Estimating the Mu Slip Curve via Extended Kalman Filtering

Robert Rudd

Automated braking system design relies on knowledge of how tire friction varies with the depth of the skid. This knowledge is captured empirically in the tire mu slip curve. An extended Kalman filter is designed to fit test data to the mu slip curve. An animation of the Kalman filter's convergence is presented.

■ Introduction

Both automobiles and aircraft have automated braking systems. These systems try to avoid skidding the tire under heavy braking or low friction conditions. Many automotive systems are content to avoid locking of the wheel (an anti*lock* system). Aircraft systems try to minimize stopping distance by maximizing available tire friction (an anti*skid* system). Shorter stopping distances have an economic value by expanding the number of airports where an aircraft may land.

Most of us are familiar with the static and dynamic coefficients [1] of friction associated with sliding surfaces. Understanding rolling friction requires knowledge of the tire "mu slip" curve [2, 3] or how tire friction varies as a function of the relative slip between the tire and the road surface. Firsthand knowledge of the mu slip curve comes from experience in driving automobiles. As drivers depress the pedal, the vehicle deceleration increases until the car enters a skid. Physically, both the tire slip and developed friction are increasing until the skid occurs. At this point, the developed friction decreases with increasing slip. It is this nonlinearity that causes a swift locking of the wheel.

A Kalman filter-based approach to antiskid control has been formulated [4]. The Kalman filter provides real-time estimates of vehicle velocity, wheel speed, and tire and brake friction, as well as tire radius and vehicle weight. The Kalman filter is particularly well-suited to the application because it allows inclusion of the underlying physics, which reduces the number of tuning iterations during system validation, while accounting for measurement and process noise.

A second benefit comes from its recursive nature. Each time a measurement is received, an estimate is produced. Other methods of curve fitting batch process a dataset.

A simplified version of the Kalman filter-based control algorithm is presented in this article. Specifically, the scope of a Kalman filter is reduced to estimate the peak amplitude and location of the tire mu slip curve in a laboratory environment. The high-quality instrumentation, relatively consistent surface, and reasonably constant normal force available in the lab allow the shape of the tire mu slip curve to be determined during the design phase of an antiskid controller. The laboratory also enables illustration of the simplified Kalman filter's more complicated cousin.

Typically, the tire mu slip curve is obtained by instrumenting a fifth wheel on an automobile or a single wheel on a dynamometer. The instrumentation consists of vehicle speed, wheel speed, drag force, and normal force. Hydraulic pressure is gradually applied to the brake to skid the tire. Tire slip is calculated from the speed data, and developed friction is calculated from the ratio of drag to normal force. The force data is usually quite noisy. The Kalman filter [5] has been designed to estimate the parameters describing this curve in the presence of noise.

For the simplified application of fitting a curve to mu slip data, other methods could be considered. In particular, the built-in function `NonlinearRegress`, using the Levenberg-Marquardt algorithm, is shown to provide reasonable estimates, although a higher number of iterations are needed.

■ The Mu Slip Curve

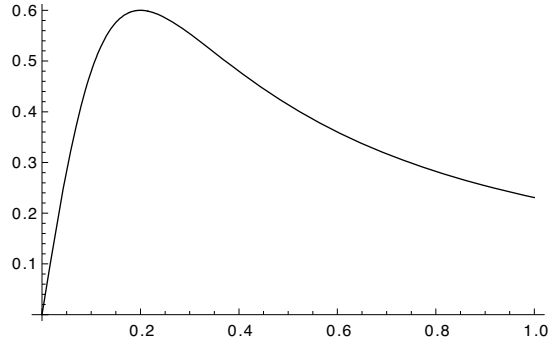
The tire mu slip curve is used to describe the developed friction of a braked tire. It is empirical in nature. A formula that has been developed with many undetermined coefficients [2, 3] is so general that it has been called the "Magic Formula." Unfortunately, the equipment needed to determine all the parameters is beyond the means of modest tire test facilities. For antiskid design, knowing only the amplitude and its location are sufficient. Therefore, we will use a much simpler formula that has only two parameters. They are the amplitude of the curve, `mup`, and the x axis location of the peak, `sp`. Here is the formula and a plot of the curve.

In[1]:=
$$\texttt{mu[s\_, mup\_, sp\_]} := \frac{2\,\texttt{mup}\,\texttt{s}}{\texttt{sp}\left(1+\left(\frac{\texttt{s}}{\texttt{sp}}\right)^2\right)}$$

In[2]:=
```
muslipplot[mup_, sp_, thickness_, color_] := Plot[mu[s, mup, sp],
    {s, 0, 1}, PlotStyle → {Thickness[thickness] , color}]
```

In[3]:=
```
muslipplot[0.6, 0.2, 0.0025, GrayLevel[0]]
```

Out[3]=



The x axis of the curve is the slip ratio. It is defined as:

slip ratio = (vehicle speed − tire circumferential speed) / vehicle speed.

The slip ratio is 0 for a free rolling tire and 1 for a locked wheel. As braking progresses, the slip ratio increases and so does the developed friction. At some point—20 percent in this case—maximum friction is reached. The goal of an antiskid system is to operate at this peak. Excursions past this peak are skids. If friction decreases without a brake pressure release, a locked wheel will swiftly and surely result.

