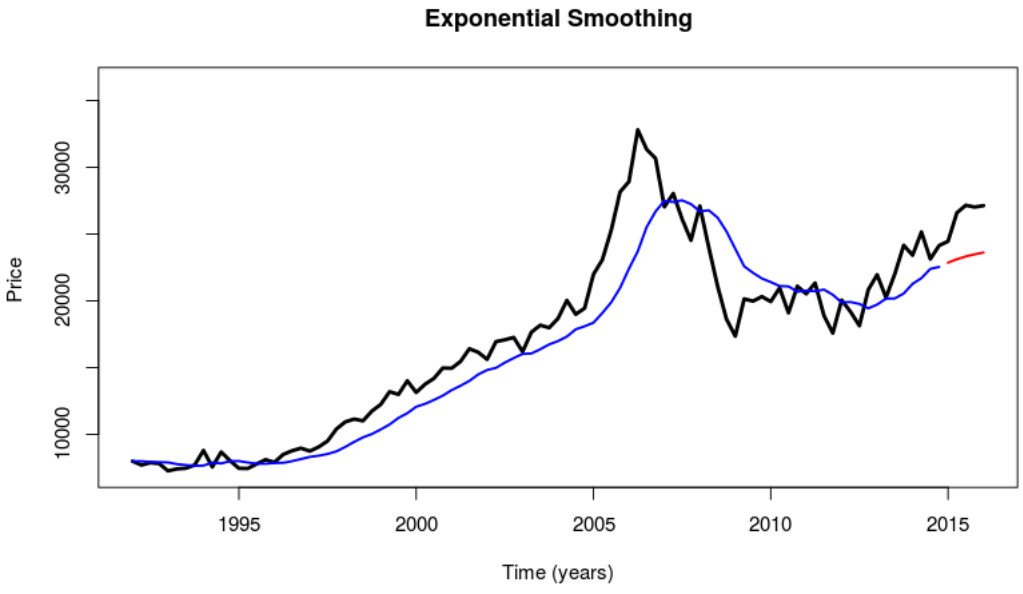

Figure 4: Exponential Smoothing

The following table shows the prediction values for the test samples. I think the figure is more informative than the table so no more comments are needed.

Time	2015	2015.25	2015.5	2015.75	2016
Prediction	22539.12	22859.49	23115.79	23320.84	23616.09
Real value	24458	26596	27155	27022	27128

Table 1: Prediction values for the Exponential Smoothing model

5 Question 1.5

Use a local linear trend model to predict the apartment prices for 2015 and 2016Q1. State the uncertainty of the predictions. Plot the estimates along the entire series and make a table with the predictions. Again use $\lambda = 0.8$ Comment on the results.

In this exercise we are using a Local Linear Trend Model to predict the apartment prices. In this model, for estimating a sample at time $N+1$, $\hat{Y}_{N+1}$, we fit a Weighted Least Squares Linear Regression model (WLS) to the N previous samples $\{Y_N, Y_{N-1}, ..., Y_1\}$ and use its parameters to predict the next samples. So the parameters $\{\theta_0, \theta_1\}$ will depend on the time instance. $\{\theta_0(N), \theta_1(N)\}$.

The WLS it a generalization of Least Square Regression, where the error contribution of each sample to $S(\theta)$, from equation (3.24) is weighted according a discrete distribution W. Expressed in matricial form, we derive equation (3.38) where Σ^{-1} has the weight vector W as its diagonal. In the exponential implementation, the weights of the samples follow a decreasing exponential sequence, where the closest samples have bigger values.

As stated in the reference book, in theorem 3.5, the solution of this model can be seen as the ML solution of a proabilistic model where we consider that the variance of the error estimates ϵ_N is different for every sample. The Inverse Covariance matrix of this probabilistic model Σ^{-1} is equal to the Weight Diagonal Matrix of the samples in the WLS model. The more weight we set a sample to have, the less variance it has in the probabilistic model.

So basically, for estimating sample at time $N+1$, $\hat{Y}_{N+1}$, we end up calculating the WLS model parameters using equation (3.38) with all previously seen samples.

$$\hat{\theta}_N = (X_N \cdot \Sigma^{-1} \cdot X_N^T)^{-1} X_N^T \cdot \Sigma^{-1} \cdot Y_N$$

Where X_N is a matrix with all the samples input values (a bias and time in this case) up to time N and Y_N is the desired output (price) of all samples up to time N. For performing estimating around sample N we just use the linear model so:

$$\hat{Y}_{N+1} = X_{N+1}^T \hat{\theta}_N$$

Of course in the book they need to complicate things a bit, so... making the assumption that all samples are separated the same time distance T, assuming that we do not lack any of them, and that the output only depends on a transformation of the time t we can derive equation (3.80). In this new notation, we perform a transformation of time, so that every sample is assigned a virtual time index j and its transformation is $f(j)$, so the prediction notation now becomes equation (3.80). In this transformation, the time instant $j = 0$ is assigned to the last observed sample N. We will only be using the Linear Trend Model given by equation (3.83).

