Estimating the Trajectory of a Satellite

Assignment 5 – 02417 Time Series Analysis – Anders Hørsted (s082382)

In this report the trajectory of an orbiting satellite will be reconstructed and predicted, based on 50 measurements $\{(r_t^m, \theta_t^m)\}_{t=1}^{50}$ of the position of the satellite given in polar coordinates. Due to imprecision in the measurement process, the real position of the satellite (r_t^p, θ_t^p) is related to the measurements by

$$r_t^m = r_t^p + \varepsilon_{rt}$$
$$\theta_t^m = \theta_t^p + \varepsilon_{\theta t}$$

where $\varepsilon_{rt} \sim N(0, 2000^2)$, $\varepsilon_{\theta t} \sim N(0, 0.03^2)$ and ε_{rt}, $\varepsilon_{\theta t}$ are independent.

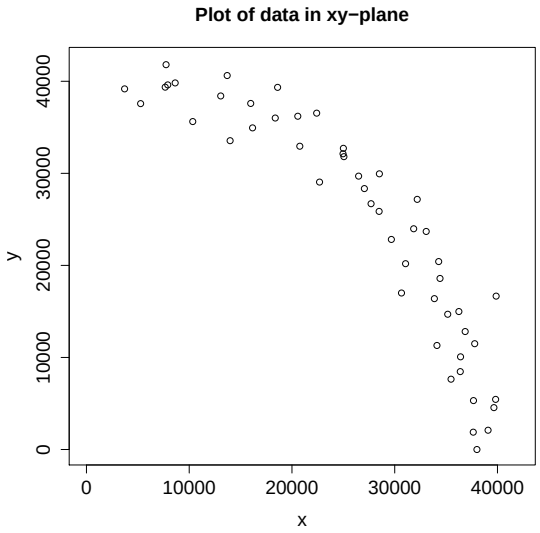

Figure 1: Plot of measured satellite position in cartesian coordinates

Formulating a model

To reconstruct and predict the trajectory of the satellite, we will create a state space model. The state vector is defined as

$$\boldsymbol{X}_t = \begin{pmatrix} r_t^p \\ \theta_t^p \\ v_{\theta t}^p \end{pmatrix}$$

where r_t^p is the true distance to the satellite, θ_t^p is the angle, and $v_{\theta t}^p$ is the angle velocity. It is assumed that the trajectory is well approximated by a circle, but to include deviations from a perfect circle, we define the random deviations

$$\varepsilon_{rt}^p \sim N(0, 500^2), \quad \varepsilon_{\theta t}^p \sim N(0, 0.005^2), \quad \varepsilon_{v\theta t}^p \sim N(0, 0.005^2)$$

and then model the dynamics of the state vector by the equations

$$\begin{aligned} r_t^p &= r_{t-1}^p + \varepsilon_{rt}^p \\ \theta_t^p &= \theta_{t-1}^p + v_{t-1}^p + \varepsilon_{\theta t}^p \\ v_t^p &= v_{t-1}^p + \varepsilon_{v\theta t}^p \end{aligned}$$

From these equation it is seen, that we model the radius and angle-velocity to be constant, as they would be for prefect uniform circular motion. To write the state space model on matrix form, we define the measurement vector by

$$\boldsymbol{Y}_t = \begin{pmatrix} r_t^m \\ \theta_t^m \end{pmatrix}$$

and two random vectors by

$$\boldsymbol{e}_1 = \begin{pmatrix} \varepsilon_{rt}^p \\ \varepsilon_{\theta t}^p \\ \varepsilon_{v\theta t}^p \end{pmatrix}, \quad \boldsymbol{e}_2 = \begin{pmatrix} \varepsilon_{rt} \\ \varepsilon_{\theta t} \end{pmatrix}$$

