Danmarks Tekniske Universitet

02417
Time Series Analysis

A4 - Multivariate Processes

Manuel Montoya Catalá (s162706)

December 2016

Contents

The GPS chip in smartphones has some flavor of a Kalman filter onboard to reduce noise and potentially also to socalled fuse accelerations and gravitational measurements into the estimated position. To mimic what goes on inside you'll get data logged from a GPS with a little added noise. The noise that was added has a standard deviation of about 1.5m. To reduce numerical problems in your implementations the data has furthermore been transformed by:

$$mlat = (lat - 55.73) \cdot 1000 \qquad mlong = (long - 12.44) \cdot 1000$$

The data can be considered milli degrees around a point. You are expected to present the data on the transformed scale as it also eases the readability of plots. Sampling was done every second and 1000 samples are in A4_gps_log.csv. You are expected to both use a marima approach and your own implementation of the Kalman filter. The last 50 observations should be left out when fitting models and only be used to compare with predictions

1 Question 4.1: Presenting the data

Present the data. Both the temporal and spatial aspects should be presented. Do comment on what you see.

Solution

As always, the first thing that is done is loading the dataset with the appropiate format. Then the data samples are divided into estimation samples (train) and the samples for prediction (test), in this case, the prediction samples are the last 50 samples of the dataset. The test samples will not be shown since it could influence our decision on the system to use.

First thing we will plot is the individual time series of the latitude and longitude. As we can appreciate in the figure below:

- There is a clear trend in both signals, so they are not stationary. We cannot model the system with a MARMA directly, first we will need to transform the signals.
- The data points are apparently very correlated. Since both have a big linear trend, they could be expresed as a linear function of one another. This perfectly defines a linear relationship between both signals. But, correlation does not mean causality, so it would be wrong to model both signals as dependent of one another.

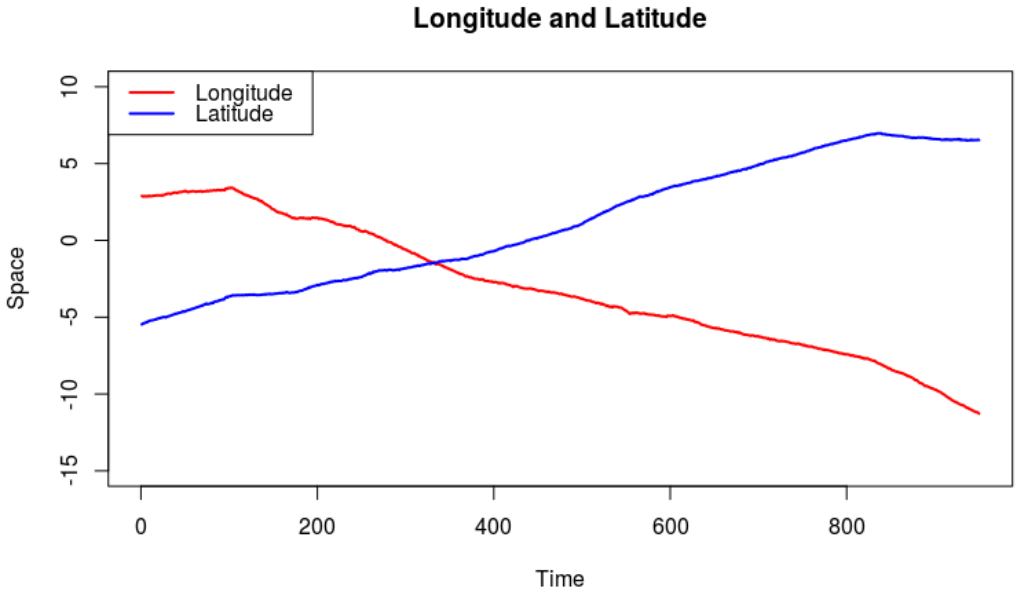

Figure 1: mlat and mlong samples for the training set in time

The next figure shows the spatial aspects of the signal, where the X-axis is the longitud and the Y-axis is the latitud, the initial upper-left point is the starting point. We can observe that:

- The data points seem to be highly correlated. We could express the lattitud as a linear regression of the longitud with a high decree of accuracy.
- We can still see the trend of course.

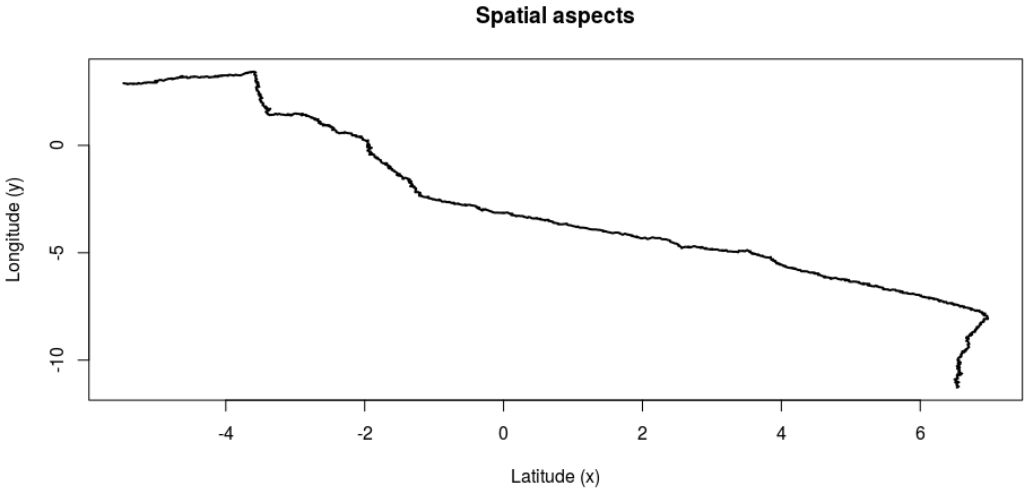

Figure 2: mlat and mlong samples for the training set in space

If we zoom into the signal, we are able to see that measurements of the positons have a noise component. Otherwise, the object is taking fast apparently random changes in direction. This can be better appreciated in the next image that shows the first 100 samples.

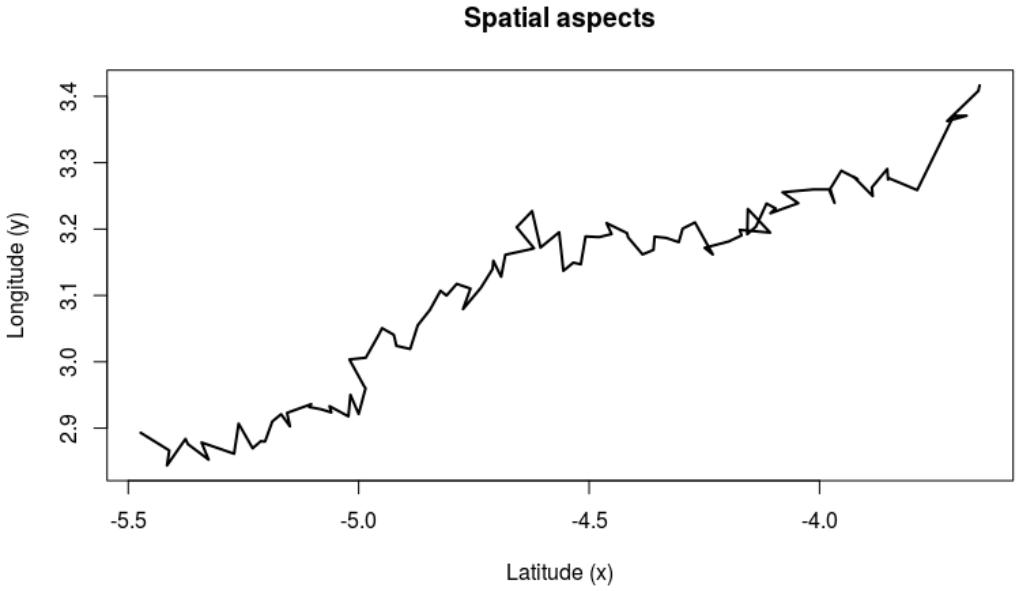

Figure 3: mlat and mlong few samples for the training set in space

Differences of the process. In order to obtain a stationary signal out of the raw data we perform a first order differentiation of both latitude and longitude. The resulting signals will be the velocity of the object at every given time instance.

The next figure shows the evolution of the object's speed with respect to time. We have also plotted a scattering of the relationship between both variables to see if there could be a pattern, such as both velocities being correlated. We can observe that:

- These signals do look like a stationary process.
- So we can fit a MARMA model to them.

- The velocities are apparently not correlated.

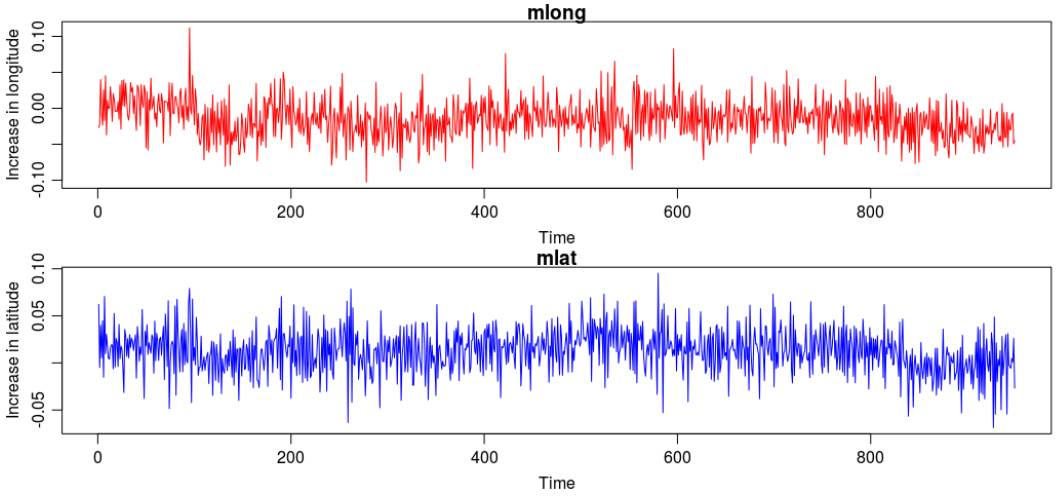

Figure 4: mlat and mlong instant velocities

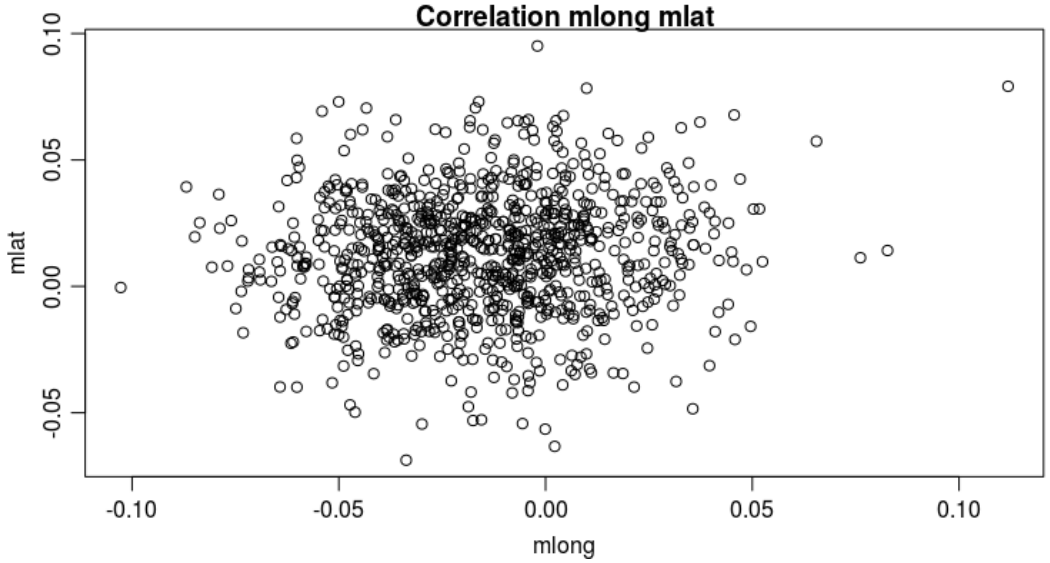

Figure 5: scatter plot of the velocities of mlong and mlat

We also ran an Augmented Dickey–Fuller test in R to test it using the function *adf.test()*. This method consists in a statistical test where the Null Hypothesis is H_0 = "The signal is stationary" by checking that the poles of the system lie within the unit circle in the Z domain. The p-value obtained for this test is $p = 0.19$ for the mlat signal and $p = 0.049$ for the mlong signal. So this test tell us that the mlat signal is not strictly stationary, but since its first order statistics are bounded, we can use it for the MARIMA model.

2 Question 4.2: ACF, PACF, CCF and PCCF

Calculate the crosscorrelation function and partial cross correlation function (Including ACF and PACF) for the data. If relevant also for differenced versions of the data.
Comment on the structures you find.

Solution

For the original signal of latitude and longitude we have the following ACF and PAFC shown in the next image shows with values up to lag 50. As we can observe:

- The ACF values of both signals do not decrease fast enough to 0, so the process is not stationary.
- The PACF is just one delta at lag 1. So we have a perfectly non-stationary AR(1) process.

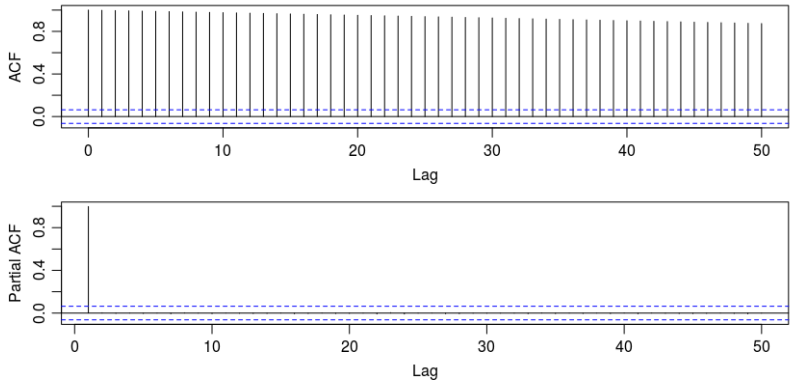

Figure 6: ACF and PACF coefficients for training values of the mlat signal

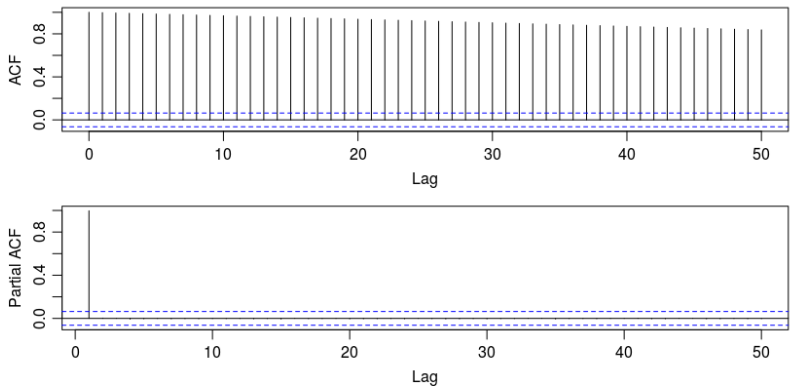

Figure 7: ACF and PACF coefficients for training values of the mlong signal

Regarding the relationship between these signals, we obtain the following CCF and PCCF. As we can observe:

- There is a high lagged crosscorrelation between both signals. This is expected since both signals follow a linear trend as explained earlier.
- The PCCF is not very informative. It presents random dampings which could mean that the signal is not stationary. It is quite non-symetric which suggests that the data is non-stationary.