Under this assumptions, you can derive some beautiful Updating rules for $\hat{\theta}_{N+1}$ given $\hat{\theta}_N$ and Y_{N+1} in equations (3.93,3.94,3.95). In the code I tried hard to implement this more computationally efficient way to compute the evolving parameters of the model but I could not get it to work, so I ended up having to recalculate $\hat{\theta}_N$ for every time instance N, I used the $f^T(j)$ transformation though.

The next figure shows the estimate of the training samples in blue, along with 2 times the standard deviation of the prediction calculated from equation (3.91).

$$Var(\epsilon_{N+l}) = \hat{\sigma}^2(1 + f^T(l) F_N^{-1} f(l))$$

Being $\hat{\sigma}^2$ the unbiased estimate of the variance for the training samples.

$$\hat{\sigma}^2 = (Y_N - X_N \hat{\theta}_N)^T \Sigma^{-1} (Y - X_N \hat{\theta}_N)/(N - p)$$

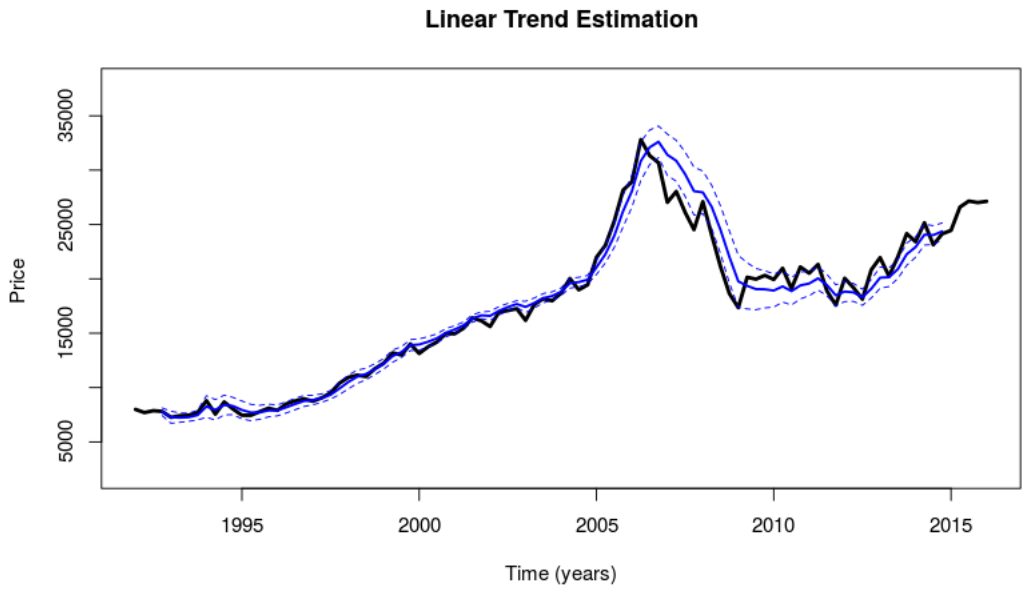

Figure 5: Estimation of price for training samples using Linear Trend Model

We can observe that whenever the trend quickly changes, the error of the estimate and its theoretical variance increases which is nice property to have. The system seems overconfident sometimes, being the variance of the estimation quite small.

In the next figure the prediction values for the test samples along with their standard deviation are plotted in red. Since we do not use the test samples to update the model, the model is just the linear regression system calculated using all the training samples. We can also see how the variance of the prediction estimate linearly increases as time grows, this was estimated using equation (3.90).

