Kalman Folding 5: Non-Linear Models and the EKF (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. In *Kalman Folding 2: Tracking*,[2] we reproduce a tracking example from the literature, showing that these advantages extend to time-dependent, linear models. Here, we extend that example to include aerodynamic drag, making the model nonlinear. We must extend the Kalman filter itself to handle such problems. The resulting class of filters are called Extended Kalman Filters or EKFs. The particular EKF designed here includes integration of non-linear equations of motion. We accomplish that integration using an internal fold whose accumulator function is a generic integrator. This design allows us to tune the integrator independently of the other parts of the filter, easing the trade-offs between computational speed and fidelity typical of numerical integration.

[1]B. Beckman, *Kalman Folding, Part 1*, to appear.
[2]B. Beckman, *Kalman Folding 2: Tracking*, to appear.

Other papers in this series feature applications of EKFs to a variety of problems including satllite navigation and pursuing a moving target.

2 Kalman Folding in the Wolfram Language

In this series of papers, we use the Wolfram language[3] 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 2: Tracking*,[2] we found the following elegant formulation for the accumulator function of a fold that implements the linear dynamic Kalman filter, that is, a filter that can track states that evolve with time[4] according to a linear transformation.

$$\begin{gathered}\text{kalmanDynamic}\left(\{\mathbf{x},\mathbf{P}\},\{\mathbf{Z},\Xi,\mathbf{\Phi},\Gamma,\mathbf{u},\mathbf{A},z\}\right)=\\ \left\{\mathbf{x}_2+\mathbf{K}\left(z-\mathbf{A}\,\mathbf{x}_2\right),\quad \mathbf{P}_2-\mathbf{K}\,\mathbf{D}\,\mathbf{K}^\intercal\right\}\end{gathered} \tag{1}$$

where

$$\mathbf{x}_2=\mathbf{\Phi}\,\mathbf{x}+\Gamma\,\mathbf{u} \tag{2}$$

$$\mathbf{P}_2=\Xi+\mathbf{\Phi}\,\mathbf{P}\,\mathbf{\Phi}^\intercal \tag{3}$$

$$\mathbf{K}=\mathbf{P}\,\mathbf{A}^\intercal\,\mathbf{D}^{-1} \tag{4}$$

$$\mathbf{D}=\mathbf{Z}+\mathbf{A}\,\mathbf{P}\,\mathbf{A}^\intercal \tag{5}$$

and all quantities are matrices:

- z is a $b \times 1$ column vector containing one multidimensional observation
- $\mathbf{x}$ and $\mathbf{x}_2$ are $n \times 1$ column vectors of *model states*
- $\mathbf{Z}$ is a $b \times b$ matrix, the covariance of observation noise
- $\mathbf{P}$ and $\mathbf{P}_2$ are $n \times n$ matrices, the theoretical covariances of $\mathbf{x}$ and $\mathbf{x}_2$, respectively
- $\mathbf{A}$ is a $b \times n$ matrix, the *observation partials*
- $\mathbf{D}$ is a $b \times b$ matrix, the Kalman denominator
- $\mathbf{K}$ is an $n \times b$ matrix, the Kalman gain
- Ξ is an $n \times n$ integral of *process noise*, namely $\int_0^{\delta t} \mathbf{\Phi}(\tau)\cdot\left(\begin{array}{c|c} 0 & 0 \\ \hline 0 & E\left[\xi\xi^\intercal\right]\end{array}\right)\cdot\mathbf{\Phi}(\tau)^\intercal\,d\tau$
- $\mathbf{\Phi}$ is an $n \times n$ integral of $\mathbf{F}$, namely $e^{\mathbf{F}\,\delta t}$
- Γ is an $n \times \dim(\mathbf{u})$ integral of *system response*, namely $\int_0^{\delta t} \mathbf{\Phi}(\tau)\cdot\mathbf{G}\,d\tau$

[3] http://reference.wolfram.com/language/

[4] In most applications, the independent variable is physical time, however, it need not be. For convenience, we use the term *time* to mean *the independent variable of the problem* simply because it is shorter to write.

