The Kalman Filter

The RLS algorithm for updating the least squares estimate given a series of vector observations looked like a "filter": new data comes in, and we use it (along with collected knowledge of the old data) to produce a new output.

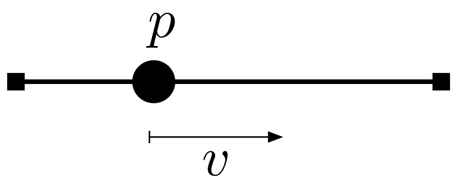
In the previous section, $\boldsymbol{x}_*$ was fixed for the entire sequence of observations. The **Kalman filter** incorporates **dynamics** for the unknown vector $\boldsymbol{x}$ into our estimation framework. It addresses the case where $\boldsymbol{x}$ **changes** from observation to observation in a manner which we can model.

The classic example is trying to estimate the position and velocity of an airplane. If the plane is in motion, these will of course change over time. But the path of the plane is somewhat predictable, and there are real physical constraints on how quickly it can turn and/or accelerate.

Our dynamical models will be linear. That is, we will assume that we can approximate the solution at the next time step $\boldsymbol{x}_{k+1}$ by applying a known $N \times N$ matrix $\boldsymbol{F}_k$ to the current solution:

$$\boldsymbol{x}_{k+1} \approx \boldsymbol{F}_k \boldsymbol{x}_k.$$

Here is a quick example of how this might work. Suppose that we are trying to estimate the position p and velocity v of a particle traveling in one dimension:



In this case, our unknowns are a two-vector:

$$\boldsymbol{x}_k = \begin{bmatrix} p_k \\ v_k \end{bmatrix}.$$

If we expect the velocity to be almost the same from measurement to measurement, our dynamics equations would look like

$$\begin{aligned} p_{k+1} &= p_k + \alpha_k v_k \quad &(+\text{ error}) \\ v_{k+1} &= v_k &(+\text{ error}), \end{aligned}$$

where α_k would be proportional to the time elapsed between $k+1$ and k. In terms of the $\boldsymbol{x}_k$, we can write

$$\boldsymbol{x}_{k+1} = \boldsymbol{F}_k \boldsymbol{x}_k, \quad \text{where} \quad \boldsymbol{F}_k = \begin{bmatrix} 1 & \alpha_k \\ 0 & 1 \end{bmatrix}.$$

Setting up the system

As before, we are making noisy observations of an unknown vector:

$$\boldsymbol{y}_k = \boldsymbol{A}_k \boldsymbol{x}_k + \boldsymbol{e}_k, \quad \boldsymbol{e}_k = \text{“measurement error”}.$$

There can be a different number of measurements at each step, but all of the $\boldsymbol{x}_k$ have length N. We will say that $\boldsymbol{y}_k \in \mathbb{R}^{M_k}$ and the $\boldsymbol{A}_k$ are $M_k \times N$ matrices.

The $\boldsymbol{x}_k$ are dynamic; we model the transition from step k to $k+1$ using

$$\boldsymbol{x}_{k+1} = \boldsymbol{F}_k \boldsymbol{x}_k + \boldsymbol{\epsilon}_k, \quad \boldsymbol{\epsilon}_k = \text{“state error”}.$$

The $\boldsymbol{F}_k$ are all $N \times N$.

It is possible to incorporate the correlation structure of the $\boldsymbol{e}_k$ and $\boldsymbol{\epsilon}_k$ into our estimates, but we will start with the standard least-squares framework.

Notation: Since the $\boldsymbol{x}_k$ are linked together through the dynamical model, the measurements $y_k = \boldsymbol{A}_k\boldsymbol{x}_k + \boldsymbol{e}_k$ also give us indirect information about $\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{k-1}$. As such, our estimate of **all** of $\boldsymbol{x}_0, \ldots, \boldsymbol{x}_k$ will change at time k. We will use the notation

$$\widehat{\boldsymbol{x}}_{\ell|k} = \text{least-squares estimate of } \boldsymbol{x}_\ell \text{ at time } k.$$

Step $k = 0$. We have observed

$$\boldsymbol{y}_0 = \boldsymbol{A}_0\boldsymbol{x}_0 + \boldsymbol{e}_0.$$

We form the least-squares estimate

$$\widehat{\boldsymbol{x}}_{0|0} = (\boldsymbol{A}_0^{\mathrm{T}}\boldsymbol{A}_0)^{-1}\boldsymbol{A}_0^{\mathrm{T}}\boldsymbol{y}_0.$$

Step $k = 1$. We have the following system of equations

$$\begin{aligned} \boldsymbol{y}_0 &= \boldsymbol{A}_0\boldsymbol{x}_0 + \boldsymbol{e}_0 \\ \mathbf{0} &= \boldsymbol{F}_0\boldsymbol{x}_0 - \boldsymbol{x}_1 + \boldsymbol{\epsilon}_0 \\ \boldsymbol{y}_1 &= \boldsymbol{A}_1\boldsymbol{x}_1 + \boldsymbol{e}_1. \end{aligned}$$

We can write this compactly as $\underline{\boldsymbol{y}} = \underline{\boldsymbol{A}}_1\underline{\boldsymbol{x}} + \underline{\boldsymbol{e}}_1$, where

$$\underbrace{\begin{bmatrix} \boldsymbol{y}_0 \\ \mathbf{0} \\ \boldsymbol{y}_1 \end{bmatrix}}_{\underline{\boldsymbol{y}}_1} = \underbrace{\begin{bmatrix} \boldsymbol{A}_0 & \mathbf{0} \\ \boldsymbol{F}_0 & -\mathbf{I} \\ \mathbf{0} & \boldsymbol{A}_1 \end{bmatrix}}_{\underline{\boldsymbol{A}}_1} \underbrace{\begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_1 \end{bmatrix}}_{\underline{\boldsymbol{x}}_1} + \underbrace{\begin{bmatrix} \boldsymbol{e}_0 \\ \boldsymbol{\epsilon}_0 \\ \boldsymbol{e}_1 \end{bmatrix}}_{\underline{\boldsymbol{e}}_1}.$$

This system is $(M_0 + N + M_1) \times 2N$; we can form the least-squares estimate using

$$\begin{bmatrix} \widehat{\boldsymbol{x}}_{0|1} \\ \widehat{\boldsymbol{x}}_{1|1} \end{bmatrix} = (\underline{\boldsymbol{A}}_1^{\mathrm{T}} \underline{\boldsymbol{A}}_1)^{-1} \underline{\boldsymbol{A}}_1^{\mathrm{T}} \underline{\boldsymbol{y}}_1.$$