In an automobile, drivers can sense the skid from sound, feel, and release pressure without too much damage. In the cockpit of a multiwheel aircraft, pilots cannot hear or feel any one wheel locking up. If a tire does lock, the tire wears through and blows in about one-third of a second due to the high loads and speed, making antiskid systems a necessity.

Knowing the shape of the tire mu slip curve during the design phase of an antiskid system is desirable. For automotive applications, the tire friction amplitude is between 0.1 and 1.0, representing snowy and dry surfaces, respectively. Aircraft tires have different design constraints, specifically low wear during touchdown and high loading. Designing for these constraints results in a friction amplitude between 0.1 and 0.6.

The factors affecting the location of the peak are not well known or understood. It is generally accepted that the location resides between 10 and 20 percent slip.

We now generate a mu slip curve corrupted by noise and use this data for our curve fit. The curve will have an amplitude, `muptrue`, of 0.35 with normally distributed noise, `sigmatrue`, of 0.04 standard deviation. The peak location, `sptrue`, resides at 15 percent. We will use 40 data, `nz`, points.

In[4]:=
```
muptrue    = 0.35;
sptrue     = 0.15;
sigmatrue = 0.04;
nz          = 40;
```

The list, `z`, contains noise-corrupted mu slip data-ordered pairs. The slip ratio data is not corrupted by noise since in the lab this data is generally of high quality.

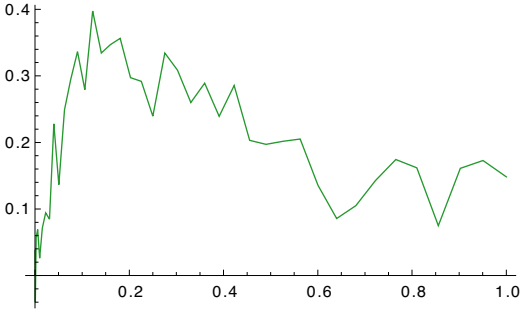
```
In[8]:= SeedRandom[2];
        z = Table[{s^2, mu[s^2, muptrue, sptrue] +
              Random[NormalDistribution[0, sigmatrue]]}, {s, 0, 1, 1/nz}];
```

For this dataset, we have set the random number generator seed so that we are all looking at the same data. We have also biased the sampling of the data, using the square function, so that there is a lower density of points on the far side of the curve. This is because the wheel will lock very quickly and result in fewer data points on the far side of the curve when sampled at a constant frequency. Here is a plot of the noise corrupted data.

```
In[10]:= measplot[nz_] := ListLinePlot[Take[z, nz],
           PlotStyle → ColorData["Legacy", "ForestGreen"]]

In[11]:= measplot[Length[z]]
```

Out[11]=



■ The Extended Kalman Filter

The Kalman filter bears the name of its inventor, Dr. Rudolf Kalman [5]. An excellent layman's explanation [6], as well as practical texts [7, 8], are available. Here we are satisfied to understand the Kalman filter as "a practical set of procedures that can be used to process numerical data to obtain estimates of parameters and variables whose values are uncertain" [9].

Here is what we need to implement a linear, nontime-varying Kalman filter:

- A state vector, x, of the parameters being estimated
- A vector of measurements, z, corrupted by noise, v
- An observation matrix, h, relating the measurements to the parameters
- An observation noise matrix, r, containing the variance of the measurements
- A covariance matrix, p, containing the variance of the difference between the estimates and true parameters

Collectively, the state vector, covariance, observation matrix, and observation noise are referred to as the model.

The standard linear Kalman algorithm recursively calculates the Kalman gain and updates the state and covariance variables with the following equation set

$$k = p_{i-1}\, h^T \left[h\, p_{i-1}\, h^T + r\right]^{-1}$$
$$x_i = x_{i-1} + k\,(z - h\, x_{i-1})$$
$$p_i = [I - k\, h]\, p_{i-1},$$

where k is the Kalman gain. These equations arise by pursuing a maximum likelihood strategy. Application of the Kalman gain minimizes the trace of the covariance matrix [7, 8]. Applying the same gain to the states minimizes the length of the error vector in the state estimates as well. Modifications to extend the filter to our nonlinear case follow.

The extended Kalman filter (EKF) algorithm first calculates the Kalman gain, `kgain`, using this equation [7, 8]:

```
In[12]:= kgain[p_, h_, r_] := p.Transpose[h].Inverse[h.p.Transpose[h] + r]
```

The measurement is incorporated into the state estimate using the following equation [7, 8],

```
In[13]:= updatex[x_, error_, k_] := x + k.error
```

where the variable, `error`, is the difference between the model and the measurement. The measurement model for a linear system is

$$z = h\, x + v,$$

where v is the normally distributed measurement noise.

Therefore, the error is

$$\text{error} = z - h\, x.$$

Finally, the covariance is updated [7, 8] to reflect that we have refined our model with outside information.

```
In[14]:= updatep[p_, h_, k_] := (IdentityMatrix[Length[p]] - k.h).p
```

The three equations are applied sequentially and recursively for each measurement.

Since we do not know a priori the amplitude or peak location of the mu slip curve, we make those the elements of our state vector:

$$x = \begin{pmatrix} \text{mup} \\ \text{sp} \end{pmatrix}.$$

For a linear system, the observation matrix is not a function of the states being estimated. For our mu slip curve there is a nonlinear relationship between the two. While convergence of a Kalman filter is assured only for a linear system, it is often applied to nonlinear systems with great success.

