Digital Filtering and Smoothing
A Student simulation Project

Eric Gossett
Bethel College, St. Paul, MN
gossett@bethel.edu

Contents

1 Introduction

This presentation began as a project to introduce upper division students to digital filtering and smoothing. The project consists of an algorithm and a simulation platform upon which to test implementations of the algorithm.

The original simulation platform was a collection of FORTRAN subroutines that provided a foundation upon which the students could build their implementation of the algorithm. The subroutines included object files that produced trajectories for several bugs.

The current simulation platform uses Mathematica 3.0. This platform has the advantages of a more natural notation, easier creation of graphs showing algorithm effectiveness, and the ability to animate the algorithm.

After the problem is introduced, the mathematical derivation of the algorithm will be presented, followed by a brief discussion of the simulation platform and some sample simulation output.

The algorithm development contains a nice blend of heuristic motivation and formal derivation.

1.1 The bug tracking problem

A small computer bug is traveling around the $x - y$ plane trying to avoid detection. We can eliminate the bug if we can produce a reasonably accurate approximation to its trajectory.

We have a bug detecting device which can be pointed at the plane. It can measure the x and y distances to the bug from where it is pointing if the bug is not too far from where it is pointing. These distances are called the *pointing errors.*

The bug does not want to be tracked, so it sends out jamming signals which somewhat corrupt the bug detecting device's measurements in a random manner. The bug can also choose to change its trajectory in unexpected ways.

Our task is to create a computer algorithm that uses the bug detecting device to:

- Screen out the measurement noise
- Point at the bug's most likely next position in the plane
- Produce an accurate record of the bug's path.

1.2 Some details

- The bug detecting device can point at a new position and return a new pair of pointing errors every .1 second.
- The bug detecting device cannot find the bug if the bug is more than 5 units away in either the x or the y direction. In this case, it will return unspecified (probably large) values for the pointing errors.

1.3 Algorithm preview

The algorithm has conflicting demands placed upon it:

- It needs to screen out the noise inherent in the position error measurements. This implies placing more reliance on an internal modal of the bug's trajectory and less reliance on the noisy position error estimates.
- It needs to quickly respond to changes in the bug's trajectory. This implies a sensitivity to the measurements and less emphasis on the internal trajectory model.

The algorithm will be a fading memory polynomial filter. It will eventually have the ability to adapt its bandwidth (noise rejecting ability) to continually provide a good mix between noise rejection and responsiveness to changes in the bug's direction. As long as the bug is going along some smooth trajectory, the algorithm should pay less attention to the measured pointing errors (since they are not terribly reliable) and place more emphasis on the recent history of the bug's motion. However, when the bug makes a change in trajectory, the algorithm needs to

give more emphasis to the measured position errors and less emphasis to the recent history. The variable bandwidth filter will address this issue.

The algorithm will be in predictor-corrector form. The predictor phase will estimate the next bug position, the bug detecting device will look at that position in the plane and return its estimate of the position errors. The corrector phase will then use these measurements to make a more accurate estimate of the current position. The cycle will then repeat.

2 Mathematical derivation of the algorithm

2.1 Notation and additional details

The location can be written in parametric form:

At time t, the bug's true position is $(x(t), y(t))$. The predicted position at time t will be denoted $(x_p(t), y_p(t))$, the corrected position by $(\overline{x}(t), \overline{y}(t))$.

We will usually drop the explicit dependence on time t. Since the corrected estimates are produced as a discrete sequence, we often indicate the time by using an integer subscript: (x_n, y_n) and $(\overline{x}_n, \overline{y}_n)$.

The pointing errors will be denoted by $(\triangle x, \triangle y)$. They consist of the difference between the actual and predicted positions, plus or minus a noise (or error) term: $\triangle x = x_n - x_p \pm e_x$ and $\triangle y = y_n - y_p \pm e_y$.

We will actually maintain estimates for the first and second derivatives as well as the position estimates.

An alternate notation for the first and second derivatives is:

$$\begin{aligned} \dot{x}(t) &\equiv x'(t) \\ \ddot{x}(t) &\equiv x''(t) \end{aligned}$$

The vectors $(x(t), \dot{x}(t), \ddot{x}(t))$ and $(y(t), \dot{y}(t), \ddot{y}(t))$ are called *state vectors.*

The prediction phase

Applying Taylor's Theorem to the functions $x(t), y(t), \dot{x}(t), \dot{y}(t), \ddot{x}(t), \ddot{y}(t)$ produces:

$$x(t) = x(t_0) + \dot{x}(t_0)h + \ddot{x}(t_0)\frac{h^2}{2} + \mathbb{O}(h^3) \quad (1)$$

$$y(t) = y(t_0) + \dot{y}(t_0)h + \ddot{y}(t_0)\frac{h^2}{2} + \mathbb{O}(h^3) \quad (2)$$

$$\dot{x}(t) = \dot{x}(t_0) + \ddot{x}(t_0)h + \mathbb{O}(h^2) \quad (3)$$

$$\dot{y}(t) = \dot{y}(t_0) + \ddot{y}(t_0)h + \mathbb{O}(h^2) \quad (4)$$

$$\ddot{x}(t) = \ddot{x}(t_0) + \mathbb{O}(h) \quad (5)$$

$$\ddot{y}(t) = \ddot{y}(t_0) + \mathbb{O}(h) \quad (6)$$

where $h = t - t_0$.

By dropping the $\mathbb{O}(h^k)$ terms, we obtain the approximations that will be used as the predicted state vectors. The subscript p is a static subscript (unlike the subscript n on the corrected state vectors).