Since $\varepsilon_{rt}^p, \varepsilon_{\theta t}^p$ and $\varepsilon_{v\theta t}^p$ are independent the covariance between any two of them is 0. ε_{rt} and $\varepsilon_{\theta t}$ are also independent and therefore the covariance between them is also 0. We then get

$$\boldsymbol{\Sigma}_1 = \mathrm{Var}[\boldsymbol{e}_1] = \begin{bmatrix} 500^2 & 0 & 0 \\ 0 & 0.005^2 & 0 \\ 0 & 0 & 0.005^2 \end{bmatrix}, \quad \boldsymbol{\Sigma}_2 = \mathrm{Var}[\boldsymbol{e}_2] = \begin{bmatrix} 2000^2 & 0 \\ 0 & 0.03^2 \end{bmatrix}$$

By setting

$$\boldsymbol{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{C} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

the state space model can now be expressed by the system equation

$$\boldsymbol{X}_t = \boldsymbol{A}\boldsymbol{X}_{t-1} + \boldsymbol{e}_1$$

and the observation equation

$$\boldsymbol{Y}_t = \boldsymbol{C}\boldsymbol{X}_t + \boldsymbol{e}_2$$

Using the defined model the trajectory can now be reconstructed and predicted by using the Kalman filter. To actually use the Kalman filter, it needs to be implemented.

Implementing the Kalman filter

The implementation of the Kalman filter is shown in listing 1. It is a function that takes the model defining parameters `A,B,C,u,S.1` and `S.2`, the initialization parameters `mu0` and `V0`, and the observations `Y`. The code is more or less self explaining and is based on equation 10.73-10.80 in [1]. The function returns a list with the reconstructed state vector for each iteration, the final covariance matrix for the one-step prediction of the state vector, the Kalman Gain from each iteration and the final one-step prediction of the state vector.

```
kalman.filter = function(A, B, C, u, S.1, S.2, mu0, V0, Y) {

    # Find the right dimensions
    num.observations = dim(Y)[1]
    observation.dim = dim(Y)[2]
    state.dim = dim(S.1)[1]

    # Init history variables
    X.hat.now.history = matrix(0, nrow=num.observations, ncol=state.dim)
    K.t.history = array(0, dim=c(state.dim, observation.dim, num.observations))

    # Initialize Kalman filter
    X.hat.step = mu0
    S.xx.step = V0
    S.yy.step = S.2

    # Run Kalman filter
    for (i in 1:num.observations) {
        Y.t = matrix(Y[i,])

        # Update Kalman gain
        K.t = S.xx.step %*% t(C) %*% solve(S.yy.step)

        # Reconstruction
        X.hat.now = X.hat.step + K.t %*% (Y.t - C %*% X.hat.step)
        S.xx.now = S.xx.step - K.t %*% C %*% S.xx.step

        # Prediction
        X.hat.step = A %*% X.hat.now + B %*% u
        S.xx.step = A %*% S.xx.now %*% t(A) + S.1
        S.yy.step = C %*% S.xx.step %*% t(C) + S.2

        # Save iteration history
        X.hat.now.history[i,] = X.hat.now
        K.t.history[,,i] = K.t
    }

    # Return information about the iterations
    return(list(X.hat.now.history=X.hat.now.history, S.xx.step=S.xx.step,
                K.t.history=K.t.history, X.hat.step=X.hat.step))

}
```

Code Listing 1: Implementation of the Kalman filter

Reconstructing the trajectory

In this section the implementation of the Kalman filter is used to reconstruct the trajectory of the satellite. To run the Kalman filter the initial parameters $\boldsymbol{\mu}_0$ and $\boldsymbol{V}_0$ needs to be defined. As long as there are many observations, the result isn't very sensitive to the initial conditions. We set

$$\boldsymbol{\mu}_0 = \begin{pmatrix} r_1^m \\ \theta_1^m \\ 0 \end{pmatrix}, \quad \boldsymbol{V}_0 = \begin{bmatrix} 0,0,0 \\ 0,0,0 \\ 0,0,0 \end{bmatrix}$$

and then run the Kalman implementation from the previous section*. Since the implementation returns the reconstructed state vector for each iteration, the reconstructed trajectory can easily be plotted† as shown in figure 2