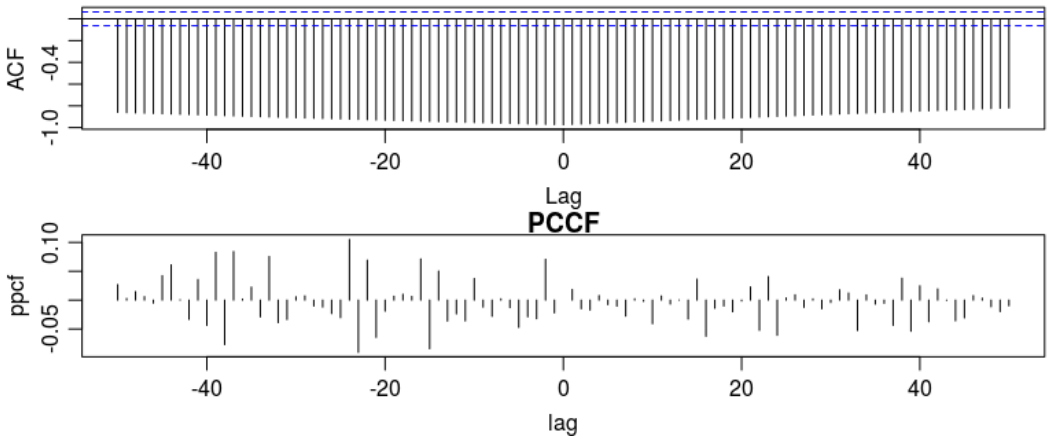

Figure 8: CCF and PCCF coefficients for training values of the positions

For the **differentiation of the signals** we get the next ACF and PACF coefficients. We can observe:

- The ACF values of both signals go fast to 0, so the process is stationary. In fact it is almost like white noise.
- The ACF values suggest that there is no MA component and the PACF values suggest an AR(5) model.

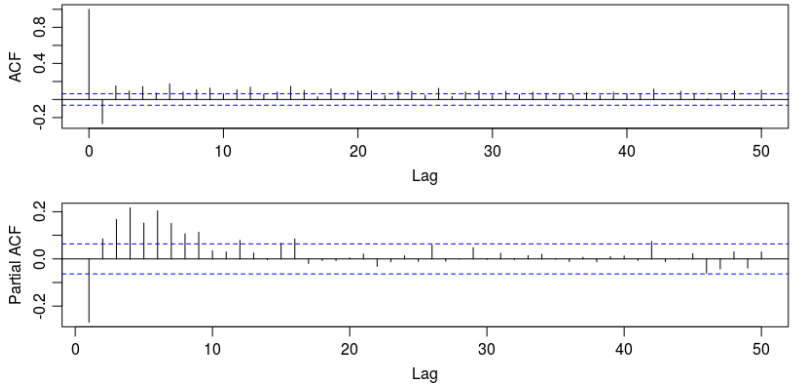

Figure 9: ACF and PACF coefficients for training values of the mlat signal

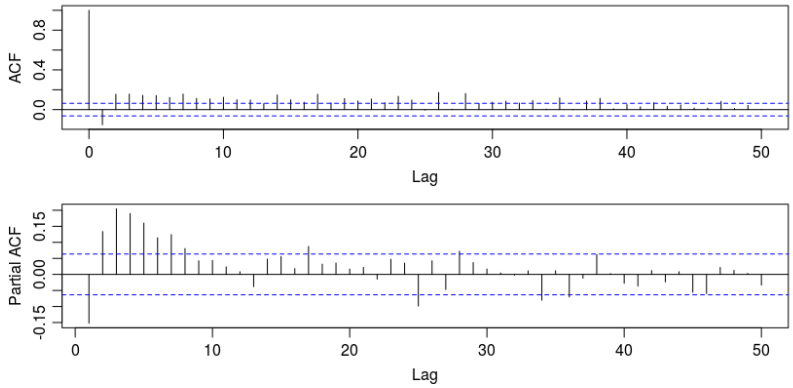

Figure 10: ACF and PACF coefficients for training values of the mlong signal

The CCF and PCCF of these velocities is plotted next. We can observe:

- The CCF of the signals seems random noise, so either they are very little correlated all the time or it is just noise.
- The PCCF suggests that there is an a small lag correlation of order up to 4, this could be because when there is a change in the position of the object, it usually affects both dimensions. It is quite symetric which suggests that the data is stationary.

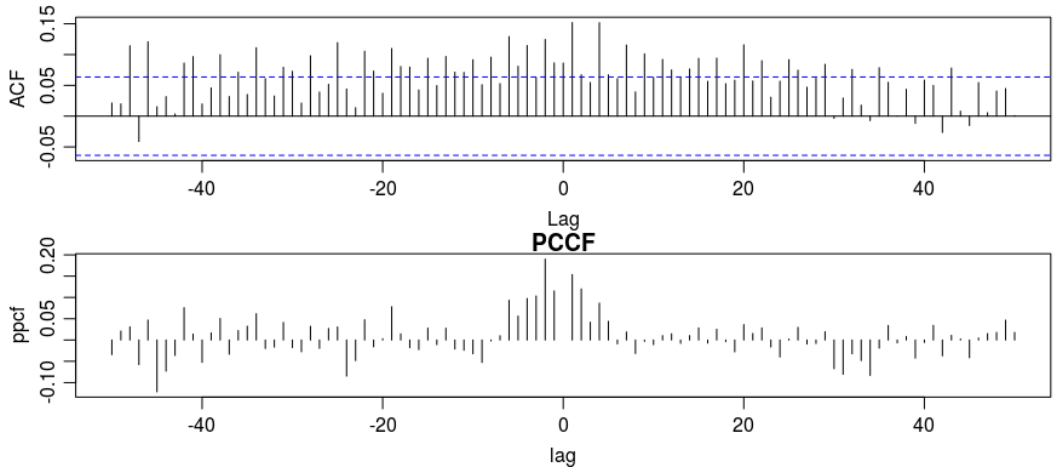

Figure 11: CCF and PCCF coefficients for training values of the differentiated signals

3 Question 4.3: MARIMA model

Find a suitable MARIMA model for the bivariate system consisting of mlat and mlong. Reduce the model and validate it. Then present the final model.

Solution

The initial MARIMA model proposed is a $MA((1,2,3),1,(1,1))$ model with all its parameters able to take non-zero values. In the implementation, we have first apply the differentiation of the signals and then applied the models, so the used model in practice is $MA((1,2,3),0,(1,1))$ over the velocities. This will require some more work when reconstructing the signal but nothing more. This model is considered big enough to capture all the potential relationships given the ACF and PACF. After choosing this model, the function *step.slow.marima.R* given by the professor is used in order to reduce the model based on the significance of the parameters, based on a penalty term.

Before using the initial model, let's see what happens if we put too many parameters that are actually clearly non-significant. The next graph shows the convergence for a $MA((1,2,3),0,(1,2))$. As we can see, it does not converge with the iterations so we cannot trust the parameters obtained.

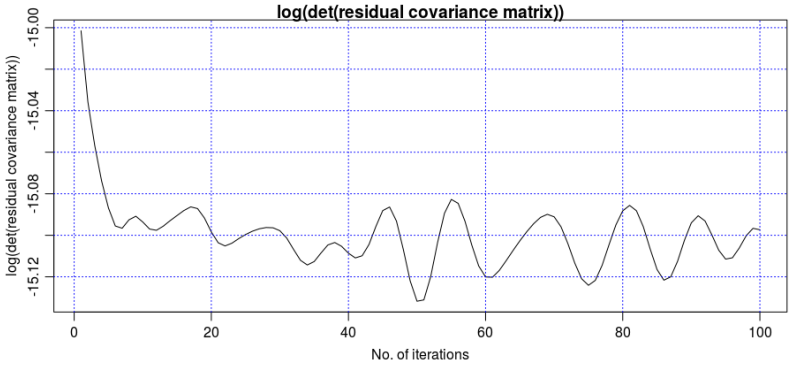

Figure 12: Convergence Model $MA((1,2,3),0,(1,2))$

The initial system that we choose then is $MA((1,2,3),0,(1))$. Even though, there is no sign of MA component, we introduce it and let the data speak for itself. Its convergence plot is shown next, as we can see, it does converge but some of the parameters are non-significative. The following summarizes the p-value of the parameters. Many of them are way bigger than 0.05 which means they probably are not significant.

```
P values of the AR parameters

Lag=1

      x1=y1    x2=y2
y1 0.000439 0.178303
```

```
y2 0.939375 0.000005

Lag=2

      x1=y1    x2=y2
y1 0.000003 0.621241
y2 0.764522 0.000003

Lag=3

      x1=y1    x2=y2
y1 0.014600 0.047515
y2 0.280264 0.003224

P values of the MA parameters

Lag=1

      x1=y1    x2=y2
y1 0.000000 0.576359
y2 0.812529 0.000000
```

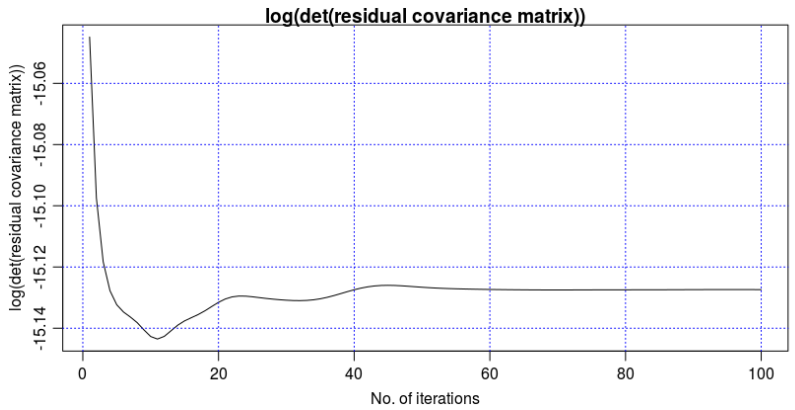

Figure 13: Convergence Model $MA((1,2,3),0,(1))$

After reducing de model, the final model is still of the form $MA(3,0,1)$ as general form but many of its parameters are set to 0. For example the MA component completely disappears. The following shows the significant values of the AR parameters. As we can see, only the AR(1) part presents dependance between the two signals, so the speeds are pretty independent.

```
values of the AR parameters of the final model.

, , Lag=1

      x1=y1     x2=y2
y1 0.275548 -0.129575
y2 0.000000  0.158754

, , Lag=2

       x1=y1     x2=y2
y1 -0.101548  0.000000
y2  0.000000 -0.162407

, , Lag=3

       x1=y1     x2=y2
```

```
y1 -0.144734  0.000000
y2  0.000000 -0.205651
```

The next image shows the convergence of the final MARIMA model.

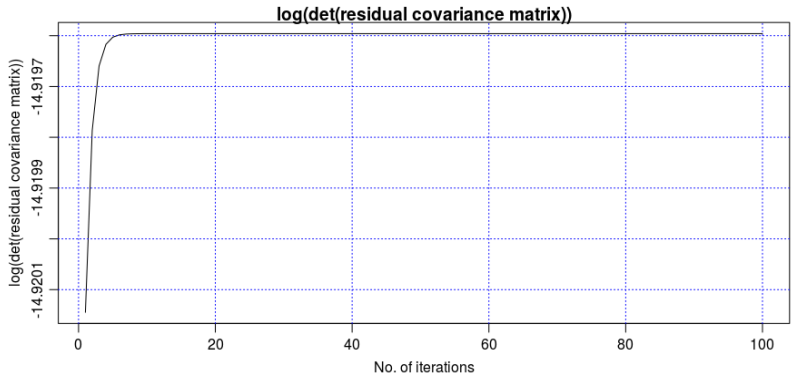

Figure 14: Convergence reduced Model $MA((1,2,3),0,(1))$

In the following, we will perform an analysis of the residuals for both lattitude and longitude sepately, in which we will boil down the residuals to a set of values that will give us different information about them.

About the residuals, we only have 949 samples because we re using the differentiated mlat and mlog (the velocities), also, the first 3 values of the residual are NAN, they cannot be calculate since we have an AR(3) model. Finally, the residual of the velocities are the same as the residual of the original signal since, transforming back would mean adding the previous value of the signal to both the prediction and the real value, and by performing the substraction, it will cancel out.

The next image shows the residuals obtained for the estimated parameters, along with their **ACF and PACF**. As we can see:

- We can observe in the residuals some basic noise (probably due to observation noise) and from time to time there are spikes (proably due to sudden changes in velocity)
- The ACF has not changed much, it is still the same as white noise. At least we did not get worse.
- The PACF is much better,but we still have some high values at high lags.
- The CCF and PCCF suggest that the residuals are not correlated.

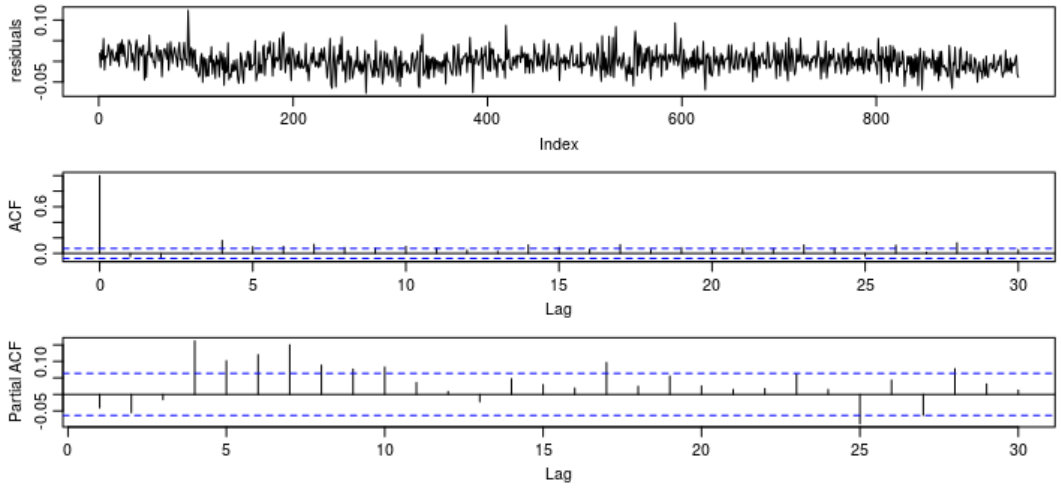

Figure 15: ACF and PACF coefficients of the mlat residual

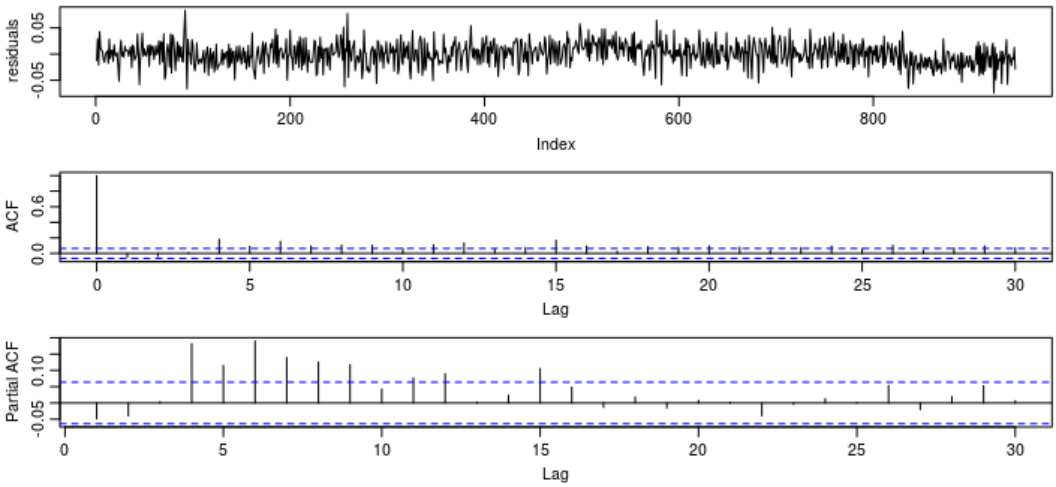

Figure 16: ACF and PACF coefficients of the mlat residual

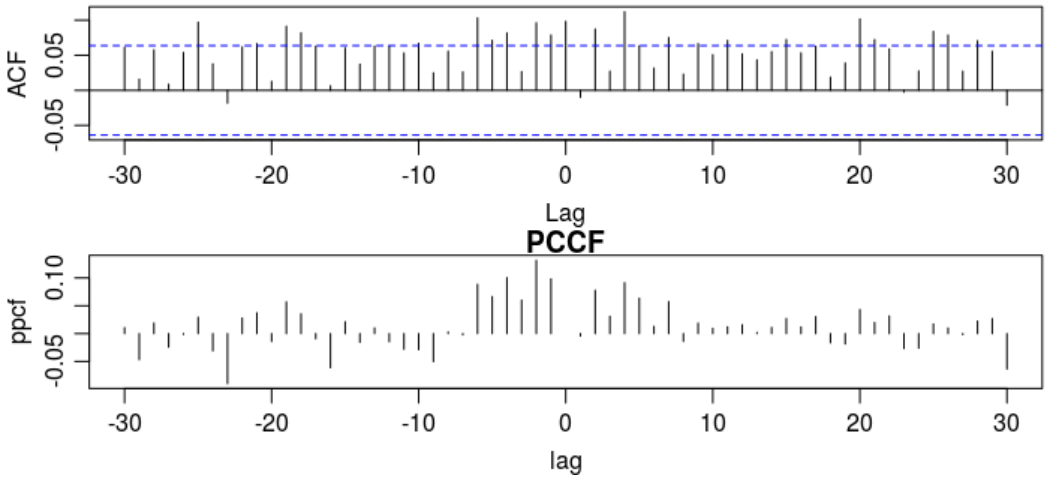

Figure 17: CCF and PCCF coefficients of the mlat mlong residual

In a White Noise process, the **probability of change in the sign** of the signal is 0.5 so we run a statistical test where the null hypotesis is that H_0: "The signal follows a White Process, and thus, the change of sign in the signal follow a Binomial distribution with probability of a change in sign $p = 0.5$". In this case, the p-value is of the mlat is $pv = 0.64$ which is way above the threshold so the residual is not white noise and the p value for the mlong is $pv = 0.037$ so it could be white noise.

