Kalman Filter Exercise

DECAR Group
Department of Mechanical Engineering, McGill University
817 Sherbrooke Street West, Montreal QC H3A 0C3

May 4, 2022

The goal of this exercise is to code a Kalman filter from scratch, starting with the ordinary differential equation of the system you are trying to estimate.

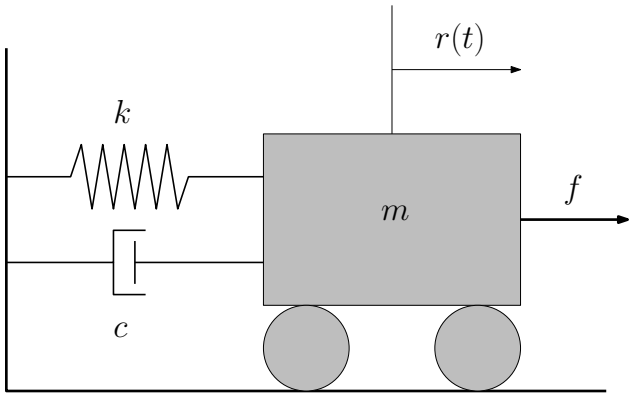

Figure 1: A mass-spring-damper system subject to an external force.

Figure 1 shows a mass-spring-damper system, consisting of a mass attached to a spring with spring constant k, a dashpot with damping constant c, and also subject to an external force f. The position of the cart from its equilibrium position is denoted as $r(t)$ and is a function of time t.

The equation of motion of the cart is given by

$$m\ddot{r}(t) + c\dot{r}(t) + kr(t) = f(t). \tag{1}$$

1 Generating "Ground Truth" Data

"Ground truth" refers to the *true* values, absent of any noise or inaccuracy, of a particular quantity of interest. In our case, since we will be doing everything in simulation, we will be generating our own ground truth values for position, velocity, acceleration, force, and then intentionally corrupting the data with noise.

1. Write the equation of motion (1) in state-space form. That is, write (1) in the form of

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)). \tag{2}$$

   The ODE in (1) happens to be linear, so you can also write it as $\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$.

2. Using Python, simulate the trajectory of the mass-spring-damper system with no external force (i.e. $f(t) = 0$), and with any non-zero initial condition, such as $r = 5, \dot{r} = 0$. You should use a numerical integration algorithm in Python, like `scipy.integrate.RK45`.

3. Simulate the trajectory again, but this time with a non-zero external force. As an example, you can let $f(t) = A\sin(\omega t)$, for some chosen value of A and ω.

You should now have data for position, velocity, and external force acting on the cart for a discrete set of times. You should code this part with the ability to easily change the time points at which you obtain your position, velocity, external force data (since we will use this to generate sensor measurements at specific frequencies).

2 Generating Sensor Measurements

Suppose we now have an accelerometer, which measures the acceleration of the cart $\ddot{r}(t)$, as well as a position sensor, which measures $r(t)$. The noisy acceleration measurement is given by

$$u^{\text{acc}}(t) = \underbrace{\frac{1}{m}\left(f(t) - kr(t) - c\dot{r}(t)\right)}_{\triangleq a(t)} + w(t), \qquad w(t) \sim \mathcal{N}(0, Q(t)), \tag{3}$$

and the noisy position measurement is given by

$$y(t) = r(t) + v(t), \qquad v(t) \sim \mathcal{N}(0, R(t)). \tag{4}$$

1. Choose the variances $Q(t), R(t)$ associated with the accelerometer and position noise, respectively. These can just be constant values, and for now, use them directly to generate sensor noise.
2. Generate accelerometer and position measurements at a frequency of your choice using your ground truth data. You should use a normally-distributed random number generator such as `numpy.random.randn`.

Now, with these sensor measurements, our next goal is to somehow use them to estimate the position and velocity of the mass, without using ground truth at all. The fact that we are using specifically an accelerometer, as opposed to some "external force sensor", makes our lives a little easier. Specifically, we actually only need to consider the following kinematics,

$$\ddot{r}(t) = u^{\text{acc}}(t), \tag{5}$$

as opposed to the full dynamics of the mass-spring-damper, shown in (1), involving the various forces, mass, etc.