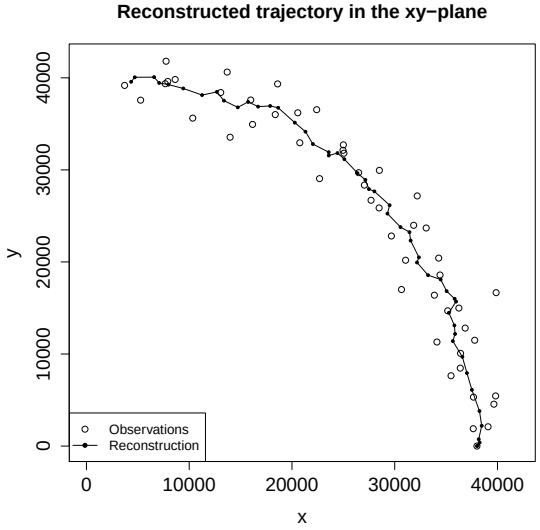

Figure 2: The reconstructed trajectory calculated with the Kalman filter

It is seen that the Kalman filter is pretty robust even when the data contains large errors. From the output of the Kalman filter function it is seen that the Kalman Gain converges to a constant value after about 30 iterations which is expected.

Predicting the trajectory

The satellite position for the 6 next time steps should now be predicted. The Kalman filter only gives one-step predictions, but using the equations

$$\widehat{\boldsymbol{X}}_{t+k+1|t} = \boldsymbol{A}\widehat{\boldsymbol{X}}_{t+k|t} + \boldsymbol{B}\boldsymbol{u}_{t+k} \tag{1}$$

$$\boldsymbol{\Sigma}^{xx}_{t+k+1|t} = \boldsymbol{A}\boldsymbol{\Sigma}^{xx}_{t+k|t}\boldsymbol{A}^T + \boldsymbol{\Sigma}_1 \tag{2}$$

*See appendix A.4 for the actual code that runs the Kalman filter

†See appendix A.5 for the code

k	$\widehat{r}_{50+k\|50}$	$1.96\sigma_{\hat{r}}$	95% CI
1	39805.96	2086.07	[37719.89; 41892.03]
2	39805.96	2304.80	[37501.16; 42110.76]
3	39805.96	2504.50	[37301.47; 42310.46]
4	39805.96	2689.40	[37116.56; 42495.37]
5	39805.96	2862.39	[36943.57; 42668.35]
6	39805.96	3025.51	[36780.45; 42831.47]

k	$\widehat{\theta}_{50+k\|50}$	$1.96\sigma_{\hat{\theta}}$	95% CI
1	1.49	0.05	[1.43; 1.54]
2	1.51	0.07	[1.44; 1.58]
3	1.54	0.09	[1.45; 1.63]
4	1.56	0.11	[1.45; 1.67]
5	1.59	0.13	[1.45; 1.72]
6	1.61	0.16	[1.46; 1.77]

Table 1: Prediction and confidence intervals for $\widehat{r}$ and $\widehat{\theta}$

the predictions for later steps can also be found. From the equations and the definition of $\boldsymbol{A}$ it is clear that the prediction of r^p_{t+k} is constant for any $k > 0$. To get all predictions of r^p_{t+k} and θ^p_{t+k} the Kalman filter is first run on the data. The output from the Kalman filter gives the 1-step prediction, and then iterating equation (1) and (2) 5 times gives the $2, 3 \ldots, 6$-step predictions. The results are shown in table 1

To visualize the predictions a plot is made and shown in figure 3. From the figure it is seen (confirmed, since we already knew it) how the predictions lie equally spaced on a perfect circle. This is probably not how the actual position is going to end up, so we would like to give an idea of the uncertainty of the predictions.