If the residual is actually white noise, then the values of its autocorrelation function will follow a Gaussian distribution $\hat{\rho}_\epsilon(k) \sim N(0, 1/N)$ for all $k > 0$. So we can perform a statistical test where the null hypotesis is $H_0 =$ The signal is white noise, so the **autocorrelation values should be white noise** as well. Under this hypotesis, the sum of the squared estimated autocorrelation values should follow a χ^2_h distribution with $h = m - n$ degrees of freedom where m is the number of lags used and n the number of parameters of the model. For this model, using the autocorrelation values up to lag 15, the p-value for both residuals is $p = 0$, so we can reject the null hypotesis. Meaning that this test tells us that the residual can not be considered white noise.

The next figure shows the **Histogram and the QQ plot** of the residual. We can appreciate that the distribution has heavier tails than the normal distribution and even some outliers (points that are very unlikely under the gaussian assumption). So the noise distribution is not specially gaussian.

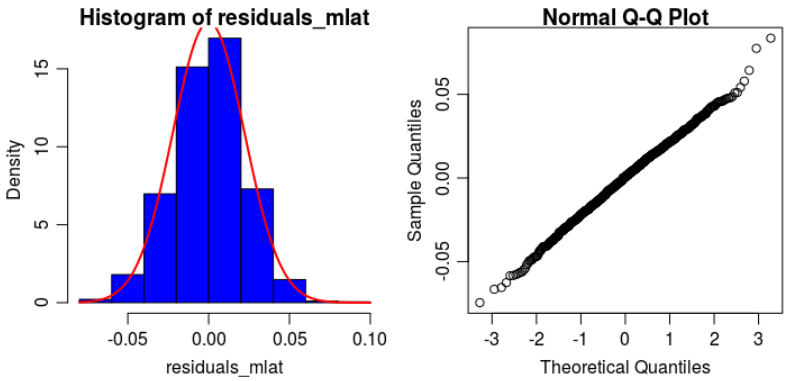

Figure 18: Histogram and QQ-plot of the residual of mlat

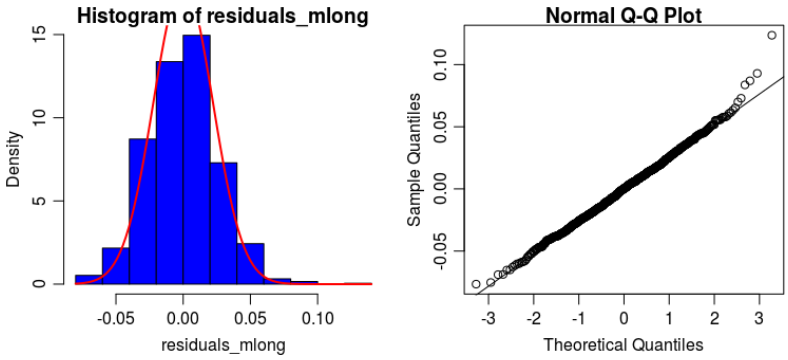

Figure 19: Histogram and QQ-plot of the residual of mlong

The next image shows the **cummulative periodogram** of the signal. We can express any signal as a sum sinusoidal functions of different frequency, modulus and phase using the Fourier Transform. The Fourier Transform of a White noise signal is a constant function of frequency, so the contribution of each frequency component is the same, thus, the cummulative sum of the contribution over the frequencies should be a straight line. The cummulative periodogram allows us to see how energy is distributed along the frequency components of our signal, and check if the signal has the frequency properties of white noise.

We can see that the energy of the frequency components of the original data is not evenly distributed so we can conclude (as we knew) that the original data is not white noise. The residuals do seem to follow the white noise distribution in the frequency domain, so according to this test the residual could be white noise. Too bad we have seen other tests that say otherwise.

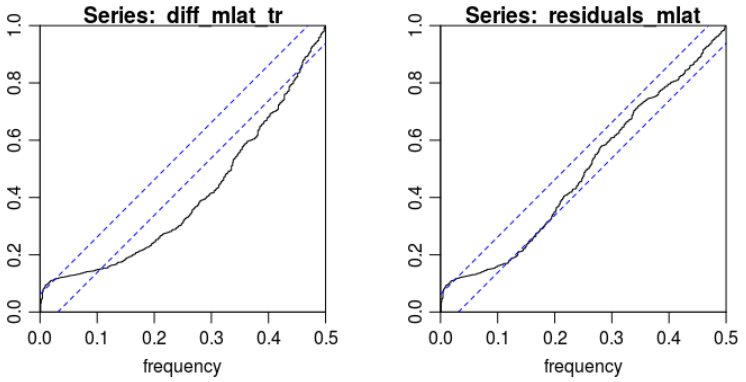

Figure 20: Cumulative periodogram of mlat and residual of mlat

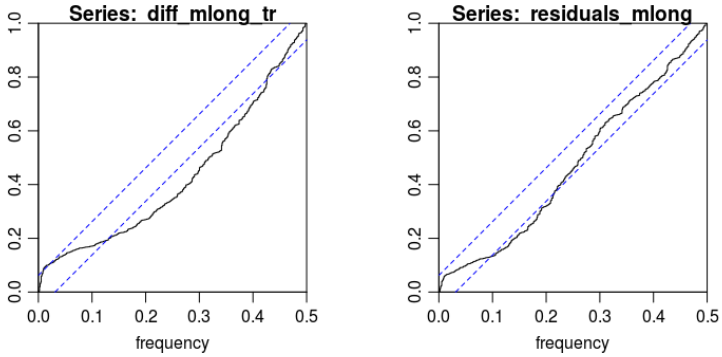

Figure 21: Cumulative periodogram of mlat and residual of mlong

Among these parameters, we can also find others that tell us more information about the system. Many of them are not very informative on their own but they can be informative compared to other models.
Variance of the Residual σ_ϵ^2: Estimated variance of the residual over the training samples.
A final parameter we will obtain for the process is the sum of the square error for the test samples. Since our model can suffer from overfitting, relying on the sum of squared error of the trianing samples is not a good practice, this error will decrease as we increase the number of parameters of our model but our validation error can increase. The mean square error for the test samples is:

$$S^{'}(\Theta)_{mlat} = 0.000488$$
$$S^{'}(\Theta)_{mlong} = 0.000677$$

Here are some other statistics:

- JB: 0.11587736
- Skewness: 0.12260113
- Kurtosis: 0.20094604
- arch.Chi-squared: 0.06155727
- serial.Chi-squared: 0.00000000

4 Question 4.4: Predicting with MARIMA model

Predict 50 steps ahead (From observation 950).
Plot the last part of the data and the predictions. Include a 95% prediction interval (Assuming the normal distribution). Include a table with the 1, 10, 25 and 50 step predictions.
Compare with the 50 observations that where left out and comment on the results.

Solution

The next image shows part of the training samples and the test samples in black and the predicted samples along with $\pm 1.96\hat{\sigma}_\epsilon$ in blue for the final MARIMA model, this area being the 95% probability confidence interval of the predictions. Since we are working with the velocities, instead of the real signal directly. In order to make the prediction we have to do a little transformation to the predicted velocities to obtain the predicted position and its standardar deviation. Basically, given the last traninig sample $x[i], i = 950$, we can obtain the position at time $i+1$, adding the velocity at position i, so we have $x[i+1] = x[i] + v[i]$. Notice that the velocity vector $v[n]$ has one less position, (we cannot differentiate the first sample with the previous one, we do not have it). So the i-th position of the $v[n]$ vector actually corresponds to $v[i] = x[i+1] - x[i]$ and so, all equations hold. To get the prediction at time $i+t$, we just have to add all previous predictions.

$$x[i+t] = x[i] + \sum_{n=0}^{t-1} v[i+n]$$

Assuming the predictions are gaussian random variables, the resulting prediction has as mean, the sum of the means, and as variance, the sum of the variances. Doing this properly, we obtain the predictions visualized in the next figures. We can observe:

- The variance of the prediction increases as $\sqrt{N}\hat{\sigma}_\epsilon$, where $\hat{\sigma}_\epsilon$ is the standard deviation of the residual. After the few first predictions of velocity, the variance of the prediction of the velocity saturates to the variance of the random process. Since the subsequent predictions are obtained adding up this random variables, we obtain this shape.
- After the last training sample, we can see the final direction of the prediction change a bit (because the system is an AR model and depends on several previous samples significantly). After that, the direction does not change. We can appreciate how the noise of the signal makes this last direction very unstable, and the last direction followed depends highly on this noise and not in the main direction.
- Due to this, the initial samples do fall into the confidence interval, but since we got the main direction wrong, as time passes, they do not anymore. The predictions of the system are too confident. They only focus on the variance itself of the velocity.

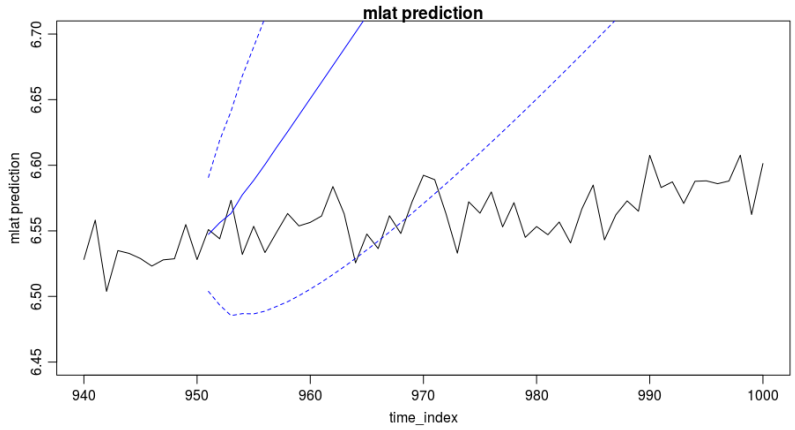

Figure 22: Predictions for the mlat signal

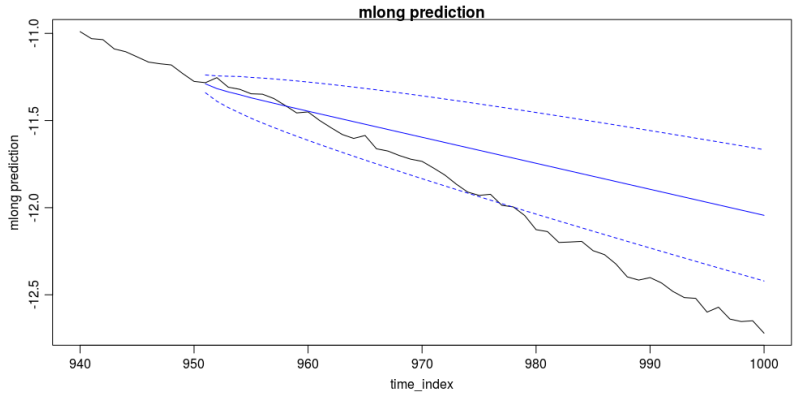

Figure 23: Predictions for the mlong signal

The next table shows the 1, 10, 25 and 50 step predictions, along with the standard deviation of the prediction and the 95% confidence interval. We can observe in it all the things said about the graphs previously.

Time	1	10	25	50
Real Value	6.551	6.5564	6.5634	6.6013
Prediction	6.547	6.6507	6.8403	7.1565
Std	0.02216	0.07409	0.1178	0.1669
Conf.Int	[6.5038,6.5907]	[6.5055,6.796]	[6.6094,7.0712]	[6.8293,7.4836]

Table 1: mlat predictions MARIMA model

Time	1	10	25	50
Real Value	-11.2826	-11.4503	-11.9293	-12.7202
Prediction	-11.2894	-11.4463	-11.6706	-12.0438
Std	0.02610	0.08537	0.1357	0.1922
Conf.Int	[-11.3406,-11.2383]	[-11.6137,-11.2790]	[-11.9365,-11.4046]	[-12.4206,-11.667]

Table 2: mlong predictions MARIMA model

5 Question 4.5: Kalman filter formulation

We choose the state vector $[x_t, v_{x,t}, y_t, v_{y,t}]^T$, where x_t represent the position and $v_{x,t}$ the velocity per sample in the x direction and similar for the y direction. The model for the x direction is given by:

$$x_t = x_{t-1} + v_{x,t-1} + \epsilon_{x,t}$$

$$v_{x,t} = v_{x,t-1} + \epsilon_{v_{x,t}}$$

where $\epsilon_{x,t} \sim N(\mu = 0; \sigma_x^2 = 0.0003)$ and $\epsilon_{v_{x,t}} \sim N(\mu = 0; \sigma_v^2 = 10e - 5)$. Again it is similar for y.
Finally, the system is observed by:

$$mlong_t = x_t + \epsilon_{1,t}$$

$$mlat_t = y_t + \epsilon_{2,t}$$

where both $\epsilon_{1,t}$ and $\epsilon_{3,t} \sim N(\mu = 0, \sigma_0^2 = 10e - 4)$.
From the supplied information you are supposed to formulate the system on state space form.

Solution

We can describe the observations of a dynamical system by means of a kalman filter. In the formulation, we asume that a given system $\{\vec{x}_t\}$ is governed by some dynamics that we know (described by matrices A and U) and some noise in the system $\{\vec{\epsilon_1}\}$ because there will always be external secondary factors that will modify the state of the system.

In the Kalman filter model, the state of the dynamical system at time t can only depend on the previous state using the matrix A to indicate such dependance, so it is a Markov process.(Of course the value at time t depends implicitly on all the previous values, but given the previous one, it does not depend on the rest).

