The Cooper Union Department of Electrical Engineering
Prof. Fred L. Fontaine
ECE416 Adaptive Algorithms
Project: Kalman Filters
April 16, 2023

In this project, you will be implementing and studying the performance of several forms of the discrete-time Kalman filter, specifically:

- Standard Kalman filter (for linear systems).
- Unscented Kalman filter.

Standard Kalman Filter

The dynamical system is:

$$\begin{aligned} x_{n+1} &= Ax_n + v_n^{(x)} \\ y_n &= Cx_n + v_n^{(y)} \end{aligned}$$

where A, C can be time-varying. You are going to implement the Kalman filter in covariance form, in the Joseph form. Specifically, you should write a function that does one iteration, taking $K(n, n-1), x(n|n-1)$ and producing $K(n,n), K(n+1,n), x(n|n), x(n+1|n)$ as output.
There are two A matrices:

$$A_1 = \begin{bmatrix} -\frac{3}{2} & -\frac{5}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix} \qquad A_2 = \begin{bmatrix} -3 & -5 \\ 1 & 1 \end{bmatrix}$$

and:

$$C = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

Also:

$$Q_x = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix} \qquad Q_y = 0.05$$

1. **Preliminary analysis:**

   Compute the eigenvalues of A_1, A_2. You will note in one case the system is stable, in the other it is unstable- which is which?

   Also compute the observability matrix $\mathcal{O}$ and find its rank to confirm each system is observable.

   Let K denote the steady-state prediction error covariance matrix, i.e., the limit of $K(n, n-1)$. The matrix K is found as the solution to the discrete time algebraic Riccatti equation (DARE). The MATLAB function *idare* solves this. However, the notation MATLAB uses in the function differs from our notation here. The following table provides the "mapping" between the notations; note that the number of states is N, number of inputs is m (here 0) and number of outputs is p.

Kalman	*idare*
K	X
A'	A
$I_{N\times N}$	E
C'	B
Q_y	R
Q_x	Q
$0_{N\times p}$	S

Call *idare* to compute K for each case. Later, you will be comparing your results to K.

2. **Kalman Filter**

We want to run the filter over 100 iterations. During the first 50, $1 \leq n \leq 50$, run it with the stable form A_1, then switch to A_2 for iterations $51 \leq n \leq 100$. Use initial condition $x_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ to generate the state trajectory and output x_n, y_n for $1 \leq n \leq 100$. Then use initial conditions for the Kalman filter $\hat{x}(1|0) = x_0$ and $K(1,0) = 0.1I$.

Here, K denotes the theoretical steady-state prediction error covariance matrix (for each A_1, A_2) found above. Then perform the following analysis of results:

(a) For $1 \leq n \leq 100$, compute $\|K(n, n-1) - K_{\text{ideal}}\|$ and plot it; here K_{ideal} is the solution to the algebraic Riccatti equation, which is DIFFERENT for $n \leq 50$ and $n \geq 51$. How well do the results match, in general, and long does it take the Kalman filter to switch to the new solution after the transition point occurs?

(b) Superimpose plots of the true trajectory x (i.e., plot $x(1,n)$ versus $x(2,n)$, where column denotes time index), and the estimated and predicted trajectories x_e, x_p, each a different color.

(c) Examine x for $n > 50$, and confirm it is "blowing up". Plot the trajectory $x(1,n)$ versus $x(2,n)$ for $51 \leq n \leq 100$, only. Then, separately, graph $\|x(:,n) - x_p(:,n)\|$ and $\|x(:,n) - x_e(:,n)\|$. **Note:** Make sure you line up the predicted and estimated values correctly, i.e., one point of x_e should estimate the true state x at some time n. You can check this by confirming your covariance matrices seem reasonable, in the next step below.