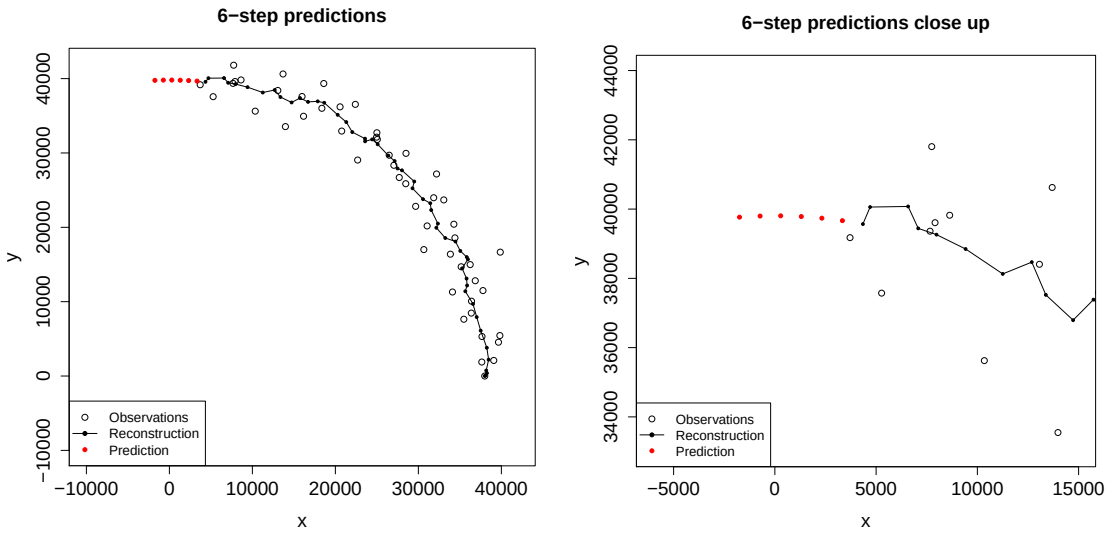

Figure 3: Predictions of the satellite position 6 step ahead

The uncertainty is already given in table 1, but to give an better idea the 6 predictions are plotted on separate plots including their respective confidence intervals. The plots are shown in figure 4 and figure 5 and it is seen how the uncertainty grows larger for each time step away from the time of prediction. This was expected both intuitively and from equation (2), but it is still worth noticing that the right side of the 6-step prediction

confidence interval is almost identical to the right side of the 1-step prediction confidence interval. With a 95% confidence, we can't reject the possibility that the satellite will be in the same spot 1 and 6 time steps ahead. Still the theory tells us that these are the best prediction we can give, as long as we don't get any new observations.