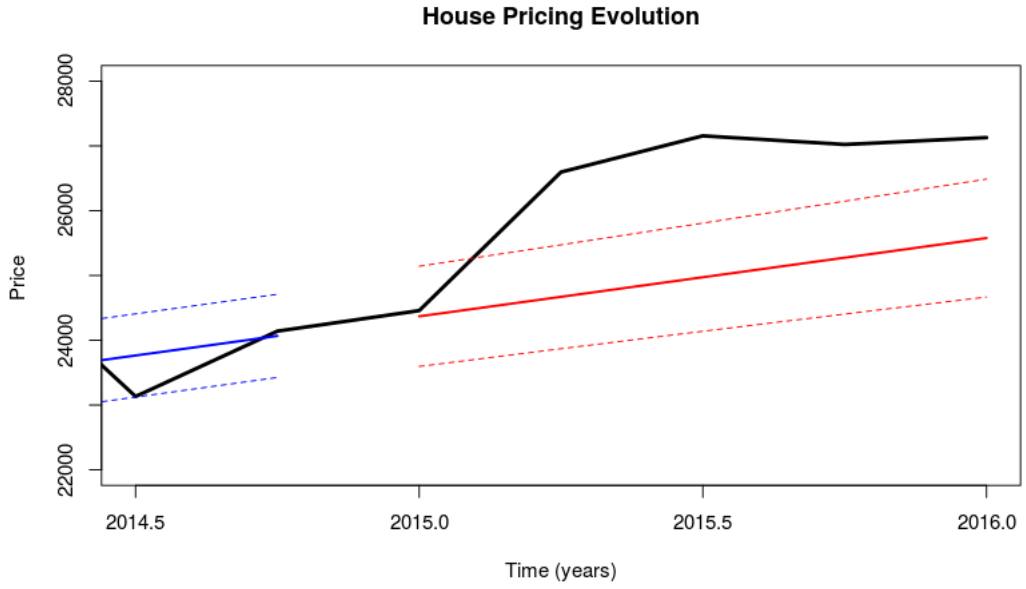

Figure 6: Prediction of price using Linear Trend Model

We can also observe that, exept for the first test sample, the test samples do not fall into the 2 times standard deviation area, which means that the model is not reliable for them. This makes sense since, if we do not keep updating the trend with test samples, and the trend changes, we will be following a false trend.

The next table shows the values for the predictions and their standard deviation derived from equation (3.90). As we can see, the standard deviation of the prediction grows as time advances.

Time	2015	2015.25	2015.5	2015.75	2016
Prediction value	24369.49	24671.49	24973.48	25275.48	25577.48
Prediction std	386.5296	400.9231	417.294	435.4193	455.0895
Real value	24458	26596	27155	27022	27128

Table 2: Prediction values and std for Local Trend model

The results are different from the ones obtained in the last question. Even though in both methods we weight the importance of the samples in the same manner, the first one only computes the weighted bias for the prediction. We can also see how the standard deviation grows with time given by equation (3.90).

6 Question 1.6

Find an optimal value of the forgetting factor for use in the local trend model suggested in the previous question. (Optimize 1-step predictions. And disregard the first 20 1-step predictions as burn in period.) Optional addon: What is the optimal value if only using data prior to 2006.

For solving this exercise we will perform an exaustive search for finding the optimal value of the forgetting factor λ that minimizes the objective function $S(\lambda)$ given by equation (3.79).

$$\hat{\lambda} = arg\min_{\lambda} S(\lambda)$$

Where the objective function $S(\lambda)$ is the sum of the square error ϵ^2_{N+1} for the training samples from $N = 20$. The error is given by equation (3.78) and the prediction is made using the Linear Trend Model from the previous question.

$$S(\lambda) = \sum_{N=20}^{Ntr} \epsilon^2_{N+1} = \sum_{N=20}^{Ntr} (Y_{N+1} - \hat{Y}_{N+1|N}(\lambda))^2 = \sum_{N=20}^{Ntr} (Y_{N+1} - X^T_{N+1}\hat{\theta}_N(\lambda))^2$$

The next figure shows the error estimate $S(\lambda)$ to minimize for different values of λ. We can see that the error is convex with respect to λ and the minimum is around $\lambda = 0.47$. We can see that the model penalizes more big values of λ than small values of λ, which could mean that the samples are very correlated to their nearby samples and they do not depend so much on older samples.