(d) Let $\varepsilon_p(n) = x(n) - \hat{x}(n|n-1)$, and ε_p is matrix whose columns are the prediction error vectors over a span of time, say M columns, then the prediction error covariance matrix is $\hat{K}_p = \frac{1}{M}\varepsilon_p\varepsilon_p^T$. Similarly, $\hat{K}_e = \frac{1}{M}\varepsilon_e\varepsilon_e^T$ where ε_e is a matrix of M columns, each column an estimation error $x(:,n) - x_e(:,n)$. Compute $\hat{K}_p, \hat{K}_e$ for the span of time BEFORE the A matrix changes, and again AFTER it changes (separately). If K denotes the solution to the algebraic Riccatti equation, compute $\left\|K - \hat{K}_p\right\|$ (each case separately). Also, let $\Delta K = K_p - K_e$, and check to see if $\Delta K > 0$ (you should know what this means and how to check it!) This confirms the estimated state is better than the predicted state. Again, look at this for each half (before and after A changes).

Remarks: Does the Kalman filter still work reasonably for the unstable system? Does the Kalman filter handle the "switch" well?

Unscented Kalman Filter

This example is taken from the following paper:
S. J. Julier, J. K. Uhlmann, "Unscented filtering and nonlinear estimation," *Proc. IEEE,* vol. 92, no. 3, March 2004, pp. 401-422."
From the article: *A vehicle enters the atmosphere at high altitude and at a very high speed. The position of the body is tracked by a radar which measures range and bearing.*
Three forces are in effect:

1. Aerodynamic drag.
2. Gravity.
3. Random terms.

A "ballistic trajectory" means motion under constant acceleration. Initially, the trajectory is ballistic, but as the atmospheric density increases the drag effects become more significant and alters the trajectory.
There are five state variables, $x_1(n), x_2(n), x_3(n), x_4(n), x_5(n)$.

- The 2-D position $(x, y) = (x_1(n), x_2(n))$. These values are with respect to the center of the earth.
- The 2-D velocity $(v_x, v_y) = (x_3(n), x_4(n))$.
- The drag depends on a *ballistic coefficient* β. This coefficient is not known precisely, and may even be variable, yet it must always be positive. This can be modeled by defining:
$$\beta(n) = \beta_0 e^{x_5(n)}$$
where β_0 is a prescribed "nominal" value. Thus, the fifth state variable is:
$$x_5(n) = \log \frac{\beta(n)}{\beta_0}$$

The dynamical equations are:

$$\begin{aligned}
\dot{x}_1(n) &= x_3(n) \\
\dot{x}_2(n) &= x_4(n) \\
\dot{x}_3(n) &= D(n) x_3(k) + G(n) x_1(n) + v_3^{(x)}(n) \\
\dot{x}_4(n) &= D(n) x_4(k) + G(n) x_2(n) + v_4^{(x)}(n) \\
\dot{x}_5(n) &= v_5^{(x)}(n)
\end{aligned}$$

where $\mathbf{v}^{(x)}$ is the process disturbance, with no (direct) disturbance on the position, i.e., $v_1^{(x)}(n) = v_2^{(x)}(n) = 0$. The nonlinearities enter through the drag D and gravity G:

$$\begin{aligned} D(n) &= -\beta(n)\, e^{(R_0 - R(n))/H_0} V(n) \\ G(n) &= -\frac{Gm_0}{R^3(n)} \end{aligned}$$

where R is the distance from the center of the earth, V is the (scalar) speed, and $Gm_0 = 3.986 \times 10^5$, $R_0 = 6374$, and $H_0 = 13.406$. Specifically:

$$\begin{aligned} \beta(n) &= \beta_0 e^{x_5(n)} \\ R(n) &= \sqrt{x_1^2(n) + x_2^2(n)} \\ V(n) &= \sqrt{x_3^2(n) + x_4^2(n)} \end{aligned}$$