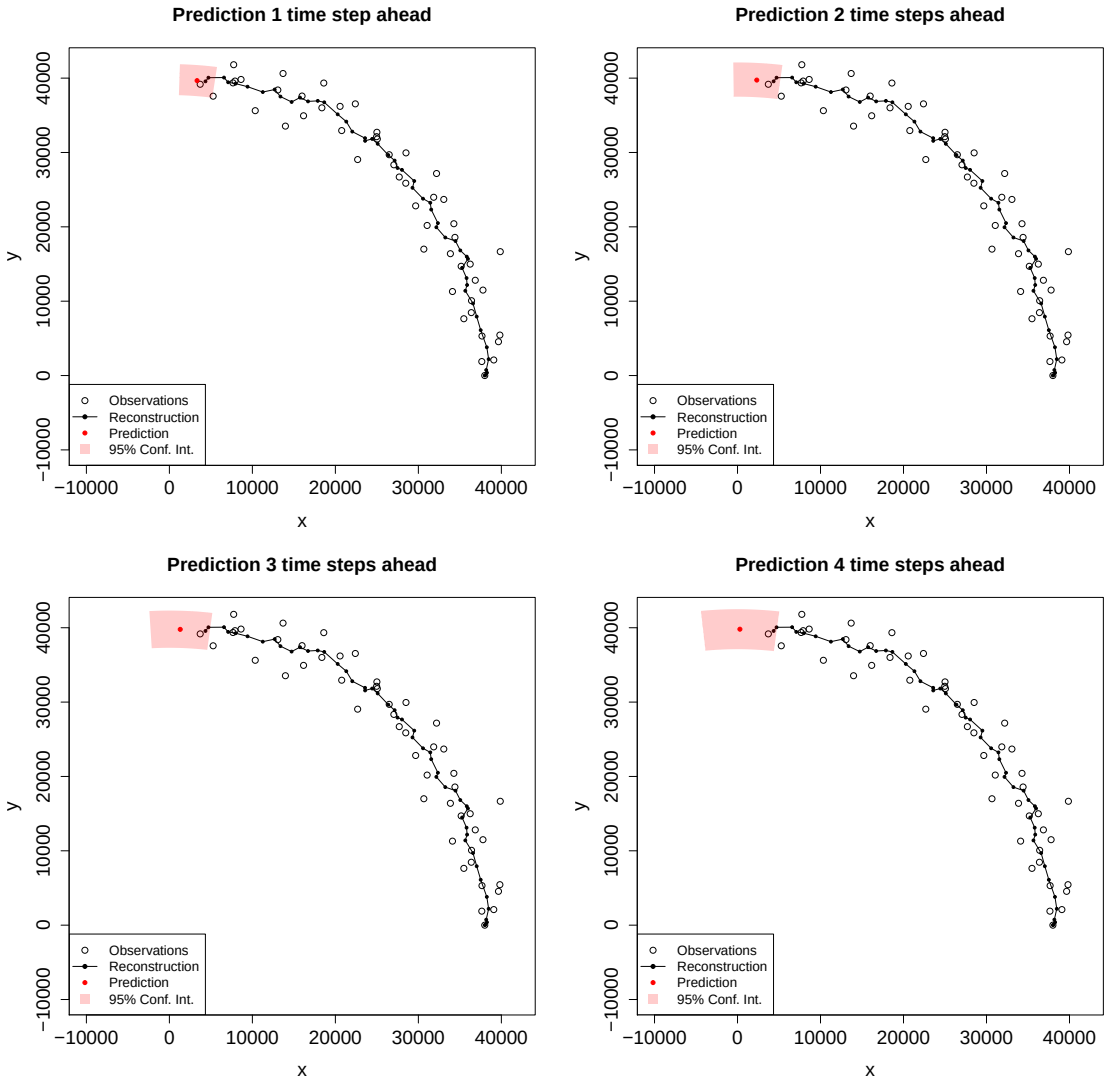

Figure 4: Plots of the 1-,2-,3-, and 4-step predictions including confidence intervals. 5- and 6-step are on the next page.

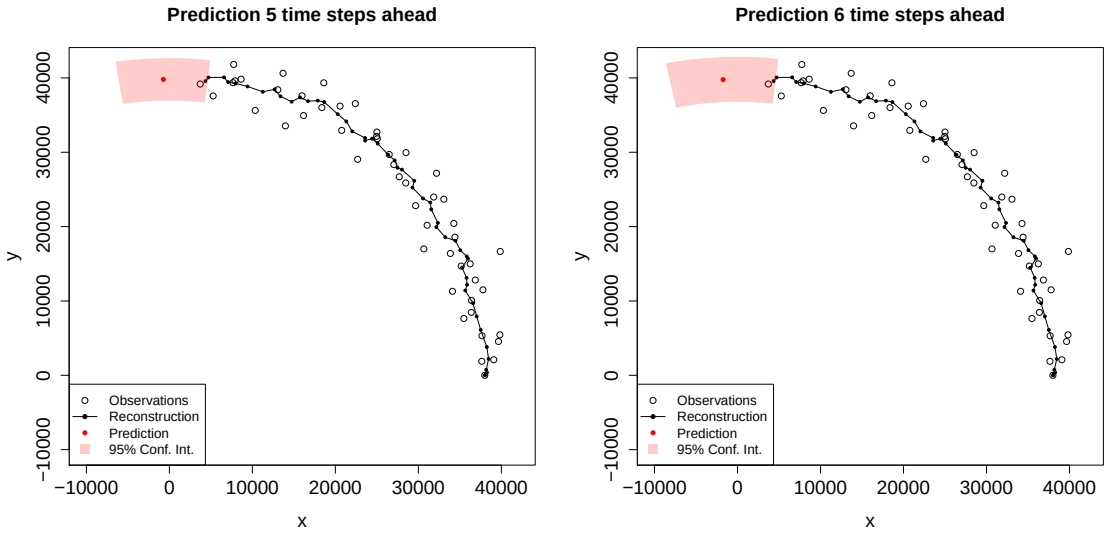

Figure 5: Plots of the 5-, and 6-step predictions including confidence intervals

A Appendices

All R code used for this assignment is included here. All source code incl. latex code for this report can be found at `https://github.com/alphabits/dtu-fall-2011/tree/master/02417/assignment-5`