Step $k = 2$**.** Now we have the following system of equations

$$\begin{aligned} \boldsymbol{y}_0 &= \boldsymbol{A}_0 \boldsymbol{x}_0 + \boldsymbol{e}_0 \\ \mathbf{0} &= \boldsymbol{F}_0 \boldsymbol{x}_0 - \boldsymbol{x}_1 + \boldsymbol{\epsilon}_0 \\ \boldsymbol{y}_1 &= \boldsymbol{A}_1 \boldsymbol{x}_1 + \boldsymbol{e}_1 \\ \mathbf{0} &= \boldsymbol{F}_1 \boldsymbol{x}_1 - \boldsymbol{x}_2 + \boldsymbol{\epsilon}_1 \\ \boldsymbol{y}_2 &= \boldsymbol{A}_2 \boldsymbol{x}_2 + \boldsymbol{e}_2, \end{aligned}$$

which we can rewrite as

$$\underbrace{\begin{bmatrix} \boldsymbol{y}_0 \\ \mathbf{0} \\ \boldsymbol{y}_1 \\ \mathbf{0} \\ \boldsymbol{y}_2 \end{bmatrix}}_{\underline{\boldsymbol{y}}_2} = \underbrace{\begin{bmatrix} \boldsymbol{A}_0 & \mathbf{0} & \mathbf{0} \\ \boldsymbol{F}_0 & -\mathbf{I} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{A}_1 & \mathbf{0} \\ \mathbf{0} & \boldsymbol{F}_1 & -\mathbf{I} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{A}_2 \end{bmatrix}}_{\underline{\boldsymbol{A}}_2} \underbrace{\begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix}}_{\underline{\boldsymbol{x}}_2} + \underbrace{\begin{bmatrix} \boldsymbol{e}_0 \\ \boldsymbol{\epsilon}_0 \\ \boldsymbol{e}_1 \\ \boldsymbol{\epsilon}_1 \\ \boldsymbol{e}_2 \end{bmatrix}}_{\underline{\boldsymbol{e}}_2}.$$

This system is $(M_0 + M_1 + M_2 + 2N) \times 3N$; we can form the least-squares estimate using

$$\begin{bmatrix} \widehat{\boldsymbol{x}}_{0|2} \\ \widehat{\boldsymbol{x}}_{1|2} \\ \widehat{\boldsymbol{x}}_{2|2} \end{bmatrix} = (\underline{\boldsymbol{A}}_2^{\mathrm{T}} \underline{\boldsymbol{A}}_2)^{-1} \underline{\boldsymbol{A}}_2^{\mathrm{T}} \underline{\boldsymbol{y}}_2.$$

In general, we have

$$\underline{\boldsymbol{A}}_k = \begin{bmatrix} \boldsymbol{A}_0 & & & & \\ \boldsymbol{F}_0 & -\mathbf{I} & & & \\ & \boldsymbol{A}_1 & & & \\ & \boldsymbol{F}_1 & & & \\ & & \ddots & & \\ & & & \boldsymbol{F}_{k-1} & -\mathbf{I} \\ & & & & \boldsymbol{A}_k \end{bmatrix}, \quad \underline{\boldsymbol{y}}_k = \begin{bmatrix} \boldsymbol{y}_0 \\ \mathbf{0} \\ \boldsymbol{y}_1 \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \boldsymbol{y}_k \end{bmatrix}; \qquad (1)$$

this system is $(M_0 + \cdots + M_k + kN) \times (k+1)N$, and we can for the least-squares estimate using

$$\begin{bmatrix} \widehat{\boldsymbol{x}}_{0|k} \\ \widehat{\boldsymbol{x}}_{1|k} \\ \vdots \\ \widehat{\boldsymbol{x}}_{k|k} \end{bmatrix} = (\underline{\boldsymbol{A}}_k^{\mathrm{T}} \underline{\boldsymbol{A}}_k)^{-1} \underline{\boldsymbol{A}}_k^{\mathrm{T}} \underline{\boldsymbol{y}}_k.$$

The $(k+1)N$-vector $\underline{\boldsymbol{A}}_k^{\mathrm{T}} \underline{\boldsymbol{y}}_k$ is

$$\underline{\boldsymbol{A}}_k^{\mathrm{T}} \underline{\boldsymbol{y}}_k = \begin{bmatrix} \boldsymbol{A}_0^{\mathrm{T}} \boldsymbol{y}_0 \\ \boldsymbol{A}_1^{\mathrm{T}} \boldsymbol{y}_1 \\ \vdots \\ \boldsymbol{A}_k^{\mathrm{T}} \boldsymbol{y}_k \end{bmatrix},$$

while the $(k+1)N \times (k+1)N$ matrix $\underline{\boldsymbol{A}}_k^{\mathrm{T}} \underline{\boldsymbol{A}}_k$ we have to invert is

$$\begin{bmatrix} \boldsymbol{A}_0^{\mathrm{T}}\boldsymbol{A}_0 + \boldsymbol{F}_0^{\mathrm{T}}\boldsymbol{F}_0 & -\boldsymbol{F}_0^{\mathrm{T}} & \mathbf{0} & & & \\ -\boldsymbol{F}_0 & \mathbf{I} + \boldsymbol{A}_1^{\mathrm{T}}\boldsymbol{A}_1 + \boldsymbol{F}_1^{\mathrm{T}}\boldsymbol{F}_1 & -\boldsymbol{F}_1^{\mathrm{T}} & & & \\ \mathbf{0} & -\boldsymbol{F}_1 & \mathbf{I} + \boldsymbol{A}_2^{\mathrm{T}}\boldsymbol{A}_2 + \boldsymbol{F}_2^{\mathrm{T}}\boldsymbol{F}_2 & -\boldsymbol{F}_2^{\mathrm{T}} & & \\ \vdots & & & \ddots & & \\ & & & & \mathbf{I} + \boldsymbol{A}_{k-1}^{\mathrm{T}}\boldsymbol{A}_{k-1} + \boldsymbol{F}_{k-1}^{\mathrm{T}}\boldsymbol{F}_{k-1} & -\boldsymbol{F}_{k-1}^{\mathrm{T}} \\ & & & & -\boldsymbol{F}_{k-1} & \mathbf{I} + \boldsymbol{A}_k^{\mathrm{T}}\boldsymbol{A}_k \end{bmatrix} \qquad (2)$$

Notice that this matrix is **block tridiagonal**; this structure is precisely what we will exploit in deriving the efficient update equations later on.

Example: Multiple measurements of a pulse

The following example illustrates the essential difference between the standard least-square framework and the Kalman filter.

We make three measurements of a patient's pulse, and want an (updated) estimate of its true value after each one. Our measurements are scalars, and have the form:

$$y = x + \text{noise}.$$

We record the three values y_0, y_1, y_2.

In the standard least-square framework, we solve the three problems

$$y_0 = 1\, x_* + e_0, \quad \begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} x_* + \begin{bmatrix} e_0 \\ e_1 \end{bmatrix}, \quad \text{and} \quad \begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} x_* + \begin{bmatrix} e_0 \\ e_1 \\ e_2 \end{bmatrix}.$$

If we call $\widehat{x}_k$ the least-squares estimate after recording y_k, a simple calculation shows that the estimate is simply the average of the measurements we have seen to date:

$$\widehat{x}_0 = y_0, \quad \widehat{x}_1 = \frac{y_0 + y_1}{2}, \quad \widehat{x}_2 = \frac{y_0 + y_1 + y_2}{3}.$$

Now suppose the we use the Kalman filtering framework to explicitly recognize that the pulse can "drift" over time. Our dynamical model is simple:

$$x_{k+1} = x_k + \epsilon_k.$$

This is saying that our best guess for the pulse at the next time step is that it takes the same value as before, but we are incorporating the fact that it will in fact not be exactly the same as this best guess.