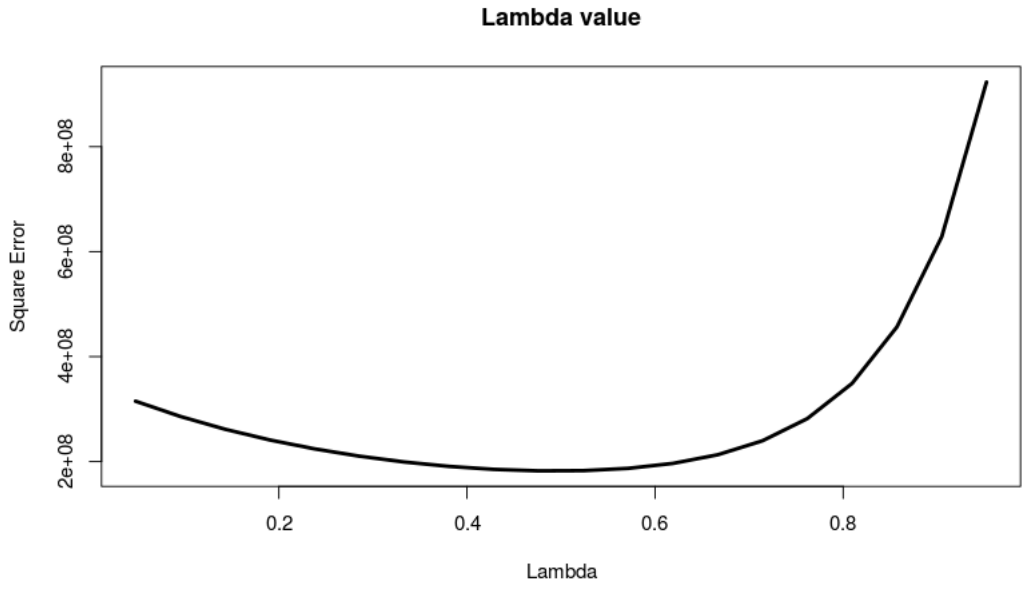

Figure 7: Square Error as a function of lambda

7 Question 1.7

Comment on the results. Which model do you prefer? Do you trust the forecasts? Do you have ideas for extending the forecast method?

We have seen a few linear methods to try to model the time series, what they do is basically try to capture some noisy linear trend (locally or globally) in the time series and estimate/predict the next samples using that model. Due to the high correlation between samples, a good prediction for the next sample $\hat{Y}_{N+1}$ is just saying that it is equal to the last sample seen Y_N, so a model that does a linear approximation of the time series around Y_N to predict Y_{N+1} should work fairly good because it can capture these relationships.

The model I prefer is the last one with the optimized λ value, because it is a model where the prediction depends heavely in the few previous samples. But I would not trust the estimated, first because of overfitting the training samples and second because the variance of the noise seems quite low so this model is too confident that a given local trend will continue in future, the variance of the predictions grows very slowly as time passes. The noisy nature of prices evolution cannot assume that such a trend will be followed far in the future.

Maybe one way of improving the models in questions 4 and 5 is to use weight values that depend on the lagged correlation between the samples. Due to the high correlation between samples, maybe working with the returns of the samples and not the samples themselves could be more informative.

