Kalman Folding 2: Tracking (WORKING DRAFT)

Extracting Models from Data, One Observation at a Time

Brian Beckman

<2016-05-03 Tue>

Contents

1 Abstract

In *Kalman Folding, Part 1*,[1] we present basic, static Kalman filtering as a functional fold, highlighting the unique advantages of this form for deploying test-hardened code verbatim in harsh, mission-critical environments. The examples in that paper are all static, meaning that the states of the model do not depend on the independent variable, often physical time.

Here, we present a time-dependent Kalman filter in the same, functional form. This filter can handle many time-dependent applications including some tracking and navigation, and is easilly extended to nonlinear and non-Gaussian forms, the Extended Kalman Filter (EKF) and Unscented Kalman Filter (UKF) respectively. Those are subjects of other papers in this Kalman-folding series. Here, we reproduce a tracking example from a well known reference, but in functional form, highlighting the advantages of that form.

[1] To appear.

2 Kalman Folding in the Wolfram Language

In this series of papers, we use the Wolfram language[2] because it excels at concise expression of mathematical code. All examples in these papers can be written in any modern mainstream language that supports higher-order functions or function pointers. For example, it is easy to write them in C, C++, Python, any Lisp, not to mention Haskell, Scala, Erlang, and OCaml.

In *Kalman Folding*,[1] we found the following elegant formulation for the accumulator function of a fold that implements the static Kalman filter:

$$\text{kalmanStatic}\left(\mathbf{Z}\right)\left(\{\mathbf{x},\mathbf{P}\},\{\mathbf{A},z\}\right)=\{\mathbf{x}+\mathbf{K}\ \left(z-\mathbf{A}\,\mathbf{x}\right),\mathbf{P}-\mathbf{K}\,\mathbf{D}\,\mathbf{K}^{\intercal}\} \tag{1}$$

where

$$\mathbf{K}=\mathbf{P}\,\mathbf{A}^{\intercal}\,\mathbf{D}^{-1} \tag{2}$$

$$\mathbf{D}=\mathbf{Z}+\mathbf{A}\,\mathbf{P}\,\mathbf{A}^{\intercal} \tag{3}$$

and all quantities are matrices:

- z is a $\mathfrak{b}\times 1$ column vector containing one multidimensional observation
- $\mathbf{x}$ is an $\mathfrak{n}\times 1$ column vector of *model states*
- $\mathbf{Z}$ is a $\mathfrak{b}\times\mathfrak{b}$ matrix, the covariance of observation noise
- $\mathbf{P}$ is an $\mathfrak{n}\times\mathfrak{n}$ matrix, the theoretical covariance of $\mathbf{x}$
- $\mathbf{A}$ is a $\mathfrak{b}\times\mathfrak{n}$ matrix, the *observation partials*
- $\mathbf{D}$ is a $\mathfrak{b}\times\mathfrak{b}$ matrix, the Kalman denominator
- $\mathbf{K}$ is an $\mathfrak{n}\times\mathfrak{b}$ matrix, the Kalman gain

In physical or engineering applications, these quantities carry physical dimensions of units of measure in addition to their matrix dimensions as numbers of rows and columns. If the physical and matrix dimensions of $\mathbf{x}$ are $[[\mathbf{x}]]\stackrel{\text{def}}{=}(\mathcal{X},\mathfrak{n}\times 1)$ and of z are $[[z]]\stackrel{\text{def}}{=}(\mathcal{Z},\mathfrak{b}\times 1)$, then

$$\begin{array}{lcccc}
[[\mathbf{Z}]] & = & (& \mathcal{Z}^2 & \mathfrak{b}\times\mathfrak{b} &) \\
[[\mathbf{A}]] & = & (& \mathcal{Z}/\mathcal{X} & \mathfrak{b}\times\mathfrak{n} &) \\
[[\mathbf{P}]] & = & (& \mathcal{X}^2 & \mathfrak{n}\times\mathfrak{n} &) \\
[[\mathbf{A}\,\mathbf{P}\,\mathbf{A}^{\intercal}]] & = & (& \mathcal{Z}^2 & \mathfrak{b}\times\mathfrak{b} &) \\
[[\mathbf{D}]] & = & (& \mathcal{Z}^2 & \mathfrak{b}\times\mathfrak{b} &) \\
[[\mathbf{P}\,\mathbf{A}^{\intercal}]] & = & (& \mathcal{X}\,\mathcal{Z} & \mathfrak{n}\times\mathfrak{b} &) \\
[[\mathbf{K}]] & = & (& \mathcal{X}/\mathcal{Z} & \mathfrak{n}\times\mathfrak{b} &)
\end{array} \tag{4}$$

In all examples in this paper, the observations z are 1×1 matrices, equivalent to scalars, so $\mathfrak{b}=1$, but the theory and code carry over to multi-dimensional vector observations.