Our observation of the dynamical system is $\{\vec{z}_t\}$. This is a linear transformation of the real state model, given by matrix C and some measuring noise $\{\vec{\epsilon_2}\}$

For one of the spacial dimensions of this problem, $\{\vec{x}_t\}$, the formulation of the system is:

$$\vec{x}_t = \begin{bmatrix} x_t \\ v_{x,t} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{t-1} \\ v_{x,t-1} \end{bmatrix} + \begin{bmatrix} \epsilon_{x,t} \\ \epsilon_{v_{x,t},t} \end{bmatrix}$$

and the formulation for the observation is:

$$\vec{z}_t = \begin{bmatrix} \hat{x}_t \end{bmatrix} = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_t \\ v_{x,t} \end{bmatrix} + \begin{bmatrix} \epsilon_{1,t} \end{bmatrix}$$

Since in our model the movement in the dimensions are independent, the combined state model is straight forward:

$$\vec{x}_t = A \cdot \vec{x}_{t-1} + \vec{\epsilon_1} = \begin{bmatrix} x_t \\ v_{x,t} \\ y_t \\ v_{y,t} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{t-1} \\ v_{x,t-1} \\ y_{t-1} \\ v_{y,t-1} \end{bmatrix} + \begin{bmatrix} \epsilon_{x,t} \\ \epsilon_{v_{x,t},t} \\ \epsilon_{y,t} \\ \epsilon_{v_{y,t},t} \end{bmatrix}$$

$$\vec{z}_t = C \cdot \vec{x}_t + \vec{\epsilon_2} = \begin{bmatrix} \hat{x}_t \\ \hat{y}_t \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_t \\ v_{x,t} \\ y_t \\ v_{y,t} \end{bmatrix} + \begin{bmatrix} \epsilon_{1,t} \\ \epsilon_{2,t} \end{bmatrix}$$

6 Question 4.6: Kalman filter implementation

Implement you own version of the Kalman filter you formulated in the previous question in R, Matlab, Python or a similar language. You should provide a listing of your code showing your implementation in the report. (Not just as an appendix)

Solution

The following code is designed to be very human readable and the comments explain the process in a high level of detail. In the following we will explain the steps that the code follows:

- First we create training observation matrix with the positions and velocities of the longitude and lattitude. The initial velocities are set to 0.
- Then we define the matrices of the formulation (A,B,C) in a human readable form. In this case B is not used. After that, we define the covariance matrix of the system noise (Sigma1) and the measuring noise (Sigma2).
- After that we need to define a prior for the prediction of the first step. We use as mean, the first sample, and as covariance of the prediction, a small value, because we have confidence that the first observation and the initial state are very similar.
- We proceed to the reconstruction of the state for the training samples. In this reconstruction, for every sample, we perform the estimation step first, where we estimate the value of the state at time t using the current observation and the prediction from the previous state. Once we have estimated the current state, we perform the prediction of the next state and finally we store all of them in data structures in order to be able to access them once the loop is finished. The equation can be found in the book (10.70, 10.71, 10.72) and (10.76, 10.77, 10.78).
- For the prediction step, we have as initial prediction, the last prediction from the reconstruction step. Then we iteratively calculate the prediction of the following steps. Once again, we save the predicted steps in data structures to be able to access them later.

```
##############################################################################################
############################## KALMAN FILTER !! ##############################################
##############################################################################################
# We create the data matrix by combining the positions and velocities in a single
# matrix !! We will need only to process these values, because for the prediction
# of the last 50 samples, we do not use them to update the model.
vel_mlat_tr = c(0,diff_mlat_tr)  # Set initial velocities to 0
vel_mlong_tr = c(0,diff_mlong_tr)

aux = rbind(c(mlat_tr),
            c(vel_mlat_tr),
            c(mlong_tr),
            c(vel_mlong_tr))

Matrix_tr = t(matrix(aux, nrow = 4))  # Nsam * 4

# System of equations A
#   Xt = A*Xt-1 + B*Ut + SystemNoise
#   Yt = Cx_k  + MeasurementNoise
# In our case B*Ut is 0, we do not have acceleration modeled.
############################## AR matrix #####################################
# We asume that x and y coordinates are independent
# and that the dependence between variables is:
# xt = x_(t-1) + vx_(t-1)
# vxt = vx_(t-1)
# This leads to the next Autocorrelation Matrix A
A <- matrix(rbind(c(1,1,0,0),
                  c(0,1,0,0),
                  c(0,0,1,1),
                  c(0,0,0,1)),
            nrow = 4)
```

```
######################### C Matrix ###################################
# Measurement matrix.
# What transformation of the real state we perfrom.
# In this case we just select the position variables xt and yt
C <- matrix(rbind(c(1,0,0,0),
                  c(0,0,1,0)),
            nrow = 2)

# Dynamic System error Covariance Matrix
# The error that ocurrs in the system due to variables that we are not taking
# into account in the model (external forces: wind, vibration)
varPos = 3*1e-4  # Variance of the position noise
varVel = 1e-5    # Variance of the velocity noise
Sigma1 <- matrix(rbind( c(varPos,0,0,0),
                        c(0,varVel,0,0),
                        c(0,0,varPos,0),
                        c(0,0,0,varVel)),
                 nrow = 4)

# Measurement error Covariance Matrix
# The covariance matrix of the measurement error observed at a given time t.
# We are only measuring position and we believe the measurements are uncorrelated
varNoise = 1e-4
Sigma2 <- matrix(rbind(c(varNoise,0),
                       c(0,varNoise)),
                 nrow = 2)

######################### Initialize Kalman Filter ############################
## Time to choose our priors of prediction !!
## Important:
#       - Variables with "hat" are estimated variables (V(t|t))
#       - Variables with "pred" are predicted variables (V(t+1|t))
# In the begining we have to define Xpred(1|0).
# Observed samples start at time t = 1. t = 0 is our prior thingy.

## We just initialize the parameters of the kalman it with a good guess.
# In this case we just use the initial obervation of Yo as mean of the prior. XOhat = YO
Xpred <- matrix(Matrix_in[1,])  # Initilization Xpred for X1 given YO

# Initilize the uncertainty of the first samples high enough if we are not sure
# or low enough if we are sure of out initial Xhat
# As more samples arise, this influence of this initial decision will be decrease.
# We choose a small uncertainty of the initializatoin
sigmapred0 = 1e-5
SigmaXXpred<- matrix(rbind( c(sigmapred0,0,0,0),   # Initilization SigmaXXpred for X1 given YO
                            c(0,sigmapred0,0,0),
                            c(0,0,sigmapred0,0),
                            c(0,0,0,sigmapred0)),
                     nrow = 4)

## Initial uncertainty of the observed variables. We say, small.
# Maybe same as the measurement noise would be appropiate.
SigmaYYpred <-  C%*%SigmaXXpred%*%t(C) + Sigma2
## Calculate covariance matrix between observarionts and latent variables
SigmaXYpred <- SigmaXXpred %*% t(C)

######### Initialize variables just to keep intermediate results ##########
XhatList <- matrix(0,nrow = 4, ncol = Ntr) # Store here the estimations of X
SigmaXXhatList <- list()        # Store here the estimated Sigma of X
```

```
XpredList <- matrix(0,nrow = 4, ncol = Ntr) # Store here the predictions of X
SigmaXXpredList <- list()        # Store here the predicted Sigma of X

################################################################################
################## RECONSTRUCTION OF THE STATE ################################
################################################################################
for(n in 1:Ntr){
  ##############   ESTIMATING STEP  ####################
  # Estimation of theparameters ("hat" variables V(t|t))
  # Calculated from the predicted values and the new sample
  K = SigmaXXpred %*% t(C) %*% solve(SigmaYYpred)  # Kalman Gain
  Xhat = Xpred + K %*% (C %*% Matrix_tr[n,] - C %*% Xpred)
  SigmaXXhat <- SigmaXXpred - K%*%SigmaYYpred%*%t(K)

  ##############   PREDICTION STEP  ####################
  ## Predict the variables for the next instance  V(t+1|t)
  Xpred <- A%*%Xhat  # We use the correlation bet
  SigmaXXpred <- A%*%SigmaXXhat%*%t(A) + Sigma1
  SigmaYYpred <- C%*%SigmaXXpred%*%t(C) + Sigma2
  SigmaXYpred <- SigmaXXpred%*%t(C)

  ################## Storing data  #################
  XhatList[,n] <-Xhat
  SigmaXXhatList[[n]] <- SigmaXXhat

  XpredList[,n] <-Xpred
  SigmaXXpredList[[n]] <- SigmaXXpred
}
################################################################################
################## PREDICTION OF THE STATE ####################################
################################################################################

# Now, using the final state of training (last estimations), we will predict
# the state and observations for the last 50 samples (test samples)

## The initial hat is the one predicted for the last training sample
# We initilize with last parameters calculated
Xhattest = matrix(XhatList[,Ntr], ncol = 1)
SigmaXXhattest <- SigmaXXhatList[[Ntr]]
Xpredtest = matrix(XpredList[,Ntr], ncol = 1)
SigmaXXpredtest <- SigmaXXpredList[[Ntr]]

## Variables to store the results
XpredtestList <- matrix(0,nrow = 4, ncol = Ntst)
SigmaXXpredtestList <- list()

# The first prediction is the one calculated last in the previous loop
XpredtestList[,1] <- Xpredtest
SigmaXXpredtestList[[1]] <- SigmaXXpredtest
# We calculate the rest
for(n in 2:(Ntst)){
  # Perform future predictions
  Xpredtest <- A%*%Xpredtest
  SigmaXXpredtest <- A%*%SigmaXXpredtest%*%t(A) + Sigma1
  XpredtestList[,n] <-Xpredtest
  SigmaXXpredtestList[[n]] <- SigmaXXpredtest
}
```

7 Question 4.7: Using your Kalman filter implementation

Use the implementation to reconstruct (estimate) the position of the gps logger including a 95% confidence interval. Present the reconstructions. Predict the position 50 steps ahead and present the predictions as for the MARIMA model. Comment on the results.

Solution

The next two figures show the reconstruction for the mlat and mlong signals during its first 50 samples. The original signal is black and the reconstruction is in blue, including also 2 intermitent blue lines corresponding to the boundaries of the 95% confidence interval of the estimation of the state. In intermitent red lines we also have the 95% confidence interval of the observation, which include the noise of the sampling. As we can see:

- The reconstruction fits the original signal almost perfectly. This could be a bad thing, since we might be overfitting the sampling noise. The default prior for the sampling noise might be too small.
- We can observe that the initial variance is lower, since our prior had a low variance. And then it increases until reaching the one given by the system.
- Obviously, the state confidence interval is always narrower than the obervation confidence interval because of the sampling noise.
- All observations fall into the 95% confidence interval, both the state and the observation intervals.
- The standard deviation of the prediction saturates to 0.00908 after the first few samples (influence of the prior) for both dimensions.

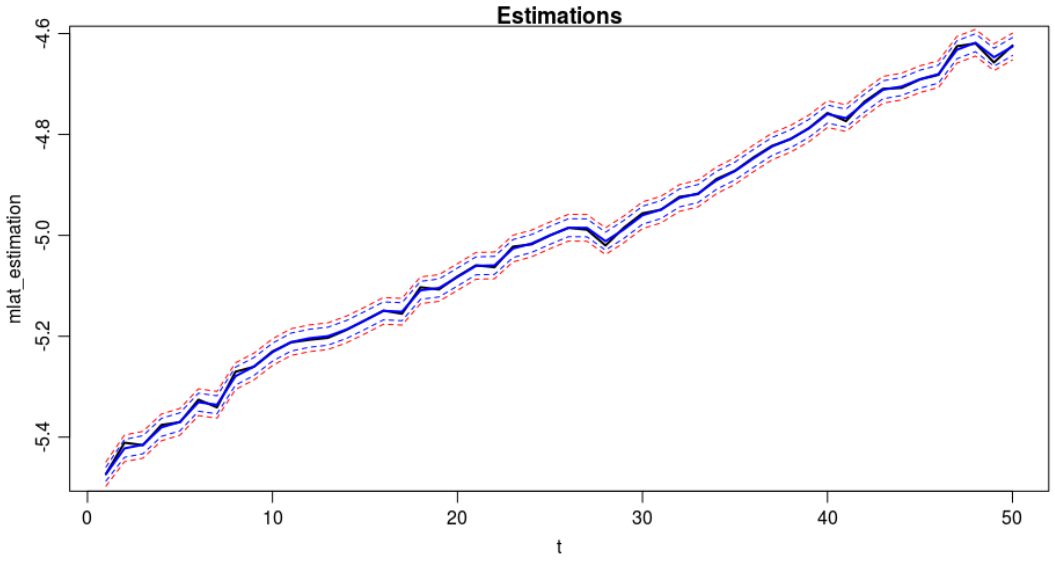

Figure 24: First 50 samples estimation of the mlat signal

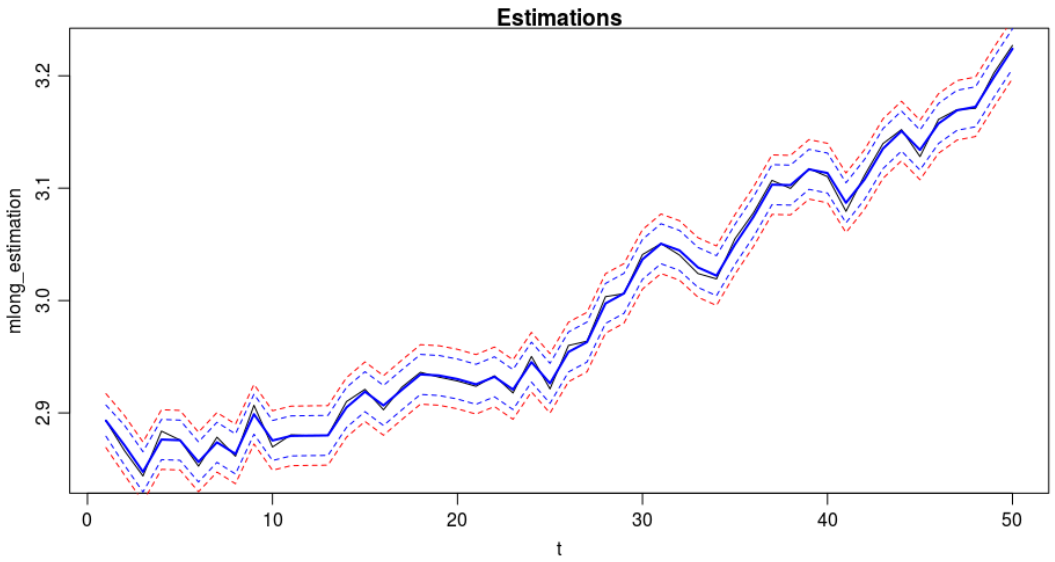

Figure 25: First 50 samples estimation of the mlong signal

If we increase the prior variance of the noise by a factor of 10 for example, the next reconstruction of the mlong signal is found. As we can see, the reconstruction is much more smooth because the system does not trust the small changes in position (and its derived velocity) that ocurr due to noise. Of couse the confident interval of the observation expands and the confidence interval of the system also expands, since we have noisier observations, we trust our estimations less. As we see now, the prediction does not follow the quick changes of the observation as much as it used to do.

