Chapter 7

Extended Kalman Filtering

Introduction

SO FAR we have seen how linear Kalman filters are designed and how they perform when the real world can be described by linear differential equations expressed in state-space form and when the measurements are linear functions of the states. In most realistic problems the real world may be described by nonlinear differential equations, and the measurements may not be a function of those states. In this chapter we will introduce extended Kalman filtering for a sample problem in which the measurements are a linear function of the states, but the model of the real world is nonlinear.

Theoretical Equations[1]

To apply extended Kalman-filtering techniques, it is first necessary to describe the real world by a set of nonlinear differential equations. These equations also can be expressed in nonlinear state-space form as a set of first-order nonlinear differential equations or

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}) + \boldsymbol{w}$$

where $\boldsymbol{x}$ is a vector of the system states, $f(\boldsymbol{x})$ is a nonlinear function of those states, and $\boldsymbol{w}$ is a random zero-mean process. The process-noise matrix describing the random process $\boldsymbol{w}$ for the preceding model is given by

$$\boldsymbol{Q} = E(\boldsymbol{w}\boldsymbol{w}^T)$$

Finally, the measurement equation, required for the application of extended Kalman filtering, is considered to be a nonlinear function of the states according to

$$\boldsymbol{z} = h(\boldsymbol{x}) + \boldsymbol{v}$$

where $\boldsymbol{v}$ is a zero-mean random process described by the measurement noise matrix $\boldsymbol{R}$, which is defined as

$$\boldsymbol{R} = E(\boldsymbol{v}\boldsymbol{v}^T)$$

For systems in which the measurements are discrete, we can rewrite the nonlinear measurement equation as

$$z_k = h(\boldsymbol{x}_k) + \boldsymbol{v}_k$$

The discrete measurement noise matrix $\boldsymbol{R}_k$ consists of a matrix of variances representing each measurement noise source. Because the system and measurement equations are nonlinear, a first-order approximation is used in the continuous Riccati equations for the systems dynamics matrix $\boldsymbol{F}$ and the measurement matrix $\boldsymbol{H}$. The matrices are related to the nonlinear system and measurement equations according to

$$\boldsymbol{F} = \left.\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}}\right|_{x=\hat{x}}$$
$$\boldsymbol{H} = \left.\frac{\partial h(\boldsymbol{x})}{\partial \boldsymbol{x}}\right|_{x=\hat{x}}$$

The fundamental matrix, required for the discrete Riccati equations, can be approximated by the Taylor-series expansion for $\exp(\boldsymbol{F}T_s)$ and is given by

$$\Phi_k = I + \boldsymbol{F}T_s + \frac{\boldsymbol{F}^2 T_s^2}{2!} + \frac{\boldsymbol{F}^3 T_s^3}{3!} + \cdots$$

where T_s is the sampling time and $\boldsymbol{I}$ is the identity matrix. Often the series is approximated by only the first two terms or

$$\Phi_k \approx I + \boldsymbol{F}T_s$$

In our applications of extended Kalman filtering, the fundamental matrix will only be used in the calculation of the Kalman gains. Because the fundamental matrix will not be used in the propagation of the states, we have already demonstrated in Chapter 5 that adding more terms to the Taylor-series expansion for the fundamental matrix will not generally improve the performance of the overall filter.

For linear systems, the systems dynamics matrix, measurement matrix, and fundamental matrix are all linear. However, these same matrices for the extended Kalman filter may be nonlinear because they depend on the system state estimates. The Riccati equations, required for the computation of the Kalman gains, are identical to those of the linear case and are repeated here for convenience:

$$\boldsymbol{M}_k = \Phi_k \boldsymbol{P}_{k-1} \Phi_k^T + \boldsymbol{Q}_k$$
$$\boldsymbol{K}_k = \boldsymbol{M}_k \boldsymbol{H}^T (\boldsymbol{H}\boldsymbol{M}_k \boldsymbol{H}^T + \boldsymbol{R}_k)^{-1}$$
$$\boldsymbol{P}_k = (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H})\boldsymbol{M}_k$$

where $\boldsymbol{P}_k$ is a covariance matrix representing errors in the state estimates after an update and $\boldsymbol{M}_k$ is the covariance matrix representing errors in the state estimates before an update. Because Φ_k and $\boldsymbol{H}$ are nonlinear functions of the state estimates, the Kalman gains cannot be computed off line as is possible with a linear Kalman filter. The discrete process-noise matrix $\boldsymbol{Q}_k$ can still be found from the continuous process-noise matrix according to

$$\boldsymbol{Q}_k = \int_0^{T_s} \Phi(\tau)\boldsymbol{Q}\Phi^T(\tau)\,\mathrm{d}t$$

If the dynamical model of a linear Kalman filter is matched to the real world, the covariance matrix $\boldsymbol{P}_k$ can not only be used to calculate Kalman gains but can also provide predictions of the errors in the state estimates. The extended Kalman filter offers no such guarantees, and, in fact, the extended Kalman filter covariance matrix may indicate excellent performance when the filter is performing poorly or is even broken.

As was already mentioned, the preceding approximations for the fundamental and measurement matrices only have to be used in the computation of the Kalman gains. The actual extended Kalman-filtering equations do not have to use those approximations but instead can be written in terms of the nonlinear state and measurement equations. With an extended Kalman filter the new state estimate is the old state estimate projected forward to the new sampling or measurement time plus a gain times a residual or

$$\hat{\boldsymbol{x}}_k = \bar{\boldsymbol{x}}_k + \boldsymbol{K}_k[\boldsymbol{z}_k - \boldsymbol{h}(\bar{\boldsymbol{x}}_k)]$$

In the preceding equation the residual is the difference between the actual measurement and the nonlinear measurement equation. The old estimates that have to be propagated forward do not have to be done with the fundamental matrix but instead can be propagated directly by integrating the actual nonlinear differential equations forward at each sampling interval. For example, Euler integration can be applied to the nonlinear system differential equations yielding

$$\bar{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_{k-1} + \hat{\dot{\boldsymbol{x}}}_{k-1}T_s$$

where the derivative is obtained from

$$\hat{\dot{\boldsymbol{x}}}_{k-1} = f(\hat{\boldsymbol{x}}_{k-1})$$

In the preceding equation the sampling time T_s is used as an integration interval. In problems where the sampling time is large, T_s would have to be replaced by a smaller integration interval, or possibly a more accurate method of integration has to be used. The best way to show how an extended Kalman filter is implemented and performs is by doing an example.