[2] `http://reference.wolfram.com/language/`

The function in equation 1 *lambda-lifts*[3] $\mathbf{Z}$, meaning that it is necessary to call *kalmanStatic* with a constant $\mathbf{Z}$ to get the actual accumulator function used in folds. This is desirable when $\mathbf{Z}$ does not depend on the independent variable, time in this paper, to reduce coupling between the accumulator function and its calling environment. It is better to pass in an explicit constant quantity than to implicitly close over[4] ambient constants, and it is good to keep the number of parameters in the observation packet $\{\mathbf{A}, \mathbf{z}\}$ as small as possible. In other applications, $\mathbf{Z}$ can depend on the independent variable, in which case we pass it around in the observation packet along with $\mathbf{A}$ and $\mathbf{z}$.

In Wolfram, this function is

```
kalman[Zeta_][{x_, P_}, {A_, z_}] :=
 Module[{D, K},
  D = Zeta + A.P.Transpose[A];
  K = P.Transpose[A].Inverse[D];
  {x2 + K.(z - A.x), P - K.D.Transpose[K]}]
```

We can test it on a small case

```
Fold[kalman[IdentityMatrix[1]],
  {ColumnVector[{0, 0, 0, 0}], IdentityMatrix[4]*1000.0},
  {{{{1,  0., 0.,  0.}}, { -2.28442}},
   {{{1,  1., 1.,  1.}}, { -4.83168}},
   {{{1, -1., 1., -1.}}, {-10.46010}},
   {{{1, -2., 4., -8.}}, {  1.40488}},
   {{{1,  2., 4.,  8.}}, {-40.8079}}}
  ] // Chop
~~>
```

$$\mathbf{x} = \begin{bmatrix} -2.97423 \\ 7.2624 \\ -4.21051 \\ -4.45378 \end{bmatrix} \tag{5}$$

$$\mathbf{P} = \begin{bmatrix} 0.485458 & 0 & -0.142778 & 0 \\ 0 & 0.901908 & 0 & -0.235882 \\ -0.142778 & 0 & 0.0714031 & 0 \\ 0 & -0.235882 & 0 & 0.0693839 \end{bmatrix}$$

expecting results within one or two standard deviations of the ground truth $\aleph = \begin{bmatrix} -3 & 9 & -4 & -5 \end{bmatrix}^\intercal$, where the standard deviations can be found by taking the square roots of the diagonal elements of $\mathbf{P}$. For details about this test case, see the first paper in the series, *Kalman Folding, Part 1*.[1]

In another paper in this series, *Kalman Folding 3: Derivations*,[5] we present a full derivation of this static accumulator function.

[3] https://en.wikipedia.org/wiki/Lambda_lifting

[4] https://en.wikipedia.org/wiki/Closure_(computer_programming)

[5] To appear.

3 A Tracking Example

Let us reproduce an example from Zarchan and Musoff,[6] to track a falling object in one position dimension, with no aerodynamic drag. Handling drag requires an extended Kalman filter (EKF), subject of a separate paper, because a model with drag is nonlinear.

We will need a time-dependent Kalman filter, which applies an additional, linear dynamic model to the states.

3.1 Time-Dependent States

Suppose the states $\mathbf{x}$ suffer time evolution by a linear transformation $\mathbf{F}$ and an additional *disturbance* or *control* input $\mathbf{u}$, linearly transformed by $\mathbf{G}$. These new quantities may be functions of time, but not of $\mathbf{x}$ lest the equations be non-linear. Write the time derivative of $\mathbf{x}$ as

$$\dot{\mathbf{x}}(t) = \mathbf{F}\,\mathbf{x}(t) + \mathbf{G}\,\mathbf{u}(t)$$

We often leave off the explicit denotation of time-dependence for improved readability:

$$\dot{\mathbf{x}} = \mathbf{F}\,\mathbf{x} + \mathbf{G}\,\mathbf{u}$$

Generalize by adding *random process* noise ξ to the state derivative:

$$\dot{\mathbf{x}} = \mathbf{F}\,\mathbf{x} + \mathbf{G}\,\mathbf{u} + \xi \tag{6}$$