For a linear system, the observation matrix for the state and covariance solutions are the same. They are different for a nonlinear observation. In fact, for the state equation portion of the algorithm, there is no observation matrix per se. Instead, we calculate the difference between the measurement and our nonlinear model directly.

```
In[15]:= muerror[z_, s_, mup_, sp_] := z - mu[s, mup, sp]
```

The observation matrix for the nonlinear measurement is determined by Taylor series linearization.

```
In[16]:= Series[f[mup, sp], {mup, mup0, 1}, {sp, sp0, 1}]
```

$$\begin{aligned}\text{Out[16]=}\ & \left(f[\text{mup0}, \text{sp0}] + f^{(0,1)}[\text{mup0}, \text{sp0}]\ (\text{sp} - \text{sp0}) + O[\text{sp} - \text{sp0}]^2\right) + \\ & \left(f^{(1,0)}[\text{mup0}, \text{sp0}] + f^{(1,1)}[\text{mup0}, \text{sp0}]\ (\text{sp} - \text{sp0}) + O[\text{sp} - \text{sp0}]^2\right) \\ & (\text{mup} - \text{mup0}) + O[\text{mup} - \text{mup0}]^2\end{aligned}$$

The zero order term represents our current estimate of the function while the difference between the true and current estimated measurement is represented by the $(x - x_0)$-like terms. Therefore, the partial derivatives represent the observation matrix for the covariance update.

```
In[17]:= eh[s_, mup_, sp_] = {{∂mup mu[s, mup, sp], ∂sp mu[s, mup, sp]}}
```

$$\text{Out[17]=}\ \left\{\left\{\frac{2\,s}{\left(1 + \frac{s^2}{sp^2}\right) sp},\ \frac{4\,mup\,s^3}{\left(1 + \frac{s^2}{sp^2}\right)^2 sp^4} - \frac{2\,mup\,s}{\left(1 + \frac{s^2}{sp^2}\right) sp^2}\right\}\right\}$$

Inspection of the observation matrix elements shows that we need to know the true parameters to evaluate the observation matrix. Since we do not know the parameters, we are forced to use our current, usually incorrect, estimates. There are a few heuristic techniques available to minimize the effect of this approximation. We will use the technique of "pseudonoise" or assume that the measurement is less accurate than it actually is. This slows down the convergence of the filter allowing time to recover from the approximation. There is some theoretical basis for pseudonoise [10], but a combination of theory and educated trial and error is usually used. Both are used for our EKF.

The first component of our pseudonoise is determined by inspecting the second derivatives of the measurement linearization.

```
In[18]:= Series[f[mup, sp], {mup, mup0, 2}, {sp, sp0, 2}]
```

$$\begin{aligned}\text{Out[18]=}\ & \left(f[\text{mup0}, \text{sp0}] + f^{(0,1)}[\text{mup0}, \text{sp0}]\ (\text{sp} - \text{sp0}) + \right. \\ & \left. \frac{1}{2} f^{(0,2)}[\text{mup0}, \text{sp0}]\ (\text{sp} - \text{sp0})^2 + O[\text{sp} - \text{sp0}]^3\right) + \\ & \left(f^{(1,0)}[\text{mup0}, \text{sp0}] + f^{(1,1)}[\text{mup0}, \text{sp0}]\ (\text{sp} - \text{sp0}) + \right.\end{aligned}$$

$$\frac{1}{2} f^{(1,2)}[\text{mup0}, \text{sp0}]\,(\text{sp} - \text{sp0})^2 + O[\text{sp} - \text{sp0}]^3\Big)\,(\text{mup} - \text{mup0}) +$$
$$\Big(\frac{1}{2} f^{(2,0)}[\text{mup0}, \text{sp0}] + \frac{1}{2} f^{(2,1)}[\text{mup0}, \text{sp0}]\,(\text{sp} - \text{sp0}) + \frac{1}{4} f^{(2,2)}[\text{mup0}, \text{sp0}]$$
$$(\text{sp} - \text{sp0})^2 + O[\text{sp} - \text{sp0}]^3\Big)\,(\text{mup} - \text{mup0})^2 + O[\text{mup} - \text{mup0}]^3$$

While we do not know the $(x - x_0)^2$-like terms, mathematically they look like measurement noise:

$$z = \text{zero order terms} + \text{first order terms} + \text{second order terms} + \text{noise}.$$

The second order terms are combined with the measurement noise giving:

$$z = \text{zero order terms} + \text{first order terms} + \text{pseudonoise}.$$

We can approximate the second order terms statistically with the appropriate terms from the covariance matrix. We form the pseudonoise variance by summing the square of the individual terms. The cross-covariances are not used directly. Since our intent is to increase the measurement noise, a negative correlation coefficient could hinder that goal. Instead the cross-covariance is computed from diagonal terms assuming the correlation coefficient is 1.

```
In[19]:= pseudonoise[s_, mup_, sp_, p11_, p22_] =
          Simplify[(1 / 2 ∂{mup, 2} mu[s, mup, sp] p11)^2 +
            (∂mup,sp mu[s, mup, sp] Sqrt[p11 p22])^2 +
            (1 / 2 ∂{sp, 2} mu[s, mup, sp] p22)^2]
```

$$\text{Out[19]=}\quad \frac{4\,\text{p22}\,\text{s}^2\left(\text{mup}^2\,\text{p22}\,\text{sp}^2\,(-3\,\text{s}^2 + \text{sp}^2)^2 + \text{p11}\,(\text{s}^4 - \text{sp}^4)^2\right)}{(\text{s}^2 + \text{sp}^2)^6}$$