A.1 Load data script

```
dat = read.csv('../data/Satelliteorbit.csv', header=FALSE, sep=",")
dat = data.matrix(dat)
dat.xy = polar.to.cartesian(dat[,1], dat[,2])
dat.x = dat.xy$x
dat.y = dat.xy$y
```

A.2 Helper functions

```
polar.to.cartesian = function(r, theta) {
    x = r*cos(theta)
    y = r*sin(theta)
    return(list(x=x, y=y))
}

display.confidence.interval = function(r.min, r.max, theta.min, theta.max, ...) {
    rs = seq(r.min, r.max, length.out=30)
    thetas = seq(theta.min, theta.max, length.out=400)
    polygon.sides = list(
        list(rs, theta.min),
        list(r.max, thetas),
        list(rev(rs), theta.max),
        list(r.min, rev(thetas))
```

```
    )
    x = c()
    y = c()

    for (side in polygon.sides) {
        tmp = polar.to.cartesian(side[[1]], side[[2]])
        x = c(x, tmp$x)
        y = c(y, tmp$y)
    }

    polygon(x, y, border=NA, col="#FF000033", ...)
}

plot.data = function(dat.x, dat.y, plot.lim, ...) {
    plot(dat.x, dat.y, xlim=plot.lim, ylim=plot.lim, xlab="x", ylab="y",
         cex.main=1.4, cex.lab=1.4, cex.axis=1.4, ...)
}

plot.predictions = function(dat.x, dat.y, pred.x, pred.y, ...) {
    plot.lim = c(-10000,42000)
    plot.data(dat.x, dat.y, plot.lim, ...)
    points(pred.x, pred.y, pch=20, col="red")
}

plot.reconstruction = function(recon.x, recon.y) {
    points(recon.coords$x, recon.coords$y, pch=20, cex=0.7)
    lines(recon.coords$x, recon.coords$y)
}

add.legend = function(show.prediction=TRUE, show.conf.interval=FALSE) {
    texts = c("Observations", "Reconstruction")
    pchs = c(21, 20)
    cols = c("black", "black")
    ltys = c(0, 1)
    cex = c(1, 1)
    if (show.prediction) {
        texts = c(texts, "Prediction")
        pchs = c(pchs, 20)
        cols = c(cols, "red")
        ltys = c(ltys, 0)
        cex = c(cex, 1)
    }
    if (show.conf.interval) {
        texts = c(texts, "95% Conf. Int.")
        pchs = c(pchs, 15)
        cols = c(cols, "#FF000033")
        ltys = c(ltys, 0)
        cex = c(cex, 1.5)
    }
    legend("bottomleft", texts, pch=pchs, col=cols, lty=ltys, pt.cex=cex)
}
```

A.3 Plot data script

```
source('loaddata.R')
source('functions.R')


pdf('../plots/data.pdf', 7, 7)
```

```
plot.data(dat.x, dat.y, c(0, 42000), main="Plot of data in xy-plane", show.legend=FALSE)
dev.off()
```

A.4 Running the Kalman filter

```
source('loaddata.R')
source('kalmanfilter.R')


# Setup model matrices
A.dat = c(1,0,0,
          0,1,1,
          0,0,1)
A = matrix(A.dat, nrow=3, ncol=3, byrow=TRUE)

B = diag(c(0,0,0))

C.dat = c(1,0,0,
          0,1,0)
C = matrix(C.dat, nrow=2, ncol=3, byrow=TRUE)

u = matrix(c(0,0,0))

# Init error covariance matrices
S.1 = diag(c(500^2, 0.005^2, 0.005^2))
S.2 = diag(c(2000^2, 0.03^2))

# Init intial input
mu0 = matrix(c(dat[1,], 0))
V0 = diag(c(0,0,0))

# Run kalman filter
kalman.results = kalman.filter(A, B, C, u, S.1, S.2, mu0, V0, dat)

# Define shorter variable names
S.xx.step = kalman.results$S.xx.step
X.hat.now.history = kalman.results$X.hat.now.history
X.hat.step = kalman.results$X.hat.step
# Reconstruction coordinates
recon.coords = polar.to.cartesian(X.hat.now.history[,1], X.hat.now.history[,2])
```