Drag Acting on Falling Object

Let us again reconsider the one-dimensional example of a high-velocity object falling on a tracking radar, as was shown in Fig. 4.26. Recall that the drag-free object was initially at 400,000 ft above the radar and had a velocity of 6000 ft/s toward the radar that was located on the surface of a flat Earth. In that example only gravity g (i.e., $g = 32.2\,\text{ft/s}^2$) acted on the object. Now we will add one degree of complexity to the problem by also considering the influence of drag. Drag will make the resultant equation describing the acceleration of the object nonlinear. However, we will assume that the amount of drag acting on the object is known. As was the case in Chapter 4, let us pretend that the radar measures the range from the radar to the object (i.e., altitude of the object) with a 1000-ft standard deviation measurement accuracy. The radar takes altitude measurements 10 times a second for 30 s. In this example the object will initially be at 200,000 ft above the radar, as is shown in Fig. 7.1. We would like to build a filter to estimate the altitude and velocity of the object.

If x is the distance from the radar to the object, then the acceleration acting on the object consists of gravity and drag[1,2] or

$$\ddot{x} = \text{Drag} - g = \frac{Q_p g}{\beta} - g$$

where β is the ballistic coefficient of the object and Q_p is the dynamic pressure. The ballistic coefficient, which is considered to be a constant in this problem, is a term describing the amount of drag acting on the object. Small values of β indicate high drag, and high values of β indicate low drag. Setting β equal to infinity eliminates the drag, and we end up with the problem of Chapter 4, where only gravity acts on the object. The dynamic pressure Q_p is given by

$$Q_p = 0.5\rho\dot{x}^2$$

where ρ is the air density and $\dot{x}$ is the velocity of the object. For this sample problem we can assume that the air density is an exponential function of altitude or[2]

$$\rho = 0.0034\mathrm{e}^{-x/22000}$$

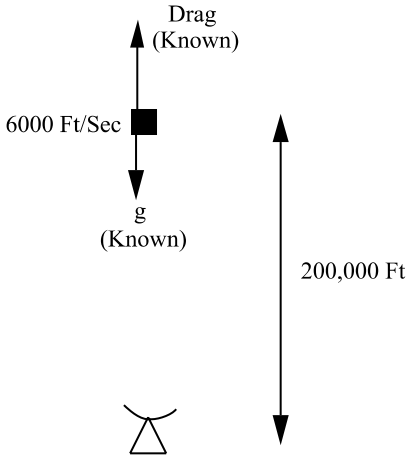

Fig. 7.1 Radar tracking falling object in presence of drag.

where x is altitude. Therefore, the equation expressing the acceleration acting on the object can now be expressed as the nonlinear second-order differential equation

$$\ddot{x} = \frac{Q_p g}{\beta} - g = \frac{0.5 g \rho \dot{x}^2}{\beta} - g = \frac{0.0034 g e^{-x/22000} \dot{x}^2}{2\beta} - g$$

A program was written to integrate numerically the preceding nonlinear differential equation using the second-order Runge–Kutta numerical integration technique described in Chapter 1. Listing 7.1 shows that the object is initially at 200,000-ft altitude traveling downward with an initial velocity of 6000 ft/s, as was shown in Fig. 7.1. The listing also shows that the ballistic coefficient β is made a parameter. The position X, velocity XD, and acceleration XDD of the object are printed out every 0.1 s for 30 s.

Cases were run with Listing 7.1 in which the ballistic coefficient BETA was varied from infinity to 500 lb/ft^2. We can see from Fig. 7.2 that there is not much difference in the final location of the object after 30 s for all ballistic coefficients considered. If there is no drag, the object descends from 200,000 to 5,500 ft in 30 s. If the ballistic coefficient is 500 lb/ft^2 (i.e., the most drag considered in this example), the object only descends to 25,400 ft in the same 30 s. In other words, increasing the drag only decreases the lowest altitude that can be reached in 30 sec.

As expected, Fig. 7.3 shows that objects with the most drag slow up the most. If there is no drag, we can see that the object actually speeds up from its initial value of 6000 ft/s to nearly 7000 ft/s as a result of gravity. With a ballistic coefficient of 2000 lb/ft^2 the object slows up to approximately 5000 ft/s. Reducing the ballistic coefficient to 1000 lb/ft^2 causes the object to slow up to 4100 ft/s. Finally, reducing the ballistic coefficient even further to 500 lb/ft^2 reduces the speed of the object from 6000 ft/s at the beginning of the flight to only 3300 ft/s at the end of the 30-s flight.

Figure 7.4 shows that, as expected, if there is no drag, the maximum acceleration experienced by the object is that of gravity (i.e., -32.2 ft/s^2). However, the presence of drag can cause the object to go through decelerations in excess of 10 g (i.e., 322 ft/s^2). We can see from Fig. 7.4 that the maximum acceleration increases with increasing drag (i.e., decreasing ballistic coefficient).