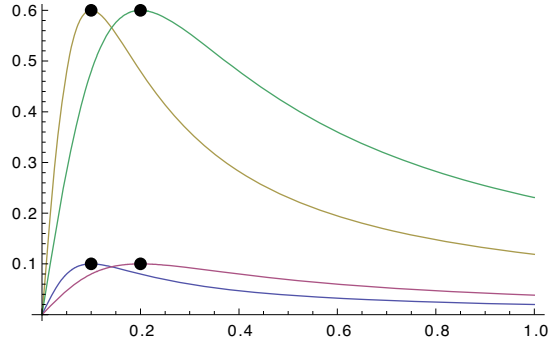
The second element of our pseudonoise is a constant increase in our measurement standard deviation. A value of three times the true value was determined by trial and error across the anticipated range of parameters. During the animation of our results in the next section, we would like the true values within the one sigma contours of the covariance function.

```
In[20]:= sigmaekf := 3 sigmatrue;
```

Experience with this estimator shows that the best overall performance is obtained when the filter is initialized to the high end of its range. Here is a plot of the range of mu slip curves we expect to encounter in an aircraft application.

```
In[21]:= Plot[{mu[s, 0.1, 0.1], mu[s, 0.1, 0.2],
           mu[s, 0.6, 0.1], mu[s, 0.6, 0.2]}, {s, 0, 1},
          Epilog → {PointSize[0.025], Point[{0.1, 0.1}],
            Point[{0.1, 0.6}], Point[{0.2, 0.1}], Point[{0.2, 0.6}]}]
```

Out[21]=



We will therefore initialize the state estimates at the high end of the range and the covariance to cover the range.

In[22]:=
```
x0 := {{0.6}, {0.2}};
p0 := {{0.6^2, 0}, {0, 0.2^2}};
```

Here is the extended Kalman algorithm for the mu slip curve estimator.

In[24]:=
```
muekf[z_, x0_, p0_, r_] := Module[
  {zslip, zmu, xmup, xsp, h, er, error, p11, p22}, x[0] = x0;
  p[0] = p0; Do[{xmup = x[i - 1][[1, 1]], xsp = x[i - 1][[2, 1]],
    zslip = z[[i, 1]], zmu = z[[i, 2]], h = eh[zslip, xmup, xsp],
    error = muerror[zmu, zslip, xmup, xsp],
    p11 = p[i - 1][[1, 1]], p22 = p[i - 1][[2, 2]],
    er = r + pseudonoise[zslip, xmup, xsp, p11, p22],
    k = kgain[p[i - 1], h, {{er}}], p[i] = updatep[p[i - 1], h, k],
    x[i] = updatex[x[i - 1], {{error}}, k]},
   {i, 1, Length[z]}]; {x, p}]
```

The following line executes the Kalman filter and assigns the state vector and covariance matrix to the variables, x and p, respectively.

In[25]:=
```
{x, p} = muekf[z, x0, p0, sigmaekf^2];
```

■ The Results

We present the results of our Kalman filtering example as an animation. The animation has the true and noise-corrupted mu slip curves in black and green, respectively. The estimated mu slip curve is shown in red after each measurement is incorporated. Similarly, we show the one sigma covariance contour on a frame-by-frame basis. We also place points at the peak locations and the current measurement for emphasis.

We defined the measured data plot previously. Here are plot definitions for the true and estimated curves.

In[26]:=
```
ptrue[muptrue_, sptrue_] :=
  muslipplot[muptrue, sptrue, 0.00375, GrayLevel[0]]
```

```
In[27]:= xplot[i_] := muslipplot[x[i][[1, 1]], x[i][[2, 1]], 0.00125, Hue[1]]
```

The one sigma contour is defined with the aid of the bivariate normal distribution function,

$$f[dx, p] = \frac{1}{2\pi \left| p^{-1} \right|} e^{\left[-\frac{1}{2}\left(dx^T p^{-1} dx\right)\right]},$$

where dx is the variation from the mean and p is the covariance matrix.

We create the one sigma contour of the bivariate normal distribution by setting the argument of the exponential function to 0.5 in the `ContourPlot` function. Here is the definition of the argument and the one sigma contour plot.

```
In[28]:= bivgdfexp[dx_, p_] := 1/2 Transpose[dx].Inverse[p].dx
```

```
In[29]:= pplot[i_] :=
          ContourPlot[bivgdfexp[{{mu}, {s}} - x[i], p[i]][[1, 1]],
           {s, x[i][[2, 1]] - Sqrt[p[i][[2, 2]]], x[i][[2, 1]] + Sqrt[p[i][[2, 2]]]},
           {mu, x[i][[1, 1]] - Sqrt[p[i][[1, 1]]], x[i][[1, 1]] + Sqrt[p[i][[1, 1]]]},
           ContourShading → False, Contours → {0.5},
           ContourStyle → Hue[0.65], PlotPoints → {32, 32}]
```

A smooth contour is obtained by scaling the plot range based upon the state and covariance data.