8 R code

This assignment has been entirely coded in R, since some variables are reused from one Question to the next, here I presen the code for all questions jointly. The code is fully commented and easy to follow.

```
# Power of a matrix function
powA = function(a,n) {
  if (n < 0){
    n = -n
    a = solve(a)
  }
  if (n==1)  return (a)
  if (n==2)  return (a%*%a)
  if (n>2) return ( a%*%powA(a,n-1))
}

################ Question 1 ######################
## Loading and preparing the data:
myData = read.csv("./apartment_prices.csv", sep = " ") # read the csv
time = as.matrix(myData[,2])   # Get the time
price = as.matrix(myData[,3])  # Get the price
Nsam = dim(time)[1]             # Number of samples
# Vars for plotting
minPrice = min(price)
maxPrice = max(price)
rangePrice = maxPrice - minPrice
### Split data into train and test
tr_indx = as.matrix(which(time[,1] < 2015))  # Indexes for estimating
Ntr = dim(tr_indx)[1]

tst_indx = (Ntr+1):Nsam
Ntst = Nsam - Ntr

# Obtain data
time_tr = as.matrix(time[tr_indx,1])
time_tst = as.matrix(time[tst_indx,1])
price_tr = as.matrix(price[tr_indx,1])
price_tst = as.matrix(price[tst_indx,1])

## Do the plotting and saving it into an image
# Save the image in this file, with res =
png(file="Price_time.png",width=800, res=90)
# Size and aspect ratio could be width=400,height=350,res=45
plot(time, price,
     type = "l",                    # Draw as a line
     lwd= 3,                        # Line width
     main="Apartment prices", # Title of the graph
     xlab="Time (years)",           # x label
     ylab="Price (Kr.)")                 # y label
dev.off()

################ Question 2 ######################
# Obtain the mean and the std for estimation
mu = mean(price_tr)               # Calculate mean
Ytr = matrix(mu, Ntr)            # Output
error_tr = Ytr - price_tr      # Obtain error estimate
```

```
# Obtain theoretical error std
std_error_tr = sqrt(t(error_tr)%*%error_tr / (Ntr - 1))


# Obtain prediction and empirical std of prediction
Ytst = matrix(mu, Ntst)          # Obtain output prediction
error_tst = Ytst - price_tst      # Obtain error estimate
error_tst = Ytst - price_tst      # Obtain error estimate

# Obtain empirical error std
std_error_tst = sqrt(t(error_tst)%*%error_tst / (Ntst - 1))

## Do the plotting and saving it into an image
png(file="Mean_Model.png",width=800, res=90)  # Save the image in this file, with res =
plot(time, price, type = "l",lwd= 3,main="CMM model for price estimation",
     xlab="Time (years)", ylab="Price ",
     ylim = c(minPrice - rangePrice/2, maxPrice + rangePrice/3))

## Plot mean +- 2std lines
## Train prediction and std of tr error
lines(time_tr, Ytr,col = "blue", lwd=2) # Extend the values to create vector and plot
lines(time_tr, Ytr + 2 * matrix(std_error_tr, Ntr), col = "blue", type = "l", lty=2, lwd=1)
lines(time_tr, Ytr - 2 * matrix(std_error_tr, Ntr), col = "blue", type = "l", lty=2, lwd=1)

## Test prediction and std of tst empirical error
lines(time_tst, Ytst,col = "red", lwd=2) # Extend the values to create vector and plot
lines(time_tst, Ytst + 2 * matrix(std_error_tst, Ntst), col = "red", type = "l", lty=2, lwd=1)
lines(time_tst, Ytst - 2 * matrix(std_error_tst, Ntst), col = "red", type = "l", lty=2, lwd=1)
dev.off()

################ Question 3 ######################
# Use a linear solver as y = w0 + w1 * t + e to predict the samples
Xtr <- cbind(matrix(1,Ntr),time_tr)   # Add a vector of 1s for the bias
thetahat <- solve(t(Xtr) %*% Xtr) %*% t(Xtr) %*% price_tr  # Calculate regression coefficients

Ytr = Xtr %*% thetahat        # Obtain output prediction
error_tr = Ytr - price_tr      # Obtain error estimate

# Obtain the unbiased estimator of tr error (estimation error). error_tr = [92,1]
NdimParam = dim(thetahat)[1]
std_error_tr = sqrt(t(error_tr)%*%error_tr / (Ntr - NdimParam))  # Obtain error std

# Obtain test estimates (prediction estimates):
Xtst = cbind(matrix(1,Ntst),time_tst)
Ytst = Xtst %*% thetahat        # Obtain output prediction
error_tst = Ytst - price_tst      # Obtain error estimate
error_tst = Ytst - price_tst      # Obtain error estimate

std_error_tst = sd(error_tst, na.rm = FALSE)   # Obtain empirical error std

# Plot the predicted and the real data.
## Do the plotting and saving it into an image
png(file="Linear_Model.png",width=800, res=90)  # Save the image in this file, with res =
plot(time, price, type = "l",lwd= 3,main=" SLM model for price estimation", xlab="Time (years)",
     ylab="Price ", ylim = c(minPrice - rangePrice/5, maxPrice + rangePrice/5))