The radar is located at $(x_r, y_r) = (R_0, 0)$ and measures range and bearing:

$$\begin{aligned} r(n) &= \sqrt{(x_1(n) - x_r)^2 + (x_2(n) - y_r)^2} + v_1^{(y)}(n) \\ \theta(n) &= \tan^{-1} \frac{x_2(n) - y_r}{x_1(n) - x_r} + v_2^{(y)}(n) \end{aligned}$$

where $\mathbf{v}^{(y)}$ is the measurement disturbance, and our output measurement vector is:

$$\mathbf{y}(n) = \begin{bmatrix} r(k) \\ \theta(k) \end{bmatrix}$$

We assume the signal is samples at $f_s = 10Hz$, i.e., once every 0.1 sec.
One of the issues is to obtain a discrete-time model for:

$$\dot{x}(t) = f\left(x(t), t, v^{(x)}(t), u(t)\right)$$

where $u(t)$ can denote an input or, more generally, a set of **known** parameter values; any unknown quantities are represented by the disturbance $v^{(x)}$. Assuming time is discretized so $t = ndt$ (here $dt = 0.1$ sec), the simplest discretization is the *Euler approximation:*

$$x_{n+1} = x_n + dt \cdot f\left(x_n, ndt, v_n^{(x)}, u_n\right)$$

In our case, the disturbance is additive, so in the discrete form it is scaled by dt. In any case, the authors suggest this is not sufficiently accurate for the dynamical system at hand. A better approach is a *midpoint approximation,* which basically does a two-step update $t \to t + \frac{1}{2}dt \to t + dt$, so:

$$x_{n+1} = x_n + dt \cdot f\left(x_n + \frac{1}{2}dt \cdot f\left(x_n, ndt, 0, u_n\right), n + \frac{1}{2}dt, v_n^{(x)}, u_n\right)$$

MATLAB code is provided to you for this: *uhlprocsim* provides either Euler or midpoint simulation of the process equation, calling *uhlproc* internally. The code for the measurement equation, *uhlmeas,* is also provided to you. In order to either simulate the trajectory, or run

a UKF on the system, you just need to call *uhlprocsim* and *uhlmeas*. For this project, make sure the midpoint technique is used.
For the UKF, I suggest using the sigma point generation method as in the article. For the case of an N_x-dimensional random vector X with $\mathbf{0}$ mean and covariance I, the $2N_x + 1$ sigma-points are:

$$\begin{aligned}
\boldsymbol{\chi}_0 &= \mathbf{0},\, w_0 = \frac{1}{3} \\
\chi_n &= \sqrt{\frac{N_x}{1-w_0}}\mathbf{e}_n,\, w_n = \frac{1-w_0}{2N_x},\, 1 \le n \le N_x \\
\chi_{N_x+n} &= -\sqrt{\frac{N_x}{1-w_0}}\mathbf{e}_n,\, w_n = \frac{1-w_0}{2N_x},\, 1 \le n \le N_x
\end{aligned}$$

where $\mathbf{e}_n \in \mathbb{R}^{N_x}$ has 1 in the n^{th} coordinate, and 0 otherwise. For a general mean μ and covariance Σ, the usual transformation $\chi_n \to \mu + \Sigma^{1/2}\chi_n$ where $\Sigma^{1/2}$ is the Cholesky factor should be applied (i.e., the sigma points are $\pm$ the columns of the Cholesky factor, scaled by a constant and translated by the mean).
As we run the system, here are the default values.
Actual system: Assume initial conditions for the vehicle are:

$$x\left(0\right) = \begin{bmatrix} 6400.4 \\ 349.14 \\ -1.8093 \\ -6.7967 \\ 0.6932 \end{bmatrix}$$

and the disturbance covariance matrices are:

$$\begin{aligned}
Q_x &= diag\left\{10^{-8}, 10^{-8}, 2.404 \times 10^{-5}, 2.404 \times 10^{-5}, 10^{-8}\right\} \\
Q_y &= diag\left\{1, 17 \times 10^{-3}\right\}
\end{aligned}$$