In the Kalman framework, the three systems we solve look like

$$y_0 = 1\,x_0 + e_0,$$

$$\begin{bmatrix} y_0 \\ 0 \\ y_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} + \begin{bmatrix} e_0 \\ \epsilon_0 \\ e_1 \end{bmatrix},$$

$$\begin{bmatrix} y_0 \\ 0 \\ y_1 \\ 0 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} e_0 \\ \epsilon_0 \\ e_1 \\ \epsilon_1 \\ e_2 \end{bmatrix}.$$

After the first measurement, our estimate for x_0 is the same:

$$\widehat{x}_{0|0} = y_0.$$

After the second measurement, we have

$$\begin{aligned} \begin{bmatrix} \widehat{x}_{0|1} \\ \widehat{x}_{1|1} \end{bmatrix} &= \left(\begin{bmatrix} 1 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \right)^{-1} \begin{bmatrix} 1 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} y_0 \\ 0 \\ y_1 \end{bmatrix} \\ &= \left(\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} \right)^{-1} \begin{bmatrix} y_0 \\ y_1 \end{bmatrix} \\ &= \begin{bmatrix} 2/3 & 1/3 \\ 1/3 & 2/3 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \end{bmatrix} \\ &= \begin{bmatrix} \frac{2y_0+y_1}{3} \\ \frac{y_0+2y_1}{3} \end{bmatrix} \end{aligned}$$

So the best guess for where the pulse was is now a weighted average between the two samples; the estimate for x_1 weighs the most recent measurement more heavily than the first one. This is natural, since we are saying the pulse is changing between measurements.

After the third measurement, we have

$$\begin{aligned}\begin{bmatrix}\widehat{x}_{0|2}\\ \widehat{x}_{1|2}\\ \widehat{x}_{2|2}\end{bmatrix} &= \left(\begin{bmatrix}2 & -1 & 0\\ -1 & 3 & -1\\ 0 & -1 & 2\end{bmatrix}\right)^{-1}\begin{bmatrix}y_0\\ y_1\\ y_2\end{bmatrix}\\ &= \begin{bmatrix}\frac{5y_0+2y_1+y_2}{8}\\ \frac{y_0+2y_1+y_2}{4}\\ \frac{y_0+2y_1+5y_2}{8}\end{bmatrix}.\end{aligned}$$

Again, we see that as a direct result of allowing the pulse to drift, recent measurements are weighed more heavily in the current estimate.

Block tridiagonal factorization

To find the least-squares estimate of $x_0, \ldots, x_{N-1}$, we need to solve the $kN \times kN$ system given by (2) a few pages back. In this section, we will show how this this type of system allows a nice factorization which makes it clear how to solve it in a "streaming" manner.

Consider the following general symmetric block tridiagonal system:

$$\begin{bmatrix} \boldsymbol{D}_0 & \boldsymbol{C}_0^{\mathrm{T}} & \mathbf{0} & \cdots & & \mathbf{0} \\ \boldsymbol{C}_0 & \boldsymbol{D}_1 & \boldsymbol{C}_1^{\mathrm{T}} & \mathbf{0} & \cdots & \vdots \\ \mathbf{0} & \boldsymbol{C}_1 & \boldsymbol{D}_2 & \ddots & & \\ \vdots & & \ddots & \ddots & \ddots & \\ & & & \ddots & \ddots & \boldsymbol{C}_{k-1}^{\mathrm{T}} \\ \mathbf{0} & \cdots & & & \boldsymbol{C}_{k-1} & \boldsymbol{D}_k \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_1 \\ \vdots \\ \vdots \\ \boldsymbol{x}_{k-1} \\ \boldsymbol{x}_k \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_0 \\ \boldsymbol{b}_1 \\ \vdots \\ \vdots \\ \boldsymbol{b}_{k-1} \\ \boldsymbol{b}_k \end{bmatrix}$$

The system is $(k+1)N \times (k+1)N$, and each of the $\boldsymbol{D}_i$ and $\boldsymbol{C}_i$ are $N \times N$. Likewise, $\boldsymbol{x}_i \in \mathbb{R}^N$ and $\boldsymbol{b}_i \in \mathbb{R}^N$.

The matrix on the left-hand side above as the product of block lower triangular and block upper triangular matrices:

$$\begin{bmatrix} \boldsymbol{Q}_0 & \mathbf{0} & \cdots & & \mathbf{0} \\ \boldsymbol{C}_0 & \boldsymbol{Q}_1 & \mathbf{0} & & \\ \mathbf{0} & \boldsymbol{C}_1 & \boldsymbol{Q}_2 & \ddots & \\ \vdots & & \ddots & \ddots & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \boldsymbol{C}_{k-1} & \boldsymbol{Q}_k \end{bmatrix} \begin{bmatrix} \mathbf{I} & \boldsymbol{U}_0 & \mathbf{0} & & \\ \mathbf{0} & \mathbf{I} & \boldsymbol{U}_1 & \mathbf{0} & \\ \vdots & & \ddots & \ddots & \\ & & & \ddots & \boldsymbol{U}_{k-1} \\ \mathbf{0} & & & \mathbf{0} & \mathbf{I} \end{bmatrix},$$

where the $\boldsymbol{C}_i$ are the same as in the original system, and the $\boldsymbol{Q}_i$ and $\boldsymbol{U}_i$ are also all $N \times N$. By inspecting the blocks, we see that we can compute the $\boldsymbol{Q}_i$ and $\boldsymbol{U}_i$ using the following iterative algorithm:

```
$\boldsymbol{Q}_0 = \boldsymbol{D}_0$
for $i = 1, 2, \ldots, k$
    $\boldsymbol{U}_{i-1} = \boldsymbol{Q}_{i-1}^{-1}\boldsymbol{C}_{i-1}^{\mathrm{T}}$
    $\boldsymbol{Q}_i = \boldsymbol{D}_i - \boldsymbol{C}_{i-1}\boldsymbol{U}_{i-1}$, meaning that $\boldsymbol{Q}_i = \boldsymbol{D}_i - \boldsymbol{C}_{i-1}\boldsymbol{Q}_{i-1}^{-1}\boldsymbol{C}_{i-1}^{\mathrm{T}}$
end
```

If the factorization is already in place, we can solve for the $\boldsymbol{x}_i$ by using forward substitution followed by back substitution. First, we solve

$$\begin{bmatrix} \boldsymbol{Q}_0 & \mathbf{0} & \cdots & & \mathbf{0} \\ \boldsymbol{C}_0 & \boldsymbol{Q}_1 & \mathbf{0} & & \\ \mathbf{0} & \boldsymbol{C}_1 & \boldsymbol{Q}_2 & \ddots & \\ \vdots & & \ddots & \ddots & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \boldsymbol{C}_{k-1} & \boldsymbol{Q}_k \end{bmatrix} \begin{bmatrix} \boldsymbol{w}_0 \\ \boldsymbol{w}_1 \\ \vdots \\ \\ \boldsymbol{w}_k \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_0 \\ \boldsymbol{b}_1 \\ \vdots \\ \\ \boldsymbol{b}_k \end{bmatrix}$$

working from the top down:

```
$\boldsymbol{w}_0 = \boldsymbol{Q}_0^{-1}\boldsymbol{b}_0$
for $i = 1, 2, \ldots, k$
    $\boldsymbol{w}_i = \boldsymbol{Q}_i^{-1}(\boldsymbol{b}_i - \boldsymbol{C}_{i-1}\boldsymbol{w}_{i-1})$
end
```