First Attempt at Extended Kalman Filter

We now have enough of a feel for the dynamics of the falling high-speed body to proceed with the design of an extended Kalman filter. Let us assume that the ballistic coefficient of the object we are tracking is known in advance. Under these circumstances it is only necessary to estimate the altitude and velocity of the object in order to track it. Thus, the proposed states for the filter design are given by

$$\boldsymbol{x} = \begin{bmatrix} x \\ \dot{x} \end{bmatrix}$$

Listing 7.1 Simulation of falling object under influence of drag and gravity

```
IMPLICIT REAL*8 (A-H)
IMPLICIT REAL*8 (O-Z)
G=32.2
X=200000.
XD=-6000.
BETA=500.
OPEN(1,STATUS='UNKNOWN',FILE='DATFIL')
TS=.1
TF=30.
T=0.
S=0.
H=.001
WHILE(T<=TF)
          XOLD=X
          XDOLD=XD
          XDD=.0034*G*XD*XD*EXP(-X/22000.)/(2.*BETA)-G
          X=X+H*XD
          XD=XD+H*XDD
          T=T+H
          XDD=.0034*G*XD*XD*EXP(-X/22000.)/(2.*BETA)-G
          X=.5*(XOLD+X+H*XD)
          D=.5*(XDOLD+XD+H*XDD)
          S=S+H
          IF(S>=(TS-.00001))THEN
                    S=0.
                    WRITE(9,*)T,X,XD,XDD
                    WRITE(1,*)T,X,XD,XDD
          ENDIF
END DO
PAUSE
CLOSE(1)
END
```

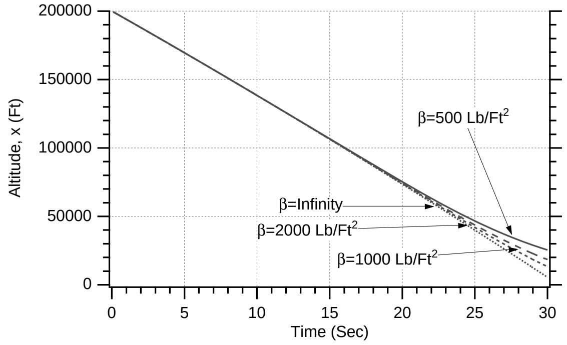

Fig. 7.2 Increasing drag decreases lowest altitude that can be reached in 30 s.

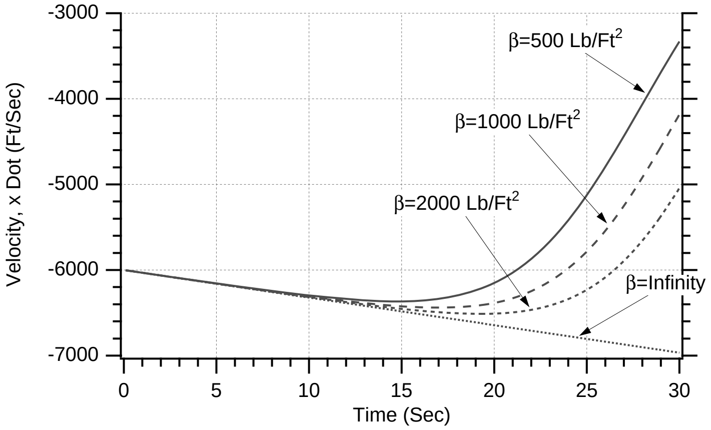

Fig. 7.3 Increasing drag reduces velocity at lower altitudes.

Because we have already shown that the acceleration acting on the object is a nonlinear function of the states according to

$$\ddot{x} = \frac{0.0034ge^{-x/22000}\dot{x}^2}{2\beta} - g$$

we can linearize the preceding equation by saying that

$$\begin{bmatrix} \Delta\dot{x} \\ \Delta\ddot{x} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial\dot{x}}{\partial x} & \dfrac{\partial\dot{x}}{\partial\dot{x}} \\ \dfrac{\partial\ddot{x}}{\partial x} & \dfrac{\partial\ddot{x}}{\partial\dot{x}} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ u_s \end{bmatrix}$$

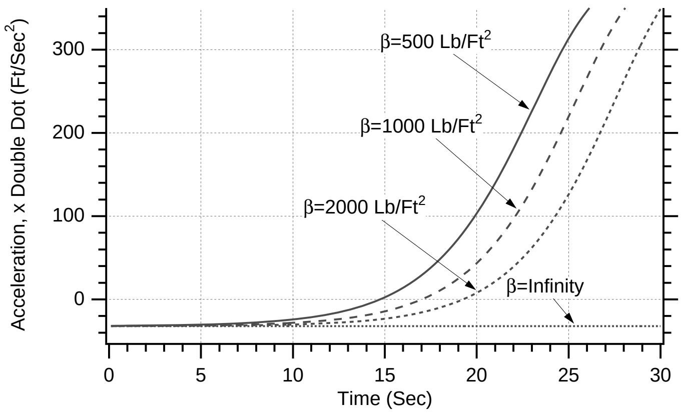

Fig. 7.4 Drag causes high decelerations.

In the preceding equation u_s is a white process noise that has been added to the acceleration equation for possible future protection. We can see from the preceding linearized equation that the systems dynamics matrix is the matrix of partial derivatives or

$$\boldsymbol{F} = \begin{bmatrix} \dfrac{\partial \dot{x}}{\partial x} & \dfrac{\partial \dot{x}}{\partial \dot{x}} \\ \dfrac{\partial \ddot{x}}{\partial x} & \dfrac{\partial \ddot{x}}{\partial \dot{x}} \end{bmatrix}_{x=\hat{x}}$$

Here we can see that the partial derivatives are evaluated at the current state estimates. We can also see from the linearized state-space equation that the continuous process-noise matrix is given by

$$\boldsymbol{Q} = E(\boldsymbol{w}\boldsymbol{w}^T)$$

Therefore, by inspection of the linearized state-space equation, we can say for this example that the continuous process-noise matrix is

$$\boldsymbol{Q}(t) = \begin{bmatrix} 0 & 0 \\ 0 & \Phi_s \end{bmatrix}$$

where Φ_s is the spectral density of the white noise sources assumed to be on acceleration. To evaluate the systems dynamics matrix, we must take the necessary partial derivatives or

$$\frac{\partial \dot{x}}{\partial x} = 0$$

$$\frac{\partial \dot{x}}{\partial \dot{x}} = 1$$

$$\frac{\partial \ddot{x}}{\partial x} = -\frac{-0.0034e^{-x/22000}\dot{x}^2 g}{2\beta(22{,}000)} = \frac{-\rho g \dot{x}^2}{44{,}000\beta}$$

$$\frac{\partial \ddot{x}}{\partial \dot{x}} = -\frac{2{*}0.0034e^{-x/22000}\dot{x} g}{2\beta} = \frac{\rho g \dot{x}}{\beta}$$