This is standard *state-space form*[7] for differential equations. Solving these equations is beyond the scope of this paper, but suffice it to say that we need certain time integrals of $\mathbf{F}$, $\mathbf{G}$, and ξ as inputs to the filter. These are

$$\mathbf{\Phi}(\delta t) \stackrel{\text{def}}{=} e^{\mathbf{F}\,\delta t} = \mathbf{1} + \frac{\mathbf{F}^2\delta t^2}{2!} + \frac{\mathbf{F}^3\delta t^3}{3!} + \cdots \tag{7}$$

$$\mathbf{\Gamma}(\delta t) \stackrel{\text{def}}{=} \int_0^{\delta t} \mathbf{\Phi}(\tau)\cdot\mathbf{G}\,d\tau \tag{8}$$

$$\mathbf{\Xi}(\delta t) \stackrel{\text{def}}{=} \int_0^{\delta t} \mathbf{\Phi}(\tau)\cdot\begin{pmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \mathrm{E}\left[\xi\xi^{\mathsf{T}}\right] \end{pmatrix}\cdot\mathbf{\Phi}(\tau)^{\mathsf{T}}\,d\tau \tag{9}$$

where δt is an increment of time used to advance the filter discretely. Again, we frequently omit denoting of explicit dependence on δt for improved readability.

[6] Zarchan and Musoff, *Fundamentals of Kalman Filtering, A Practical Approach, Fourth Edition*, Ch. 4

[7] `https://en.wikipedia.org/wiki/State-space_representation`

3.2 Recurrences for Dynamics

The transitions of a state (and its covariance) from time t to the next state (and covariance) at time $t + \delta t$ follow these recurrences:

$$\mathbf{x} \leftarrow \mathbf{\Phi}\,\mathbf{x} + \mathbf{\Gamma}\,\mathbf{u} \tag{10}$$

$$\mathbf{P} \leftarrow \mathbf{\Xi} + \mathbf{\Phi}\,\mathbf{P}\,\mathbf{\Phi}^{\mathsf{T}} \tag{11}$$

These equations appear plausible on inspection. We encourage you to verify that they satisfy a discretization of equation 6.

3.3 The Foldable Filter

These tiny changes are all that is needed to add state evolution to the Kalman filter:

```
kalman[Zeta_][{x_, P_}, {Xi_, Phi_, Gamma_, u_, A_, z_}] :=
 Module[{x2, P2, D, K},
  x2 = Phi.x + Gamma.u;
  P2 = Xi + Phi.P.Transpose[Phi];
  (* after this, it's identical to the static filter *)
  D = Zeta + A.P2.Transpose[A];
  K = P2.Transpose[A].inv[D];
  {x2 + K.(z - A.x2), P2 - K.D.Transpose[K]}]
```

3.3.1 Test

Check that it reproduces the test case above for the static filter:

```
With[{ (* make some constant matrices *)
   Xi = zero[4], Zeta = id[1],
   Phi = id[4], Gamma = zero[4, 1], u = zero[1]},
  Fold[
   kalman[Zeta],
   {col[{0, 0, 0, 0}], id[4]*1000.0},
   Map[ Join[{Xi, Phi, Gamma, u}, #]&,
    {{{{1,  0., 0.,  0.}}, { -2.28442}},
     {{{1,  1., 1.,  1.}}, { -4.83168}},
     {{{1, -1., 1., -1.}}, {-10.46010}},
     {{{1, -2., 4., -8.}}, {  1.40488}},
     {{{1,  2., 4.,  8.}}, {-40.8079}}}]]]
```

3.4 Dynamics of a Falling Object

Let $h(t)$ be the height of the falling object, and let the state vector $\mathbf{x}(t)$ contain $h(t)$ and its first derivative, $\dot{h}(t)$, the speed of descent.[8]

[8] A state-space form containing a position and derivative is commonplace in second-order dynamics like Newton's Second Law. We usually employ state-space form to reduce n-th-order differential equations to first-order differential equations by stacking the dependent variable on $n - 1$ of its derivatives in the state vector.

$$\mathbf{x} = \begin{bmatrix} h(t) \\ \dot{h}(t) \end{bmatrix}$$

The system dynamics are elementary:

$$\begin{bmatrix} \dot{h}(t) \\ \ddot{h}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} h(t) \\ \dot{h}(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} g \end{bmatrix}$$

where g is the acceleration of Earth's gravitation, about -32.2ft/s^2 (note the minus sign). We read out the dynamics matrices:

$$\mathbf{F} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad \mathbf{G} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} g \end{bmatrix}$$

and their integrals from equations 7, 8, and 9

$$\mathbf{\Phi} = \begin{bmatrix} 1 & \delta t \\ 0 & 1 \end{bmatrix}, \quad \mathbf{\Gamma} = \begin{bmatrix} \delta t^2/2 \\ \delta t \end{bmatrix}, \quad \mathbf{\Xi} = E\left[\xi\xi^T\right] \begin{bmatrix} \delta t^3/3 & \delta t^2/2 \\ \delta t^2/2 & \delta t \end{bmatrix}$$

We test this filter over a sequence of fake observations tracking an object from an initial height of $400,000$ ft and initial speed of $-6,000\,\text{ft/s}$ and from time $t = 0\text{s}$ to $t = 57\,5\text{s}$, just before impact at $h = 0\text{ft}$. We take one observation every tenth of a second, so $\delta t = 0.10\,\text{s}$. We compare the two states $h(t)$ and $\dot{h}(t)$ with ground truth and their residuals with the theoretical sum of squared residuals in the matrix $\mathbf{P}$. The results are shown in figure 1, showing good statistics over five consecutive runs and qualitatively matching the results in the reference.

