More Train Fun

Having missed your train several times you now want to improve your estimates of where the train is in the dense woodland. You have decided to use the velocity information included with the real time online GPS data for the train you found online. It remains that, due to the tree cover, the error in the position of the train from the GPS can be thought of as coming from a normal distribution of $\mu = 0$ and standard deviation $\sigma = 10mi.$ The same is true of the velocity measurements. This time we will do things a bit differently. We no longer trust the rail company which says the train travels with an average speed of 100 mph $\pm$ 15 as we have been burned too many times missing our train. Instead we will model the trains velocity as a Markov process tracking its change in distance in times steps of 1 min. That is we will think of its velocity v(t) as,

$$v_{t+1} = v_t + z$$

Here z is a normally distributed white noise term. This way we are letting the velocity come into play instead of assuming we are always around 100mph. To estimate the noise term for the velocity we will take the uncertainty of 15 mph to be its standard deviation. This will allow us to model the position of the train as a discrete time equation given by,

$$x_{t+1} = x_t + \frac{v_t}{60}.$$

Let's revisit the approach we took before to find the optimal combination of data and model prediction, that is we will take the analysis vector $\vec{x}^a =< x^a, v^a >$ to be one which minimizes some cost functional. In the 1-d case we used:

$$J(x) = \frac{1}{(\sigma^f)^2}\left(x - x^f\right)^2 + \frac{1}{(\sigma^0)^2}(x - x^0)^2$$

The n dimensional analog is:

$$J(x) = \left\langle \vec{x} - \vec{x}^f, B^{-1}\left(\vec{x} - \vec{x}^f\right)\right\rangle + \langle \vec{x} - \vec{x}^o, R^{-1}(\vec{x} - \vec{x}^o)\rangle$$

or equivalently,

$$J(x) = \left(\vec{x} - \vec{x}^f\right)^T B^{-1}\left(\vec{x} - \vec{x}^f\right) + (\vec{x} - \vec{x}^0)^T R^{-1}(\vec{x} - \vec{x}^0).$$

Here B is the matrix of error covariances relating to the forecast and R the error covariances for the observations.

1) Find $\vec{x}^a = argmin\left(J(\vec{x})\right)$ and write it in the form $\vec{x}^a = \vec{x}^f + K\left(\vec{x}^0 - \vec{x}^f\right)$ in the end you should obtain $K = B(R + B)^{-1}$.

It is important to note here that we have assumed that our observations the same as our state variables. That is, we assumed we are directly observing x and v . This does not need to be the case though, we may instead indirectly observe the variables or only one of them. For example, perhaps our observation for velocity is given by a radar bounce back time, in this case we would need to convert that time to a velocity using some operator. It may also be that we only have measurements of position but not velocity in which case the dimension of our state space is not the same as our observation space. To

account for issues like this we use an observation operator H which maps our state space to our observation space, that is,

$$\vec{y}^f = H\vec{x}^f\,, \qquad H: R^n \to R^m$$

In this case we can write our cost function as,

$$J(x) = \left(\vec{x} - \vec{x}^f\right)^T B^{-1}\left(\vec{x} - \vec{x}^f\right) + (\vec{y} - \vec{y}^0)^T R^{-1}(\vec{y} - \vec{y}^0) \text{ where } \vec{y} = H\vec{x}.$$

2) Find $\vec{x}^a = argmin(J(\vec{x}))$ and write it in the form $\vec{x}^a = \vec{x}^f + K(\vec{y}^0 - H\vec{x}^f)$ in the end you should obtain $K = (B^{-1} + H^T R^{-1} H)^{-1}(R^{-1}H)^T$.
3) Show $K = (B^{-1} + H^T R^{-1} H)^{-1}(R^{-1}H)^T = BH^T(HBH^T + R)^{-1}$

It should be noted here that we have assumed that H is linear and that is needed for the Kalman filter. Often, though, H is nonlinear in which case it must be linearized which relates to the Google-able term Extended Kalman Filter.

One difficult piece of the puzzle is estimating B and R, R often comes from knowledge about how the measurements were taken, the accuracy of the device used etc. Estimating B can be done in several ways, if not known, one which we will talk about later with the Ensemble Kalman Filter. Here we know that the error in the GPS position and velocity data come from normal distributions with mean zero and standard deviations of 10 mi and 10mph. Since it is the tree cover causing the errors we will assume they are independent of each other. In this case we can take R to be,

$$R = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}.$$

For B we will look at what we know about the velocity. We can assume that the velocity from one minute to the next comes from a normal distribution with mean zero and standard deviation of 15. That is:

$$v_{t+1} = v_t + z$$

Then recalling how we will evolve the position,

$$x_{t+1} = x_t + \frac{v_t}{60},$$

We can plug in our model for v we can write,

$$x_{t+1} = \frac{v_t}{60}x_t + \frac{z}{60}$$

so that we can think of the train's position as a Markov Process with white noise term z/60 or we will call it w and let it have mean zero with standard deviation σ =15/60 . With this in mind, we can write our system as:

$$\vec{x}_{t+1} = A\vec{x} + Z$$

Where $A = \begin{bmatrix} 1 & 1/60 \\ 0 & 1 \end{bmatrix}$ and $Z = \begin{pmatrix} z/60 \\ z \end{pmatrix}$. That is we will say this is the true process. We can take our forecasts using $\vec{x}^f{}_{t+1} = A\overrightarrow{x_t}$. With we can calculate the covariance matrix of noise term Z as,

$$Q = E[(Z-\bar{Z})(Z-\bar{Z})^T] = E\left[\begin{pmatrix}\frac{z}{60}\\ z\end{pmatrix}\begin{pmatrix}\frac{z}{60} & z\end{pmatrix}\right] = E\left[\begin{bmatrix}\frac{z^2}{60^2} & \frac{z^2}{60}\\ \frac{z^2}{60} & z^2\end{bmatrix}\right] = \begin{bmatrix}\frac{\sigma^2}{60^2} & \frac{\sigma^2}{60}\\ \frac{\sigma^2}{60} & \sigma^2\end{bmatrix}.$$

since Z has mean zero. For the first time step this might approximate B fairly well. However, as we will propagate our model forward we will want to think about how the error changes from one time step to the next using our forecast model vs the true model, That is we want to calculate B_t , at a given time as

$$B_t = E\left[\left(\vec{x}_t - \vec{x}_t{}^f\right)\left(\vec{x}_t - \vec{x}_t{}^f\right)^T\right] = E\left[\left(A\left(\vec{x}_{t-1} - \vec{x}_{t-1}{}^f\right) + Z\right)\left(A\left(\vec{x}_{t-1} - \vec{x}_{t-1}{}^f\right) + Z\right)^T\right] = AB_{t-1}A^T + Q$$

The last equality is not totally obvious, but can be shown. (Left as a bonus exercise!)

We now have the final pieces to do our sequential Kalman Filter.

-We have our predictive equations:

$$\vec{x}^f{}_{t+1} = A\overrightarrow{x_t}$$

$$B_t = AB_{t-1}A^T + Q$$

And our Kalman update equations:

$$\vec{x}_t^a = \vec{x}^f + K\left(y^0 + H\vec{x}^f\right), \qquad K = B_tH^T(HB_tH^T + R)^{-1}$$

And one I have not mentioned!

$$B_t^a = B_t - KHB_t$$

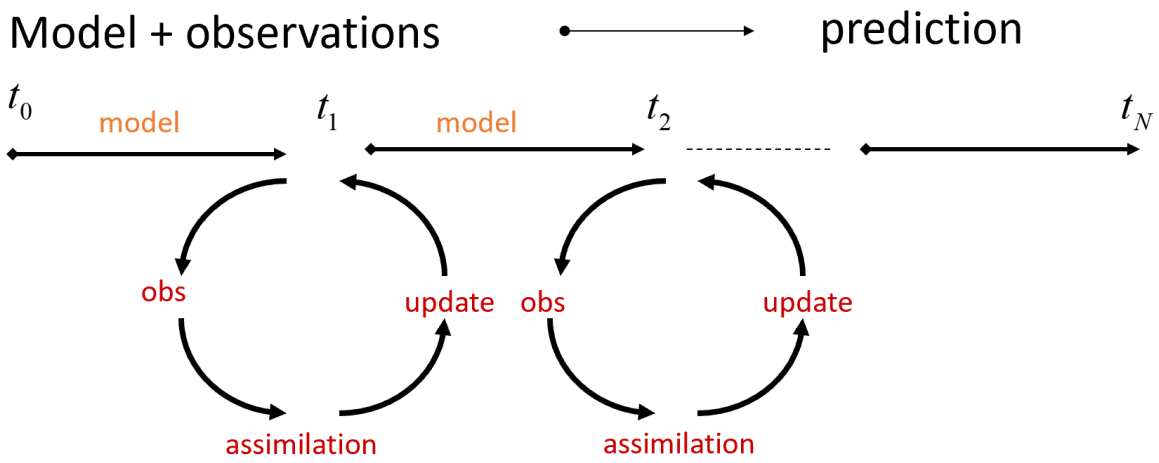
For an understanding of this have a look at this reference,

https://www.cl.cam.ac.uk/~rmf25/papers/Understanding%20the%20Basis%20of%20the%20Kalman%20Filter.pdf

It does a train problem from a probabilistic approach, it turns out we can get the same equations thinking of the state as a gaussian distribution with the analysis state given by the mean!

