ELEC 3004/7312: Digital Linear Systems: Signals and Control!

Tutorial 5 [Week 10] - State Space, the Kalman Filter and Linear Quadratic Regulation

Todays tutorial is almost exlusively in Matlab - as soon as you arrive fire Matlab up and download the course materials. This tutorial may extend into Tutorial 6.

Part 1: state space

State space is a convienent format for representing linear systems. Instead of considering a system as a laplace equation directly, We re-write the system as a set of four matrices. The matrix form of the system convieniety provide the intermediate values for each pole of the system.

We are going to use Problem set 1 Q5 as the basis for our first steady state system. Open problem set 1 from the class website.

Before we start solving any math we want to decide on the **inputs**, the **outputs** and the **state vector**. The input in this case should be the road displacement R(t). this is the independent, external variable that affects the deflection of the suspension. Initally, the output will be Y(t), the position of the main car body. Finally the state vector is an internal set of variables (by internal we mean we don't observe them directly) which fully describe the dynamic of the system. To decide on the state vector, we are going to have to look at what forces are acting upon the internals of the system, and that means looking the the equations from lecture 4 p 36. you will see the system is fully described by the variables:

$$R(t)$$
$$Y(t)$$
$$X(t)$$
$$\dot{Y}(t)$$
$$\dot{X}(t)$$
$$\ddot{Y}(t)$$
$$\ddot{X}(t)$$

When we look at how to find the **state transition matrix**, the first of our four matrices, you will see why we can describe the internal state completely with the following state vector:

$$Y(t)$$
$$X(t)$$
$$\dot{Y}(t)$$
$$\dot{X}(t)$$

The **state transition matrix** A descibes how to find the next state given the prior state. The **Control Input matrix** B describes how our control inputs (in this case the road) affects the state. As this system is continuous, we will only derive a continuous state transition matrix in the tutorials, and use matlab to find the discrete equivalent [1]. We define the state update function as follows:

[1] The discrete equivalent is not so very different, if you are interested see `http://www.engr.iupui.edu/~skoskie/ECE595_f05/handouts/discretization.pdf`

$$\dot{x} = A \cdot x + B \cdot u$$

where x is the **state vector** and u the **input vector**, which in this case is $R(t)$. given we have four internal states, A is a $4x4$ matrix and given we have one input B is a $4x1$ matrix. The tutors will now go through the derivation of the A matrix and B matrix from the equations described in lecture 4 p 36

Finally the **output matrix** C describes how the output is linked to the state vector, and the **feed forward** matrix D describes how the input links directly to the output. in general, there is no feed forward and D = 0, and C observes a subset of the states directly. The output function is as follows

$$y = C \cdot x + D \cdot u$$

the tutors will quickly derive C now for the problem from PS1 Q5.

Now open the CarSS.m file from the tutorial materials. Here you will find the car problem written in state space. $ABCD$ are all filled in, and are parametrically defined by the variables on lines one to 5. You now have two tasks:

*Find the maximum step displacement of the **road**, given the suspensions spring has a maximum displacement of 10cm before bottoming out. IE find $Disp_{Spring} = (Y(t) - X(t))$; find the maximum displacement using stepinfo for a unit step, and $Disp_max = 0.1/Peak$*

Modify the wheel spring constant to be a much more realistic 500kN/m . Now experiment with the damping constant b to minimise rise time (5000 is close to critically damped), then redo the analysis above. Is this more realistic?

Part 2: The Kalman filter

So we have seen how a state space model can be applied to a linear system and used to derive the dynamics. We saw how simple it was to derive $Y(t) - X(t)$ and compute the true displacement of the suspension, not the approximation from PS1. Now we will see how filters may be implemented in state space.