3 Discretization

1. Rewrite equation (5) and (4) as

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}a(t) + \mathbf{L}w(t), \qquad w(t) \sim \mathcal{N}(0, Q(t)), \tag{6}$$
$$y(t) = \mathbf{C}\mathbf{x}(t) + v(t), \qquad v(t) \sim \mathcal{N}(0, R(r)), \tag{7}$$

   respectively.

2. Using a Zero-Order-Hold, write the above equations as

$$\mathbf{x}_k = \mathbf{A}_{k-1}\mathbf{x}_{k-1} + \mathbf{B}_{k-1}a_{k-1} + \mathbf{w}_{k-1}, \qquad \mathbf{w}_{k-1} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_{k-1}), \tag{8}$$
$$y_k = \mathbf{C}_k\mathbf{x}_k + v_k, \qquad v_k \sim \mathcal{N}(0, R_k). \tag{9}$$

   That is, form a discrete time state-space system. You do not need to generate an analytical expression for $\mathbf{Q}_{k-1}$ (although it is possible), but rather simply use Van Loan's method. For R_k, simply use the same value you used to generate the sensor measurements in the first place.

4 The Kalman Filter

Now that we have a discrete time state space model, along with corresponding covariances, we can readily implement a Kalman filter.

1. Implement a Kalman filter to produce an estimate of the state $\hat{\mathbf{x}}_k$ at all time steps, along with a corresponding covariance $\mathbf{P}_k$, using the accelerometer measurements, position measurements, an initial guess of the state $\hat{\mathbf{x}}_0$, and an initial guess of the covariance $\mathbf{P}_0$.

   Choose your own initial guess. You can set it to be the true value of the state, or some other state near it, and you should see the Kalman filter converge to the true state estimate.

2. Plot the components of the true state $\mathbf{x}$ versus estimated state $\hat{\mathbf{x}}$.

3. Plot the components of the estimation error $\mathbf{e} = \hat{\mathbf{x}} - \mathbf{x}$ versus time, along with the corresponding $\pm 3\sigma_k$ bounds. The values for σ_k can be obtained from the square root of the diagonal entries of $\mathbf{P}_k$.

4. Implement a Kalman filter with position measurements occuring at 1/10th the frequency of the accelerometer measurements, and repeat the above plot.

5 Extended Kalman Filter

Now, instead of position measurements, suppose we actually have distance measurements to some point on the wall at height h, where the wall is a distance d away from the cart's equilibrium position. The measurement model becomes

$$y(t) = \sqrt{(x+d)^2 + h^2)} + v(t), \qquad v(t) \sim \mathcal{N}(0, R(t)). \tag{10}$$

1. With the new measurement model, form a discrete-time state space system of the form,

$$\mathbf{x}_k = \mathbf{A}_{k-1}\mathbf{x}_{k-1} + \mathbf{B}_{k-1}a_{k-1} + \mathbf{w}_{k-1}, \qquad \mathbf{w}_{k-1} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_{k-1}), \tag{11}$$

$$y_k = \mathbf{g}(\mathbf{x}_k) + v_k, \qquad v_k \sim \mathcal{N}(0, R_k). \tag{12}$$

2. Linearize the measurement model to produce a linearized state-space system of the from

$$\delta\mathbf{x}_k = \mathbf{A}_{k-1}\delta\mathbf{x}_{k-1} + \mathbf{B}_{k-1}\delta a_{k-1} + \delta\mathbf{w}_{k-1}, \qquad \delta\mathbf{w}_{k-1} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_{k-1}), \tag{13}$$

$$\delta y_k = \mathbf{C}_k\delta\mathbf{x}_k + \delta v_k, \qquad \delta v_k \sim \mathcal{N}(0, R_k). \tag{14}$$

   The process model is already linear, so linearizing it is trivial.

3. Implement an extended Kalman filter on the linearized discrete-time state space system, plotting the estimation error and $\pm 3\sigma$ bounds.

6 Tips and Hints

1. Try to get the math correct *before* you start coding. Sometimes though, this can be an iterative process.

2. For debugging purposes, it is often useful to test the filter in the absence of noise, and test the filter with the initial estimate set to the ground truth value. It can be helpful to code a toggle so that you can easily turn off or on noise in the measurement generation.

3. Back up your work using git!