The ground truth is

$$h(t) = h_0 + \dot{h}_0\, t + g\, t^2/2$$

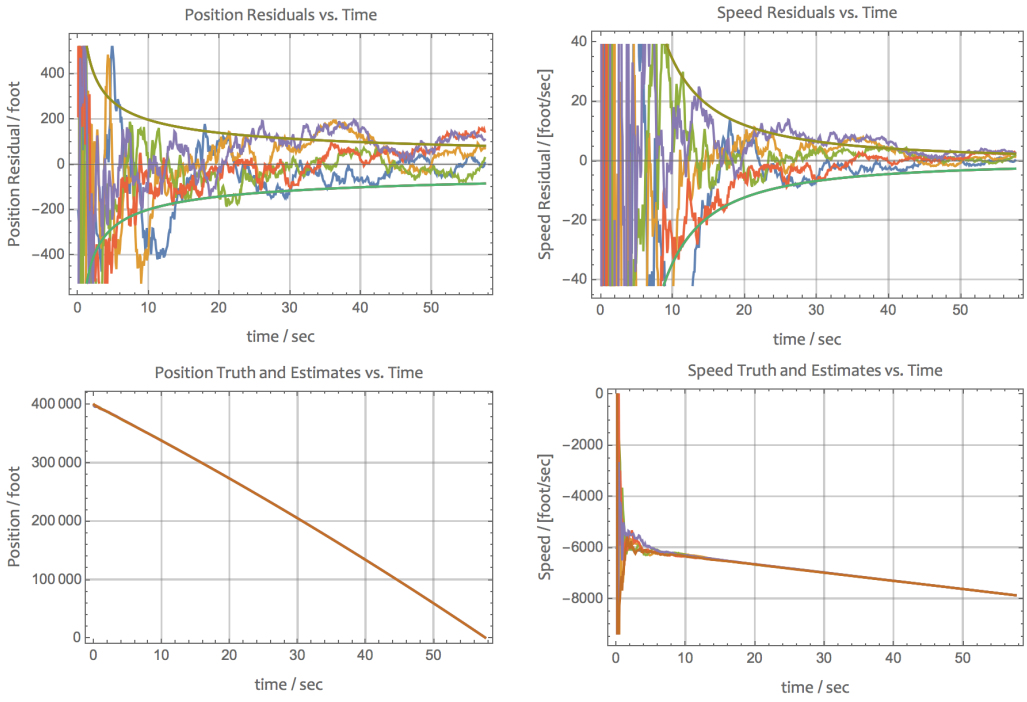
where

$$h_0 = 400,000\,\text{ft}, \quad \dot{h}_0 = -6,000\,\text{ft/sec}$$

and we generate fake noisy observations by sampling a Gaussian distribution of zero mean and standard deviation $1,000$ ft. We do not need process noise for this example. It's often added during debugging and of a Kalman filter to compensate for underfitting or overfitting an inappropriate model. It's also appropriate when we know that the process is stochastic or noisy and have an estimate of its covariance.

4 Concluding Remarks

It's easy to add system dynamics to a static Kalman filter. Expressed as the accumulator function for a fold, the filter is decoupled from the environment in which it runs. We can run exactly the same code, even and especially the same binary, over arrays in memory, lazy streams, asynchronous observables, any data source that can support a *fold* operator. Such flexibility of deployment allows us to address the difficult issues of modeling, statistics, and numerics in friendly environments where we have large memories and powerful debugging tools, then to deploy with confidence in unfriendly, real-world environments where we have small memories, asynchronous, real-time data delivery, and seldom more than logging for forensics. Emacs 24.5.1 (Org mode 8.3.4)



% position resids < σ (E >= 68)	79.3056
position resids >= σ	596.
position resids < σ	2284.
average position resid	-5.3446
theoretical std dev	83.2249
std dev of position resids	174.874

% speed resids < σ (E >= 68)	79.3056
speed resids >= σ	596.
speed resids < σ	2284.
average speed resid	-90.6453
theoretical std dev	2.50586
std dev of speed resids	1993.8

Figure 1: Simulated tracking of a falling object