Notice the implicit assumption that accelerations are constant.

$$x_p \simeq \overline{x}_{n-1} + h\dot{\overline{x}}_{n-1} + \frac{h^2}{2}\ddot{\overline{x}}_{n-1} \tag{7}$$

$$y_p \simeq \overline{y}_{n-1} + h\dot{\overline{y}}_{n-1} + \frac{h^2}{2}\ddot{\overline{y}}_{n-1} \tag{8}$$

$$\dot{x}_p \simeq \dot{\overline{x}}_{n-1} + h\ddot{\overline{x}}_{n-1} \tag{9}$$

$$\dot{y}_p \simeq \dot{\overline{y}}_{n-1} + h\ddot{\overline{y}}_{n-1} \tag{10}$$

$$\ddot{x}_p \simeq \ddot{\overline{x}}_{n-1} \tag{11}$$

$$\ddot{y}_p \simeq \ddot{\overline{y}}_{n-1} \tag{12}$$

The correction phase

The corrected state vectors add the corresponding predicted state component to a scaled term involving the position error. The scaling variables α_n, β_n, and γ_n are small when we want to discount the measured position errors and larger when we want to give more credence to the position errors. They are artifacts of the mechanism used to determine $\overline{x}_n$, $\dot{\overline{x}}_n$, and $\ddot{\overline{x}}_n$.

$$\overline{x}_n = x_p + \alpha_n \triangle x \tag{13}$$

$$\overline{y}_n = y_p + \alpha_n \triangle y \tag{14}$$

$$\dot{\overline{x}}_n = \dot{x}_p + \beta_n \frac{\triangle x}{h} \tag{15}$$

$$\dot{\overline{y}}_n = \dot{y}_p + \beta_n \frac{\triangle y}{h} \tag{16}$$

$$\ddot{\overline{x}}_n = \ddot{x}_p + 2\gamma_n \frac{\triangle x}{h^2} \tag{17}$$

$$\ddot{\overline{y}}_n = \ddot{y}_p + 2\gamma_n \frac{\triangle y}{h^2} \tag{18}$$

Fading memory

How should $\overline{x}_n$, $\dot{\overline{x}}_n$, and $\ddot{\overline{x}}_n$ be chosen? One reasonable approach is to choose them so that the trajectory they determine closely approximates the true past trajectory. It also seems reasonable that recent data points should be weighted more heavily than older data points.

In order to more precisely define "closely approximates", notice that if we have estimates for the state vector at time n, we can use equation (7) (with $t - t_0 = -ih$) to retroactively estimate the position at time $n - i$ as:

$$x_{n-i} \simeq \overline{x}_n - ih\dot{\overline{x}}_n + \frac{i^2h^2}{2}\ddot{\overline{x}}_n$$

The difference between the position estimated by the state vector $\left(\overline{x}_n, \dot{\overline{x}}_n, \ddot{\overline{x}}_n\right)$ and the true position x_{n-i} at a time i increments in the past would then be:

$$\left(\overline{x}_n - ih\dot{\overline{x}}_n + \frac{i^2h^2}{2}\ddot{\overline{x}}_n\right) - x_{n-i}$$

Our goal is to choose $\overline{x}_n, \dot{\overline{x}}_n$, and $\ddot{\overline{x}}_n$ so that the following weighted sum is minimized in a least squares sense.

$$R = \sum_{i=0}^{n} B^i [\left(\overline{x}_n - ih\dot{\overline{x}}_n + \frac{i^2h^2}{2}\ddot{\overline{x}}_n\right) - x_{n-i}]^2 \tag{19}$$

for $0 < B < 1$.

Notice that the weighting factor B places more weight on recent positions and less weight on older positions. We assume that B is constant for this derivation (although it can be experimentally determined once the filter has been coded).

2.2 Derivation of α_n, β_n, and γ_n

We need to find values for $\overline{x}_n, \dot{\overline{x}}_n$, and $\ddot{\overline{x}}_n$ that minimize

$$R = \sum_{i=0}^{n} B^i [\left(\overline{x}_n - ih\dot{\overline{x}}_n + \frac{i^2h^2}{2}\ddot{\overline{x}}_n\right) - x_{n-i}]^2$$

This will eventually be accomplished by making proper choices for α_n, β_n, and γ_n in equations (13) through (18).

The normal equations

Initially, we minimize R by taking partial derivatives with respect to $\overline{x}_n, \dot{\overline{x}}_n$, and $\ddot{\overline{x}}_n$ and setting the partials equal to zero. The values of the variables that solve the resulting system of *normal equations* will minimize[1] R.

The partial derivatives are:

$$\frac{\partial R}{\partial \overline{x}_n} = 2\sum_{i=0}^{n} B^i(\overline{x}_n - ih\dot{\overline{x}}_n + \frac{1}{2}i^2h^2\ddot{\overline{x}}_n - x_{n-i})$$

$$\frac{\partial R}{\partial \dot{\overline{x}}_n} = -2h\sum_{i=0}^{n} iB^i(\overline{x}_n - ih\dot{\overline{x}}_n + \frac{1}{2}i^2h^2\ddot{\overline{x}}_n - x_{n-i})$$

$$\frac{\partial R}{\partial \ddot{\overline{x}}_n} = h^2\sum_{i=0}^{n} i^2B^i(\overline{x}_n - ih\dot{\overline{x}}_n + \frac{1}{2}i^2h^2\ddot{\overline{x}}_n - x_{n-i})$$