A.5 Plot reconstructed trajectory

```
source('runkalman.R')


pdf('../plots/reconstruction.pdf', 7, 7)
plot.data(recon.coords$x, recon.coords$y, plot.lim=c(0,42000), type="l",
          main="Reconstructed trajectory in the xy-plane")
points(recon.coords$x, recon.coords$y, pch=20, cex=0.7)
points(dat.x, dat.y)
add.legend(show.prediction=FALSE, show.conf.interval=FALSE)
dev.off()
```

A.6 Predicting trajectory

```
source('functions.R')
source('runkalman.R')


k = 6

# Setup variables with first prediction
X.hat.pred = matrix(0, k, 3)
X.hat.pred[1,] = X.hat.step
S.xx.pred = array(0, dim=c(3, 3, k))
S.xx.pred[,,1] = S.xx.step

# Make predictions
for (i in 2:k) {
    X.hat.pred[i,] = A %*% X.hat.pred[i-1,]
    S.xx.pred[,,i] = A %*% S.xx.pred[,,i-1] %*% t(A) + S.1
}

r.pred = X.hat.pred[,1]
theta.pred = X.hat.pred[,2]
r.pred.err = 1.96*sqrt(S.xx.pred[1,1,])
theta.pred.err = 1.96*sqrt(S.xx.pred[2,2,])

pred.cartesian = polar.to.cartesian(r.pred, theta.pred)
x.pred = pred.cartesian$x
y.pred = pred.cartesian$y

theta.min = theta.pred - theta.pred.err
theta.max = theta.pred + theta.pred.err
r.min = r.pred - r.pred.err
r.max = r.pred + r.pred.err

# Plot all predictions
pdf(sprintf('../plots/all-predictions.pdf', 7, 7))
plot.predictions(dat.x, dat.y, x.pred, y.pred, main="6-step predictions")
plot.reconstruction(recon.coords$x, recon.coords$y)
add.legend()
dev.off()

# Plot all predictions magnified
pdf(sprintf('../plots/all-predictions-magnified.pdf', 7, 7))
plot(dat.x, dat.y, xlim=c(-6000, 15000), ylim=c(33000, 44000), xlab="x", ylab="y",
     cex.main=1.4, cex.lab=1.4, cex.axis=1.4, main="6-step predictions close up")
points(x.pred, y.pred, pch=20, col="red")
plot.reconstruction(recon.coords$x, recon.coords$y)
add.legend()
dev.off()

# Plot each prediction with confidence interval
for (i in 1:k) {
    pdf(sprintf('../plots/prediction-%d.pdf', i), 7, 7)
    plural = if (i == 1) '' else 's'
    main = sprintf("Prediction %d time step%s ahead", i, plural)
    plot.predictions(dat.x, dat.y, x.pred[i], y.pred[i], main=main)
    plot.reconstruction(recon.coords$x, recon.coords$y)
    display.confidence.interval(r.min[i], r.max[i], theta.min[i], theta.max[i])
    add.legend(show.conf.interval=TRUE)
    dev.off()
}
```

```
# Save predictions and confidence intervals in latex table
r.table.lines = c()
theta.table.lines = c()
line.template = "%d & %.02f & %.02f & [%.02f; %.02f] %s"
for (i in 1:k) {
    delim = if (i==k) '' else '\\\\'
    r.line = sprintf(line.template, i, r.pred[i], r.pred.err[i], r.min[i],
                     r.max[i], delim)
    r.table.lines = c(r.table.lines, r.line)
    theta.line = sprintf(line.template, i, theta.pred[i], theta.pred.err[i],
                         theta.min[i], theta.max[i], delim)
    theta.table.lines = c(theta.table.lines, theta.line)
}

r.table.file = file('../tables/r-predictions.tex')
writeLines(r.table.lines, r.table.file)
close(r.table.file)

theta.table.file = file('../tables/theta-predictions.tex')
writeLines(theta.table.lines, theta.table.file)
close(theta.table.file)
```

References

[1] Henrik Madsen, *Time Series Analysis*. Chapman & Hall/CRC, 1st Edition, 2008.