- $\mathbf{u}$ is a vector of external *disturbances* or *control inputs*
- δt is an increment of time (or, more generally, then independent variable of the problem)

and the time-evolving states satisfy the following differential equation in *state-space form*:

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

$\mathbf{F}$, $\mathbf{G}$, and $\mathbf{u}$ may depend on time, but not on $\mathbf{x}$; that is the meaning of "linear dynamic" in this context. In this paper, we relieve those restrictions by explicitly integrating non-linear equations of motion and by using Taylor-series approximations for the gain $\mathbf{K}$ and $\mathbf{D}$ denominator matrices.

2.1 Dimensional Arguments

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. Both kinds of dimensions are aspects of the *type* of a quantity. Dimensional arguments, like type-arguments more generally, are invaluable for checking equations.

If the physical and matrix dimensions of $\mathbf{x}$ are $[[\mathbf{x}]] \overset{\text{def}}{=} (\mathcal{X}, n \times 1)$, of $\mathbf{z}$ are $[[\mathbf{z}]] \overset{\text{def}}{=} (\mathcal{Z}, b \times 1)$, of δt are $[[\delta t]] \overset{\text{def}}{=} (\mathcal{T}, \text{scalar})$, and of $\mathbf{u}$ are $[[\mathbf{u}]] \overset{\text{def}}{=} (\mathcal{U}, \dim(\mathbf{u}) \times 1)$, then

$$\begin{array}{lcclcl}
[[\mathbf{Z}]] & = & (& \mathcal{Z}^2 & b \times b &) \\
[[\mathbf{A}]] & = & (& \mathcal{Z}/\mathcal{X} & b \times n &) \\
[[\mathbf{P}]] & = & (& \mathcal{X}^2 & n \times n &) \\
[[\mathbf{A}\,\mathbf{P}\,\mathbf{A}^\intercal]] & = & (& \mathcal{Z}^2 & b \times b &) \\
[[\mathbf{D}]] & = & (& \mathcal{Z}^2 & b \times b &) \\
[[\mathbf{P}\,\mathbf{A}^\intercal]] & = & (& \mathcal{X}\mathcal{Z} & n \times b &) \\
[[\mathbf{K}]] & = & (& \mathcal{X}/\mathcal{Z} & n \times b &) \\
[[\mathbf{F}]] & = & (& 1/\mathcal{T} & n \times n &) \\
[[\mathbf{\Phi}]] & = & (& \text{dimensionless} & n \times n &) \\
[[\mathbf{G}]] & = & (& \mathcal{X}/(\mathcal{U}\mathcal{T}) & n \times \dim(\mathbf{u}) &) \\
[[\mathbf{\Gamma}]] & = & (& \mathcal{X}/\mathcal{U} & n \times \dim(\mathbf{u}) &) \\
[[\mathbf{\Xi}]] & = & (& \mathcal{X}^2 & n \times n &)
\end{array} \tag{7}$$

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

3 Tracking with Drag

Let us reproduce Zarchan and Musoff's[5] example of tracking a falling object in one position dimension, with aerodynamic drag. Throughout this series of papers, we refer to the examples in that reference because they provide solid benchmarks against which to prove our contribution. The difference between our approach and typical presentations of Kalman-type filters is functional decomposition: if you write code in functional style, you gain the ability to deploy it verbatim, even and especially at the binary level, in both laboratory and field. We show, in this series of papers,

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

that this ability can make the difference in a successful application because seemingly insignificant changes, even instruction order, can make qualitative differences in filter behavior due to floating-point issues.

To accommodate nonlinearity, we replace equation 2 for time-propagation of the state x with explicit numerical integration of the nonlinear equations of motion. We use an internal fold over a generic integrator interface to demonstrate easily changing the integrator from one that diverges the filter to one that converges it.