With the $\boldsymbol{w}_i$ in hand, we can now solve

$$\begin{bmatrix} \mathbf{I} & \boldsymbol{U}_0 & \mathbf{0} & & \\ \mathbf{0} & \mathbf{I} & \boldsymbol{U}_1 & \mathbf{0} & \\ \vdots & & \ddots & \ddots & \\ & & & \ddots & \boldsymbol{U}_{k-1} \\ \mathbf{0} & & & \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_1 \\ \vdots \\ \\ \boldsymbol{x}_k \end{bmatrix} = \begin{bmatrix} \boldsymbol{w}_0 \\ \boldsymbol{w}_1 \\ \vdots \\ \\ \boldsymbol{w}_k \end{bmatrix}$$

for the $\boldsymbol{x}_i$ by working bottom up:

```
$\boldsymbol{x}_k = \boldsymbol{w}_k$
for $i = k-1, k-2, \ldots, 0$
    $\boldsymbol{x}_i = \boldsymbol{w}_i - \boldsymbol{U}_i\boldsymbol{x}_{i+1}$
end
```

Of course, we can combine the factorization with the forward step of the solve above to yield the following algorithm for solving symmetric[1] block tridiagonal systems:

```
Symmetric Block Tridiagonal Solve
  Input: $N \times N$ matrices $\boldsymbol{D}_0, \ldots, \boldsymbol{D}_k$ and $\boldsymbol{C}_0, \ldots, \boldsymbol{C}_{k-1}$;
         Right-hand side $N$-vectors $\boldsymbol{b}_0, \ldots, \boldsymbol{b}_k$
  Initialize: $\boldsymbol{Q}_0 = \boldsymbol{D}_0$
              $\boldsymbol{w}_0 = \boldsymbol{Q}_0^{-1}\boldsymbol{b}_0$
  for $i = 1, 2, \ldots, k$ do
      $\boldsymbol{U}_{i-1} = \boldsymbol{Q}_{i-1}^{-1}\boldsymbol{C}_{i-1}^{\mathrm{T}}$
      $\boldsymbol{Q}_i = \boldsymbol{D}_i - \boldsymbol{C}_{i-1}\boldsymbol{U}_{i-1}$
      $\boldsymbol{w}_i = \boldsymbol{Q}_i^{-1}(\boldsymbol{b}_i - \boldsymbol{C}_{i-1}\boldsymbol{w}_{i-1})$
  end for
  $\boldsymbol{x}_k = \boldsymbol{w}_k$
  for $i = k-1, k-2, \ldots, 0$ do
      $\boldsymbol{x}_i = \boldsymbol{w}_i - \boldsymbol{U}_i\boldsymbol{x}_{i+1}$
  end for
```

[1]This is easily adapted to non-symmetric block tridiagonal, but we only need the symmetric case for what we do below.

Kalman filter update equations

Recalling equation (2) from earlier in these notes, we can find the least-squares solution $\widehat{\boldsymbol{x}}_{0|k}, \widehat{\boldsymbol{x}}_{1|k}, \ldots, \widehat{\boldsymbol{x}}_{k|k}$ at step k using the tridiagonal solver from the previous section with

$$\boldsymbol{C}_i = -\boldsymbol{F}_i, \quad i = 0, \ldots, k-1$$
$$\boldsymbol{D}_i = \begin{cases} \boldsymbol{A}_0^{\mathrm{T}}\boldsymbol{A}_0 + \boldsymbol{F}_0^{\mathrm{T}}\boldsymbol{F}_0, & i = 0 \\ \mathbf{I} + \boldsymbol{A}_i^{\mathrm{T}}\boldsymbol{A}_i + \boldsymbol{F}_i^{\mathrm{T}}\boldsymbol{F}_i, & i = 1, \ldots, k-1 \\ \mathbf{I} + \boldsymbol{A}_k^{\mathrm{T}}\boldsymbol{A}_k, & i = k. \end{cases}$$

That is great, but what is even better is that we can quickly move from the current best estimate $\widehat{\boldsymbol{x}}_{k|k}$ to the estimate at the next time step $\widehat{\boldsymbol{x}}_{k+1|k+1}$ very quickly.

Suppose that we have in our hands the solution to (2), consisting of the estimates

$$\widehat{\boldsymbol{x}}_{0|k}, \widehat{\boldsymbol{x}}_{1|k}, \ldots, \widehat{\boldsymbol{x}}_{k|k} \tag{3}$$

When the signal evolves ($\boldsymbol{x}_{k+1} = \boldsymbol{F}_k\boldsymbol{x}_k + \boldsymbol{\epsilon}_k$) and new measurements come in ($\boldsymbol{y}_{k+1} = \boldsymbol{A}_{k+1}\boldsymbol{x}_{k+1} + \boldsymbol{e}_{k+1}$), we break the update into two stages. The first is the "predict" stage, where we add the evolution equations — we solve,

$$\underline{\widetilde{\boldsymbol{A}}}_{k+1} = \begin{bmatrix} \boldsymbol{A}_0 & & & & & \\ \boldsymbol{F}_0 & -\mathbf{I} & & & & \\ & \boldsymbol{A}_1 & & & & \\ & \boldsymbol{F}_1 & & & & \\ & & \ddots & & & \\ & & & \boldsymbol{F}_{k-1} & -\mathbf{I} & \\ & & & & \boldsymbol{A}_k & \mathbf{0} \\ & & & & \boldsymbol{F}_k & -\mathbf{I} \end{bmatrix}, \quad \underline{\widetilde{\boldsymbol{y}}}_{k+1} = \begin{bmatrix} \boldsymbol{y}_0 \\ \mathbf{0} \\ \boldsymbol{y}_1 \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \boldsymbol{y}_k \\ \mathbf{0} \end{bmatrix}.$$

This system has another block column, but since the last block column in non-zero only in the $k+1$ block, the $\boldsymbol{x}_{k+1}$ variables are essentially decoupled. The solution to $\widetilde{\underline{\boldsymbol{A}}}_{k+1}^{\mathrm{T}}\widetilde{\underline{\boldsymbol{A}}}_{k+1}\bar{\boldsymbol{x}} = \widetilde{\underline{\boldsymbol{A}}}_{k+1}^{\mathrm{T}}\widetilde{\underline{\boldsymbol{y}}}_{k+1}$ will be exactly the same as before, i.e. (3), but with an additional component given by

$$\widehat{\boldsymbol{x}}_{k+1|k} = \boldsymbol{F}_k\widehat{\boldsymbol{x}}_{k|k}.$$

We can interpret this as a **prediction** for $\widehat{\boldsymbol{x}}_{k+1|k+1}$ which we will update when we incorporate the measuremets $\boldsymbol{y}_{k+1}$.

The factorization is updated with

$$\begin{aligned}\widetilde{\boldsymbol{D}}_k &= \boldsymbol{D}_k + \boldsymbol{F}_k^{\mathrm{T}}\boldsymbol{F}_k = \mathbf{I} + \boldsymbol{A}_k^{\mathrm{T}}\boldsymbol{A}_k^{\mathrm{T}} + \boldsymbol{F}_k^{\mathrm{T}}\boldsymbol{F}_k \\ \boldsymbol{C}_k &= -\boldsymbol{F}_k \\ \widetilde{\boldsymbol{D}}_{k+1} &= \mathbf{I}.\end{aligned}$$

How exactly this works is thoroughly detailed in the Technical Details section below.