The normal equations that result are:

$$\overline{x}_n\sum_{i=0}^{n} B^i - h\dot{\overline{x}}_n\sum_{i=0}^{n} iB^i + \frac{1}{2}h^2\ddot{\overline{x}}_n\sum_{i=0}^{n} i^2B^i = \sum_{i=0}^{n} B^i x_{n-i} \tag{20}$$

$$\overline{x}_n\sum_{i=0}^{n} iB^i - h\dot{\overline{x}}_n\sum_{i=0}^{n} i^2B^i + \frac{1}{2}h^2\ddot{\overline{x}}_n\sum_{i=0}^{n} i^3B^i = \sum_{i=0}^{n} iB^i x_{n-i} \tag{21}$$

$$\overline{x}_n\sum_{i=0}^{n} i^2B^i - h\dot{\overline{x}}_n\sum_{i=0}^{n} i^3B^i + \frac{1}{2}h^2\ddot{\overline{x}}_n\sum_{i=0}^{n} i^4B^i = \sum_{i=0}^{n} i^2B^i x_{n-i} \tag{22}$$

The form of the normal equations should look familiar to those who have previously seen least squares derivations for discrete data sets. The term B^i in the summations is new.

[1] Second order partials are necessary to confirm that the solution is actually a minimum.

It is possible to use standard linear algebra techniques to solve this system for $\overline{x}_n, \dot{\overline{x}}_n$, and $\ddot{\overline{x}}_n$. However, those techniques are inappropriate in this context because we need to solve the system in real-time; there will be another system to solve in less than .1 second.

What is needed is some way to express the solution to the normal equations at time n in terms of the solution at time $n-1$. This recursive form will avoid the usual disadvantages of recursive algorithms because the previous case will already have been solved.

The weight summations

The five summations defined next play an important role in the recursive form of the filter.

$$S_0(n) = \sum_{i=0}^{n} B^i$$

$$S_1(n) = \sum_{i=0}^{n} iB^i$$

$$S_2(n) = \sum_{i=0}^{n} i^2B^i$$

$$S_3(n) = \sum_{i=0}^{n} i^3B^i$$

$$S_4(n) = \sum_{i=0}^{n} i^4B^i$$

Because we wish to find a recursive solution to the normal equations, it will be useful to express the summations at time n in terms of the summations at time $n-1$, and vice versa.

Using a simple change in the summation index, it can be shown that the summations $S_i(n)$ can be written in terms of the summations $S_i(n-1)$:

$$\sum_{i=0}^{n} B^i = 1 + B\sum_{i=0}^{n-1} B^i \tag{23}$$

$$\sum_{i=0}^{n} iB^i = B\left(\sum_{i=0}^{n-1} iB^i + \sum_{i=0}^{n-1} B^i\right) \tag{24}$$

$$\sum_{i=0}^{n} i^2B^i = B\left(\sum_{i=0}^{n-1} i^2B^i + 2\sum_{i=0}^{n-1} iB^i + \sum_{i=0}^{n-1} B^i\right) \tag{25}$$

$$\sum_{i=0}^{n} i^3B^i = B\left(\sum_{i=0}^{n-1} i^3B^i + 3\sum_{i=0}^{n-1} i^2B^i + 3\sum_{i=0}^{n-1} iB^i + \sum_{i=0}^{n-1} B^i\right) \tag{26}$$

$$\sum_{i=0}^{n} i^4B^i = B\left(\sum_{i=0}^{n-1} i^4B^i + 4\sum_{i=0}^{n-1} i^3B^i + 6\sum_{i=0}^{n-1} i^2B^i + 4\sum_{i=0}^{n-1} iB^i + \sum_{i=0}^{n-1} B^i\right) \tag{27}$$

This can be expressed in a more succinct manner as:

$$S_0(n) = 1 + BS_0(n-1) \tag{28}$$

$$S_1(n) = B[S_1(n-1)+S_0(n-1)] \tag{29}$$

$$S_2(n) = B[S_2(n-1)+2S_1(n-1)+S_0(n-1)] \tag{30}$$

$$S_3(n) = B[S_3(n-1)+3S_2(n-1)+3S_1(n-1)+S_0(n-1)] \tag{31}$$

$$S_4(n) = B[S_4(n-1)+4S_3(n-1)+6S_2(n-1)+4S_1(n-1)+S_0(n-1)] \tag{32}$$

It is also not hard to show that

$$S_0(n-1) = \frac{1}{B}[S_0(n)-1] \tag{33}$$

$$S_1(n-1) = \frac{1}{B}[S_1(n)-S_0(n)+1] \tag{34}$$

$$S_2(n-1) = \frac{1}{B}[S_2(n)-2S_1(n)+S_0(n)-1] \tag{35}$$

$$S_3(n-1) = \frac{1}{B}[S_3(n)-3S_2(n)+3S_1(n)-S_0(n)+1] \tag{36}$$

$$S_4(n-1) = \frac{1}{B}[S_4(n)-4S_3(n)+6S_2(n)-4S_1(n)+S_0(n)-1] \tag{37}$$

A recursive solution of the normal equations

The previous equations show how to express $S_i(n)$ in terms of the summations $S_i(n-1)$. This enables us to express the left-hand sides of the normal equations in a recursive manner. It will also be necessary to express the right-hand sides recursively.