Assume the initial conditions for the UKF are:

$$\hat{x}\left(1|0\right) = \begin{bmatrix} 6400 \\ 350 \\ -2 \\ -7 \\ 0.65 \end{bmatrix}$$

and:

$$K\left(1,0\right) = diag\left\{10^{-4}, 10^{-4}, 10^{-4}, 10^{-4}, 1\right\}$$

It is not unreasonable to assume we have approximate knowledge of the initial position and velocity, for example. The ballistic coefficient is harder to determine, hence we are building in a larger initial error in it. Also setting the corresponding value in the initial $K\left(1,0\right)$ larger enables the UKF to start with a wider "net" if you will (think of the spread of the sigma points), corresponding to the larger initial uncertainty we have in that parameter.
Finally, the simulation should be over 50 seconds, which means 500 time steps.
Theory and Setup

1. Following the work in the lecture notes, assuming 0 mean and covariance I, shown that the empirical mean of the sigma points is 0, the empirical covariance is I, the skewness of each component is 0, and the Kurtosis of each component is $\frac{N_x}{1-w_0}$. In this case, $w_0 = 1/3$ and $N_x = 5$; show this yields a Kurtosis of 3, which matches the Gaussian. **Remark:** The form given in the notes matches Kurtosis regardless of dimensionality, but would give negative w_0 which can be disadvantageous. The advantage here is, though it only works for this specific N_x, it does have all positive weights.

2. Write a function *[w,sig]=sigmapts(mu,Cchol)* that will generate the sigma points and weights using the above process given input mean vector and Cholesky factor (i.e., not the direct covariance matrix). [Don't call the Cholesky factor *chol* because that is the name of the MATLAB function that computes the Cholesky factor!]

3. Write a function *sigmu= sigmamean(w,sig)* that will compute the empirical mean from given sigma points and weights.

4. Write a function called *sigcov= sigmacov(w,sigx,sigy)* that will compute the empirical cross-covariance for X, Y prescribed by sigma points $sigx, sigy$ respectively, with common weights in w. This function, when called simply as *sigmacov(w,sigx)* should compute the auto-covariance matrix. [It should internally call *sigmamean*]

5. Write a function that performs one iteration of the UKF: *[xpnew,Kpnew,xe,Kxxe]= ukfiter(xp,Kp,Qx,Qy,ymeas,t,dt)* Here xp, Kp are initial predicted state and prediction error covariance matrix, respectively, and it outputs both the estimated values xe, Ke and the updated predicted values $xpnew, Kpnew$. The process and measurement covariance matrices are Qx, Qy, $ymeas$ is the measured vector, and t is the discrete iteration index, and dt is the parameter for the discretization. Internally, it should call *uhlprocsim,uhlmeas,sigmapts,sigmamean,sigmacov.*

Simulating the System / UKF

You should simulate the system and simultaneously run the UKF. As you update each state and produce the measurement (r, θ), run the UKF to keep up with estimated/predicted states and covariance matrices.

Then create three graphs:

1. Plot the actual position and estimated trajectory (different colors, superimposed). That is, plot state variable x_1 versus state variable x_2. Place a marker and text label at the initial point, so we can see where it starts.

2. Plot the actual velocity trajectory and its estimate (different colors, superimposed). That is, plot state variable x_3 versus state variable x_4. Again, place a marker at the starting point.

3. Plot the actual $\beta = \beta_0 e^{x_5}$ and the estimated value (i.e., using the estimated x_5 state variable), superimposed. Although we are not directly performing a parameter estimation with a JUKF, in effect inclusion of β as a state variable (actually, parametrized in a way to enforce positivity in β, which is a common technique) **is** in fact how parametric estimation would be done anyway! As a remark, since this is embedded with the

other states, the covariance matrices the UKF generates implies we are not assuming the uncertainty in β is uncorrelated with that of the other states, hence this is closer in form to a JUKF than a DUKF (which would run a separate UKF for the β and for the other states).

Now put a whole wrapper around the whole thing and repeat it 5 times– in the end there will be 5 sets of 3 graphs. [You don't have to store things between runs] You are doing this to see how much variability there may be, informally.