Therefore, the systems dynamics matrix turns out to be

$$\boldsymbol{F}(t) = \begin{bmatrix} 0 & 1 \\ \dfrac{-\hat{\rho} g \hat{\dot{x}}^2}{44{,}000\beta} & \dfrac{\hat{\rho}\hat{\dot{x}} g}{\beta} \end{bmatrix}$$

where gravity g and the ballistic coefficient β are assumed to be known perfectly and the estimated air density is a function of the estimated altitude and is given by

$$\hat{\rho} = 0.0034e^{-\hat{x}/22000}$$

If we assume that the systems dynamics matrix is approximately constant between sampling instants, the fundamental matrix can be found by the infinite Taylor-series approximation

$$\Phi(t) = \boldsymbol{I} + \boldsymbol{F}t + \frac{\boldsymbol{F}^2t^2}{2!} + \frac{\boldsymbol{F}^3t^3}{3!} + \cdots$$

If we define

$$f_{21} = \frac{-\hat{\rho}g\hat{\dot{x}}^2}{44{,}000\beta}$$
$$f_{22} = \frac{\hat{\rho}\hat{\dot{x}}g}{\beta}$$

then the systems dynamics matrix can be written as

$$\boldsymbol{F}(t) = \begin{bmatrix} 0 & 1 \\ f_{21} & f_{22} \end{bmatrix}$$

If we only take the first two terms of the Taylor-series expansion for the fundamental matrix, we get

$$\Phi(t) = \boldsymbol{I} + \boldsymbol{F}t = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ f_{21} & f_{22} \end{bmatrix} t = \begin{bmatrix} 1 & t \\ f_{21}t & 1 + f_{22}t \end{bmatrix}$$

or more simply

$$\Phi(t) = \begin{bmatrix} 1 & t \\ f_{21}t & 1 + f_{22}t \end{bmatrix}$$

Therefore, the discrete fundamental matrix can be found by substituting T_s for t and is given by

$$\Phi_k = \begin{bmatrix} 1 & T_s \\ f_{21}T_s & 1 + f_{22}T_s \end{bmatrix}$$

In this example the altitude measurement is a linear function of the states and is given by

$$x_k^* = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_k \\ \dot{x}_k \end{bmatrix} + v_k$$

where v_k is the measurement noise. We can see from the preceding equation that the measurement matrix is given by

$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

The discrete measurement noise matrix is given by

$$\boldsymbol{R}_k = E(\boldsymbol{v}_k \boldsymbol{v}_k^T)$$

Therefore, we can say that for this problem the discrete measurement noise matrix turns out to be a scalar and is simply

$$\boldsymbol{R}_k = \sigma_v^2$$

where σ_v^2 is the variance of the measurement noise. Similarly, we have already shown that the continuous process-noise matrix is given by

$$\boldsymbol{Q}(t) = \begin{bmatrix} 0 & 0 \\ 0 & \Phi_s \end{bmatrix}$$

where Φ_s is the spectral density of the white noise sources assumed to be on acceleration. The discrete process-noise matrix can be derived from the continuous process-noise matrix according to

$$\boldsymbol{Q}_k = \int_0^{T_s} \Phi(\tau) \boldsymbol{Q} \Phi^T(\tau) \, \mathrm{d}t$$

Therefore, substitution of the appropriate matrices into the preceding equation yields

$$\boldsymbol{Q}_k = \Phi_s \int_0^{T_s} \begin{bmatrix} 1 & \tau \\ f_{21}\tau & 1 + f_{22}\tau \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & f_{21}\tau \\ \tau & 1 + f_{22}\tau \end{bmatrix} \mathrm{d}\tau$$

After multiplying out the matrices, we get

$$\boldsymbol{Q}_k = \Phi_s \int_0^{T_s} \begin{bmatrix} \tau^2 & \tau + f_{22}\tau^2 \\ \tau + f_{22}\tau^2 & 1 + 2f_{22}\tau + f_{22}^2\tau^2 \end{bmatrix} \mathrm{d}\tau$$

Finally, after integration we obtain the final expression for the discrete process-noise matrix as

$$\boldsymbol{Q}_k = \Phi_s \begin{bmatrix} \frac{T_s^3}{3} & \frac{T_s^2}{2} + f_{22}\frac{T_s^3}{3} \\ \frac{T_s^2}{2} + f_{22}\frac{T_s^3}{3} & T_s + f_{22}T_s^2 + f_{22}^2\frac{T_s^3}{3} \end{bmatrix}$$

We now have defined all of the matrices required to solve the matrix Riccati equations. The next step is to write down the equations for the Kalman-filter section. First we must be able to propagate the states from the present sampling time to the next sampling time. This cannot be done in this example exactly with the fundamental matrix because the fundamental matrix is approximate (i.e., only two terms of a Taylor-series expansion were used), and the fundamental matrix was also based on a linearization of the problem. Therefore, we will use Euler numerical integration of the nonlinear differential equations until the next update, using the sampling time T_s as the integration interval. Therefore, given the nonlinear acceleration equation

$$\bar{\ddot{x}}_{k-1} = \frac{0.0034ge^{-\hat{x}_{k-1}/22000}\hat{\dot{x}}_{k-1}^2}{2\beta} - g$$

the projected velocity and position are determined by a one-step Euler integration to the next sampling interval as

$$\bar{\dot{x}}_k = \hat{\dot{x}}_{k-1} + T_s\bar{\ddot{x}}_{k-1}$$
$$\bar{x}_k = \hat{x}_{k-1} + T_s\bar{\dot{x}}_{k-1}$$

Now the Kalman-filtering equations can be written as

$$\hat{x}_k = \bar{x}_k + K_{1_k}(x_k^* - \bar{x}_k)$$
$$\hat{\dot{x}}_k = \bar{\dot{x}}_k + K_{2_k}(x_k^* - \bar{x}_k)$$

where x_k^* is the noisy measurement of altitude. Again, note that the matrices required by the Riccati equations in order to obtain the Kalman gains were based on linearized equations, while the Kalman filter made use of the nonlinear equations.