## Plot mean +- 2std lines
lines(time_tr, Ytr,col = "blue", lwd=2) # Extend the values to create vector and plot
lines(time_tr, Ytr + 2 * matrix(std_error_tr, Ntr), col = "blue", type = "l", lty=2, lwd=1)
lines(time_tr, Ytr - 2 * matrix(std_error_tr, Ntr), col = "blue", type = "l", lty=2, lwd=1)
## Test prediction and std of tst empirical error
lines(time_tst, Ytst,col = "blue", lwd=2) # Extend the values to create vector and plot
lines(time_tst, Ytst + 2 * matrix(std_error_tst, Ntst), col = "red", type = "l", lty=2, lwd=1)
lines(time_tst, Ytst - 2 * matrix(std_error_tst, Ntst), col = "red", type = "l", lty=2, lwd=1)
dev.off()

################ Question 4 ######################

# Exponential smoothing: We predict the value in instant N + 1 from a
# weighted average of the previous samples. The weights follow a exponential
# decreasing.
# Y^_N+1+1|N+1 = (1-lambda)Y_N+1 + Y^_N+1|N

lambda = 0.8
priceHatTr = matrix(price[1],1,1) #Initial value is the sample itself
for(i in 1:Ntr){
  yhat_i = (1-lambda)*price_tr[i]+lambda*priceHatTr[i]
  priceHatTr = cbind(priceHatTr, yhat_i)
}

# Predictions for data Test
last_Tr = price[Ntr]    # Last known price
priceHatTst = matrix(priceHatTr[Ntr],1,1)  # Price estimate for tst.
# First value will be removed after
for(i in 1:Ntst){
  pred = (1-lambda)*last_Tr+lambda*priceHatTst[i]
  priceHatTst = cbind(priceHatTst, pred)
}

## Plot the initial
```

```
png(file="Exp_Smoothing.png",width=800, res=90)  # Save the image in this file, with res =
plot(time, price, type = "l",lwd= 3,main="Exponential Smoothing", xlab="Time (years)", ylab="Price ",
     ylim = c(min(price)*(1- 1/10), max(price) + 200)*(1+ 1/10))
lines(time_tr, priceHatTr[1:Ntr], col="blue",lwd=2)

## Plot the predicted data !!
lines(time_tst, priceHatTst[-1],col="red",lwd=2)
dev.off()



#####################################################################
################ Question 5.1 Unefficient way ######################
#####################################################################

## Brute force, no updating.
# Calculate the estimation error in the training set
# Create "imaginary" time instances for ALL time instances
X_j = cbind(matrix(1,Nsam), matrix(-seq(Nsam-1,0,-1)))
lambda = 0.47

# Transition matrix
L = matrix(1,2,2)
L[1,2] = 0
# f(0): Imaginary time in instant 0
f0 = matrix(1,2,1) #f(0)
f0[2] = 0
fj = L %*% f0

## ESTIMATION (GET Initial parameters)
## Create weight matrix
Ninit = 3
Nest = Ntr - Ninit   # Number of samples to use for updating
# Main variables where to store the predictions and theoretical std at every point.
Ypred = matrix(1,Nest,1)
Ypred_std = matrix(1,Nest,1)
for (Nini in (Ninit:Ntr)){
  X_j = cbind(matrix(1,Nini), matrix(-seq(Nini-1,0,-1)))
  CovInv = matrix(0,Nini,Nini)  # Inverse covariance Matrix of the samples. (ML model)
  for(i in 0:(Nini-1)){
    CovInv[Nini-i,Nini-i] = lambda^(i)
  }
  # Solve the system to obtain the GLS parameters.
  # N * Covariance of the dimensions of f(x)_j = [1 j] (Imaginary time)
  FMatrix = t(X_j[1:Nini,]) %*% CovInv %*% X_j[1:Nini,]
  H = t(X_j[1:Nini,]) %*% CovInv %*% price_tr[1:Nini,]      # Samples * Weight_Matrix * Y
  thetahat = solve(FMatrix) %*% H  # Same as GLS
  Ytr = X_j %*% thetahat

  # Get training variance
  erorr_tr = price_tr[1:Nini] - Ytr
  Sigma2 = (t((erorr_tr)) %*%CovInv %*% (erorr_tr))
  Sigma2 = Sigma2 / (Nini - Ndim)

  # Get next test and its std
  Ytst = t(powA(L,1)%*%f0)%*%thetahat
  Ypred[Nini] = Ytst

  fl = powA(L,1) %*% f0
  var_error_tst = Sigma2 %*% (1 + t(fl)%*%solve(FMatrix)%*%fl)
  std_error_tst = sqrt(var_error_tst)

  Ypred_std[Nini] = std_error_tst
}

## Do the plotting and saving it into an image
png(file="LT_tr_est.png",width=800, res=90)  # Save the image in this file, with res =
plot(time, price, type = "l",lwd= 3,main="Linear Trend Estimation", xlab="Time (years)", ylab="Price ",
     ylim = c(minPrice - rangePrice/5, maxPrice + rangePrice/5))
# ylim = c(min(price)*(1- 1/10), max(price) + 200)*(1+ 1/10))
#ylim = c(24000,28000)

## Plot mean +- 2std lines
lines(time_tr[Ninit:Ntr+1], Ypred[Ninit:Ntr+1], col = "blue", lwd=2)
# Extend the values to create vector and plot
lines(time_tr[Ninit:Ntr+1], Ypred[Ninit:Ntr+1] - 2*Ypred_std[Ninit:Ntr+1] ,
      col = "blue", type = "l", lty=2, lwd=1)
lines(time_tr[Ninit:Ntr+1], Ypred[Ninit:Ntr+1] + 2*Ypred_std[Ninit:Ntr+1] ,
      col = "blue", type = "l", lty=2, lwd=1)
dev.off()

################ Question 5.2 Prediction ######################
# Create "imaginary" time instances
Xtr_j = cbind(matrix(1,Ntr), matrix(-seq(Ntr-1,0,-1)))