Until recently, filters have most likely been described to you from a Fourier viewpoint, where we want to manipulate a signal made up of a bunch of sinewaves, and recover only the sinewaves were interested in. But a filter has a more fundamental concept, we want to remove as much unwanted signal as possible, and *estimate* some underlying signal. The kalman filter solves this problem exceptionally well in the case that our unwanted noise is **additive gaussian** in nature, and our signal is a linear system being driven by an underlying dynamic that is **also a gaussian**.

were going to need to extend the state space model we described in part 1 a little to accomodate these gaussian noise sources. we will extend our state transition function as follows:

$$\dot{x} = A \cdot x + B \cdot u + v$$

where v is known as the process noise. This is the unmodellable change in the state x caused by external factors. If we put this in the example of Drone from PS2 Q5, this is wind shear and turbulence pushing the drone off course. Now we extend our measurement model:

$$y = C \cdot x + D \cdot u + w$$

where w is known as the measurement noise. This is the noise inherent to our gyroscope due to thermal fluctuations, power supply noise, and uncorrected interference.

The two noise sources cannot be found directly. If they could we would use that knowledge to find the postion of the system exactly. But we can estimate their **covariance**, which can be used to infer the statistics of the system we are investigating.

We will not derive the kalman filter again, but will go over its basic principle now. The kalman filter can be seen as iteratively solving two problems, Which tutors will describe quickly now[2].

*Find an A-Priori **Prediction** of the current state of the system, given the prior state and the state transition model*

*Update the current state of the system to an **a posteriori** estimate based on the computed prediction and the observed position of the system state. The kalman gain tells us how much we need to correct our prediction by*

The Kalman.m file implements 3 forms of the Kalman filter. The first uses the convienient dsp.KalmanFilter function provided by matlab, The second follows the wikipedia implementation of the kalman filter directly, updating the kalman gain at each stage. This is incredibly expensive for a microcontroller, as there is a matrix inverse and a large number of redundant operations, when you realise that the kalman gain converges to a steady-state value after a short number of cycles. In practise, the third implementation is the most common. Here a kalman gain is pre-computed using the Lupanyov expansion, and the fixed kalman gain used to update the estimate. This is a significantly lower computational load and gives identical steady state performance.

[2]see lectures: `http://robotics.itee.uq.edu.au/~elec3004/lectures/L14-AaptiveFilters.pdf`

Part 3: The Linear Quadratic Regulator

You will see the implementation of a controller using PID in the Prac. Here you get the chance to see the state space implementation of a controller, namely a Linear Quadratic Regulator and hopefully get a feeling for the advantages and disadvantages of both. Namely, that while an LQR requires accurate modelling of the system inputs and dynamics, it very direcly results in a controller which accounts for coupled dynamics. PID controllers generally require more feeling about in the dark to find the right coefficents to get it to work, and are not very suited to systems with coupled dynamics.

The Linear Quadratic regulator takes a state space system, and Two cost matrices, Q and R. The Q matrix describes the cost of a state variable vaying from the desired value of that state. The R matrix describes how much it costs us to drive our inputs. think of this as associating a cost with how much the gas pedal is depressed, the more we push on the pedal, the more fuel (IE dollars) we burn in the motor. Here the function lqrd has been used to solve for the required gain[3].

Open the LQR.m script. here you will see the same state space model from the Kalman filter example, but if you look at lines 68 to 76 you will see the definition of matrices Q and R, the two cost matrices. Then the for loop from 87 to 98 implements the state update, computes the kalman filter estimate of the state, and computes the ideal input control effort for the system.

The tutors will go over the steps the simulation is undertaking to control the current state. you can see from the plot that the system converges in two seconds to stay within 2m of the origin. The LQR is regulating the input as can be seen in figure 2, which plots the input vector, to drive the system towards the origin.

modify Q and R to obtain different controller performance. What change will result in a controller that uses less gas, IE the total control effort is low.

Can you push the controller too far, and make it unstable?. what difference does changing line 98 - input(:,i) = -LQRGain estimatedState(:,i); to input(:,i) = -LQRGain* state(:,i); make? The result is much better, but why is this never possible in practise?*

[3] see either `https://ocw.mit.edu/courses/mechanical-engineering/2-154-maneuvering-and-control-of-surface-and-underwater-vehicles-13-49-fall-2004/lecture-notes/lec19.pdf` or `https://stanford.edu/class/ee363/lectures/clqr.pdf` for the underlying math