To that end, notice that the right-hand sides of the normal equations at iteration n can be written in terms of the right-hand sides of the normal equations at iteration $n-1$:

$$\sum_{i=0}^{n} B^i x_{n-i} = x_n + B\sum_{i=0}^{n-1} B^i x_{n-i-1} \tag{38}$$

$$\sum_{i=0}^{n} iB^i x_{n-i} = B\sum_{i=0}^{n-1} iB^i x_{n-i-1} + B\sum_{i=0}^{n-1} B^i x_{n-i-1} \tag{39}$$

$$\sum_{i=0}^{n} i^2 B^i x_{n-i} = B\sum_{i=0}^{n-1} i^2 B^i x_{n-i-1} + 2B\sum_{i=0}^{n-1} iB^i x_{n-i-1} + B\sum_{i=0}^{n-1} B^i x_{n-i-1} \tag{40}$$

The terms $\sum_{i=0}^{n-1} i^k B^i x_{n-i-1}$ on the right-hand sides of equations (38) through (40) are the right-hand sides of the normal equations at iteration $n-1$.They can therefore be expressed in terms of the left-hand sides of those normal equations:

$$\sum_{i=0}^{n-1} B^i x_{n-i} = \overline{x}_{n-1} \sum_{i=0}^{n-1} B^i - h\dot{\overline{x}}_{n-1} \sum_{i=0}^{n-1} iB^i + \frac{1}{2}h^2\ddot{\overline{x}}_{n-1} \sum_{i=0}^{n-1} i^2 B^i \tag{41}$$

$$\sum_{i=0}^{n-1} iB^i x_{n-i} = \overline{x}_{n-1} \sum_{i=0}^{n-1} iB^i - h\dot{\overline{x}}_{n-1} \sum_{i=0}^{n-1} i^2 B^i + \frac{1}{2}h^2\ddot{\overline{x}}_{n-1} \sum_{i=0}^{n-1} i^3 B^i \tag{42}$$

$$\sum_{i=0}^{n-1} i^2 B^i x_{n-i} = \overline{x}_{n-1} \sum_{i=0}^{n-1} i^2 B^i - h\dot{\overline{x}}_{n-1} \sum_{i=0}^{n-1} i^3 B^i + \frac{1}{2}h^2\ddot{\overline{x}}_{n-1} \sum_{i=0}^{n-1} i^4 B^i \tag{43}$$

Substituting equations (41) - (43) into equations (38) - (40) and then substituting the resulting equations into the normal equations and finally using equations (33) - (37) to replace summations $S_i(n-1)$ with summations $S_i(n)$, we can express the normal equations as:

$$\begin{aligned}&\overline{x}_n S_0(n) - h\dot{\overline{x}}_n S_1(n) + \frac{1}{2}h^2\ddot{\overline{x}}_n S_2(n)\\ &= x_n + \overline{x}_{n-1}(S_0(n)-1) - h\dot{\overline{x}}_{n-1}(S_1(n)-S_0(n)+1) + \frac{1}{2}h^2\ddot{\overline{x}}_{n-1}(S_2(n)-2S_1(n)+S_0(n)-1)\end{aligned} \tag{44}$$

$$\begin{aligned}&\overline{x}_n S_1(n) - h\dot{\overline{x}}_n S_2(n) + \frac{1}{2}h^2\ddot{\overline{x}}_n S_3(n)\\ &= \overline{x}_{n-1} S_1(n) - h\dot{\overline{x}}_{n-1}(S_2(n)-S_1(n)) + \frac{1}{2}h^2\ddot{\overline{x}}_{n-1}(S_3(n)-2S_2(n)+S_1(n))\end{aligned} \tag{45}$$

$$\begin{aligned}&\overline{x}_n S_2(n) - h\dot{\overline{x}}_n S_3(n) + \frac{1}{2}h^2\ddot{\overline{x}}_n S_4(n)\\ &= \overline{x}_{n-1} S_2(n) - h\dot{\overline{x}}_{n-1}(S_3(n)-S_2(n)) + \frac{1}{2}h^2\ddot{\overline{x}}_{n-1}(S_4(n)-2S_3(n)+S_2(n))\end{aligned} \tag{46}$$

The rather messy solution to the linear system (44) - (46) can be manipulated into the form

$$\overline{x}_n = x_p + \alpha_n \triangle x$$

$$\overline{y}_n = y_p + \alpha_n \triangle y$$

$$\dot{\overline{x}}_n = \dot{x}_p + \beta_n \frac{\triangle x}{h}$$

$$\dot{\overline{y}}_n = \dot{y}_p + \beta_n \frac{\triangle y}{h}$$

$$\ddot{\overline{x}}_n = \ddot{x}_p + 2\gamma_n \frac{\triangle x}{h^2}$$

$$\ddot{\overline{y}}_n = \ddot{y}_p + 2\gamma_n \frac{\triangle y}{h^2}$$

where

$$\alpha_n = \frac{C_1}{D} \tag{47}$$

$$\beta_n = \frac{C_2}{D} \tag{48}$$

$$\gamma_n = \frac{C_3}{D} \tag{49}$$

$$C_1 = S_2(n)S_4(n) - S_3(n)S_3(n) \tag{50}$$

$$C_2 = S_1(n)S_4(n) - S_2(n)S_3(n) \tag{51}$$

$$C_3 = S_1(n)S_3(n) - S_2(n)S_2(n) \tag{52}$$

$$D = C_1S_0(n) - C_2S_1(n) + C_3S_2(n) \tag{53}$$

2.3 An adaptive, fading-memory, polynomial filter

The filter presented in equations (7) - (18) and (47) - (53) is a fading-memory polynomial filter. As time proceeds, less and less emphasis is placed on the current measured position error and more emphasis is placed on the predicted trajectory (remember the assumption about constant accelerations). This is often unrealistic (recall that the bug can change its trajectory). In this section, the filter is modified so that it can adapt to non-constant accelerations.

An adaptive filter

The filtering level is determined by the filter variables α_n, β_n, and γ_n. These variables are in turn completely determined by the summations $S_i(n)$. The key idea is to disconnect the summations $S_i(k)$ and the time index n. When k is small, α_k, β_k, and γ_k are relatively large and when k is large, α_k, β_k, and γ_k are relatively small.