lambda = 0.80

# Transition matrix
L = matrix(1,2,2)
L[1,2] = 0
# f(0): Imaginary time in instant 0
f0 = matrix(1,2,1) #f(0)
```

```
f0[2] = 0

# Weight matrix for assigning weights to the samples.
# Older samples have decreasing weight using exponential decrease.
# It can be seen as WLS where Sigma inverse is a diagonal matrix.
# So it is also a ML for the Multivariate case of all samples, where
# all the samples are independent (diagonal matrix), but have differente
# variances (the more variance, the less weight )

## ESTIMATION (GET Initial parameters)
## Create weight matrix
CovInv = matrix(0,Ntr,Ntr)  # Inverse covariance Matrix of the samples. (ML model)
for(i in 0:(Ntr-1)){
  CovInv[Ntr-i,Ntr-i] = lambda^(i)
}
# Solve the system to obtain the GLS parameters.
# N * Covariance of the dimensions of f(x)_j = [1 j] (Imaginary time)
FMatrix = t(Xtr_j) %*% CovInv %*% Xtr_j
H = t(Xtr_j) %*% CovInv %*% price_tr      # Samples * Weight_Matrix * Y
thetahat = solve(FMatrix) %*% H  # Same as GLS
Ytr = Xtr_j %*% thetahat # Prediction for training

# The unbias estimator for the variance of the noise in the training samples is:
Ndim = dim(fj)[1]
Sigma2 = (t((price_tr - Ytr)) %*%CovInv %*% (price_tr - Ytr)) %/% (Ntr - Ndim)
std_error_tr = sqrt(Sigma2)

## PREDICTION (GET Initial parameters)
# Now we predict the next samples, using the linear model with respecto to j (time transform)
Ytst = matrix(t(fj)%*%thetahat,1,1)
# For every new sample we get the new time and obtain the output.
fj = f0  # fj for calculating fj for different js
for(i in 1:Ntst){
  fj = L%*%fj     # Obtain new time instance (transformation of time j)
  Ytst_j = t(fj)%*%thetahat
  Ytst = cbind(Ytst, Ytst_j)
}

## Std of the prediction error
# Tha variance for a prediction "l" samples away from the last training point is
var_error_tst = matrix(0,1,1)  # Price estimate for tst.
for(l in 1:Ntst){
  fl = powA(L,l) %*% f0
  var_error_tst_l = Sigma2 %*% (1 + t(fl)%*%solve(FMatrix)%*%fl)
  var_error_tst = cbind(var_error_tst, var_error_tst_l)
}
std_error_tst = sqrt(var_error_tst)

## Do the plotting and saving it into an image
png(file="LT_test.png",width=800, res=90)  # Save the image in this file, with res =
plot(time, price, type = "l",lwd= 3,main="House Pricing Evolution", xlab="Time (years)", ylab="Price ",
     ylim = c(22000,28000), xlim = c(2014.5,2016))
# ylim = c(min(price)*(1- 1/10), max(price) + 200)*(1+ 1/10))
#ylim = c(24000,28000)

## Plot mean +- 2std lines
lines(time_tr, Ytr,col = "blue", lwd=2) # Extend the values to create vector and plot
lines(time_tr, Ytr + 2 * matrix(std_error_tr, Ntr), col = "blue", type = "l", lty=2, lwd=1)
lines(time_tr, Ytr - 2 * matrix(std_error_tr, Ntr), col = "blue", type = "l", lty=2, lwd=1)
## Test prediction and std of tst empirical error
lines(time_tst, Ytst[-1],col = "red", lwd=2) # Extend the values to create vector and plot
lines(time_tst, Ytst[-1] + 2 * matrix(std_error_tst[-1], Ntst), col = "red", type = "l", lty=2, lwd=1)
lines(time_tst, Ytst[-1] - 2 * matrix(std_error_tst[-1], Ntst), col = "red", type = "l", lty=2, lwd=1)
dev.off()



######################################################################
################ Question 6. Unefficient way ####################
######################################################################

## Brute force, no updating.
# Calculate the estimation error in the training set
# Transition matrix

L = matrix(1,2,2)
L[1,2] = 0
# f(0): Imaginary time in instant 0
f0 = matrix(1,2,1) #f(0)
f0[2] = 0
fj = L %*% f0

# Search of lambda
Npoints = 20;
lambda_values = (1:Npoints)/(Npoints+1)
S_values = matrix(0,Npoints,1)   # Where to store the errors :)

Sindx = 0
for (lambda in (lambda_values)){
  Sindx = Sindx +1
  ## ESTIMATION (GET Initial parameters)
```

```
  ## Create weight matrix
  Ninit = 20
  Nest = Ntr - Ninit   # Number of samples to use for updating
  # Main variables where to store the predictions and theoretical std at every point.
  Ypred = matrix(1,Nest,1)
  Ypred_std = matrix(1,Nest,1)

  S_aux = 0
  for (Nini in (Ninit:(Ntr-1))){
    X_j = cbind(matrix(1,Nini), matrix(-seq(Nini-1,0,-1)))
    CovInv = matrix(0,Nini,Nini)  # Inverse covariance Matrix of the samples. (ML model)
    for(i in 0:(Nini-1)){
      CovInv[Nini-i,Nini-i] = lambda^(i)
    }
    # Solve the system to obtain the GLS parameters.
    # N * Covariance of the dimensions of f(x)_j = [1 j] (Imaginary time)
    FMatrix = t(X_j[1:Nini,]) %*% CovInv %*% X_j[1:Nini,]
    H = t(X_j[1:Nini,]) %*% CovInv %*% price_tr[1:Nini,]      # Samples * Weight_Matrix * Y