In the second stage, the "update" stage, we incorporate the measurements and solve

$$\underline{\boldsymbol{A}}_{k+1} = \begin{bmatrix} \boldsymbol{A}_0 & & & & & \\ \boldsymbol{F}_0 & -\mathbf{I} & & & & \\ & \boldsymbol{A}_1 & & & & \\ & \boldsymbol{F}_1 & & & & \\ & & \ddots & & & \\ & & & \boldsymbol{F}_{k-1} & -\mathbf{I} & \\ & & & & \boldsymbol{A}_k & \mathbf{0} \\ & & & & \boldsymbol{F}_k & -\mathbf{I} \\ & & & & \mathbf{0} & \boldsymbol{A}_{k+1} \end{bmatrix}, \quad \underline{\boldsymbol{y}}_{k+1} = \begin{bmatrix} \boldsymbol{y}_0 \\ \mathbf{0} \\ \boldsymbol{y}_1 \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \boldsymbol{y}_k \\ \mathbf{0} \\ \boldsymbol{y}_{k+1} \end{bmatrix}$$

This update works by changing the last block diagonal element to

$$\boldsymbol{D}_{k+1} = \widetilde{\boldsymbol{D}}_{k+1} + \boldsymbol{A}_{k+1}^{\mathrm{T}}\boldsymbol{A}_{k+1} = \mathbf{I} + \boldsymbol{A}_{k+1}^{\mathrm{T}}\boldsymbol{A}_{k+1}.$$

we can then quickly compute $\widehat{\boldsymbol{x}}_{k+1|k+1} = \boldsymbol{w}_{k+1}$ as

$$\widehat{\boldsymbol{x}}_{k+1|k+1} = \widehat{\boldsymbol{x}}_{k+1|k} + \boldsymbol{G}_{k+1}(\boldsymbol{y}_{k+1} - \boldsymbol{A}_{k+1}\widehat{\boldsymbol{x}}_{k+1|k}),$$

where $\boldsymbol{G}_{k+1}$ is the "Kalman gain" matrix (this is computed below). Notice that if the measurements $\boldsymbol{y}_{k+1}$ match our prediction $\widehat{\boldsymbol{x}}_{k+1|k}$, then we don't update this prediction at all — otherwise we pass the difference through the gain matrix $\boldsymbol{G}_{k+1}$ and add it to $\widehat{\boldsymbol{x}}_{k+1|k}$.

Here are the updating equations, which are carefully derived in the Technical Details section:

```
Kalman Filter
  Initialize: $\widehat{\boldsymbol{x}}_{0|0} = (\boldsymbol{A}_0^{\mathrm{T}}\boldsymbol{A}_0)^{-1}\boldsymbol{A}_0^{\mathrm{T}}\boldsymbol{y}_0$
              $\boldsymbol{P}_{0|0} = (\boldsymbol{A}_0^{\mathrm{T}}\boldsymbol{A}_0)^{-1}$
  for $k = 0, 1, 2, \ldots$ do
      State update extrapolation: $\widehat{\boldsymbol{x}}_{k+1|k} = \boldsymbol{F}_k\widehat{\boldsymbol{x}}_{k|k}$
      Info matrix extrapolation: $\boldsymbol{P}_{k+1|k} = \boldsymbol{F}_k\boldsymbol{P}_{k|k}\boldsymbol{F}_k^{\mathrm{T}} + \mathbf{I}$
      Kalman gain: $\boldsymbol{G}_{k+1} = \boldsymbol{P}_{k+1|k}\boldsymbol{A}_{k+1}^{\mathrm{T}}(\boldsymbol{A}_{k+1}\boldsymbol{P}_{k+1|k}\boldsymbol{A}_{k+1}^{\mathrm{T}} + \mathbf{I})^{-1}$
      State update: $\widehat{\boldsymbol{x}}_{k+1|k+1} = \widehat{\boldsymbol{x}}_{k+1|k} + \boldsymbol{G}_{k+1}(\boldsymbol{y}_{k+1} - \boldsymbol{A}_{k+1}\widehat{\boldsymbol{x}}_{k+1|k})$
      Info matrix update: $\boldsymbol{P}_{k+1|k+1} = (\mathbf{I} - \boldsymbol{G}_{k+1}\boldsymbol{A}_{k+1})\boldsymbol{P}_{k+1|k}$
  end for
```

So the cost of moving from $\widehat{\boldsymbol{x}}_{k|k}$ to $\widehat{\boldsymbol{x}}_{k+1|k+1}$ is $O(N^3) + O(M_{k+1}N^2)$, which is about the same as it would cost to simply compute the standard least-squares solution for $\boldsymbol{x}_{k+1}$ using only $\boldsymbol{y}_{k+1}$.

System Identification

In this section, we will see how we can use least-squares to **learn** the impulse response of a linear time-invariant system by observing its input and output. We have seen already how we might set this up as a linear inverse problem (see the example at the beginning of Chapter II of these notes). Now, using the work we just did on recursive least-squares, we can describe how we could solve this problem in a streaming manner.

Specifically, suppose we observe the convolution

$$y[n] = \sum_{k=0}^{N-1} h_*[k]u[n-k] + \text{ noise},$$

where

- $u[n]$ is the input signal (observed),
- $y[n]$ is the output signal (observed), and
- $h_*[n]$ is the impulse response of (finite) length N (unknown).

Our goal is to estimate $\boldsymbol{h}_* \in \mathbb{R}^N$ after observing $u[n]$ and $y[n]$ over some amount of time.

Note that we can equivalently write our observations as

$$y[n] = \boldsymbol{A}_n \boldsymbol{h}_* \text{ noise},$$

where $\boldsymbol{A}_n$ is the $1 \times N$ matrix

$$\boldsymbol{A}_n = \begin{bmatrix} u[n] & u[n-1] & \dots & u[n-N+1] \end{bmatrix} = \boldsymbol{u}_n^{\mathrm{T}}.$$

With all of the measurement vectors up to time M stacked up:

$$\underline{\boldsymbol{A}}_M = \begin{bmatrix} \boldsymbol{A}_0 \\ \boldsymbol{A}_1 \\ \vdots \\ \boldsymbol{A}_M \end{bmatrix} = \begin{bmatrix} \boldsymbol{u}_0^{\mathrm{T}} \\ \boldsymbol{u}_1^{\mathrm{T}} \\ \vdots \\ \boldsymbol{u}_M^{\mathrm{T}} \end{bmatrix}, \quad \underline{\boldsymbol{y}}_M = \begin{bmatrix} y[0] \\ y[1] \\ \vdots \\ y[M] \end{bmatrix},$$

we could then form the least-squares estimate

$$\widehat{\boldsymbol{h}} = (\underline{\boldsymbol{A}}_M^{\mathrm{T}} \underline{\boldsymbol{A}}_M)^{-1} \underline{\boldsymbol{A}}_M^{\mathrm{T}} \underline{\boldsymbol{y}}_M.$$

Alternatively, we could apply the recursive least-squares algorithm from page 55 of these notes, giving us the following iteration (we have moved things around to make things as efficient as possible for this special case):

$$\begin{aligned} \boldsymbol{v}_n &= \boldsymbol{P}_{n-1}\boldsymbol{u}_n \\ \boldsymbol{k}_n &= \frac{1}{1 + \boldsymbol{u}_n^{\mathrm{T}}\boldsymbol{v}_n}\boldsymbol{v}_n \\ \boldsymbol{h}_n &= \boldsymbol{h}_{n-1} + (y[n] - \boldsymbol{u}_n^{\mathrm{T}}\boldsymbol{h}_{n-1})\boldsymbol{k}_n \\ \boldsymbol{P}_n &= \boldsymbol{P}_{n-1} - \boldsymbol{k}_n\boldsymbol{v}_n^{\mathrm{T}}\boldsymbol{P}_{n-1}. \end{aligned}$$

Note that this algorithm is much more efficient than re-solving the entire least-squares problem from scratch with each new measurement, but it still requires a matrix-vector multiplication at each iteration. It might be hard to imagine anything that would require even less computation, but there are some surprisingly effective alternatives that are even cheaper! We will talk about one particularly important example.