A convenient indicator of how well the current predictions match the actual trajectory is the size of the pointing errors $\triangle x$ and $\triangle y$. If these values are fairly small, it seems appropriate to let k continue to grow larger thereby filtering out more of the measurement noise. Thus, at the next iteration, the filter should use α_{k+1}, β_{k+1}, and γ_{k+1}.

If, however, the pointing errors are fairly large, it seems appropriate to use a smaller value for k. This will place more weight on the pointing errors and hopefully pull the predicted trajectory into closer alignment with the actual trajectory. Thus, at the next iteration, the filter should use α_{k-1}, β_{k-1}, and γ_{k-1}. Since these parameters depend only on the values of the summations $S_i(k-1)$, equations (33) - (37) can be used to recursively calculate $S_i(k-1)$ from the current summations $S_i(k)$.

If the pointing errors are extremely large, it seems appropriate to place maximum weight on the measurements and much less weight on the predicted trajectory. In this case, it seems appropriate to reset k to some predetermined initial value n_0. This is necessary in order to not lose the bug completely. However, such a drastic step will cause the predicted velocities and accelerations to become unreliable for many subsequent iterations.

The final filter

The filter starts with $k = n_0$ and initial state vectors $(\overline{x}_0, \dot{\overline{x}}_0, \ddot{\overline{x}}_0)$ and $(\overline{y}_0, \dot{\overline{y}}_0, \ddot{\overline{y}}_0)$.

At each new time increment (of duration h), n is incremented by 1 and the new state is predicted:

$$
\begin{aligned}
x_p &\simeq \overline{x}_{n-1} + h\dot{\overline{x}}_{n-1} + \frac{h^2}{2}\ddot{\overline{x}}_{n-1} \\
y_p &\simeq \overline{y}_{n-1} + h\dot{\overline{y}}_{n-1} + \frac{h^2}{2}\ddot{\overline{y}}_{n-1} \\
\dot{x}_p &\simeq \dot{\overline{x}}_{n-1} + h\ddot{\overline{x}}_{n-1} \\
\dot{y}_p &\simeq \dot{\overline{y}}_{n-1} + h\ddot{\overline{y}}_{n-1} \\
\ddot{x}_p &\simeq \ddot{\overline{x}}_{n-1} \\
\ddot{y}_p &\simeq \ddot{\overline{y}}_{n-1}
\end{aligned}
$$

The bug detecting device is pointed at the predicted position (x_p, y_p) and the position errors Δx and Δy are measured.

If the position errors are small, k is incremented by 1. If the position errors are moderately large, k is decremented by 1. If the position errors are very large, k should be set to n_0 again.

In any case, the corrected state vectors $(\overline{x}_n, \dot{\overline{x}}_n, \ddot{\overline{x}}_n)$ and $(\overline{y}_n, \dot{\overline{y}}_n, \ddot{\overline{y}}_n)$ are calculated using:

$$
\begin{aligned}
\overline{x}_n &= x_p + \alpha_k \Delta x \\
\overline{y}_n &= y_p + \alpha_k \Delta y \\
\dot{\overline{x}}_n &= \dot{x}_p + \beta_k \frac{\Delta x}{h} \\
\dot{\overline{y}}_n &= \dot{y}_p + \beta_k \frac{\Delta y}{h} \\
\ddot{\overline{x}}_n &= \ddot{x}_p + 2\gamma_k \frac{\Delta x}{h^2} \\
\ddot{\overline{y}}_n &= \ddot{y}_p + 2\gamma_k \frac{\Delta y}{h^2}
\end{aligned}
$$

Where

$$
\begin{aligned}
\alpha_k &= \frac{C_1}{D} \\
\beta_k &= \frac{C_2}{D} \\
\gamma_k &= \frac{C_3}{D} \\
C_1 &= S_2(k)S_4(k) - S_3(k)S_3(k) \\
C_2 &= S_1(k)S_4(k) - S_2(k)S_3(k) \\
C_3 &= S_1(k)S_3(k) - S_2(k)S_2(k) \\
D &= C_1S_0(k) - C_2S_1(k) + C_3S_2(k)
\end{aligned}
$$

and

$$
\begin{aligned}
S_0(k) &= 1 + BS_0(k-1) \\
S_1(k) &= B\,[S_1(k-1)+S_0(k-1)] \\
S_2(k) &= B\,[S_2(k-1)+2S_1(k-1)+S_0(k-1)] \\
S_3(k) &= B\,[S_3(k-1)+3S_2(k-1)+3S_1(k-1)+S_0(k-1)] \\
S_4(k) &= B\,[S_4(k-1)+4S_3(k-1)+6S_2(k-1)+4S_1(k-1)+S_0(k-1)]
\end{aligned}
$$

$$
\begin{aligned}
S_0(k-1) &= \frac{1}{B}\,[S_0(k)-1] \\
S_1(k-1) &= \frac{1}{B}\,[S_1(k)-S_0(k)+1] \\
S_2(k-1) &= \frac{1}{B}\,[S_2(k)-2S_1(k)+S_0(k)-1] \\
S_3(k-1) &= \frac{1}{B}\,[S_3(k)-3S_2(k)+3S_1(k)-S_0(k)+1] \\
S_4(k-1) &= \frac{1}{B}\,[S_4(k)-4S_3(k)+6S_2(k)-4S_1(k)+S_0(k)-1]
\end{aligned}
$$

and $S_0(1)= 1 + B$, $S_1(1)= S_2(1)= S_3(1)= S_4(1)= B$.

The filter is now ready to increment n and make the next prediction.