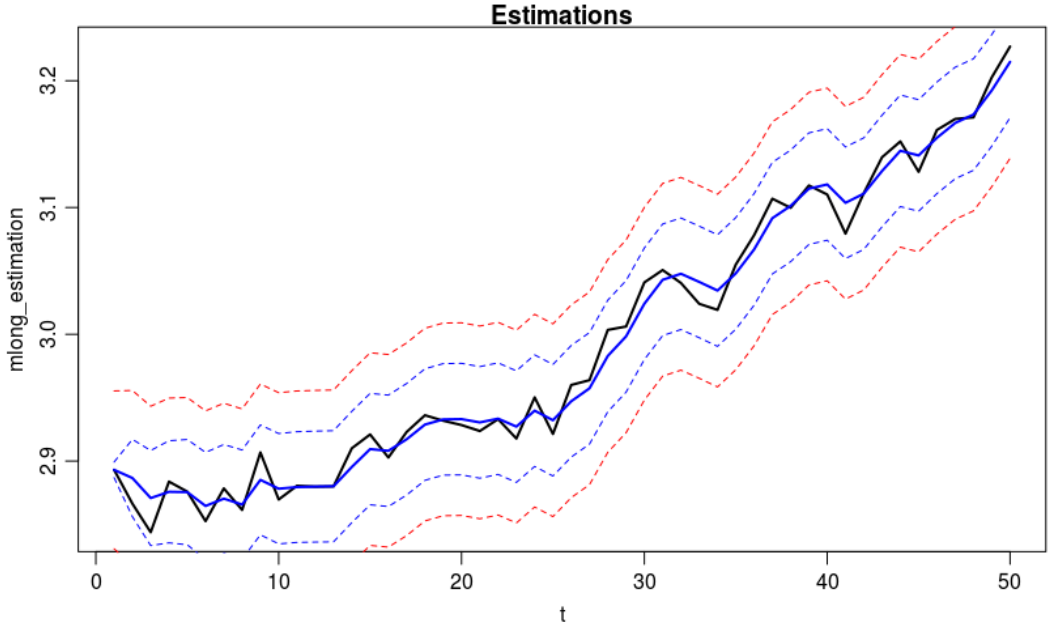

Figure 26: First 100 samples estimation of the mlong signal using a more noisy prior

Just in case there is grading related to it, the next two images show the reconstruction for the whole training dataset in the same way as before for the initial noise parameters. Of course at this degree of zoom, we can only say that the reconstruction feets the real data almostp perfectly.

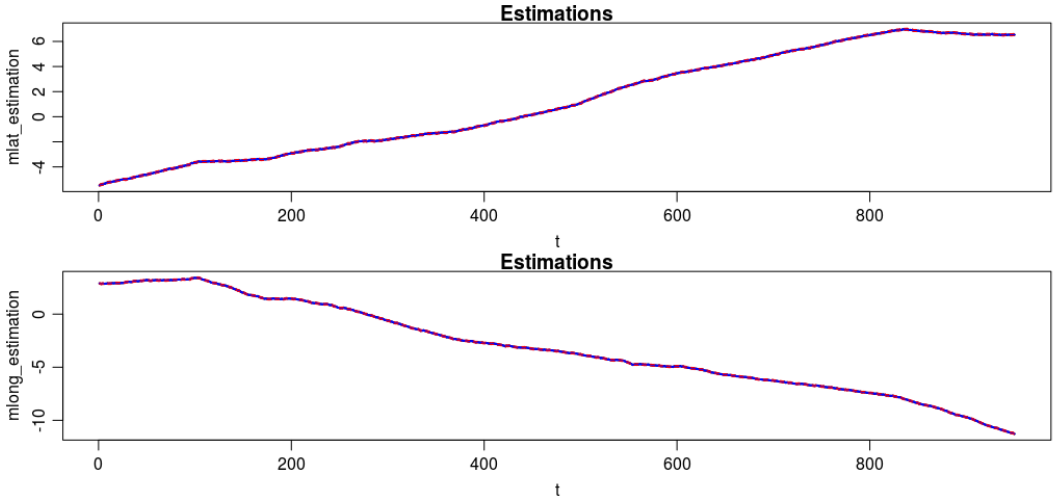

Figure 27: Reconstruction of the Kalman filter

Regarding the **predictions**, the next graph shows part of the training samples and the test samples in black. The predicted state in blue, along with the 95% confidence interval, drawn by 2 discontinuous blue lines with $\pm 1.96\hat{\sigma}_\epsilon$. The 95% confidence interval of the observation is also plotted in red. Some observations are:

- For the predictions, we can see that they all fall into the 95% confidence interval. This model generates predictions with more variance than the MARIMA model.
- The main direction of the prediciton is very good, as oposed to the ones obtained by the MARIMA model. We could say that the Kalman Filter is able to estimate the real direction of the object since it takes into account that there is sampling noise, so it does not trust the instant changes of direction as much as the MARIMA model does.
- The standard deviation of the observation and the one of the state are very close, since the sampling noise variance becomes small compared to the variance of the prediction of the state.
- The standard deviation of the prediction grows in what seems an exponential function.
- The images themselves are pretty cool, like lassers.

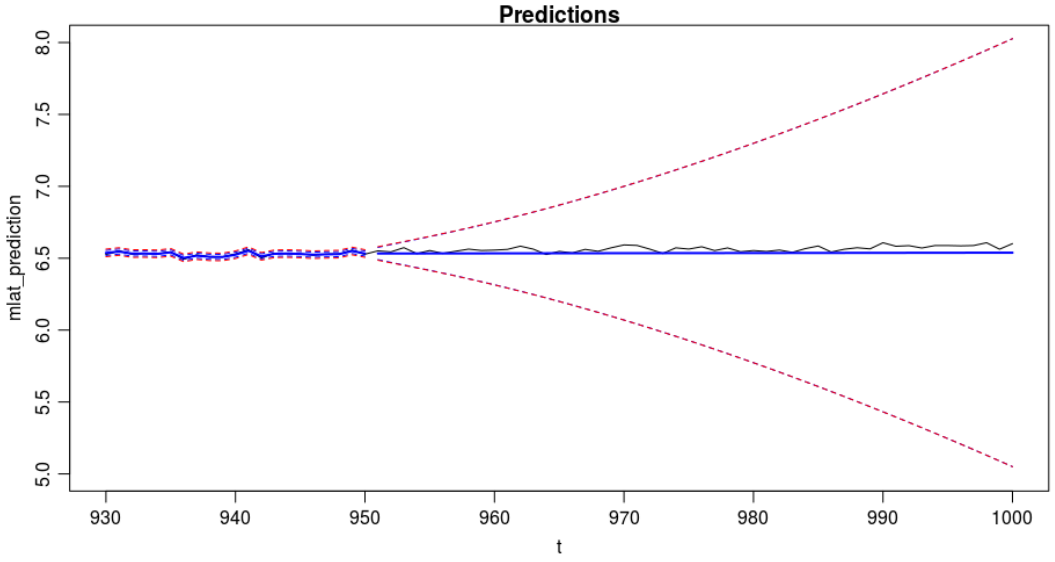

Figure 28: Prediction mlat Kalman Filter

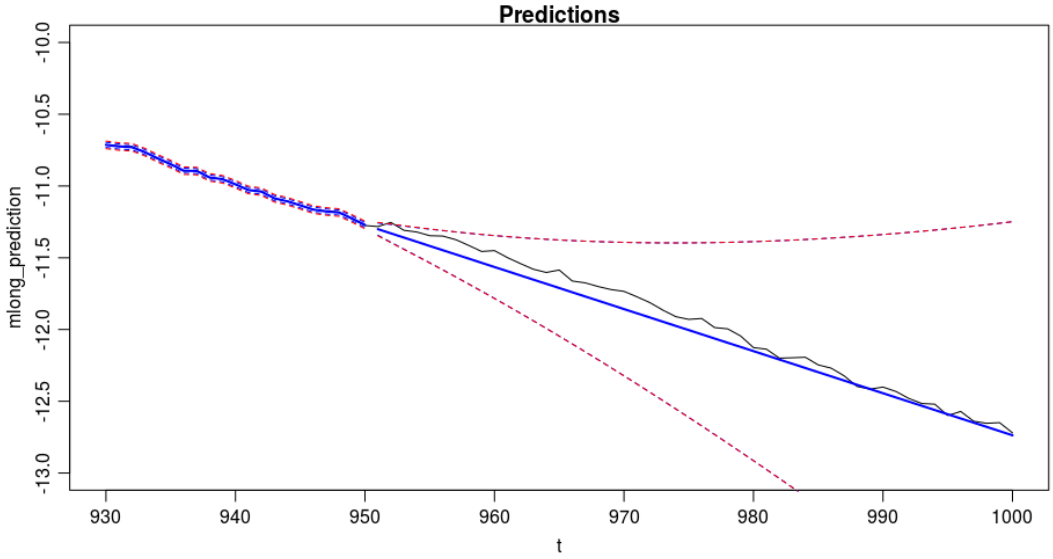

Figure 29: Prediction mlong Kalman Filter

Also, we could see the prediction samples along with their covariance matrix in a spatial form. The next image shows the real testvalues in black, the predictions in blue and the area of 95% of confidence interval for some samples. In this picture we can appreciate how the variance grows and also that the samples are not related to each other.

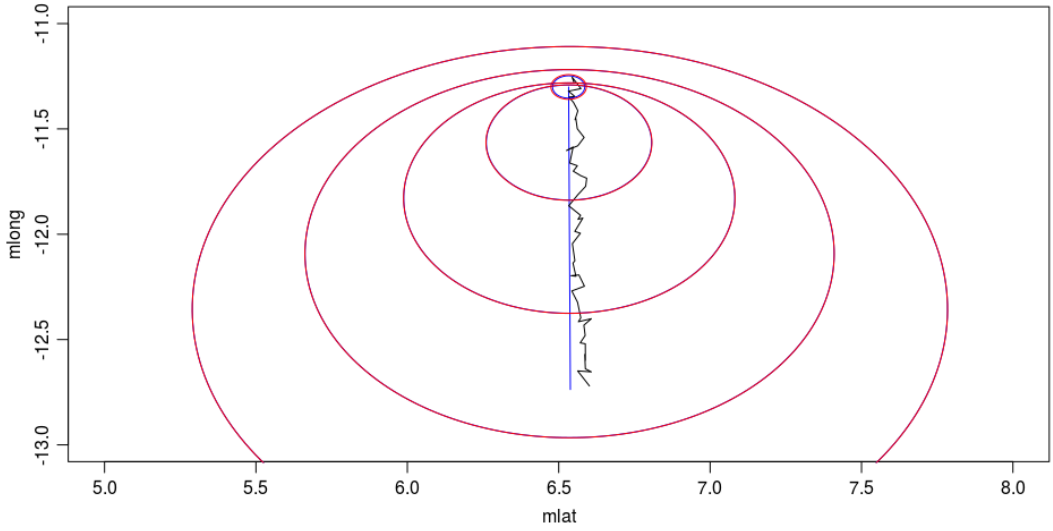

Figure 30: Prediction mlong mlat

The next table shows the 1, 10, 25 and 50 step predictions, along with the standard deviation of the prediction and the 95% confidence interval for the observations. We can observe in it all the things said about the graphs previously. We can also observe that the variance of the prediction is the same for both dimensions.

Time	1	10	25	50
Real Value	6.551	6.5564	6.5634	6.6013
Prediction	6.5326	6.5337	6.5354	6.5383
Std	0.0239	0.1119	0.3103	0.7593
Conf.Int	[6.485837, 6.579533]	[6.314310, 6.753150]	[5.927190, 7.143754]	[5.050010, 8.026741]

Table 3: mlat predictions Kalman model

Time	1	10	25	50
Real Value	-11.2826	-11.4503	-11.9293	-12.7202
Prediction	-11.300	-11.564	-12.004	-12.737
Std	0.0239	0.1119	0.3103	0.7593
Conf.Int	[-11.3476,-11.2539]	[-11.784,-11.3451]	[-12.6126,-11.3960]	[-14.2255,-11.2488]

Table 4: mlong predictions Kalman model

8 Question 4.8: Bonus: Optimize noise parameters

The provided values for the variances are not the optimal values. If you have time for an extra challenge then find the maximum likelihood estimates of the three variances (σ_x^2, σ_v^2, and σ_0^2).

Solution

So the time has come to question our priors, more specifically, the priors on the system and sampling noise. As it was pointed out earlier, it seems like the sampling noise might have been underestimated, leading to an estimation of the state that fits some measuring noise.

To select a better value for these hyperparameters we need a function that tells us how good they are, this function being in this case the likelihood function, defined in the book by (10.92). This is the probability of observing our dataset, given our model basically (by model we could say the parameters of the Kalman Filter). This is computed from the prediction error at time t, given the current state and all previous observations. The best set of parameters is the one that maximizes the product of the likelihood probabilities of the training samples.

Taking logarithm of this product, we reach the equation:

$$log(L(\vec{y}_1^T|model)) = -\frac{T}{2}log(2\pi) - \frac{1}{2}\sum_{t=1}^{T} log(|\Sigma_{t|t-1}|) - \frac{1}{2}\sum_{t=1}^{T}(y_t - \hat{y}_{t|t-1})' \Sigma_{t|t-1}^{-1}(y_t - \hat{y}_{t|t-1})$$

where T is the number of samples, $\hat{y}_{t|t-1}$ is the prediction of the observation at time t given all previous observations and $\Sigma_{t|t-1}$ the prediction covariance matrix of the observation.

A basic method to search for the optimal parameters using this function is just computing a grid search of them and at the end selecting the set that maximizes the function. Since for this case, the first term of the equation is the same for all the models, we do not need to use it for the maximization purposes.

The following grid search has been performed, it is not very fine, but it should be representative.

```
varPoses = exp(seq(log(1e-7),log(1e-5),length=10))
varVeles = exp(seq(log(1e-6),log(1e-5),length=10))
varNoises = exp(seq(log(1e-3),log(1e-1),length=10))
```

We get that the best values, the ones that maximize the log likelihood are: $\sigma_x^2 = 1e-07, \sigma_v^2 = 2.15e-06$, and $\sigma_0^2 = 0.02154$. Which is basically telling us that there is not much system noise, and that there is more for the position than for the velocity, this could be logical, since the velocity is calculated as a substraction of 2 random variables so the variances add up. It also has a high sampling noise, which means that the randomness we observe comes from the sampling.

The following graphs show the reconstruction of the first 100 samples. As we can see, there is a lot of more smoothing, because we trust less the observations, leading to a better estimation of the main direction. We can also appreciate how the influence of the prior prediction for the first sample fades after the first 10 samples.