The function `points` plots one frame of color-coded points data.

```
In[30]:= points[i_] := Graphics[
          {PointSize[0.025`], Point[{z[[i, 1]], mu[z[[i, 1]], muptrue, sptrue]}],
           Point[{sptrue, muptrue}], Hue[1], Point[{x[i][[2, 1]], x[i][[1, 1]]}],
           ColorData["Legacy", "ForestGreen"],
           PointSize[0.0125`], Point[z[[i]]]}]
```

One frame of results is assembled in the function `showresult`.

```
In[31]:= showresult[i_] :=
          Show[{xplot[i], ptrue[muptrue, sptrue], measplot[i], xplot[i],
            pplot[i], points[i]}, PlotRange → {{0, 1}, {0, 0.65}},
           Frame → True, FrameLabel → {"Slip Ratio", "Friction"},
           Background → GrayLevel[1],
           Epilog → {Text[
              Column[
               {Style["Measured", ColorData["Legacy", "ForestGreen"]],
                Style["True", GrayLevel[0]], Style["Estimated", Hue[1]],
                Style["One Sigma", Hue[0.65]]}], {0.6, 0.2}, {-1, -1}]}]
```

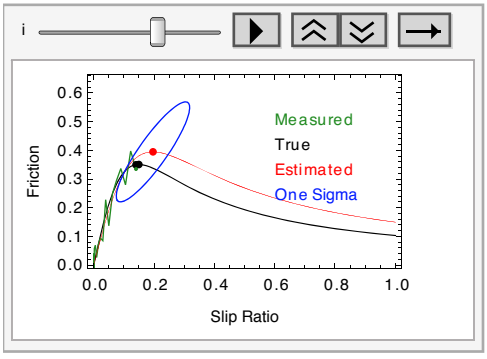
`showresults` generates the animation using every di^{th} frame.

```
In[32]:= showresults[di_] := Animate[showresult[i],
           {i, 1, Length[z], di}, SaveDefinitions → True]
```

Here is the animation.

```
In[33]:= animateekf = showresults[1]
```

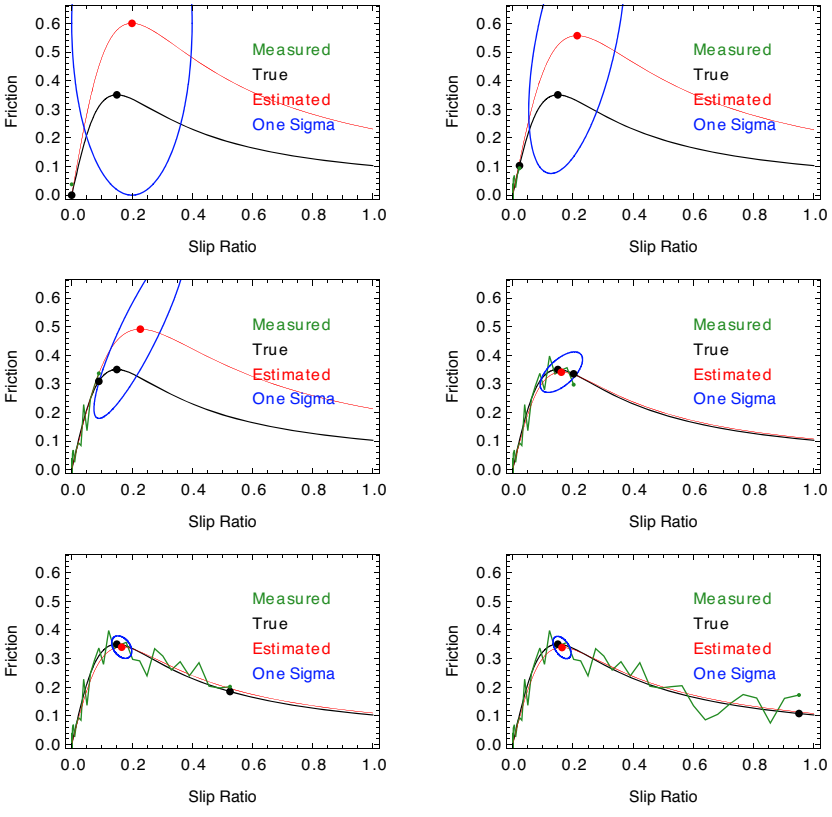
Out[33]=



Here is an array of selected frames.

```
In[34]:= GraphicsArray[
           Map[showresult[##] &, {{1, 7}, {13, 19}, {30, 40}}, {2}],
           ImageSize → {400, Automatic}]
```

Out[34]=



■ Discussion

Inspection of the animation shows that estimated state variables converge to the true value within the one sigma accuracy. It may be helpful to think of the one sigma contour as the area where the filter is looking or searching for the most likely solution. As we incorporate more and more measurements, our model is refined and the search area is reduced.

As we move up the mu slip curve, the estimation process proceeds and the unknown parameters become "correlated," indicated by the rotation of the one sigma contour. Correlation perhaps can be best thought of by considering the statistical correlation of two sinusoids. If they are in phase, the correlation coefficient is 1, and when plotted versus each other they produce an ellipse (a Lissajous figure) with a slope of 1 with 0 thickness in one dimension. This is displayed in red with a slight phase shift to show that it is an ellipse. If they are out of phase (blue), the slope and correlation coefficient are −1. A sine and cosine would have a correlation coefficient of 0 and produce an equal dimension ellipse with 0 slope (black). If the correlation coefficient of the two sinusoids is known to be 1 and we measure and correct one, then we know we can correct the other the same amount. In the animation, we see a positive correlation is followed by a negative correlation relating to the slope of the mu slip curve.

```
In[35]:= ParametricPlot[
          {{Sin[t], Sin[t + 0.007 Pi]}, {Sin[t], Sin[t + Pi / 2]},
           {Sin[t], Sin[t + 0.993 Pi]}}, {t, 0, 2 Pi},
          AspectRatio → 1, PlotStyle → {Hue[1], GrayLevel[0], Hue[.65]}]
```