Least Mean Squares (LMS) Filter

The **Least Mean Squares (LMS) filter** is an important algorithm that is even cheaper (computationally) than recursive least-squares. This algorithm, introduced in 1960, is also the first example of a **stochastic gradient** algorithm, which is currently a hot topic in large-scale machine learning.

One way to think of the LMS filter is as a (very rough) approximation to using steepest descent to solve our least-squares optimization problem. To be more precise, recall that the optimization problem

$$\underset{\boldsymbol{h}\in\mathbb{R}^N}{\text{minimize}} \;\; \|\underline{\boldsymbol{A}}_M\boldsymbol{h} - \underline{\boldsymbol{y}}_M\|_2^2.$$

can be solved via an iterative update of the form

$$\boldsymbol{h}_{k+1} = \boldsymbol{h}_k + \alpha_k \boldsymbol{r}_k,$$

where

$$\boldsymbol{r}_k = \underline{\boldsymbol{A}}_M^{\mathrm{T}}\underline{\boldsymbol{y}}_M - \underline{\boldsymbol{A}}_M^{\mathrm{T}}\underline{\boldsymbol{A}}_M\boldsymbol{h}_k.$$

Note that we can also write

$$\boldsymbol{r}_k = \sum_{n=0}^{M}(y[n] - \boldsymbol{u}_n^{\mathrm{T}}\boldsymbol{h}_k)\boldsymbol{u}_n \tag{4}$$

Now, note that the most costly aspects of this algorithm are:

1. Computing the sum over all M terms in the gradient in (4), and
2. Running this algorithm to convergence (requiring many iterations) each time a new measurement $y[n]$ arrives.

The LMS filter does away with both of these restrictions by taking the following (seemingly crazy!) strategy: when a new measurement $y[n]$ arrives, take only a single gradient step, and ignore every term in (4) except the one corresponding to the most recent measurement! To be concrete, given $y[n]$ and an existing estimate $\boldsymbol{h}_n$, we produce an update via the iteration:

$$\boldsymbol{h}_{n+1} = \boldsymbol{h}_n + \alpha \left(y[n] - \boldsymbol{u}_n^T \boldsymbol{h}_n \right) \boldsymbol{u}_n.$$

Note that $\boldsymbol{u}_n^T \boldsymbol{h}_n$ is the output of your candidate filter whose taps are the current weights $\boldsymbol{h}_n$. We have also fixed the stepsize.

```
LMS Filter
  Initialize: $\boldsymbol{h}_0 = \mathbf{0}$
  for $n = 1, 2, 3, \ldots$ do
      (observe input $u[n]$ and output $y[n]$)
      $\boldsymbol{u}_n = \begin{bmatrix} u[n] & u[n-1] & \cdots & u[n-N+1] \end{bmatrix}^{\mathrm{T}}$
      $\boldsymbol{h}_n = \boldsymbol{h}_{n-1} + \mu \left( y[n] - \boldsymbol{u}_n^T \boldsymbol{h}_{n-1} \right) \boldsymbol{u}_n$
  end for
```

It can be shown that the LMS algorithm converges when the stepsize is sufficiently small (depending on the statistics of the signal $u[n]$).There are also many results that give the speed of convergence under various assumptions on the true $\boldsymbol{h}_*$ and $u[n]$. In general, these results say that to get within ϵ relative error, we need either $\sim 1/\epsilon$ or $\sim 1/\epsilon^2$ iterations (depending on the assumptions). Notice that this is dramatically worse than the $\sim \log(1/\epsilon)$ required for steepest descent.[2] But also notice that the iterations are much cheaper. Sim-

[2] Suppose $\epsilon = 10^{-3}$. What are $\log(1/\epsilon)$, $1/\epsilon$, $1/\epsilon^2$?

ilarly, the RLS algorithm tends to converge much more quickly than LMS, but each iteration is much more computationally expensive.

There is now a rich literature on the generalization of LMS where you implement steepest descent by, at each iteration, randomly selecting only a few of the terms in the sum that computes the gradient. This is now known as stochastic gradient descent (stochastic because the terms are typically chosen at random) and is a big part of the success of modern machine learning algorithms. (Computing the gradient in many algorithms involves taking a sum over every item in your dataset – if you start working with datasets of millions of images, speeding this up becomes critical!)

Adaptive Filtering

In many practical settings, the impulse response may gradually (or not so gradually!) drift over time. Both of the methods we have described above can handle this setting as well.

First, notice that the core update in the LMS filter depends only on the current output value and the last N input samples. This means that by approximating the gradient with only the last term, LMS naturally adapts to changing filter coefficients. The convergence results mentioned above can be very easily translated into characterizations of how closely we can track dynamic $\boldsymbol{h}_*$.

To make the RLS filter more agile to dynamic $\boldsymbol{h}_*$, we must find a similar way to slowly forget old inputs. This can be achieved by considering a regularized weighted least-squares problem. At time n, we can choose $\boldsymbol{h}_n$ to solves

$$\underset{\boldsymbol{h}}{\text{minimize}} \;\; \sum_{m=0}^{n} \lambda^m \left|y[m] - \boldsymbol{u}_m^{\mathrm{T}}\boldsymbol{h}\right|^2 + \delta\lambda^n \|\boldsymbol{h}\|_2^2.$$

This is a Tikhonov-regularized least-squares problem, where the regularization parameter is getting smaller as n increases. We can again apply the matrix inversion lemma to obtain the following algorithm:

```
RLS Adaptive Filter
  Initialize: $\boldsymbol{h}_0 = \mathbf{0}$,  $\boldsymbol{P}_0 = \delta^{-1}\mathbf{I}$.
  for $n = 1, 2, 3, \ldots$ do
    (observe input $u[n]$ and output $y[n]$)
    $\boldsymbol{u}_n = \begin{bmatrix} u[n] & u[n-1] & \cdots & u[n-N+1] \end{bmatrix}^{\mathrm{T}}$
    $\boldsymbol{v}_n = \boldsymbol{P}_{n-1}\boldsymbol{u}_n$
    $\boldsymbol{k}_n = \frac{1}{\lambda + \boldsymbol{u}_n^{\mathrm{T}}\boldsymbol{v}_n}\boldsymbol{v}_n$
    $\boldsymbol{h}_n = \boldsymbol{h}_{n-1} + (y[n] - \boldsymbol{u}_n^{\mathrm{T}}\boldsymbol{h}_{n-1})\boldsymbol{k}_n$
    $\boldsymbol{P}_n = \lambda^{-1}\boldsymbol{P}_{n-1} - \lambda^{-1}\boldsymbol{k}_n\boldsymbol{v}_n^{\mathrm{T}}\boldsymbol{P}_{n-1}$
  end for
```