    thetahat = solve(FMatrix) %*% H  # Same as GLS
    Ytr = X_j %*% thetahat

    # Get next test and its std
    Ytst = t(powA(L,1)%*%f0)%*%thetahat

    S_aux = S_aux + (Ytst - price_tr[Nini+1])^(2)
    print (S_aux)
  }
  S_values[Sindx] = S_aux
}

best_lambda = lambda_values[which.min(S_values)]

## Do the plotting and saving it into an image
png(file="S_value.png",width=800, res=90)  # Save the image in this file, with res =
plot(lambda_values, S_values, type = "l",lwd= 3,main="Lambda value", xlab="Lambda", ylab="Square Error")

dev.off()

######################################################################
################ Question 5 Efficcient Update Misserably Failed ##
######################################################################

lambda = 0.99
# Transition matrix
L = matrix(1,2,2)
L[1,2] = 0
# f(0): Imaginary time in instant 0
f0 = matrix(1,2,1) #f(0)
f0[2] = 0
fj = L %*% f0
# Weight matrix for assigning weights to the samples.
# Older samples have decreasing weight using exponential decrease.
# It can be seen as WLS where Sigma inverse is a diagonal matrix.
# So it is also a ML for the Multivariate case of all samples, where
# all the samples are independent (diagonal matrix), but have differente
# variances (the more variance, the less weight )

## ESTIMATION (GET Initial parameters)
## Create weight matrix
Nini = 20; # Initial number of samples to perform the initial GLS
# Create "imaginary" time instances for ALL time instances
X_j = cbind(matrix(1,Nsam), matrix(-seq(Nsam-1,0,-1)))
CovInv = matrix(0,Nini,Nini)  # Inverse covariance Matrix of the samples. (ML model)
for(i in 0:(Nini-1)){
  CovInv[Nini-i,Nini-i] = lambda^(i)
}

# Solve the system to obtain the GLS parameters.
# N * Covariance of the dimensions of f(x)_j = [1 j] (Imaginary time)
FMatrix = t(X_j[1:Nini,]) %*% CovInv %*% X_j[1:Nini,]
H = t(X_j[1:Nini,]) %*% CovInv %*% price_tr[1:Nini,]      # Samples * Weight_Matrix * Y
thetahat = solve(FMatrix) %*% H  # Same as GLS
Ytr = X_j[1:Nini,] %*% thetahat # Prediction of the previous samples

# The unbias estimator for the variance of the last samples is:
Ndim = dim(fj)[1]
Sigma2 = (t((price[1:Nini,] - Ytr)) %*%CovInv %*% (price[1:Nini,] - Ytr)) %/% (Nini - Ndim)
std_error_tr = sqrt(Sigma2)

## Estimation of the next sample.
# Now we predict only the next sample, using the linear model with respecto to j (time transform)
Ytst = matrix(t(fj)%*%thetahat,1,1)
# For every new sample we get the new time and obtain the output.
fj = L%*%f0   # fj for calculating fj for different js
Ytst = t(fj)%*%thetahat

## Std of the prediction error
# Tha variance for a predictnion "l" samples away from the last training point is
fl = powA(L,l) %*% f0
var_error_tst_l = Sigma2 %*% (1 + t(fl)%*%FMatrix%*%fl)
std_error_tst = sqrt(var_error_tst)
```

```
#####################################################
##### UPDATING OF PARAMETERS WITH NEW SAMPLES #####
#####################################################

Nest = Ntr - Nini   # Number of samples to use for updating
# Main variables where to store the predictions and theoretical std at every point.
Ypred = matrix(1,Nest,1)
initialthetahat = thetahat
initialFMatrix = FMatrix
initialH = H

### REAL UPDATION
thetahat = initialthetahat
FMatrix = initialFMatrix
H = initialH

for(i in 1:(Nest)){   # For every new sample
  #  print(thetahat)
  fi = powA(L,1) %*% f0  # We update the 0 point when we update this
  #### Get the estimate for the sample (from the past updating)
  Ypred[i] = t(fi)%*%thetahat
  #  print (Ypred[i])
  #### UPDATE THE parameters with the new sample !
  # We update F, then H and then the parameters
  if (i < 5){
    print ("--------------------------")
    print(FMatrix)
    print (H)
    print (thetahat)
  }
  # Get the H and F
  fl = powA(L,-(Nini + i -1)) %*% f0
  FMatrix = FMatrix + lambda^(Nini + i -1) * fl%*%t(fl)
  H = lambda * solve(L) %*% H + f0 * price[Nini + i, ]
  thetahat = solve(FMatrix) %*% H  # Same as GLS
  # thetahat = t(L)%*%thetahat + solve(FMatrix) %*% f0 *(price[Nini + i, ] - Ypred[i])  # Same as GLS
}

## Do the plotting and saving it into an image
png(file="Local_Trend.png",width=800, res=90)  # Save the image in this file, with res =
plot(time, price, type = "l",lwd= 3,main="House Pricing Evolution", xlab="Time (years)", ylab="Price ")
# ylim = c(min(price)*(1- 1/10), max(price) + 200)*(1+ 1/10))
#ylim = c(24000,28000)

## Plot mean +- 2std lines
lines(time[Nini:(Ntr-1)], Ypred[,1] ,col = "blue", lwd=2) # Extend the values to create vector and plot
dev.off()
```