Out[35]=

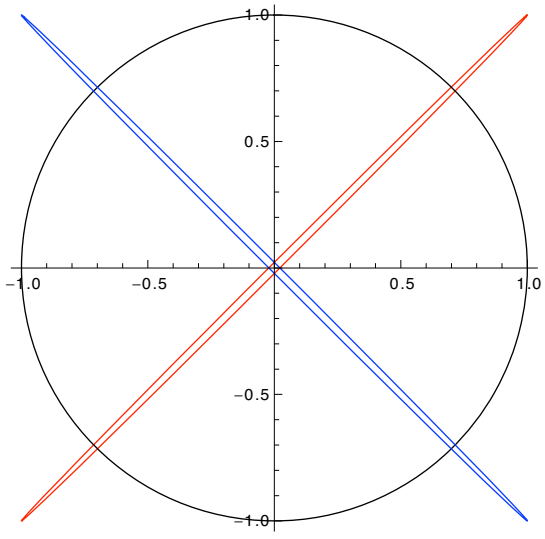


Figure 1 shows a more conventional way of viewing the covariance information. The diagonal elements are normalized by their initial values and the off-diagonal term is viewed as the correlation coefficient.

```
In[36]:= figure1
```

Out[36]=

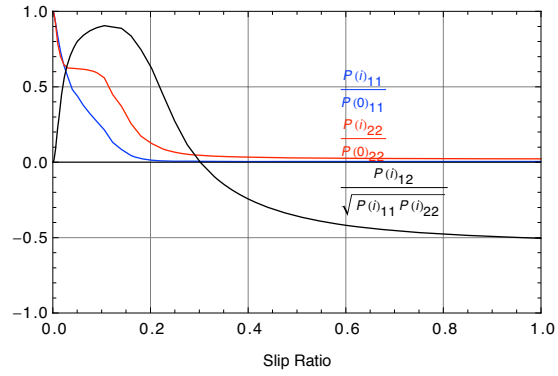


Figure 1. Covariance convergence.

The curve representing amplitude steadily reduces while the curve representing the location of the peak becomes constant for slip ratios of 0.05 to 0.15. The correlation coefficient changes sign at about 0.25. We will investigate this further.

In general, a Kalman filter begins by assigning a portion of the error to the measuring device. A simple way of getting a feel for this is to look at the Kalman gain for a single state when that state is being directly measured (the observation matrix is 1). Since we are trying to observe the state, its covariance matrix can be considered the signal and the measurement contains the noise. Assuming the state covariance is a multiple of the observation noise, we can calculate the Kalman gain as a function of the signal-to-noise ratio.

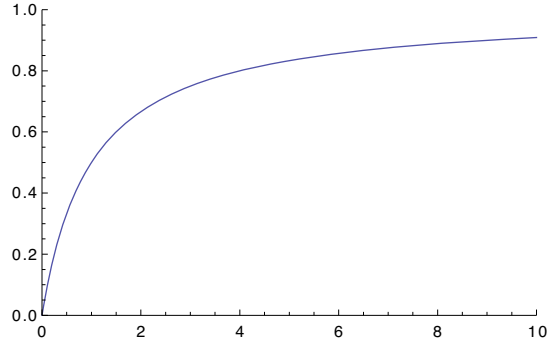
```
In[37]:= kex[snr_] = Simplify[kgain[{{snr r}}, {{1}}, {{r}}][[1, 1]]]
```

$$\text{Out[37]=}\quad \frac{\text{snr}}{1+\text{snr}}$$

Here is a plot of the Kalman gain for this simple case.

```
In[38]:= Plot[kex[snr], {snr, 0, 10}, PlotRange → {{0, 10}, {0, 1}}]
```

Out[38]=



If the measurement noise is much larger than the covariance (a low signal-to-noise ratio), the Kalman gain is close to 0 and the state receives little correction. It is sometimes said that the filter is rejecting the measurement, but it is merely recognizing that its internal model is more accurate than the measurement and relying on the model instead. On the other hand, if the measurement is more accurate than the covariance (a high signal-to-noise ratio), then the Kalman gain is near 1 and the model receives a strong correction.

The incorporation of external data into the model is also recognized in the covariance matrix. For a single state estimator, the covariance is reduced by

$$\frac{p_i}{p_{i-1}} = 1 - k.$$

A large Kalman gain, due to an accurate measurement, results in a large reduction in the estimate's uncertainty. As the estimation progresses, the reduced uncertainty in the model will result in a smaller gain in the next Kalman cycle (assuming the same measurement noise).