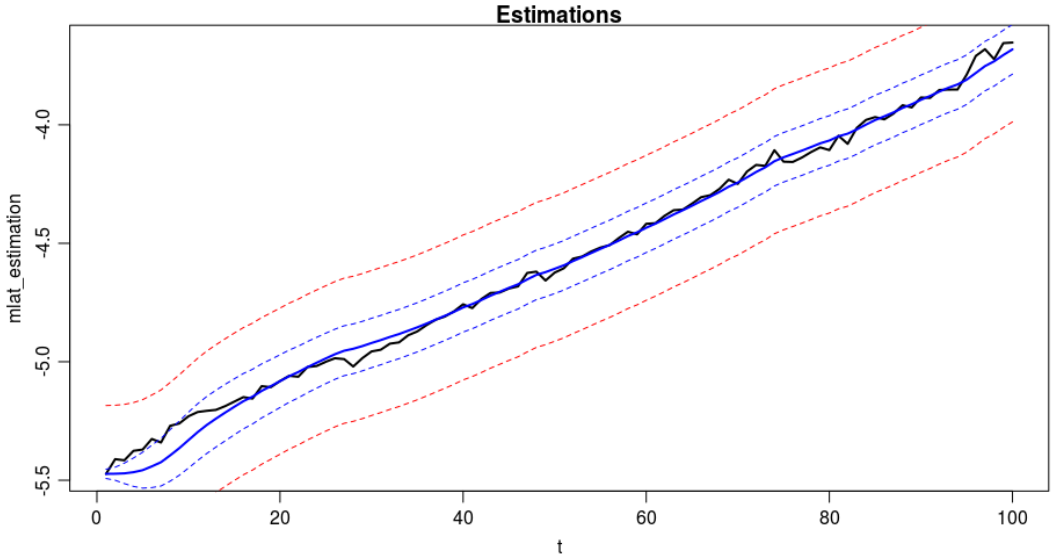

Figure 31: First 100 samples estimation of the mlat signal optimized

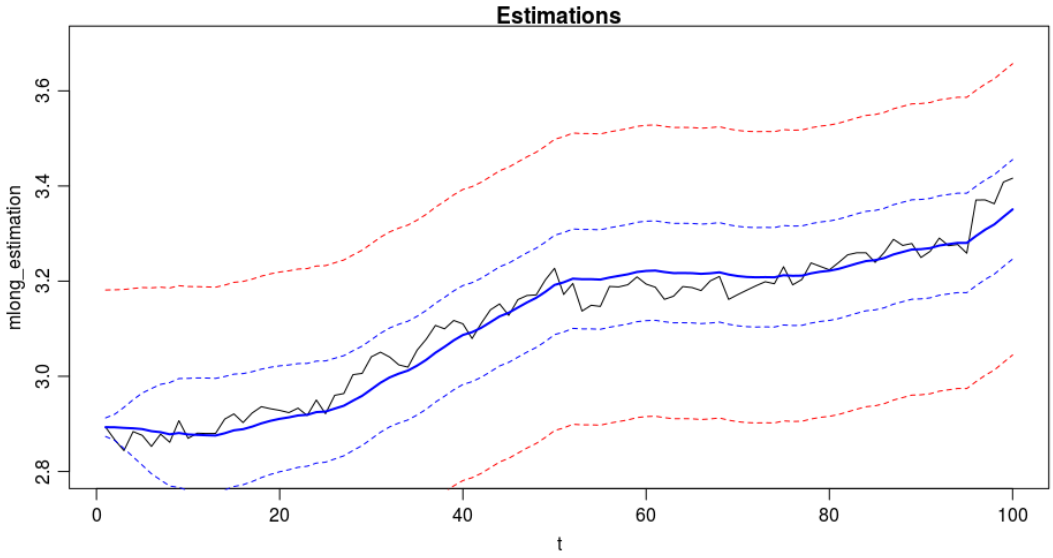

Figure 32: First 100 samples estimation of the mlong signal optimized

The next graphs show the predictions in the same way as before. We can happily see how the variance of the prediction is reduced significantly, while the main prediction is as correct as the previous one.

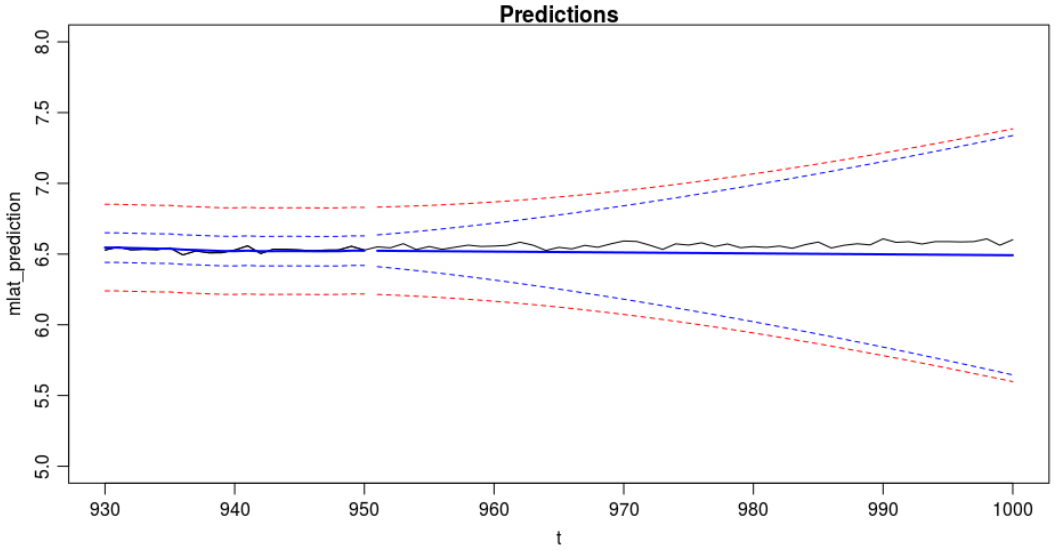

Figure 33: Prediction of the mlat signal optimized

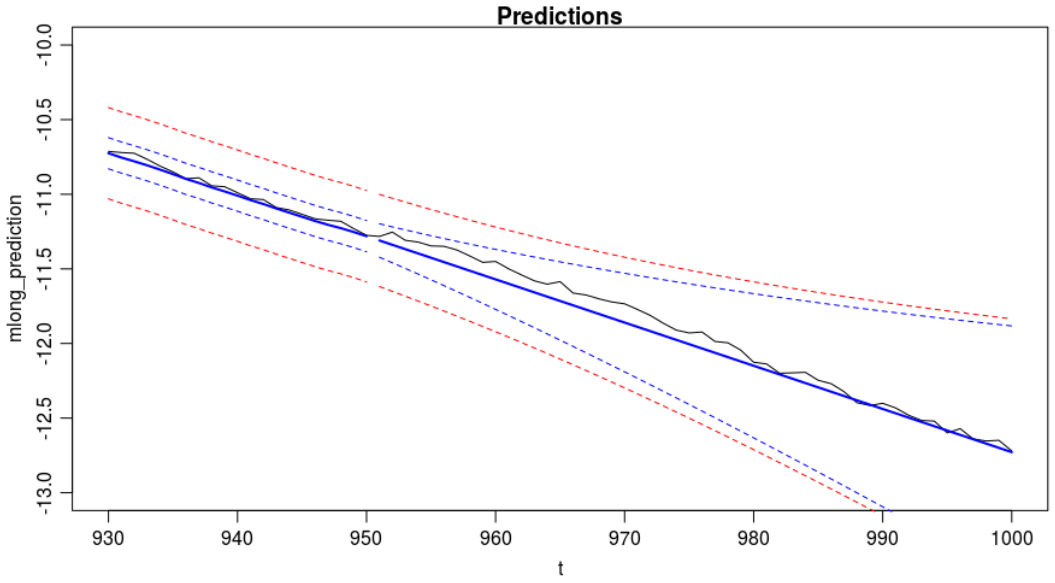

Figure 34: Prediction of the mlong signal optimized

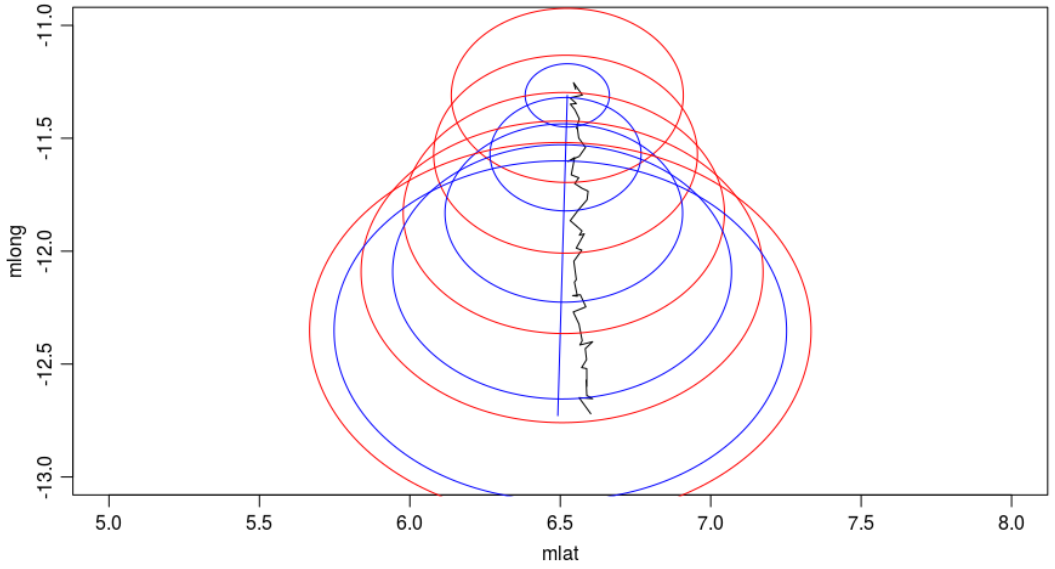

Figure 35: Prediction mlong mlat in spatial form optimized

9 Question 4.9: Comparison

Comment on the performance of the two modelling approaches. Suggest alternative formulations of the Kalman filter that may perform better

Solution

As a starting point, the Kalman Filter as it is stated has less capabilities to learn patterns than the MARMA model, since the Kalman Filter can only depend on the previous value of the rest of the variables. But it allows to first, use with non-stationary signals, and second, apply our understanding of the signal into the model. It has some drawbacks like that we have to select priors, thus increasing the number of hyperparameters of the model and we make assumptions. But if our prior knowledge (dynamic system + noises) is correct, we have a much more robust system and computationally cheap in both time and memory, as it can be easily updated and does not require to store previous samples.

The confidence of the MARIMA model is too high regarding predictions, maybe due to its "stationarity" assumptions of the signals. While the Kalman Filter has a better estimation of the uncertainty. Also, we can tell the Kalman Filter about the sampling noise, so that it does not trust the observed samples. The MARIMA model trusts them too much, thus misleading the predictions.

If the object usually moved also in a given pattern, like (we had to take into account the rotation of the Earth), probably we can put that knowledge into the Kalman filter and get a better estimate.

As better alternatives for the Kalman filter formulation we could propose:

- Model also the acceleration. Create another variable of the state being the acceleration if the object is usually suffering it smoothly (maybe due to rotation of the Earth).
- Take into account the theory of relativity when measuring and put it into the model.
- Modify the transformation matrix C. Maybe there is a distortion in the reading from the sensors, or some coupling that could be solved usign another transformation matrix.
- Assume that there are correlations in the system noise and measurement noise and put them into the system. Basically, modify the covariance matrices of the noises.

Maybe it is harder to implement seasonality in a kalman filter.