The preceding equations for the Kalman filter, along with the Riccati equations, were programmed and are shown in Listing 7.2, along with a simulation of the real world. We can see that the process-noise matrix is set to zero in this example (i.e., $\Phi_s = 0$). We have initialized the states of the filter close to the true values. The initial filter estimate of altitude XH is in error by 100 ft, and the initial filter estimate of velocity XDH is in error by 10 ft/s. We can see from Listing 7.2 that the initial covariance matrix reflects those errors. We can also see that nominally there is 1000 ft of measurement noise (i.e., SIGNOISE = 1000).

Listing 7.2 First attempt at extended Kalman filter for tracking a falling object under the influence of both drag and gravity

```
C THE FIRST THREE STATEMENTS INVOKE THE ABSOFT RANDOM
  NUMBER GENERATOR ON THE MACINTOSH
      GLOBAL DEFINE
            INCLUDE 'quickdraw.inc'
      END
      IMPLICIT REAL*8 (A-H)
      IMPLICIT REAL*8 (O-Z)
      REAL*8 PHI(2,2),P(2,2),M(2,2),PHIP(2,2),PHIPPHIT(2,2),GAIN(2,1)
      REAL*8 Q(2,2),HMAT(1,2),HM(1,2),MHT(2,1)
      REAL*8 PHIT(2,2)
      REAL*8 HMHT(1,1),HT(2,1),KH(2,2),IDN(2,2),IKH(2,2)
      INTEGER ORDER
      SIGNOISE=1000.
      X=200000.
      XD=-6000.
      BETA=500.
      XH=200025.
      XDH=-6150.
      OPEN(1,STATUS='UNKNOWN',FILE='DATFIL')
      OPEN(2,STATUS='UNKNOWN',FILE='COVFIL')
      ORDER=2
      TS=.1
      TF=30.
      PHIS=0.
      T=0.
      S=0.
      H=.001
      DO 1000 I=1,ORDER
      DO 1000 J=1,ORDER
            PHI(I,J)=0.
            P(I,J)=0.
            Q(I,J)=0.
            IDN(I,J)=0.
1000  CONTINUE
      IDN(1,1)=1.
      IDN(2,2)=1.
      P(1,1)=SIGNOISE*SIGNOISE
      P(2,2)=20000.
      DO 1100 I=1,ORDER
            HMAT(1,I)=0.
            HT(I,1)=0.
1100  CONTINUE
      HMAT(1,1)=1.
      HT(1,1)=1.
      WHILE(T<=TF)
           XOLD=X
           XDOLD=XD
           XDD=.0034*32.2*XD*XD*EXP(-X/22000.)/(2.*BETA)-32.2
           X=X+H*XD
```

(*continued*)

Listing 7.2 (*Continued*)

```
              XD=XD+H*XDD
              T=T+H
              XDD=.0034*32.2*XD*XD*EXP(-X/22000.)/(2.*BETA)-32.2
              X=.5*(XOLD+X+H*XD)
              XD=.5*(XDOLD+XD+H*XDD)
              S=S+H
              IF(S>=(TS-.00001))THEN
              S=0.
              RHOH=.0034*EXP(-XH/22000.)
              F21=-32.2*RHOH*XDH*XDH/(44000.*BETA)
              F22=RHOH*32.2*XDH/BETA
              PHI(1,1)=1.
              PHI(1,2)=TS
              PHI(2,1)=F21*TS
              PHI(2,2)=1.+F22*TS
              Q(1,1)=PHIS*TS*TS*TS/3.
              Q(1,2)=PHIS*(TS*TS/2.+F22*TS*TS*TS/3.)
              Q(2,1)=Q(1,2)
              Q(2,2)=PHIS*(TS+F22*TS*TS+F22*F22*TS*TS*TS/3.)
              CALL MATTRN(PHI,ORDER,ORDER,PHIT)
              CALL MATMUL(PHI,ORDER,ORDER,P,ORDER,ORDER,
                PHIP)
              CALL MATMUL(PHIP,ORDER,ORDER,PHIT,ORDER,
 1              ORDER,PHIPPHIT)
              CALL MATADD(PHIPPHIT,ORDER,ORDER,Q,M)
              CALL MATMUL(HMAT,1,ORDER,M,ORDER,ORDER,HM)
              CALL MATMUL(HM,1,ORDER,HT,ORDER,1,HMHT)
              HMHTR=HMHT(1,1)+SIGNOISE*SIGNOISE
              HMHTRINV=1./HMHTR
              CALL MATMUL(M,ORDER,ORDER,HT,ORDER,1,MHT)
              DO 150 I=1,ORDER
                   GAIN(I,1)=MHT(I,1)*HMHTRINV
150           CONTINUE
              CALL MATMUL(GAIN,ORDER,1,HMAT,1,ORDER,KH)
              CALL MATSUB(IDN,ORDER,ORDER,KH,IKH)
              CALL MATMUL(IKH,ORDER,ORDER,M,ORDER,
                ORDER,P)
              CALL GAUSS(XNOISE,SIGNOISE)
              XDDB=.0034*32.2*XDH*XDH*EXP(-XH/22000.)/
 1              (2.*BETA)-32.2
              XDB=XDH+XDDB*TS
              XB=XH+TS*XDB
              RES=X+XNOISE-XB
              XH=XB+GAIN(1,1)*RES
              XDH=XDB+GAIN(2,1)*RES
              ERRX=X-XH
              SP11=SQRT(P(1,1))
              ERRXD=XD-XDH
              SP22=SQRT(P(2,2))
```

(*continued*)

Listing 7.2 (*Continued*)

```
                        WRITE(9,*)T,X,XH,XD,XDH,RES,GAIN(1,1),GAIN(2,1)
                        WRITE(1,*)T,X,XH,XD,XDH,RES,GAIN(1,1),GAIN(2,1)
                        WRITE(2,*)T,ERRX,SP11,-SP11,ERRXD,SP22,-SP22
                ENDIF
        END DO
        PAUSE
        CLOSE(1)
        END

C SUBROUTINE GAUSS IS SHOWN IN LISTING 1.8
C SUBROUTINE MATTRN IS SHOWN IN LISTING 1.3
C SUBROUTINE MATMUL IS SHOWN IN LISTING 1.4
C SUBROUTINE MATADD IS SHOWN IN LISTING 1.1
C SUBROUTINE MATSUB IS SHOWN IN LISTING 1.2
```

The nominal case of Listing 7.2 was run where the measurement noise standard deviation was 1000 ft and the process noise was set to zero. We can see from Fig. 7.5 that the error in the estimate of altitude appears to be within the covariance matrix predictions (i.e., square root of P_{11}) approximately 68% of the time. Figure 7.6 also shows that the error in the estimate of velocity also appears to be within the covariance matrix bounds (i.e., square root of P_{22}) the right percentage of time. Thus, at first glance, it appears that our extended Kalman filter appears to be working properly.