For a multistate filter, the signal is given by projecting the state covariance to measurement space. Review of the denominator of the Kalman gain equation gives this term as

```
In[39]:= pz[p_, h_] := (h.p.Transpose[h])[[1, 1]]
```

Figure 2 shows a plot of the signal and noise for the mu slip curve. The pre-update signal is blue and the post-update signal is red. The observation noise, which includes the pseudonoise, is black.

```
In[40]:= figure2
```

Out[40]=

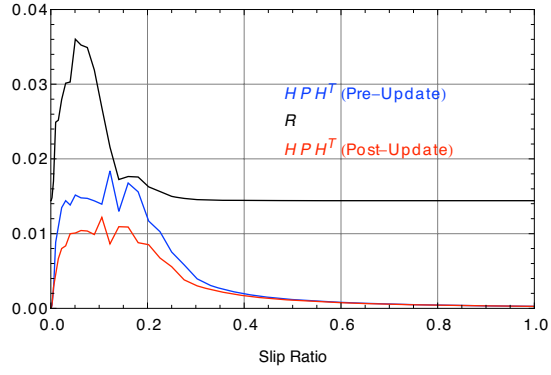


Figure 2. Signal and noise variance.

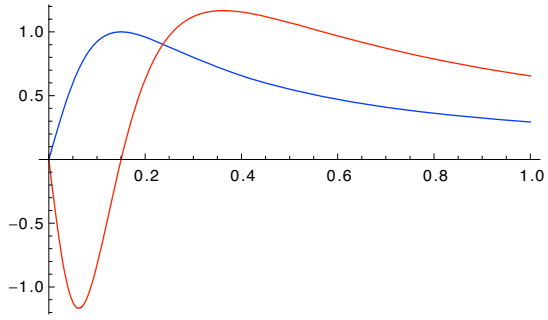
In comparison to our true measurement noise of 0.04^2, the pseudonoise is large at nearly 0.19^2. This indicates the second order terms are quite strong for the size of the covariance at the time. The net effect is to make the Kalman filter think it has a much less accurate measurement and slow down convergence.

Early on, we see that our pseudonoise term is larger than our signal term, but not so much that a significant improvement in model accuracy does not occur. At around a slip ratio of 7.5 percent, the pseudonoise term drops quickly. The filter thinks it has a more accurate measurement and applies a strong correction to the model. Once the peak is passed, the filter thinks the model is more accurate than the measurements and only minor corrections are made.

The behavior of the filter at low slip ratios can be understood by plotting the observation matrix versus slip ratio.

```
In[41]:= Plot[Evaluate[eh[s, muptrue, sptrue][[1]]],
          {s, 0, 1}, PlotStyle → {Hue[0.65], Hue[1]}]
```

Out[41]=



The observation matrix sets the relative weight that each state contributes to the measurement. For example, at the peak of the curve, the observation matrix element for the amplitude of the curve is 1 and it is 0 for the location of the peak. This is telling the filter to attribute all of the modeling error to the amplitude state. Conversely, there are points on either side of the peak where the observation matrix elements are equal in magnitude. At these points, the modeling error is split equally between the states.

The two state variables correlate initially, which is another way of saying that we do not have enough information to tell one from the other. As we reach slip ratios between 5 and 10 percent, the derivative of the mu slip curve with respect to the peak location (red) has an extrema or a point where it has its maximum effect on the measurement. It has its minimum effect on the measurement as the peak is crossed and the derivative of the mu slip curve with respect to the amplitude (blue) is at a maximum. This is pleasing heuristically. We may know we are near the top of a mountain, but if it is relatively flat, we may not be able to tell exactly where the top is.

As we cross the peak, the derivative of the mu slip curve with respect to the peak location changes sign. The variables decorrelate, which is another way of saying we have enough information to tell the variables apart. Rapid convergence occurs when this happens.

This note is not meant to imply that a Kalman filter is the only viable method of fitting the mu slip curve. The built-in function, `NonLinearRegress`, also does quite well at fitting the data.

```
In[42]:= Needs["Statistics`NonlinearFit`"] // Quiet

In[43]:= bfp[maxiter_] :=
          BestFitParameters /. NonlinearRegress[z, mu[s, mup, sp],
            {s}, {{mup, 0.6, 0, 0.6}, {sp, 0.2, 0.1, 0.2}},
            RegressionReport → BestFitParameters, MaxIterations → maxiter]
```

The option `maxiter` is the maximum number of iterations. After five passes through the dataset, the built-in routine produces the following estimates.

```
In[44]:= bfp[5]

During evaluation of In[44]:=
```

> FindFit::cvmit :
> Failed to converge to the requested accuracy or precision within 5 iterations. >>

```
Out[44]= {mup → 0.34876, sp → 0.156885}
```

The results are slightly better than the Kalman filter's estimates.

```
In[45]:= x[Length[z]]

Out[45]= {{0.338894}, {0.164908}}
```

This is not really an apples-to-apples comparison though, as the built-in routine regresses on the entire dataset up to `maxiter` times, while the Kalman filter performs the work in one pass. For a single pass, the built-in routine returns the following.

```
In[46]:= bfp[1]

During evaluation of In[46]:=
```

> FindFit::cvmit :
> Failed to converge to the requested accuracy or precision within 1 iterations. >>

```
Out[46]= {mup → 0.467086, sp → 0.1}
```

If we up the number of iterations to two, we get a reasonable result.

```
In[47]:= bfp[2]

During evaluation of In[47]:=
```

> FindFit::cvmit :
> Failed to converge to the requested accuracy or precision within 2 iterations. >>

```
Out[47]= {mup → 0.414491, sp → 0.135667}
```

Similarly, we could take the final Kalman filter estimates from the first pass and use them for initialization on a second pass to refine our estimate.

References

[1] R. A. Ibrahim and E. Rivin, eds., "Friction Induced Vibration," *Transactions of the American Society of Mechanical Engineers*, *Applied Mechanics Reviews*, **47**(7), July, 1994.