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 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 x lest the equations be non-linear. Write the time derivative of x as

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

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

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

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

$$\dot{x} = \mathbf{F}\,x + \mathbf{G}\,\mathbf{u} + \xi \tag{8}$$

This is standard *state-space form*[6] 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

$$\mathbf{\Phi}(\delta t) \overset{\text{def}}{=} e^{\mathbf{F}\,\delta t} = \mathbf{1} + \frac{\mathbf{F}^2\delta t^2}{2!} + \frac{\mathbf{F}^3\delta t^3}{3!} + \cdots \tag{9}$$

$$\mathbf{\Gamma}(\delta t) \overset{\text{def}}{=} \int_0^{\delta t} \mathbf{\Phi}(\tau)\cdot\mathbf{G}\,d\tau \tag{10}$$

$$\mathbf{\Xi}(\delta t) \overset{\text{def}}{=} \int_0^{\delta t} \mathbf{\Phi}(\tau)\cdot\begin{pmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & E\left[\xi\xi^{\mathsf{T}}\right] \end{pmatrix}\cdot\mathbf{\Phi}(\tau)^{\mathsf{T}}\,d\tau \tag{11}$$

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]`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:

$$\mathbf{x} \leftarrow \mathbf{\Phi}\,\mathbf{x} + \mathbf{\Gamma}\,\mathbf{u} \tag{12}$$

$$\mathbf{P} \leftarrow \Xi + \mathbf{\Phi}\,\mathbf{P}\,\mathbf{\Phi}^{\intercal} \tag{13}$$

These equations appear plausible on inspection and you can verify that they satisfy equation 8.

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.[7]

[7] 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.

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

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 9, 10, and 11

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

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 $-6,000$ ft/s and from time $t = 0$s to $t = 57.5$s, just before impact at $h = 0$ft. We take one observation every tenth of a second, so $\delta t = 0.10$ 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

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

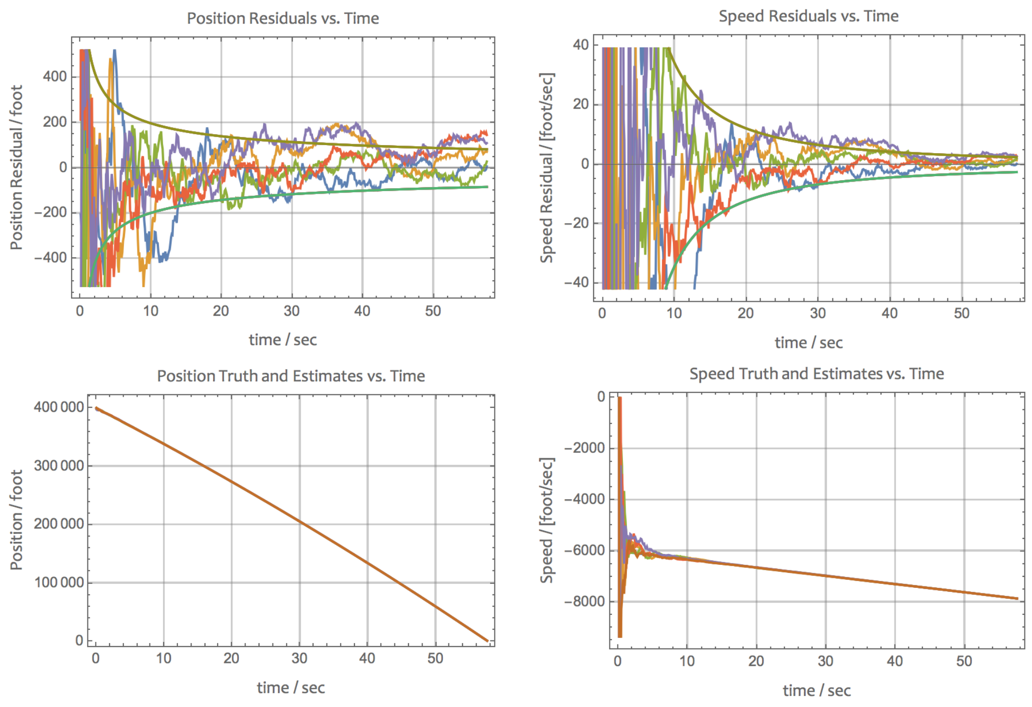
where

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

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