Technical Details: Kalman Filter Updates

For the first stage of the update, the "predict stage", the system of equations moves from $\underline{\boldsymbol{A}}_k$ in (1) to

$$\widetilde{\underline{\boldsymbol{A}}}_{k+1} = \begin{bmatrix} \boldsymbol{A}_0 & & & & & \\ \boldsymbol{F}_0 & -\mathbf{I} & & & & \\ & \boldsymbol{A}_1 & & & & \\ & \boldsymbol{F}_1 & & & & \\ & & \ddots & & & \\ & & & \boldsymbol{F}_{k-1} & -\mathbf{I} & \\ & & & & \boldsymbol{A}_k & \mathbf{0} \\ & & & & \boldsymbol{F}_k & -\mathbf{I} \end{bmatrix}, \quad \widetilde{\underline{\boldsymbol{y}}}_{k+1} = \begin{bmatrix} \boldsymbol{y}_0 \\ \mathbf{0} \\ \boldsymbol{y}_1 \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \boldsymbol{y}_k \\ \mathbf{0} \end{bmatrix}$$

The least-squares solution to this system will be unchanged, except for an additional term in the $k+1$st block, which we call $\widehat{\boldsymbol{x}}_{k+1|k}$.

With our previous factorization in hand, we have

$$\boldsymbol{P}_{k|k} := \boldsymbol{Q}_k^{-1}$$

computed already. The addition of the $\boldsymbol{F}_k$ above changes the $\boldsymbol{D}_k$ in our block tridiagonal system to

$$\widetilde{\boldsymbol{D}}_k = \boldsymbol{D}_k + \boldsymbol{F}_k^{\mathrm{T}}\boldsymbol{F}_k,$$

and so the factorization is updated with

$$\begin{aligned} \widetilde{\boldsymbol{Q}}_k &= \boldsymbol{Q}_k + \boldsymbol{F}_k^{\mathrm{T}}\boldsymbol{F}_k \\ \widetilde{\boldsymbol{w}}_k &= \widetilde{\boldsymbol{Q}}_k^{-1}(\boldsymbol{b}_k + \boldsymbol{F}_{k-1}\boldsymbol{w}_{k-1}). \end{aligned}$$

This new system also has $k+1$ blocks, so we need to compute the new terms in our running factorization: $\boldsymbol{U}_k$, $\widetilde{\boldsymbol{Q}}_{k+1}$, and $\widetilde{\boldsymbol{w}}_{k+1}$.

By the matrix inversion lemma

$$\widetilde{\boldsymbol{Q}}_k^{-1} = \boldsymbol{Q}_k^{-1} - \boldsymbol{Q}_k^{-1}\boldsymbol{F}_k^{\mathrm{T}}(\mathbf{I} + \boldsymbol{F}_k\boldsymbol{Q}_k^{-1}\boldsymbol{F}_k^{\mathrm{T}})^{-1}\boldsymbol{F}_k\boldsymbol{Q}_k^{-1},$$

so we can rewrite $\widetilde{\boldsymbol{w}}_k$ as

$$\widetilde{\boldsymbol{w}}_k = \boldsymbol{w}_k - \boldsymbol{Q}_k^{-1}\boldsymbol{F}_k^{\mathrm{T}}(\mathbf{I} + \boldsymbol{F}_k\boldsymbol{Q}_k^{-1}\boldsymbol{F}_k^{\mathrm{T}})^{-1}\boldsymbol{F}_k\boldsymbol{w}_k.$$

We now have $\boldsymbol{C}_k = -\boldsymbol{F}_k$ and $\widetilde{\boldsymbol{D}}_{k+1} = \mathbf{I}$, and $\widetilde{\boldsymbol{b}}_{k+1} = \mathbf{0}$, so we move forward with the running factorization by solving

$$\begin{aligned}
\boldsymbol{U}_k &= -\widetilde{\boldsymbol{Q}}_k^{-1}\boldsymbol{F}_k^{\mathrm{T}} \\
\widetilde{\boldsymbol{Q}}_{k+1} &= \mathbf{I} - \boldsymbol{F}_k\widetilde{\boldsymbol{Q}}_k^{-1}\boldsymbol{F}_k^{\mathrm{T}} \\
\widetilde{\boldsymbol{w}}_{k+1} &= \widetilde{\boldsymbol{Q}}_{k+1}^{-1}\boldsymbol{F}_k\widetilde{\boldsymbol{w}}_k \\
&= \widetilde{\boldsymbol{Q}}_{k+1}^{-1}(\mathbf{I} - \boldsymbol{F}_k\boldsymbol{Q}_k^{-1}\boldsymbol{F}_k^{\mathrm{T}}(\mathbf{I} + \boldsymbol{F}_k\boldsymbol{Q}_k^{-1}\boldsymbol{F}_k^{\mathrm{T}})^{-1})\boldsymbol{F}_k\boldsymbol{w}_k \\
&= \widetilde{\boldsymbol{Q}}_{k+1}^{-1}(\mathbf{I} + \boldsymbol{F}_k\boldsymbol{Q}_k^{-1}\boldsymbol{F}_k^{\mathrm{T}})^{-1}\boldsymbol{F}_k\boldsymbol{w}_k \qquad (5) \\
&= \boldsymbol{F}_k\boldsymbol{w}_k \qquad (6)
\end{aligned}$$

where the second to last step (5) follows from the identity for symmetric matrices

$$\mathbf{I} - \boldsymbol{A}(\mathbf{I} + \boldsymbol{A})^{-1} = (\mathbf{I} + \boldsymbol{A})^{-1}, \qquad (7)$$

(just multiply both sides on the right by $(\mathbf{I} + \boldsymbol{A})$), and the last step (6) follows from an application of the matrix inversion lemma

$$\begin{aligned}
(\mathbf{I} + \boldsymbol{F}_k\boldsymbol{Q}_k^{-1}\boldsymbol{F}_k^{\mathrm{T}})^{-1} &= \mathbf{I} - \boldsymbol{F}_k(\boldsymbol{Q}_k + \boldsymbol{F}_k^{\mathrm{T}}\boldsymbol{F}_k)^{-1}\boldsymbol{F}_k^{\mathrm{T}} \\
&= \mathbf{I} - \boldsymbol{F}_k\widetilde{\boldsymbol{Q}}_k^{-1}\boldsymbol{F}_k \\
&= \widetilde{\boldsymbol{Q}}_{k+1}.
\end{aligned}$$

Since $\widetilde{\boldsymbol{w}}_{k+1} = \widehat{\boldsymbol{x}}_{k+1|k}$ and $\boldsymbol{w}_k = \widehat{\boldsymbol{x}}_{k|k}$, we can rewrite the conclusion above as

$$\widehat{\boldsymbol{x}}_{k+1|k} = \boldsymbol{F}_k \widehat{\boldsymbol{x}}_{k|k}.$$

We will also denote

$$\begin{aligned}\boldsymbol{P}_{k+1|k} := \widetilde{\boldsymbol{Q}}_{k+1}^{-1} &= \mathbf{I} + \boldsymbol{F}_k \boldsymbol{Q}_k^{-1} \boldsymbol{F}_k \\ &= \mathbf{I} + \boldsymbol{F}_k \boldsymbol{P}_{k|k} \boldsymbol{F}_k.\end{aligned}$$