[2] E. Bakker, L. Nyborg, and H. Pacejka, "Tyre Modelling for Use in Vehicle Dynamics Studies," Warrendale, PA: *Society of Automotive Engineers*, *Technical Paper No.* 870421, 1987 pp. 190–204.

[3] E. Bakker, H. Pacejka, and L. Lindner, "A New Tire Model with an Application in Vehicle Dynamics Studies," Warrendale, PA: *Society of Automotive Engineers*, *Technical Paper No.* 890087, 1989 pp. 101–113.

[4] R. Rudd, *Antiskid Brake Control System Using Kalman Filtering*, US Patent 5,918,951, filed May 9, 1997, and issued July 6, 1999 www.google.com/patents?id=3aMYAAAAEBAJ&dq=5918951.

[5] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Transactions of the American Society of Mechanical Engineers, Journal of Basic Engineering*, **82**(1)B, March, 1960 pp. 35–45 www.cs.unc.edu/~welch/kalman/media/pdf/Kalman1960.pdf.

[6] R. M. du Plessis, "Poor Man's Explanation of Kalman Filtering or How I Stopped Worrying and Learned to Love Matrix Inversion," *Rockwell International Technical Note*, Anaheim, CA: Rockwell International, Autonetics Division, 1967 www.taygeta.com/kalman.html.

[7] A. Gelb, ed., *Applied Optimal Estimation*, Cambridge, MA: The MIT Press, 1974.

[8] P. S. Maybeck, *Stochastic Models, Estimation, and Control*, Vol. 1, New York: Academic Press, 1979 www.cs.unc.edu/~welch/kalman/maybeck.html.

[9] H. W. Sorenson, ed., *Kalman Filtering: Theory and Application*, Los Alamitos, CA: IEEE Press, 1985.

[10] C. Hutchinson, J. A. D'Appolito, and K. J. Roy, "Applications of Minimum Variance Reduced-State Estimators," *IEEE Transactions on Aerospace and Electronic Systems,* September 1975, pp. 785–794.

R. Rudd, "Estimating the Mu Slip Curve via Extended Kalman Filtering," *The Mathematica Journal*, 2011. dx.doi.org/doi:10.3888/tmj.11.1–5.

About the Author

Robert Rudd works for the Goodrich Corporation, Fuel and Utility Systems, in Vergennes, Vermont. Rudd received bachelor's and master's degrees in mechanical engineering from Rensselaer Polytechnic Institute in Troy, New York, and has worked at the Unisys and Singer corporations. He has designed and tested antiskid control algorithms for the Lockheed Martin F-16, the Airbus A321, the Boeing 777, the Northrop Grumman RQ-4B Global Hawk, and regional business jets. He has also designed Kalman filters and smoothers for use with the Boeing B1-B.

Robert Rudd
Goodrich Fuel and Utility Systems
100 Panton Road
Vergennes, VT 05491
Rob.Rudd@goodrich.com

■ Implementation

```
Needs["ErrorBarPlots`"]

figure1 :=
 ListLinePlot[{Table[{z[[i, 1]], p[i][[1, 1]]/p[0][[1, 1]]}, {i, 1, Length[z]}],
   Table[{z[[i, 1]], p[i][[2, 2]]/p[0][[2, 2]]}, {i, 1, Length[z]}], Table[
    {z[[i, 1]], p[i][[1, 2]]/Sqrt[p[i][[1, 1]] p[i][[2, 2]]]}, {i, 1, Length[z]}]},
  Frame → True, FrameLabel → {"Slip Ratio", None},
  GridLines → Automatic,
  PlotStyle → {Hue[0.65], Hue[1], GrayLevel[0]},
  PlotRange → {{0, 1}, {-1, 1}},
  Epilog → {
    Text[
     Column[{Style["P (i)11/P (0)11", Hue[0.65]],
       Style["P (i)22/P (0)22", Hue[1]],
       Style["P (i)12/Sqrt[P (i)11 P (i)22]", GrayLevel[0]]
      }], {0.7, .12}]}]
```

```
figure2 := ListLinePlot[
  {Table[{z[[i, 1]], pz[p[i - 1], eh[z[[i, 1]], x[i - 1][[1, 1]],
       x[i - 1][[2, 1]]]}, {i, 1, Length[z]}],
   Table[{z[[i, 1]], sigmaekf^2 + pseudonoise[z[[i, 1]], x[i - 1][[1, 1]],
       x[i - 1][[2, 1]], p[i - 1][[1, 1]], p[i - 1][[2, 2]]]},
    {i, 1, Length[z]}], Table[{z[[i, 1]], pz[p[i],
      eh[z[[i, 1]], x[i][[1, 1]], x[i][[2, 1]]]]}, {i, 1, Length[z]}]},
  Frame → True, FrameLabel → {"Slip Ratio", None},
  GridLines → Automatic,
  PlotStyle → {Hue[0.65`], GrayLevel[0], Hue[1]},
  PlotRange → {{0, 1}, {0, 0.04`}}, Epilog → {
    Text[Column[{
      Style[HoldForm[H P H^T "(Pre-Update)"], Hue[0.65`]],
      Style[R, GrayLevel[0]], Style[HoldForm[
        H P H^T "(Post-Update)"], Hue[1]]}], {.65, .025}]}]
$Line = 0;
```