3 The simulation platform

There are two places at which the filter can be tuned to various bug characteristics. They are:

- the value of B (.995 is a good initial value)
- the quantification of the pointing error sizes used to determine whether to use α_{k+1}, α_{k-1}, or α_{n_0}.

A simulation platform enables students to validate their filter implementations and to experiment with these adjustable elements. The simulation platform also enables a comparison of the actual trajectory and the estimated trajectory provided by the corrected state vector.

3.1 Platform components

The simulation platform consists of five Mathematica 3.0 files. Three of the files are Mathematica notebooks (documents/windows in which mathematical manipulations can be processed). The other two files are Mathematica packages (files containing accessory functions that can be used in a notebook without the need to type in their definitions). Brief descriptions of these five files are listed next (packages use the .m extension, notebooks use the .nb extension).

FilterSums.m This package contains functions that perform the calculations represented by equations (28) - (37) and (47) - (53). This package should be written by the students.

BugMotion.m This package (written by the instructor) contains two functions. The first calculates the true bug trajectory at time n. The second function calculates the pointing errors, given the predicted position. The returned value contains the noise component added by the bug. The noise characteristics can be set by the instructor. Examples of these functions will be given in the next section.

BugTracking.nb This is the key component of the simulation platform. This notebook provides the code to run the simulation. The major activities include initialization code, opening and closing data files for recording information about the simulation run, and code to perform the filter loop. The framework might best be done by the instructor, with students providing the filter loop.

BugPlots.nb This notebook provides a series of plots to analyze the effectiveness of the filter in a simulation run. The graphs use the data files produced by BugTracking.nb. The graphs include comparisons of the true values and filtered estimates for all six of the state variables. Also included are residual plots, showing the differences between the true and filtered state variables. The position residuals also include a graph of the noise values added to the pointing errors. Examples will be given in the next section.

BugAnimation.nb This notebook uses the data files produced by BugTracking.nb to create an animation of the bug and the filtered estimate of the bug's positions. The bug is represented by a blue dot, the filtered estimate by a red cross-hair. A series of plots from a succession of time increments are shown flip-book style. A static example will be given in the next section.

One of the features of Mathematica 3.0 that makes this a natural choice for the simulation platform is the ability to use formatted mathematical notation in the notebook (rather than only plain ascii characters). The simulation platform can use the same notation as was used in the mathematical derivation of the filter. This provides for the students a smooth transition from reading about the filter to implementing the filter in Mathematica code.

3.2 Examples

The code and graphs presented next should be sufficient to give a sense of the various components of the simulation platform.

BugMotion.m

The major function definitions have been extracted from BugMotion.m and are listed next. The bug they represent is a very active one. The noise pattern is a uniform random value between $-.25$ and $.25$.

The extracts contain Mathematica specific code. It is not necessary to be familiar with the details; the major steps in the computations should be fairly clear.

```
Location[n_,h_] :=
  Module[{t,x={0,0,0},y={0,0,0}},
     (* Set the time value from the iteration and time increment. *)
      t = n*h;

     (* position *)
     x[[1]] = 5.0 + 5.0*t-5.0*Cos[3.0*t];
     y[[1]] = 40.0 - 20.0*Sin[2.0*t];

     (* velocity *)
     x[[2]] = 5.0 + 15.0*Sin[3.0*t];
     y[[2]] = -40.0*Cos[2.0*t];

     (* acceleration *)
     x[[3]] = 45.0*Cos[3.0*t];
     y[[3]] = 80.0*Sin[2.0*t];

    Return[{x,y,t}]
  ]

PointingErrors[n_,h_,xp_List,yp_List] :=
  Module[{x,y,dx,dy,t,rndx,rndy},
   (* calculate the true position error *)
   {x,y,t} = Location[n,h];
    dx = x[[1]] - xp[[1]];
    dy = y[[1]] - yp[[1]];

    (* If the position error is more than 5 in either component, the bug
       detecting device cannot find the bug and so returns random garbage.
       Otherwise, the bug adds random jamming noise. *)

     If[(Abs[dx] > 5.0) || (Abs[dy] > 5.0),
       (* then *)
        rndx = Random[Real,{0,1000.0}];
        rndy = Random[Real,{0,1000.0}];
        dx = dx/Abs[dx]*rndx;
        dy = dy/Abs[dy]*rndy,
       (* else *)
        rndx = 0.5*(0.5 - Random[]);
        rndy = 0.5*(0.5 - Random[]);
        dx = dx + rndx;
        dy = dy + rndy];

     Write["noise.dat", NumberForm[rndx], " ", NumberForm[rndy], " ",
           NumberForm[dx], " ", NumberForm[dy]];

     Return[{dx,dy}]
  ]
```

BugTracking.nb

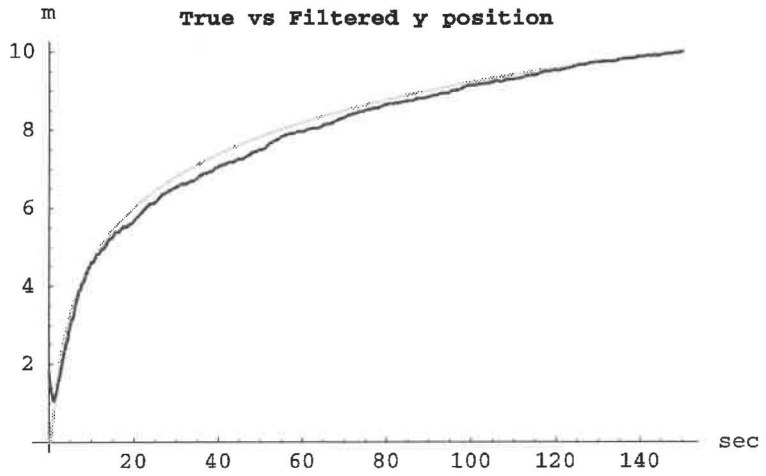
Except for numerous initialization activities, BugTracking.nb is a straight-forward implementation of the filter (with lots of information written to disk at each iteration). The statement in the main filter loop that calculates the pointing errors at time period i might look like:

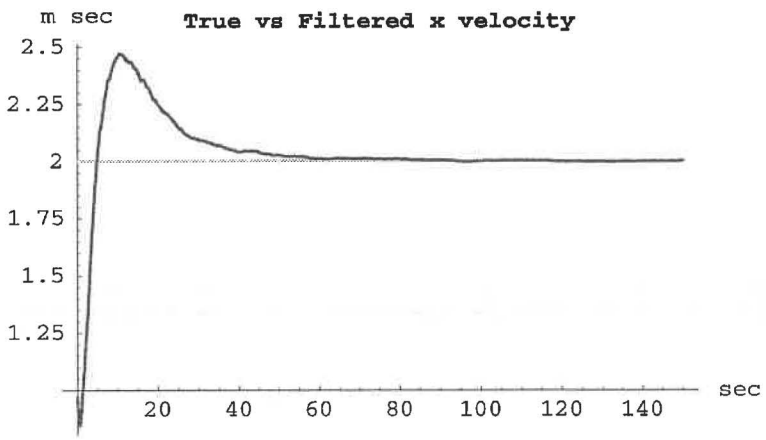
$$\{\triangle x, \triangle y\} = \text{PointingErrors}[i, h, x_p, y_p]$$

BugPlots.nb

The graphs produced in BugPlots.nb are very informative. In the graphs showing the true state versus the filtered estimate, the lighter curve represents the true state and the darker curve represents the filtered estimate. For labeling purposes, position units have been set to meters.

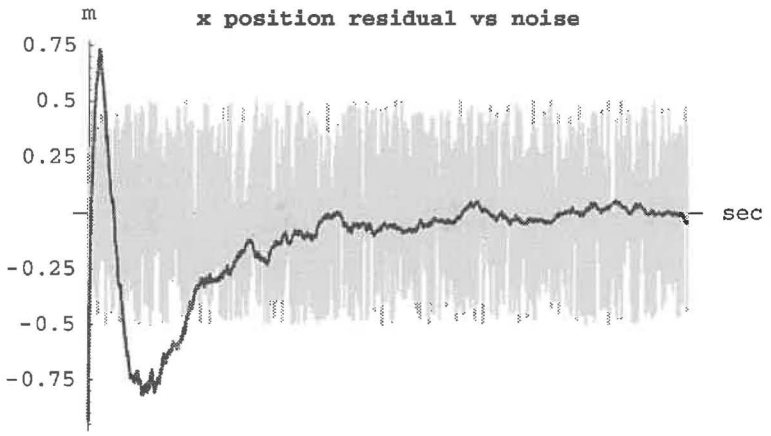
A bug with a simple trajectory The first series of graphs indicate the filter performance with a bug that has a very simple trajectory. The bug has constant velocity in the x component, and very mild acceleration changes in the y component. The major source of error is the jamming noise the bug introduces. Notice that as time proceeds, the filtered state vector gets closer and closer to the true state vector.





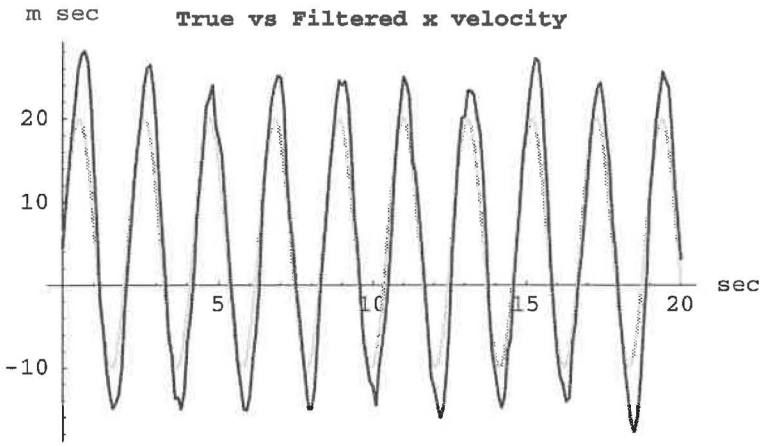
Notice the dramatic noise reduction in the x position residual plot. The noise is uniform random between $-.5$ and .5. The error settles down to approximately $\pm.05$.

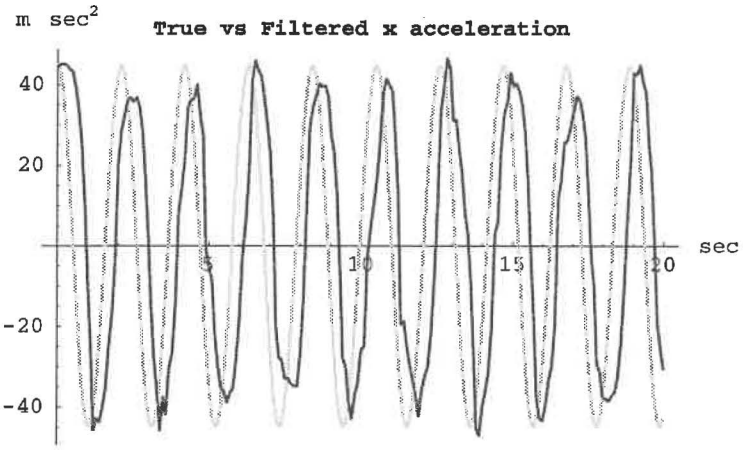
The noise is shown by the lighter curve, the residual by the darker curve.