10 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.

```
source("./graphlib.R")
source("./mystatlib.R")
source("./step.slow.marima.R")

################ Question 4-1 ######################
## Loading and preparing the data:
myData = read.csv("./A4_gps_log.csv", sep = ",", header = TRUE) # read the csv
t = as.matrix(myData[,1])   # Get the date
mlat = as.matrix(myData[,2])   # Get the hour
mlong = as.matrix(myData[,3])     # Get the NO concentration

Nsam = dim(t)[1]              # Number of samples

### Split data into train and test
Ntst = 50
tr_indx = 1:(Nsam - Ntst)  # Indexes for estimating
Ntr = length(tr_indx)
tst_indx = (Ntr + 1):Nsam   # Indexes for estimating
Ntst = length(tst_indx)

# Obtain data
mlat_tr = mlat[tr_indx]
```

```
mlat_tst = mlat[tst_indx]

mlong_tr = mlong[tr_indx]
mlong_tst = mlong[tst_indx]

t_tr = t[tr_indx]
t_tst = t[tst_indx]

## PLOT BOTH SIGNALS IN TERMS OF TIME
plot_timeSeries(t_tr, mlong_tr, "NO concentration", "Time", "Concentration", 2, 0)
lines(t_tr, mlat_tr, lwd = 5)

plot_timeSeries(mlat_tr, mlong_tr, "Title", "latitud (x)", "longitud (y)", 2, 0)

############### Transformations of the data !###############
diff_mlat_tr = diff(mlat_tr)
diff_mlong_tr = diff(mlong_tr)

diffdiff_mlat_tr = diff(diff_mlat_tr)
diffdiff_mlong_tr = diff(diff_mlong_tr)

############## Create Train Matrixes ################
aux = rbind(c(mlat_tr), c(mlong_tr))
train_Matrix = t(matrix(aux, nrow = 2))

aux = rbind(c(diff_mlat_tr), c(diff_mlong_tr))
difftrain_Matrix = t(matrix(aux, nrow = 2))

aux = rbind(c(diff_mlat_tr), c(diff_mlong_tr))
diffdifftrain_Matrix = t(matrix(aux, nrow = 2))

############## Create Test Matrixes ################
aux = rbind(c(mlat_tst), c(mlong_tst))
test_Matrix = t(matrix(aux, nrow = 2))

############## Create Both Matrixes ################
# Because of the stupid way the MARMA forecast works
aux = rbind(c(diff(mlat)), c(diff(mlong)))
dataMatrix = t(matrix(aux, nrow = 2))

################################################
######  PLOTTING
par(mfrow=c(1,1))

# Velocity Properties !!
plot_timeSeries(t_tr[-Ntr], diff_mlong_tr, "mlong", "Time",
                "Increase in longitud",2,0)
plot_timeSeries(t_tr[-Ntr], diff_mlat_tr, "mlat", "Time",
                "Increase in latitud",2,0)

# Joint changes
plot_timeSeries(diff_mlong_tr, diff_mlat_tr, "mlat", "mlong",
                "Correlation mlong mlat",2,0)

# Acceleration Poperties !!
# Velocity Properties !!
plot_timeSeries(t_tr[1:(Ntr-2)], diffdiff_mlong_tr, "mlong", "Time",
                "Increase in longitud",2,0)
plot_timeSeries(t_tr[1:(Ntr-2)], diffdiff_mlat_tr, "mlat", "Time",
                "Increase in latitud",2,0)
# Joint changes
```

```
plot_timeSeries(diffdiff_mlong_tr, diffdiff_mlat_tr, "mlat", "mlong",
                "Correlation mlong mlat",2,0)

##############################################
####### View Basic Stufff ##################
summary(myData[c("mlat","mlong")])  # Same stuff
summary(t(matrix(mlat, mlong)))

################ Question 2 ######################
# We just use the ACF and PACF functions to calculate the coefficients.
par(c(1,2))
## Original data !!
Nlag = 50
source("./graphlib.R")
plot_acfpacf(mlat_tr,"NOtr_ACFPACF.png",Nlag, 0)
plot_acfpacf(mlong_tr,"NOtr_ACFPACF.png",Nlag, 0)
plot_ccfpccf(mlat_tr,mlong_tr, "NOtr_ACFPACF.png",Nlag, 0)

## Differenced data !
plot_acfpacf(diff_mlat_tr,"NOtr_ACFPACF.png",Nlag, 0)
plot_acfpacf(diff_mlong_tr,"NOtr_ACFPACF.png",Nlag, 0)
plot_ccfpccf(diff_mlat_tr,diff_mlong_tr, "NOtr_ACFPACF.png",Nlag, 0)
##mierda = acf(cbind(diff_mlat_tr,diff_mlong_tr), lag.max = Nlag)

## Differenced data acceleration!
plot_acfpacf(diffdiff_mlat_tr,"NOtr_ACFPACF.png",Nlag, 0)
plot_acfpacf(diffdiff_mlong_tr,"NOtr_ACFPACF.png",Nlag, 0)
plot_ccfpccf(diffdiff_mlat_tr,diffdiff_mlong_tr, "NOtr_ACFPACF.png",Nlag, 0)

library(tseries)
adf.test(diff_mlat_tr, alternative = "stationary",k = 24)
adf.test(diff_mlong_tr, alternative = "stationary",k = 24)
################################################################################
################ Question 3 ####################################################
################################################################################
## Starting marima
library(marima)
##################### TRY DIFFERENT MODELS #####################################
## Let us start with a marima(1,0,1)
par(mfrow=c(1,1))

ar = c(1,2,3)
ma = c(1,1)
max.iter = 100
struct11 <- define.model(kvar=2, ar=ar, ma=ma, indep=NULL) # rem.var is to ignore the years
M1 <- marima(difftrain_Matrix, means=1, ar.pattern=struct11$ar.pattern, max.iter = max.iter,
             ma.pattern=struct11$ma.pattern, Check=FALSE, Plot="log.det", penalty=0)
M1
# print estimates
short.form(M1$ar.estimates, leading=FALSE)
short.form(M1$ar.pvalues, leading=FALSE)

short.form(M1$ma.estimates, leading=FALSE)
short.form(M1$ma.pvalues, leading=FALSE)

Mfinal = step.slow (M1, difftrain_Matrix, penalty=2, max.iter=100)
short.form(Mfinal$ar.estimates, leading=FALSE)
short.form(Mfinal$ar.pvalues, leading=FALSE)
short.form(Mfinal$ma.estimates, leading=FALSE)
short.form(Mfinal$ma.pvalues, leading=FALSE)
```

```
residuals = Mfinal$residuals[,4:(Ntr -1)]
residuals_mlat = residuals[1,]
residuals_mlong = residuals[2,]
plot_SigACFPACF(residuals_mlong, Nlag)
plot_SigACFPACF(residuals_mlat, Nlag)

plot_ccfpccf(residuals_mlat,residuals_mlong, "NOtr_ACFPACF.png",Nlag, 0)

## Error Sigma !!


SS <- sum(residuals_mlong^2)/Ntr  # Sum of square errors of the residual model 1
par(mfrow=c(1,2))

########################### Error Sigma ##############
# Histogram and draw gaussian
hist(residuals_mlong,probability=T,col='blue')
curve(dnorm(x,sd = sqrt(Mfinal$resid.cov[1,1])), col=2, lwd=2, add = TRUE)
# Do the samples quantities
qqnorm(residuals_mlong)
qqline(residuals_mlong)

########################### PERDIODOGRAM ##############
par(mfrow=c(1,2))
cpgram(diff_mlong_tr)
cpgram(residuals_mlong)

plot_ccfpccf(residuals_mlat,residuals_mlong, "NOtr_ACFPACF.png",Nlag, 0)
plot_ccfpccf(diff_mlat_tr,diff_mlong_tr, "NOtr_ACFPACF.png",Nlag, 0)

alltest(residuals_mlat)
### Check if the sum of squared acf values follow a chi-square distribution.
acfvals <- acf(residuals_mlat, type="correlation", plot=FALSE)$acf[1:15] # Get acf residuals
test.stat <- sum(acfvals^2) * (length(residuals)-1)
1 - pchisq(test.stat, length(acfvals)-6)
prob = pchisq(test.stat, length(acfvals)-6, lower.tail = FALSE)

########################### VAR library tests ##############
var1l <- VAR(t(residuals), p=1)
summary(var1l)
vars.test(var1l)

########## OTHER ADVANCES STUFF #############
## Using regression variables
Model3 <- define.model(kvar=7, ar=ar, ma=ma, rem.var=c(1), reg.var=c(6,7))
Marima3 <- marima(all.data,means=1, ar.pattern=Model3$ar.pattern,
                  ma.pattern=Model3$ma.pattern, Check=FALSE, Plot='log.det', penalty=0)
short.form(Marima3$ar.estimates, leading=FALSE) # print estimates
short.form(Marima3$ma.estimates, leading=FALSE)

## Using a penalty
Model4 <- define.model(kvar=7, ar=ar, ma=ma, rem.var=1, reg.var=c(6,7))
Marima4 <- marima(all.data[1:90,], means=1, ar.pattern=Model4$ar.pattern,
                  ma.pattern=Model4$ma.pattern, Check=FALSE, Plot='log.det', penalty=1)
short.form(Marima4$ar.estimates, leading=FALSE) # print estimates
short.form(Marima4$ma.estimates, leading=FALSE)
Marima4$ar.fvalues
Marima4$ma.fvalues

#############################################################################################
################################## Prediction ###############################
```

```
##############################################################################################

# call the forecasting function using Marima and the prepared data:
Forecasts <-  arma.forecast(t(dataMatrix), nstart=949, nstep=50, marima=Mfinal)

#################################################################
#################### MLONG ######################################
#################################################################
### From here on the plot is constructed ###
## We start in 950, dataMatrix only has 999 samples corresponding to the las 999 samples, we
# cannot calculate the diff of the first sample.
# To generate the orginial signal for diff[i] we need to add the value of the signal signal[i-1]
# In this case, since the matrix diff is already desplazated to the future 1 positons, the arrays
# already syncronized

############################ mlat ############################

# For reconstruction x[i + 1] = x[i] + v[i]. Using the index yeah
# For the predictions, we are given v[i], v[i+1], v[i+2] ans so on,
# so to get the original signal we perfrom cumsum of these increments.
# Making gaussian assumption, the variance of the sum is equal to the sum
# of the variances

Predict<- cumsum(Forecasts$forecasts[1,Ntr:(Nsam -1)]) + mlat[Ntr]
stdv<-sqrt(cumsum(Forecasts$pred.var[1,1,]))
upper.lim=Predict+stdv*1.96
lower.lim=Predict-stdv*1.96
par(mfrow=c(1,1))

## Plot the real signal !!!
Nbefore = 10

## Check for correction :)
caca = dataMatrix[(Ntr - Nbefore - 1):(Nsam - 1),1] + mlat[(Ntr - Nbefore -1):(Nsam -1)]
caca2 = mlat[(Ntr - Nbefore):Nsam]
caca - caca2
plot(t[(Ntr - Nbefore):Nsam],caca ,type="l", xlab="time_index",
     ylab="mlat prediction", main="mlat prediction", col= "black",
     ylim = c(6.45, 6.7))

##lines(t[(Ntr - Nbefore):Nsam], caca2,type="l", col= "blue")

lines(t[(Ntr +1):(Nsam)], Predict,type="l", col= "blue")
lines(t[(Ntr+1 ):(Nsam)], upper.lim, type="l", lty = 2, col = "blue")
lines(t[(Ntr+1 ):(Nsam)], lower.lim, type="l", lty = 2, col = "blue")

############################ mlong ############################
Predict<-cumsum(Forecasts$forecasts[2,Ntr:(Nsam -1)]) + mlong[Ntr]
stdv<-sqrt(cumsum(Forecasts$pred.var[2,2,]))
upper.lim=Predict+stdv*1.96
lower.lim=Predict-stdv*1.96
par(mfrow=c(1,1))

# plot results:
Nbefore = 10
plot(t[(Ntr - Nbefore):Nsam], dataMatrix[(Ntr - Nbefore - 1):(Nsam - 1),2] + mlong[(Ntr - Nbefore
     ylab="mlong prediction", main="mlong prediction", col= "black")

## Plot the real signal !!!
lines(t[(Ntr +1):Nsam ], Predict,type="l", col= "blue")
```

```
lines(t[(Ntr +1):Nsam], upper.lim, type="l", lty = 2, col = "blue")
lines(t[(Ntr +1):Nsam], lower.lim, type="l", lty = 2, col = "blue")
##############################################################################################
############################## KALMAN FILTER !! ##############################################
##############################################################################################
# We create the data matrix by combining the positions and velocities in a single
# matrix !! We will need only to process these values, because for the prediction
# of the last 50 samples, we do not use them to update the model.
vel_mlat_tr = c(0,diff_mlat_tr)  # Set initial velocities to 0
vel_mlong_tr = c(0,diff_mlong_tr)

aux = rbind(c(mlat_tr),
            c(vel_mlat_tr),
            c(mlong_tr),
            c(vel_mlong_tr))

Matrix_tr = t(matrix(aux, nrow = 4))  # Nsam * 4

# System of equations A
#   Xt = A*Xt-1 + B*Ut + SystemNoise
#   Yt = Cx_k  + MeasurementNoise
# In our case B*Ut is 0, we do not have acceleration modeled.
############################## AR matrix ###################################
# We asume that x and y coordinates are independent
# and that the dependence between variables is:
# xt = x_(t-1) + vx_(t-1)
# vxt = vx_(t-1)
# This leads to the next Autocorrelation Matrix A
A <- matrix(rbind(c(1,1,0,0),
                  c(0,1,0,0),
                  c(0,0,1,1),
                  c(0,0,0,1)),
                  nrow = 4)

######################### C Matrix ###################################
# Measurement matrix.
# What transformation of the real state we perfrom.
# In this case we just select the position variables xt and yt
C <- matrix(rbind(c(1,0,0,0),
                  c(0,0,1,0)),
                  nrow = 2)

# Dynamic System error Covariance Matrix
# The error that ocurrs in the system due to variables that we are not taking
# into account in the model (external forces: wind, vibration)
varPos = 3*1e-4  # Variance of the position noise
varVel = 1e-5    # Variance of the velocity noise
Sigma1 <- matrix(rbind( c(varPos,0,0,0),
                        c(0,varVel,0,0),
                        c(0,0,varPos,0),
                        c(0,0,0,varVel)),
                        nrow = 4)

# Measurement error Covariance Matrix
# The covariance matrix of the measurement error observed at a given time t.
# We are only measuring position and we believe the measurements are uncorrelated
varNoise = 1e-4
Sigma2 <- matrix(rbind(c(varNoise,0),
                       c(0,varNoise)),
                      nrow = 2)
```

```
######################## Initialize Kalman Filter ############################
## Time to choose our priors of prediction !!
## Important:
#       - Variables with "hat" are estimated variables (V(t|t))
#       - Variables with "pred" are predicted variables (V(t+1|t))
# In the begining we have to define Xpred(1|0).
# Observed samples start at time t = 1. t = 0 is our prior thingy.

## We just initialize the parameters of the kalman it with a good guess.
# In this case we just use the initial oberuation of Yo as mean of the prior. XOhat = YO
Xpred <- matrix(Matrix_in[1,])  # Initilization Xpred for X1 given YO

# Initilize the uncertainty of the first samples high enough if we are not sure
# or low enough if we are sure of out initial Xhat
# As more samples arise, this influence of this initial decision will be decrease.
# We choose a small uncertainty of the initializatoin
sigmapred0 = 1e-4
SigmaXXpred<- matrix(rbind( c(sigmapred0,0,0,0),   # Initilization SigmaXXpred for X1 given YO
                            c(0,sigmapred0,0,0),
                            c(0,0,sigmapred0,0),
                            c(0,0,0,sigmapred0)),
                     nrow = 4)

## Initial uncertainty of the observed variables. We say, small.
# Maybe same as the measurement noise would be appropiate.
SigmaYYpred <-  C%*%SigmaXXpred%*%t(C) + Sigma2
## Calculate covariance matrix between observarionts and latent variables
SigmaXYpred <- SigmaXXpred %*% t(C)

######### Initialize variables just to keep intermediate results ##########
XhatList <- matrix(0,nrow = 4, ncol = Ntr) # Store here the estimations of X
SigmaXXhatList <- list()        # Store here the estimated Sigma of X

XpredList <- matrix(0,nrow = 4, ncol = Ntr) # Store here the predictions of X
SigmaXXpredList <- list()        # Store here the predicted Sigma of X

##############################################################################
################## RECONSTRUCTION OF THE STATE ###############################
##############################################################################
for(n in 1:Ntr){
  #############   ESTIMATING STEP  ###################
  # Estimation of theparameters ("hat" variables V(t|t))
  # Calculated from the predicted values and the new sample
  K = SigmaXXpred %*% t(C) %*% solve(SigmaYYpred)  # Kalman Gain
  Xhat = Xpred + K %*% (C %*% Matrix_tr[n,] - C %*% Xpred)
  SigmaXXhat <- SigmaXXpred - K%*%SigmaYYpred%*%t(K)

  #############   PREDICTION STEP  ###################
  ## Predict the variables for the next instance  V(t+1|t)
  Xpred <- A%*%Xhat  # We use the correlation bet
  SigmaXXpred <- A%*%SigmaXXhat%*%t(A) + Sigma1