We now go to the second stage of the update, where we add the measurements $\boldsymbol{y}_{k+1} = \boldsymbol{A}_{k+1}\boldsymbol{x}_{x+1} + \boldsymbol{e}_{k+1}$. We add a block row to our system of equations; we now want to solve

$$\underline{\boldsymbol{A}}_{k+1} = \begin{bmatrix} \boldsymbol{A}_0 & & & & & \\ \boldsymbol{F}_0 & -\mathbf{I} & & & & \\ & \boldsymbol{A}_1 & & & & \\ & \boldsymbol{F}_1 & & & & \\ & & \ddots & & & \\ & & & \boldsymbol{F}_{k-1} & -\mathbf{I} & \\ & & & & \boldsymbol{A}_k & \mathbf{0} \\ & & & & \boldsymbol{F}_k & -\mathbf{I} \\ & & & & \mathbf{0} & \boldsymbol{A}_{k+1} \end{bmatrix}, \quad \underline{\boldsymbol{y}}_{k+1} = \begin{bmatrix} \boldsymbol{y}_0 \\ \mathbf{0} \\ \boldsymbol{y}_1 \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \boldsymbol{y}_k \\ \mathbf{0} \\ \boldsymbol{y}_{k+1} \end{bmatrix}$$

This means that we are replacing $\widetilde{\boldsymbol{D}}_{k+1} = \mathbf{I}$ with $\boldsymbol{D}_{k+1} = \mathbf{I} + \boldsymbol{A}_{k+1}^{\mathrm{T}}\boldsymbol{A}_{k+1}$ and $\widetilde{\boldsymbol{b}}_{k+1} = \mathbf{0}$ with $\boldsymbol{b}_{k+1} = \boldsymbol{A}_{k+1}^{\mathrm{T}}\boldsymbol{y}_{k+1}$. We update the factor-and-solve as follows:

$$\begin{aligned}\boldsymbol{D}_{k+1} &= \mathbf{I} + \boldsymbol{A}_{k+1}^{\mathrm{T}}\boldsymbol{A}_{k+1} \\ \boldsymbol{Q}_{k+1} &= \widetilde{\boldsymbol{Q}}_{k+1} + \boldsymbol{A}_{k+1}^{\mathrm{T}}\boldsymbol{A}_{k+1} \\ \boldsymbol{w}_{k+1} &= \boldsymbol{Q}_{k+1}^{-1}(\boldsymbol{A}_{k+1}^{\mathrm{T}}\boldsymbol{y}_{k+1} + \boldsymbol{F}_k\widetilde{\boldsymbol{w}}_k). \end{aligned} \tag{8}$$

Using the matrix-inversion lemma, we can write

$$\boldsymbol{Q}_{k+1}^{-1} = \widetilde{\boldsymbol{Q}}_{k+1}^{-1} - \widetilde{\boldsymbol{Q}}_{k+1}^{-1}\boldsymbol{A}_{k+1}^{\mathrm{T}}(\mathbf{I} + \boldsymbol{A}_{k+1}\widetilde{\boldsymbol{Q}}_{k+1}^{-1}\boldsymbol{A}_{k+1}^{\mathrm{T}})^{-1}\boldsymbol{A}_{k+1}\widetilde{\boldsymbol{Q}}_{k+1}^{-1}, \tag{9}$$

and so using the fact from above that $\widetilde{\boldsymbol{w}}_{k+1} = \widetilde{\boldsymbol{Q}}_{k+1}^{-1}\boldsymbol{F}_k\widetilde{\boldsymbol{w}}_k = \widetilde{\boldsymbol{Q}}_{k+1}^{-1}\widehat{\boldsymbol{x}}_{k+1|k}$, the two terms in the expression for $\boldsymbol{w}_{k+1} = \widehat{\boldsymbol{x}}_{k+1|k+1}$ becomes

$$\boldsymbol{Q}_{k+1}^{-1}\boldsymbol{F}_k\widetilde{\boldsymbol{w}}_k = \widehat{\boldsymbol{x}}_{k+1|k} - \widetilde{\boldsymbol{Q}}_{k+1}^{-1}\boldsymbol{A}_{k+1}^{\mathrm{T}}(\mathbf{I} + \boldsymbol{A}_{k+1}\widetilde{\boldsymbol{Q}}_{k+1}^{-1}\boldsymbol{A}_{k+1}^{\mathrm{T}})^{-1}\boldsymbol{A}_{k+1}\widehat{\boldsymbol{x}}_{k+1|k} \tag{10}$$

and

$$\begin{aligned}\boldsymbol{Q}_{k+1}^{-1}\boldsymbol{A}_{k+1}^{\mathrm{T}}\boldsymbol{y}_{k+1} &= \\ &\widetilde{\boldsymbol{Q}}_{k+1}^{-1}\boldsymbol{A}_{k+1}^{\mathrm{T}}\left(\mathbf{I} - (\mathbf{I} + \boldsymbol{A}_{k+1}\widetilde{\boldsymbol{Q}}_{k+1}^{-1}\boldsymbol{A}_{k+1}^{\mathrm{T}})^{-1}\boldsymbol{A}_{k+1}\widetilde{\boldsymbol{Q}}_{k+1}^{-1}\boldsymbol{A}_{k+1}^{\mathrm{T}}\right)\boldsymbol{y}_{k+1} \\ &= \widetilde{\boldsymbol{Q}}_{k+1}^{-1}\boldsymbol{A}_{k+1}^{\mathrm{T}}\left(\mathbf{I} + \boldsymbol{A}_{k+1}\widetilde{\boldsymbol{Q}}_{k+1}^{-1}\boldsymbol{A}_{k+1}^{\mathrm{T}}\right)^{-1}\boldsymbol{y}_{k+1}, \end{aligned} \tag{11}$$

where we have again used the identity (7). Combining (10) and (11) with (8), we have

$$\boldsymbol{w}_{k+1} = \widehat{\boldsymbol{x}}_{k+1|k+1} = \widehat{\boldsymbol{x}}_{k+1|k} + \boldsymbol{G}_{k+1}\left(\boldsymbol{y}_{k+1} - \boldsymbol{A}_{k+1}\widehat{\boldsymbol{x}}_{k+1|k}\right),$$

where

$$\begin{aligned}\boldsymbol{G}_{k+1} &= \widetilde{\boldsymbol{Q}}_{k+1}^{-1}\boldsymbol{A}_{k+1}^{\mathrm{T}}\left(\mathbf{I} + \boldsymbol{A}_{k+1}\widetilde{\boldsymbol{Q}}_{k+1}^{-1}\boldsymbol{A}_{k+1}^{\mathrm{T}}\right)^{-1} \\ &= \boldsymbol{P}_{k+1|k}\boldsymbol{A}_{k+1}^{\mathrm{T}}\left(\mathbf{I} + \boldsymbol{A}_{k+1}\boldsymbol{P}_{k+1|k}\boldsymbol{A}_{k+1}^{\mathrm{T}}\right)^{-1}.\end{aligned}$$

Finally, (9) above also gives us the update

$$\boldsymbol{P}_{k+1|k+1} := \boldsymbol{Q}_{k+1}^{-1} = \left(\mathbf{I} - \boldsymbol{G}_{k+1}\boldsymbol{A}_{k+1}\right)\boldsymbol{P}_{k+1|k}.$$