Another case was run with Listing 7.2 in which the measurement noise standard deviation SIGNOISE was reduced from 1000 to 25 ft. Although we can see from Fig. 7.7 that the error in the estimate of altitude has been significantly reduced (i.e., see Fig. 7.5), we can also see that the error in the estimate of altitude

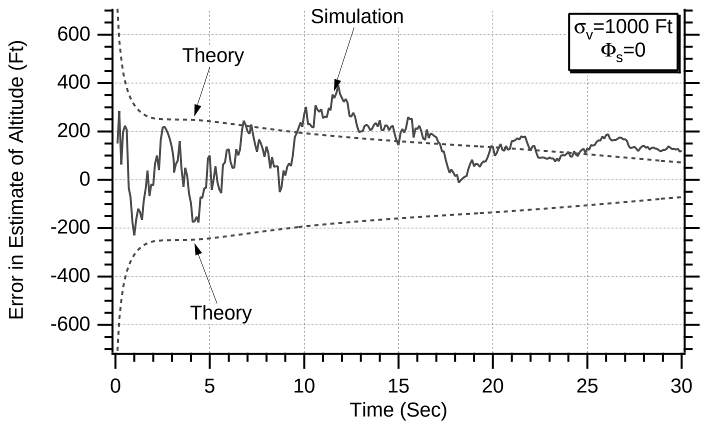

Fig. 7.5 Error in the estimate of position appears to be within the theoretical bounds.

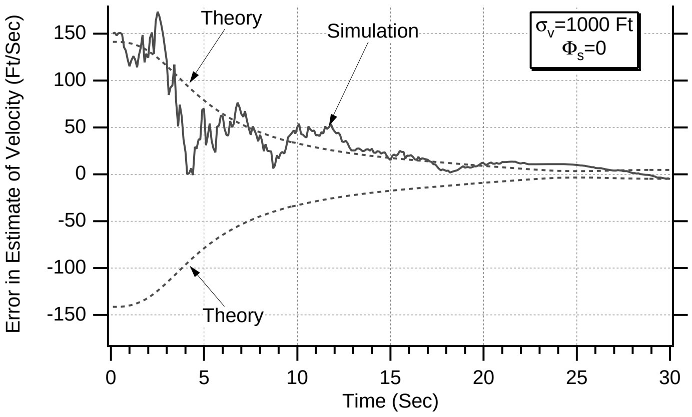

Fig. 7.6 Error in the estimate of velocity appears to be within the theoretical bounds.

is no longer within the theoretical bounds and is in fact diverging from the theoretical bounds. Similarly, Fig. 7.8 also shows that, although the error in the estimate of velocity has been reduced (i.e., see Fig. 7.6), we can no longer say that the error in the estimate of velocity is within the theoretical bounds. It, too, appears to be diverging for most of the flight. Clearly something is wrong with the design of the extended Kalman filter in this example.

Recall that the filter residual is the difference between the actual measurement and the nonlinear measurement equation. Figure 7.9 shows how the residual behaves as a function of time. Here we can see that the filter residual starts to drift from its theoretical value of zero (we will derive the fact that the residual should have zero mean in Chapter 14) after approximately 20 s. To prevent the residual from drifting, the filter should pay more attention to the measurements after 20 s. The only way for the filter to pay attention to the measurements is through the

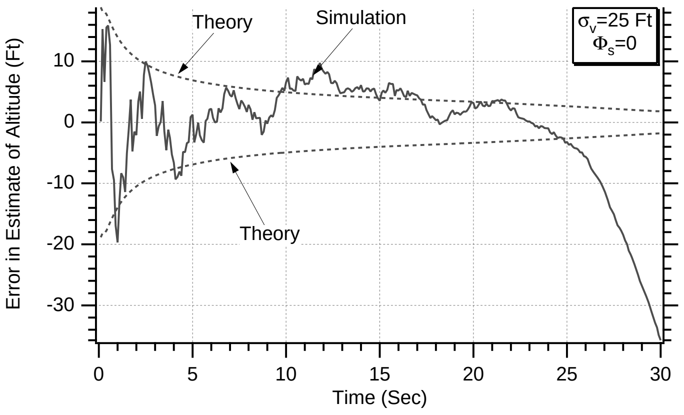

Fig. 7.7 Error in estimate of position appears to be diverging.

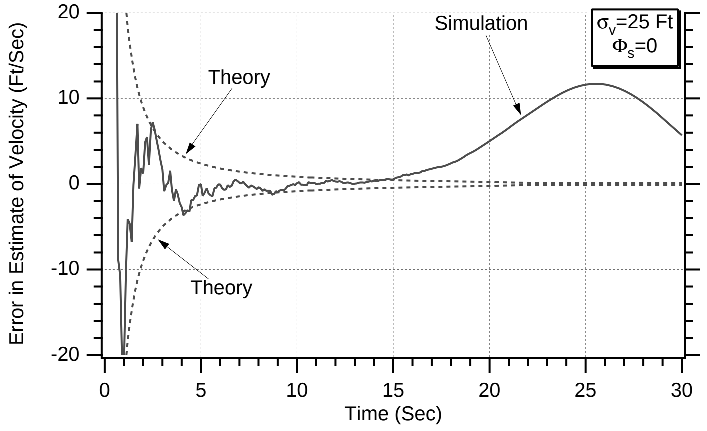

Figure 7.8 Error in estimate of velocity appears to be growing for most of the flight.