With these equations let's put together some code. The basic Kalman filter procedure is the following, also visualized below.

1. Using your model dynamics predict forward in time to your next observation point.
2. Update your solution and covariance estimates using the Kalman Gain and Observation operator.
3. Repeat.

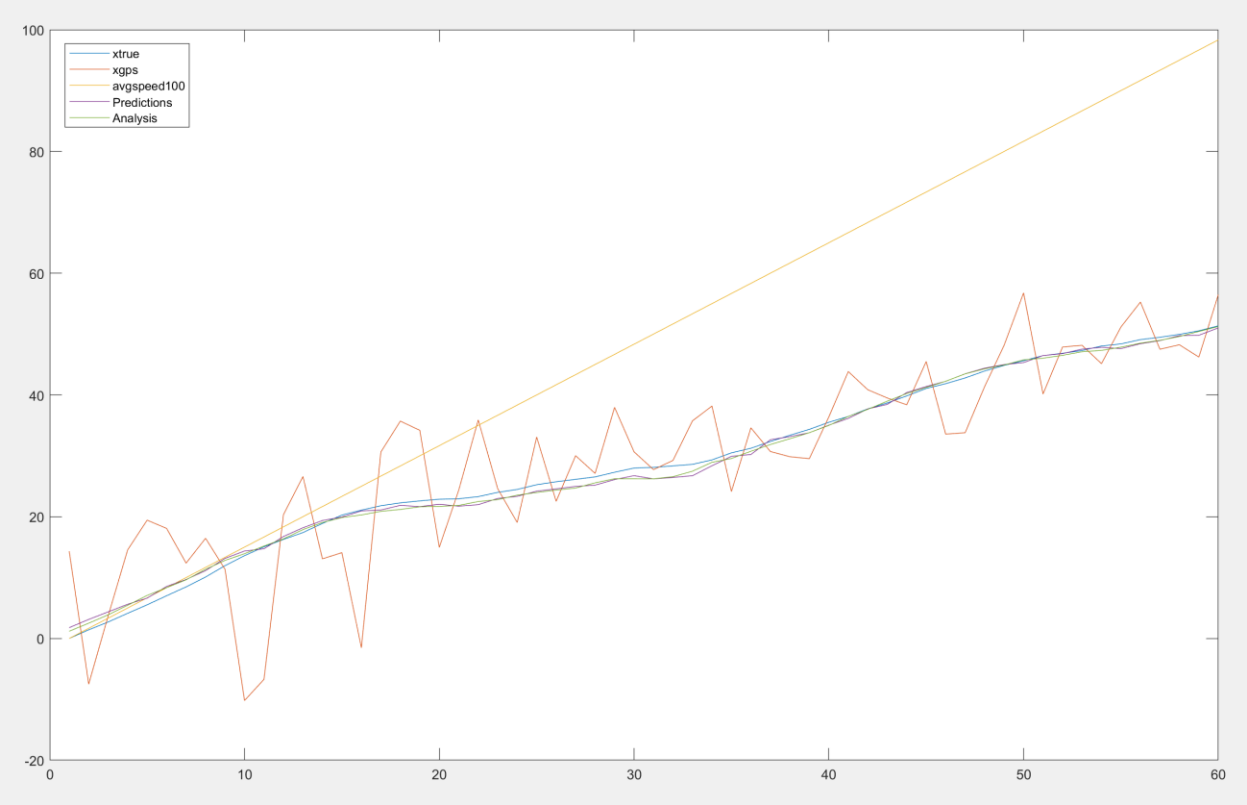


Let's get this going for our train! On Hubzero go and download KalmanTrain.zip. In the zip file you will find several matlab functions and some artificially generated data for our train. Specifically you will find

- xtrue.mat (the true position of the train for 60 min)
- vtrue.mat (the trus velocity of the train for 60 min)
- xGPS (the bad GPS observations of the trains position for 60 min)
- vGPS (the bad velocity observations for the train for 60 min)
  predict.m (makes a prediction for the model and model error covariance at the next time step)
- Kupdate.m (preforms the data assimilation update step)
- Run.m (a template for you to put all of the pieces together to get the Kalmin filter working!)
- plotstuff.m (plots the results of the scheme)
- obs.m (a function you can use to generate different observations to play around once you get the code working)
- forwardop.m (a function you can use to make new true model runs to experiment.)

Your mission will be to fill in the blanks in the right way so that run.m preforms the sequential Kalman filter process for us. Once you get it working, you should see a result like the one below.

Note this day you have lots of time for coffee!



Once you feel like you have it working try playing around a bit, what happens if you change the covariances?