  SigmaYYpred <- C%*%SigmaXXpred%*%t(C) + Sigma2
  SigmaXYpred <- SigmaXXpred%*%t(C)

  ################ Storing data  ################
  XhatList[,n] <-Xhat
  SigmaXXhatList[[n]] <- SigmaXXhat

  XpredList[,n] <-Xpred
  SigmaXXpredList[[n]] <- SigmaXXpred
```

```
}
##########################################################################
############ Compare the Y estimated with the measurements !!! #######
##########################################################################
Yhat =   t(C %*% XhatList)
Ypred = t(C %*% XpredList)
train_Matrix
diff = Yhat - train_Matrix

## Compute the variance:
mlat_stdXX = c()
mlat_stdYY = c()
mlong_stdXX = c()
mlong_stdYY = c()

for(n in 1:Ntr){
  mlat_stdXX = c(mlat_stdXX,  sqrt(SigmaXXhatList[[n]][1,1]))
  aux = C%*%SigmaXXhatList[[n]]%*%t(C) + Sigma2
  mlat_stdYY = c(mlat_stdYY, sqrt(aux[1,1]))

  mlong_stdXX = c(mlong_stdXX,  sqrt(SigmaXXhatList[[n]][3,3]))
  aux = C%*%SigmaXXhatList[[n]]%*%t(C) + Sigma2
  mlong_stdYY = c(mlat_stdYY, sqrt(aux[2,2]))
}


par(mfrow=c(2,1), mar=c(3,3,1,1), mgp=c(2,0.7,0))
par(mfrow=c(1,1))
Nex = 50
plot_timeSeries(t_tr[1:Nex], mlat_tr[1:Nex], "Estimations", "t", "mlat_estimation", 2, 0)
lines(t_tr[1:Nex], Yhat[1:Nex,1], col = "blue", lwd = 2)
lines(t_tr[1:Nex], Yhat[1:Nex,1] + 1.96 * mlat_stdXX[1:Nex], col = "blue", lwd = 1,  lty = 2)
lines(t_tr[1:Nex], Yhat[1:Nex,1] - 1.96 * mlat_stdXX[1:Nex], col = "blue", lwd = 1,  lty = 2)
lines(t_tr[1:Nex], Yhat[1:Nex,1] + 1.96 * mlat_stdYY[1:Nex], col = "red", lwd = 1,  lty = 2)
lines(t_tr[1:Nex], Yhat[1:Nex,1] - 1.96 * mlat_stdYY[1:Nex], col = "red", lwd = 1,  lty = 2)

Nex = 50
plot(t_tr[1:Nex], mlong_tr[1:Nex],main = "Estimations", xlab = "t",
     ylab = "mlong_estimation", type = "l")
lines(t_tr[1:Nex], Yhat[1:Nex,2], col = "blue", lwd = 2)
lines(t_tr[1:Nex], Yhat[1:Nex,2] + 1.96 * mlong_stdXX[1:Nex], col = "blue", lwd = 1,  lty = 2)
lines(t_tr[1:Nex], Yhat[1:Nex,2] - 1.96 * mlong_stdXX[1:Nex], col = "blue", lwd = 1,  lty = 2)
lines(t_tr[1:Nex], Yhat[1:Nex,2] + 1.96 * mlong_stdYY[1:Nex], col = "red", lwd = 1,  lty = 2)
lines(t_tr[1:Nex], Yhat[1:Nex,2] - 1.96 * mlong_stdYY[1:Nex], col = "red", lwd = 1,  lty = 2)


plot(Yhat, train_Matrix)

plot_timeSeries(t_tr[1:Nex], mlat_tr[1:Nex], "Estimations", "t", "mlat_estimation", 2, 0)
lines(t_tr[1:Nex], Ypred[1:Nex,1], col = "blue", lwd = 2)
plot(Ypred, train_Matrix)
plot(Ypred, Yhat)

Nlag = 30
Nlag2 = 30
residuals = t(Ypred[1:(Ntr-1),] - train_Matrix[2:(Ntr),]) # Right displacement

residuals_mlatMA = residuals_mlat
residuals_mlat = residuals[1,]
residuals_mlong = residuals[2,]
plot_SigACFPACF(residuals_mlong, Nlag)
```

```
tsdiag(fit1, gof.lag = Nlag2)  # You plot the residuals and their ACF and JunlgeBox

## Error Sigma !!

SS <- sum(residuals_mlat^2)/Ntr  # Sum of square errors of the residual model 1
par(mfrow=c(1,2))

########################### Error Sigma ###############
# Histogram and draw gaussian
hist(residuals_mlat,probability=T,col='blue')
curve(dnorm(x,sd = sqrt(Mfinal$resid.cov[1,1])), col=2, lwd=2, add = TRUE)
# Do the samples quantities
qqnorm(residuals_mlat)
qqline(residuals_mlat)

########################### PERDIODOGRAM ##############
par(mfrow=c(1,2))
cpgram(diff_mlat_tr)
cpgram(residuals_mlat)

alltest(residuals_mlat)
########################### VAR library tests ##############
var1l <- VAR(t(residuals), p=1)
summary(var1l)
vars.test(var1l)
#############################################################################
################## PREDICTION OF THE STATE ##########################
#############################################################################

# Now, using the final state of training (last estimations), we will predict
# the state and observations for the last 50 samples (test samples)

## The initial hat is the one predicted for the last training sample
# We initilize with last parameters calculated
Xhattest = matrix(XhatList[,Ntr], ncol = 1)
SigmaXXhattest <- SigmaXXhatList[[Ntr]]
Xpredtest = matrix(XpredList[,Ntr], ncol = 1)
SigmaXXpredtest <- SigmaXXpredList[[Ntr]]

## Variables to store the results
XpredtestList <- matrix(0,nrow = 4, ncol = Ntst)
SigmaXXpredtestList <- list()

# The first prediction is the one calculated last in the previous loop
XpredtestList[,1] <- Xpredtest
SigmaXXpredtestList[[1]] <- SigmaXXpredtest
# We calculate the rest
for(n in 2:(Ntst)){
  # Perform future predictions
  Xpredtest <- A%*%Xpredtest
  SigmaXXpredtest <- A%*%SigmaXXpredtest%*%t(A) + Sigma1
  XpredtestList[,n] <-Xpredtest
  SigmaXXpredtestList[[n]] <- SigmaXXpredtest
}

#########################################################################
############ Compare the Y predicted with the measurements !!! #######
#########################################################################
Ypredtest =   t(C %*% XpredtestList)
difftest = Ypredtest - test_Matrix
YpredSigmaYY = C %*% SigmaXXpredtestList[[1]] %*% t(C)
```

```
## Compute the variance:
mlat_stdXXtest = c()
mlat_stdYYtest = c()
mlong_stdXXtest = c()
mlong_stdYYtest = c()

for(n in 1:Ntst){
  mlat_stdXXtest = c(mlat_stdXXtest,  sqrt(SigmaXXpredtestList[[n]][1,1]))
  aux = C%*%SigmaXXpredtestList[[n]]%*%t(C) + Sigma2
  mlat_stdYYtest = c(mlat_stdYYtest, sqrt(aux[1,1]))

  mlong_stdXXtest = c(mlong_stdXXtest,  sqrt(SigmaXXpredtestList[[n]][3,3]))
  aux = C%*%SigmaXXpredtestList[[n]]%*%t(C) + Sigma2
  mlong_stdYYtest = c(mlong_stdYYtest, sqrt(aux[2,2]))
}

par(mfrow=c(1,1))
Nbefore = 20
plot(t[(Ntr- Nbefore):Nsam], mlat[(Ntr- Nbefore):Nsam], type = "l",
     main = "Predictions", xlab = "t", ylab = "mlat_prediction", ylim  = c(5.0,8.0))

lines(t_tr[(Ntr-Nbefore):Ntr], mlat_tr[(Ntr-Nbefore):Ntr])
lines(t_tr[(Ntr-Nbefore):Ntr], Yhat[(Ntr-Nbefore):Ntr,1], col = "blue", lwd = 2)
lines(t_tr[(Ntr-Nbefore):Ntr], Yhat[(Ntr-Nbefore):Ntr,1] + 1.96 * mlat_stdXX[(Ntr-Nbefore):Ntr], c
lines(t_tr[(Ntr-Nbefore):Ntr], Yhat[(Ntr-Nbefore):Ntr,1] - 1.96 * mlat_stdXX[(Ntr-Nbefore):Ntr], c
lines(t_tr[(Ntr-Nbefore):Ntr], Yhat[(Ntr-Nbefore):Ntr,1] + 1.96 * mlat_stdYY[(Ntr-Nbefore):Ntr], c
lines(t_tr[(Ntr-Nbefore):Ntr], Yhat[(Ntr-Nbefore):Ntr,1] - 1.96 * mlat_stdYY[(Ntr-Nbefore):Ntr], c

lines(t_tst, Ypredtest[,1], col = "blue", lwd = 2, ylim  = c(6.0,8.0))
lines(t_tst, Ypredtest[,1] + 1.96 * mlat_stdXXtest, col = "blue", lwd = 1,  lty = 2)
lines(t_tst, Ypredtest[,1] - 1.96 * mlat_stdXXtest, col = "blue", lwd = 1,  lty = 2)
lines(t_tst, Ypredtest[,1] + 1.96 * mlat_stdYYtest, col = "red", lwd = 1,  lty = 2)
lines(t_tst, Ypredtest[,1] - 1.96 * mlat_stdYYtest, col = "red", lwd = 1,  lty = 2)

c(Ypredtest[1,1], Ypredtest[10,1], Ypredtest[25,1], Ypredtest[50,1])
c(Ypredtest[1,2], Ypredtest[10,2], Ypredtest[25,2], Ypredtest[50,2])

c(mlat_stdYYtest[1], mlat_stdYYtest[10], mlat_stdYYtest[25],mlat_stdYYtest[50])
c(mlong_stdYYtest[1], mlong_stdYYtest[10], mlong_stdYYtest[25],mlong_stdYYtest[50])

c(Ypredtest[1,1] + 1.96*mlat_stdYYtest[1], Ypredtest[10,1] + 1.96*mlat_stdYYtest[10],
  Ypredtest[25,1] + 1.96*mlat_stdYYtest[25],Ypredtest[50,1] + 1.96*mlat_stdYYtest[50])
c(Ypredtest[1,1] - 1.96*mlat_stdYYtest[1], Ypredtest[10,1] - 1.96*mlat_stdYYtest[10],
  Ypredtest[25,1] - 1.96*mlat_stdYYtest[25],Ypredtest[50,1] - 1.96*mlat_stdYYtest[50])

c(Ypredtest[1,2] + 1.96*mlong_stdYYtest[1], Ypredtest[10,2] + 1.96*mlong_stdYYtest[10],
  Ypredtest[25,2] + 1.96*mlong_stdYYtest[25],Ypredtest[50,2] + 1.96*mlong_stdYYtest[50])

c(Ypredtest[1,2] - 1.96*mlong_stdYYtest[1], Ypredtest[10,2] - 1.96*mlong_stdYYtest[10],
  Ypredtest[25,2] - 1.96*mlong_stdYYtest[25],Ypredtest[50,2] - 1.96*mlong_stdYYtest[50])

Nbefore = 20
plot(t[(Ntr- Nbefore):Nsam], mlong[(Ntr- Nbefore):Nsam], type = "l", ylim  = c(-13,-10),
     main = "Predictions", xlab = "t", ylab = "mlong_prediction")

lines(t_tr[(Ntr-Nbefore):Ntr], mlat_tr[(Ntr-Nbefore):Ntr])
lines(t_tr[(Ntr-Nbefore):Ntr], Yhat[(Ntr-Nbefore):Ntr,2], col = "blue", lwd = 2)
lines(t_tr[(Ntr-Nbefore):Ntr], Yhat[(Ntr-Nbefore):Ntr,2] + 1.96 * mlong_stdXX[(Ntr-Nbefore):Ntr],
lines(t_tr[(Ntr-Nbefore):Ntr], Yhat[(Ntr-Nbefore):Ntr,2] - 1.96 * mlong_stdXX[(Ntr-Nbefore):Ntr],
lines(t_tr[(Ntr-Nbefore):Ntr], Yhat[(Ntr-Nbefore):Ntr,2] + 1.96 * mlong_stdYY[(Ntr-Nbefore):Ntr],
```

```
lines(t_tr[(Ntr-Nbefore):Ntr], Yhat[(Ntr-Nbefore):Ntr,2] - 1.96 * mlong_stdYY[(Ntr-Nbefore):Ntr],

lines(t_tst, Ypredtest[,2], col = "blue", lwd = 2)
lines(t_tst, Ypredtest[,2] + 1.96 * mlong_stdXXtest, col = "blue", lwd = 1,  lty = 2)
lines(t_tst, Ypredtest[,2] - 1.96 * mlong_stdXXtest, col = "blue", lwd = 1,  lty = 2)
lines(t_tst, Ypredtest[,2] + 1.96 * mlong_stdYYtest, col = "red", lwd = 1,  lty = 2)
lines(t_tst, Ypredtest[,2] - 1.96 * mlong_stdYYtest, col = "red", lwd = 1,  lty = 2)

## Plot some elipses for the noise ?
library(mixtools)
library(mvtnorm)

plot(t_tst, Ypredtest[,2])
lines(t_tst,mlong_tst)

plot(t_tst,mlat_tst, ylim = c(6.2,6.7))
lines(t_tst, Ypredtest[,1])

plot(Ypredtest[,1], Ypredtest[,2], xlim = c(5,8), ylim = c(-13,-11),
     type = "l", xlab = "mlat", ylab = "mlong",  col = "blue")
lines(mlat_tst, mlong_tst, type = "l")


for (i in 1:5){
  j = (i-1) * 9 + 1
  ellipse(mu=Ypredtest[j,], sigma= 1*C %*% SigmaXXpredtestList[[j]]  %*% t(C), alpha = .05, npoint
  ellipse(mu=Ypredtest[j,], sigma= 1*C %*% SigmaXXpredtestList[[j]]  %*% t(C) + Sigma2, alpha = .0
  }

################################################################################
###################### BONUS PART ##########################
################################################################################

varPos = varPoses[1]
varVel = varVeles[4]
varNoise = varNoises[7]

source("./mystatlib.R")

varPoses = exp(seq(log(1e-7),log(1e-5),length=10))
varVeles = exp(seq(log(1e-6),log(1e-5),length=10))
varNoises = exp(seq(log(1e-3),log(1e-1),length=10))

likies = array(0, dim=c(length(varPoses),length(varVeles),length(varNoises)))

for (i in 1:length(varPoses)){
  for (j in 1:length(varVeles)){
    for (k in 1:length(varNoises)){
      varPos = varPoses[i]
      varVel = varVeles[j]
      varNoise = varNoises[k]

Sigma1 <- matrix(rbind( c(varPos,0,0,0),
                        c(0,varVel,0,0),
                        c(0,0,varPos,0),
                        c(0,0,0,varVel)),
                 nrow = 4)

# Measurement error Covariance Matrix
# The covariance matrix of the measurement error observed at a given time t.
# We are only measuring position and we believe the measurements are uncorrelated
```

```
Sigma2 <- matrix(rbind(c(varNoise,0),
                       c(0,varNoise)),
                 nrow = 2)

###################### Initialize Kalman Filter ##########################
## Time to choose our priors of prediction !!
## Important:
#       - Variables with "hat" are estimated variables (V(t|t))
#       - Variables with "pred" are predicted variables (V(t+1|t))
# In the begining we have to define Xpred(1|0).
# Observed samples start at time t = 1. t = 0 is our prior thingy.

## We just initialize the parameters of the kalman it with a good guess.
# In this case we just use the initial obervation of Yo as mean of the prior. XOhat = YO
Xpred <- matrix(Matrix_in[1,])  # Initilization Xpred for X1 given YO

# Initilize the uncertainty of the first samples high enough if we are not sure
# or low enough if we are sure of out initial Xhat
# As more samples arise, this influence of this initial decision will be decrease.
# We choose a small uncertainty of the initializatoin
sigmapred0 = 10e-5
SigmaXXpred<- matrix(rbind( c(sigmapred0,0,0,0),   # Initilization SigmaXXpred for X1 given YO
                            c(0,sigmapred0,0,0),
                            c(0,0,sigmapred0,0),
                            c(0,0,0,sigmapred0)),
                     nrow = 4)

## Initial uncertainty of the observed variables. We say, small.
# Maybe same as the measurement noise would be appropiate.
SigmaYYpred <-  C%*%SigmaXXpred%*%t(C) + Sigma2
## Calculate covariance matrix between observarionts and latent variables
SigmaXYpred <- SigmaXXpred %*% t(C)

######### Initialize variables just to keep intermediate results ##########
XhatList <- matrix(0,nrow = 4, ncol = Ntr) # Store here the estimations of X
SigmaXXhatList <- list()        # Store here the estimated Sigma of X

XpredList <- matrix(0,nrow = 4, ncol = Ntr + 1) # Store here the predictions of X
SigmaXXpredList <- list()        # Store here the predicted Sigma of X
SigmaYYpredList <- list()
SigmaYYpredList[[1]] <- SigmaYYpred
XpredList[,0] = Xpred
##############################################################################
################## RECONSTRUCTION OF THE STATE ###############################
##############################################################################
for(n in 1:Ntr){
  ##############   ESTIMATING STEP  ###################
  # Estimation of theparameters ("hat" variables V(t|t))
  # Calculated from the predicted values and the new sample
  K = SigmaXXpred %*% t(C) %*% solve(SigmaYYpred)  # Kalman Gain
  Xhat = Xpred + K %*% (C %*% Matrix_tr[n,] - C %*% Xpred)
  SigmaXXhat <- SigmaXXpred - K%*%SigmaYYpred%*%t(K)

  ##############   PREDICTION STEP  ###################
  ## Predict the variables for the next instance  V(t+1|t)
  Xpred <- A%*%Xhat  # We use the correlation bet
  SigmaXXpred <- A%*%SigmaXXhat%*%t(A) + Sigma1
  SigmaYYpred <- C%*%SigmaXXpred%*%t(C) + Sigma2
  SigmaXYpred <- SigmaXXpred%*%t(C)
```

```
  ################## Storing data  ################
  XhatList[,n] <-Xhat
  SigmaXXhatList[[n]] <- SigmaXXhat

  XpredList[,n+1] <-Xpred
  SigmaXXpredList[[n]] <- SigmaXXpred # We displace it 1 :) Necesary
  SigmaYYpredList[[n+1]] <- SigmaYYpred  # We displace it 1 :) Necesary
}
#############################################################################
############ Compare the Y estimated with the measurements !!! #######
#############################################################################
Ypred = t(C %*% XpredList)

like = get_likelihood(Matrix_tr %*% t(C), Ypred, SigmaYYpredList)
likies[i,j,k] = like

    }
  }
}


which(likies == max(likies), arr.ind = TRUE)

#install.packages("vars")
library(vars)

get_likelihood <- function(y, ypred, SigmaYYpredList){
  # This function gets the likelihood function of the model
  Sigma2inv = solve(Sigma2)

  logpred = 0
  Nt = dim(y)[1]
  for (i in 1:Nt){
    logpred = logpred + log(det(SigmaYYpredList[[i]])) + t(y[i,] - ypred[i,])%*%solve(SigmaYYpredL
  }
  logpred = - logpred/2
  return(logpred)
}

##signTest
sign.test = function(res) {
  binom.test(sum(res[-1]*res[-length(res)]<0), length(res)-1)
}

## Wrapping all tests in one function
vars.test <- function(x){
  norm <- sapply(normality.test(x)[[2]],function(xx) xx$p.value)
  arch <- arch.test(x)[[2]]$p.value
  ser <- serial.test(x)[[2]]$p.value
  ## Return vector of p values
  return(c(norm,"arch"=arch,"serial"=ser))
}

alltest <- function(residuals){
  ### Binomial test
  # Confidence interval of the number of change of signs.
  n.residuals <- length(residuals)
  (n.residuals-1)/2
  sqrt((n.residuals-1)/4)
  (n.residuals-1)/2 + 1.96 * sqrt((n.residuals-1)/4) * c(-1,1)
```

```
  ### Binomial test
  # P-value using binomial distribution that the number of sign changes is 1/2 probable
  (N.sign.changes <- sum( residuals[-1] * residuals[-n.residuals]<0 ))
  bt = binom.test(N.sign.changes, n.residuals-1)
  p_value_bintest = bt$p.value
  int_conf = bt$conf.int

  return (list(p_value_bintest))
}
```