Kalman gains. A high set of gains will enable the filter to make use of the measurements, whereas a low set of gains will cause the filter to ignore the measurements. We can see from Fig. 7.10 that the first Kalman gain is very small and the second Kalman gain is approximately zero after 20 s. Thus, we can conclude that after a while the filter is ignoring the measurements, which causes the residual to drift.

We have seen in preceding chapters that the addition of process noise can often help account for the fact that our filter may not be matched to the real world. A case was run in which the spectral density of the process noise was increased from 0 to 100 when the measurement noise standard deviation was 25 ft. No attempt was made to optimize the amount of process noise required. We can see

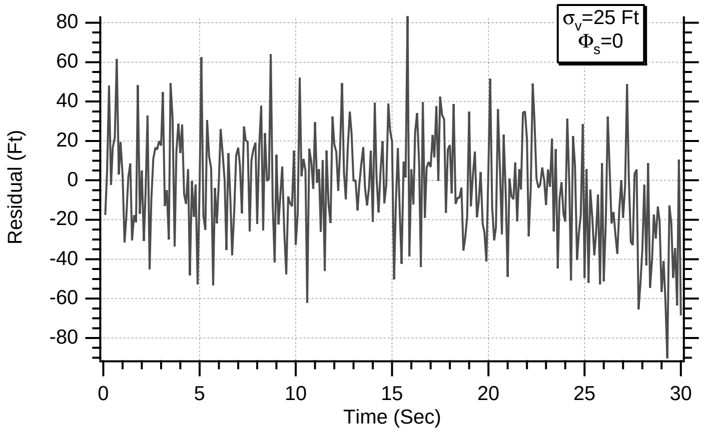

Fig. 7.9 Residual drifts from zero after 20 s.

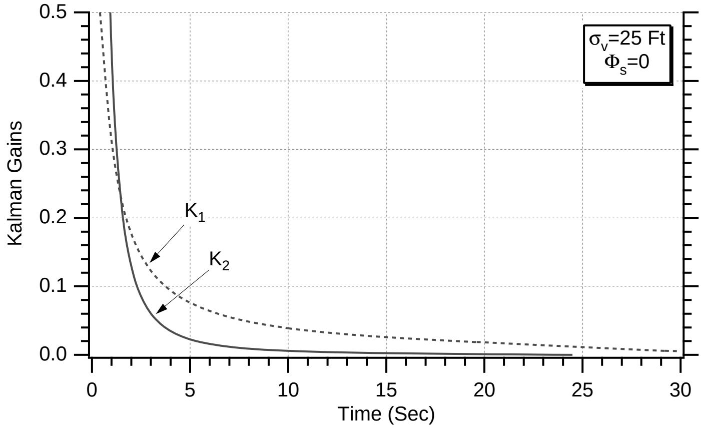

Fig. 7.10 Both Kalman gains approach zero.

from Fig. 7.11 that the addition of process noise has now made the residual zero mean. The reason for this is shown in Fig. 7.12, where the Kalman gains no longer approach zero. The larger Kalman gains (i.e., compared to the case in which there was no process noise) enable the filter to pay more attention to the measurements and thus prevent the residual from drifting off.

The elimination of the divergence problem can also be seen when we monitor the errors in the estimates of both states. We can see from Figs. 7.13 and 7.14 that the addition of process noise eliminated filter divergence. Comparing Fig. 7.7 with Fig. 7.13 shows that the error in the estimate of position has been reduced from 30 to approximately 10 ft. Comparing Fig. 7.8 with Fig. 7.14 shows that the error in the estimate of velocity remains at approximately 10 ft/s, except that now the error in the estimate of velocity is no longer behaving strangely. We also can

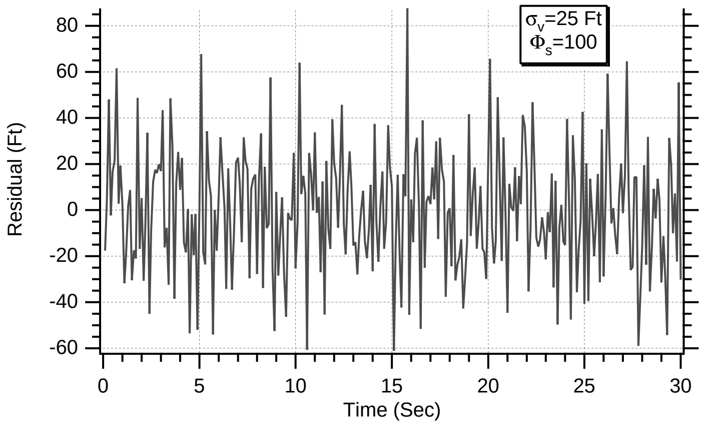

Fig. 7.11 Adding process noise prevents residual from drifting away from zero.

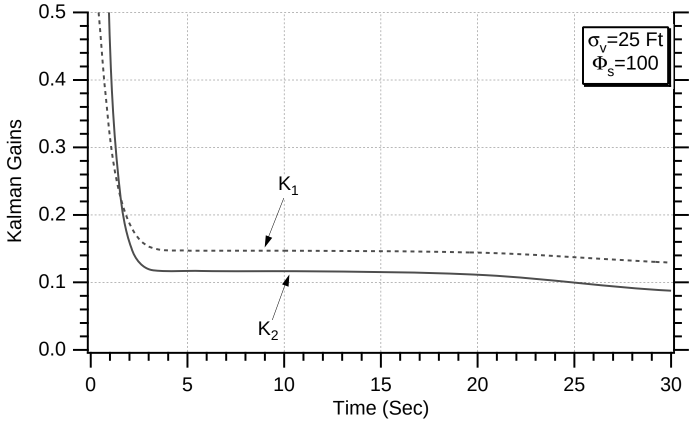

Fig. 7.12 Process noise prevents Kalman gains from going to zero.

see from Figs. 7.13 and 7.14 that the addition of process noise causes the errors in the estimates to no longer approach zero as more measurements are taken but simply to approach a steady-state value.