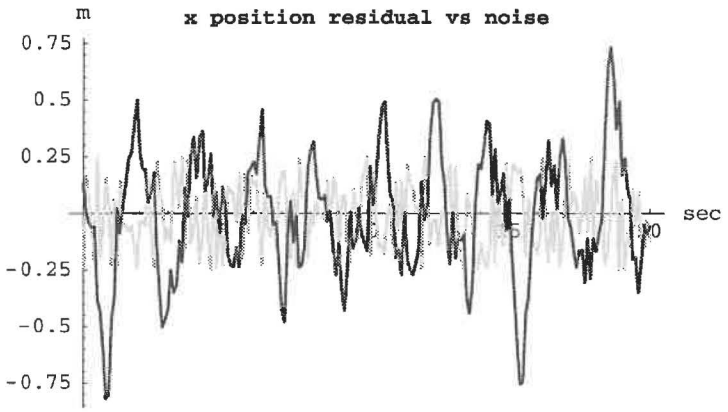
A very active bug The next collection of graphs show the filter performance with the more active bug whose trajectory equations were presented in the discussion of BugMotion.m. Recall that the bug has a very dynamic trajectory and sends out measurement noise that is uniform random bewteen $-.25$ and .25.

The scale on the position graphs is so large that there is no clear distinction between the true and filtered curves. In the velocity and acceleration components, there are obvious overshoots and lags related to the bug's constantly changing trajectory.

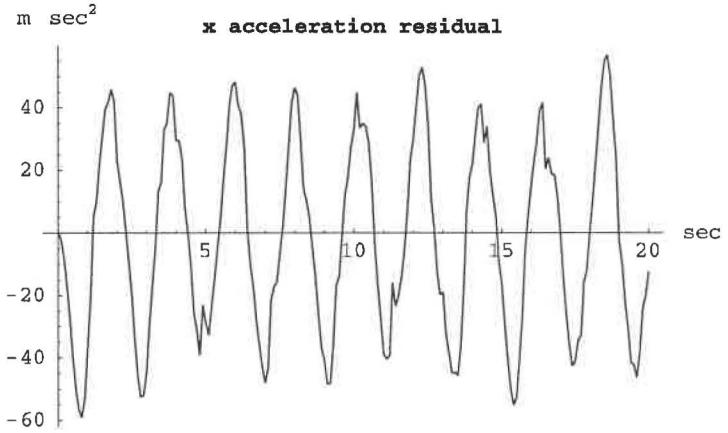




A comparison of the x residual with a plot of the noise indicates that the major component of the error in the filtered estimate is due to the bug's motion rather than to the noise in the position error measurements.

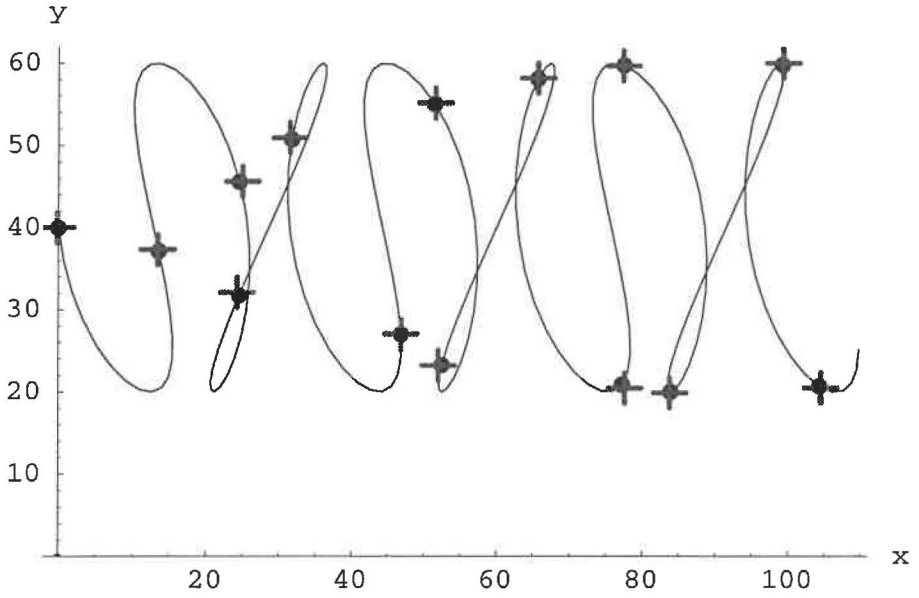


Notice the poor approximation to acceleration for this bug.



BugAnimation.nb

It is harder to demonstrate the BugAnimation.nb notebook using a static format. The graph that follows has been created by super-imposing a subset of the individual frames from the animation of the active bug. The small disk represents the bug (moving generally left to right). The cross-hair represents the filtered estimate of the bug's position. Normally, the upper right-hand corner contains the filter number k for each frame. A curve showing the bug's true trajectory has been added to serve as a visual reference.



4 Conclusion

The graphs in the previous section indicate that the filter seems to have met its objectives fairly well. It screens noise from the measurements while adapting to changes in trajectory. Clearly it cannot do either perfectly. It will be more effective at eliminating noise when the bug has a trajectory that is fairly consistent with the constant acceleration assumption.

The material is accessible to upper-division students who have taken the standard sophomore level mathematics classes. The entire project can be presented by the instructor and the simulation completed by the students in a two to three week period.