Second Attempt at Extended Kalman Filter

We saw in the preceding section that without process noise the extended Kalman filter's errors in the estimates diverged rather than got smaller as more measurements were taken. Adding process noise appeared to be the engineering fix for making the divergence problem go away. However, in the model of the real world for this problem there was no process noise. Therefore, the extended Kalman filter was apparently matched to the real world, and there should not have

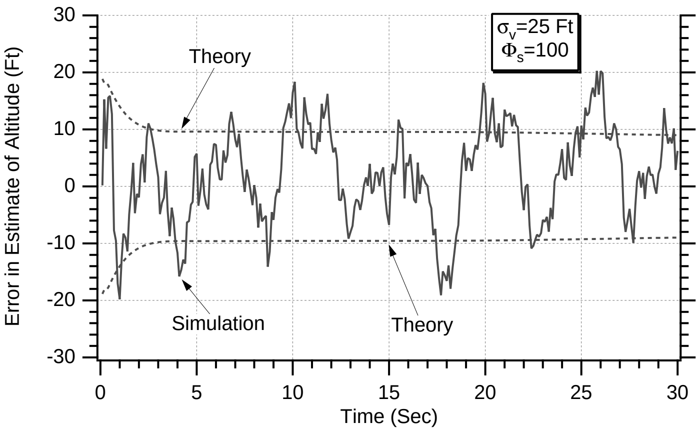

Fig. 7.13 Adding process noise eliminates filter divergence in position.

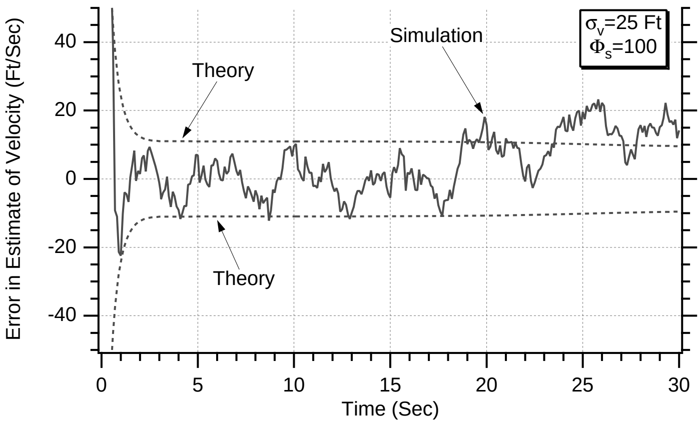

Fig. 7.14 Adding process noise eliminates filter divergence in velocity.

been any divergence. In this section we shall attempt to make the extended Kalman filter even better in an attempt to remove the divergence in the errors in the state estimates without resorting to the addition of process noise.

One possible cause of the divergence is that we only used two terms to compute the fundamental matrix. In reality, the fundamental matrix can be expressed by the infinite Taylor-series expansion

$$\Phi_k = I + \boldsymbol{F}T_s + \frac{\boldsymbol{F}^2 T_s^2}{2!} + \frac{\boldsymbol{F}^3 T_s^3}{3!} + \cdots$$

For this example we have shown that the systems dynamics matrix is given by

$$\boldsymbol{F}(t) = \begin{bmatrix} 0 & 1 \\ f_{21} & f_{22} \end{bmatrix}$$

where f_{21} and f_{22} can be written in terms of the state estimates as

$$f_{21} = \frac{-\hat{\rho} g \hat{\dot{x}}^2}{44{,}000\beta}$$

$$f_{22} = \frac{\hat{\rho} \hat{\dot{x}} g}{\beta}$$

We previously showed that, assuming the systems dynamics matrix to be approximately constant between sampling times, the two-term Taylor-series approximation to the continuous fundamental matrix was

$$\Phi(t) = \boldsymbol{I} + \boldsymbol{F}t = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ f_{21} & f_{22} \end{bmatrix} t = \begin{bmatrix} 1 & t \\ f_{21}t & 1 + f_{22}t \end{bmatrix}$$

or more simply

$$\Phi(t) = \begin{bmatrix} 1 & t \\ f_{21}t & 1 + f_{22}t \end{bmatrix}$$

Therefore, the discrete fundamental matrix can be found by substituting T_s for t and is given by

$$\Phi_k = \begin{bmatrix} 1 & T_s \\ f_{21}T_s & 1 + f_{22}T_2 \end{bmatrix}$$

To get better approximations to the fundamental matrix, we first have to find $\boldsymbol{F}^2$ and $\boldsymbol{F}^3$ or

$$\boldsymbol{F}^2 = \begin{bmatrix} 0 & 1 \\ f_{21} & f_{22} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ f_{21} & f_{22} \end{bmatrix} = \begin{bmatrix} f_{21} & f_{22} \\ f_{22}f_{21} & f_{21} + f_{22}^2 \end{bmatrix}$$

$$\boldsymbol{F}^3 = \begin{bmatrix} 0 & 1 \\ f_{21} & f_{22} \end{bmatrix} \begin{bmatrix} f_{21} & f_{22} \\ f_{22}f_{21} & f_{21} + f_{22}^2 \end{bmatrix} = \begin{bmatrix} f_{22}f_{21} & f_{21} + f_{22}^2 \\ f_{21}^2 + f_{22}^2 f_{21} & 2f_{22}f_{21} + f_{22}^3 \end{bmatrix}$$

Therefore, the three-term Taylor-series approximation to the fundamental matrix is given by

$$\Phi_{k_{3\ \mathrm{Term}}} = \begin{bmatrix} 1 & T_s \\ f_{21}T_s & 1 + f_{22}T_s \end{bmatrix} + \begin{bmatrix} f_{21} & f_{22} \\ f_{22}f_{21} & (f_{21} + f_{22}^2) \end{bmatrix} \frac{T_s^2}{2}$$

or more simply

$$\Phi_{k_{3\ \mathrm{Term}}} = \begin{bmatrix} 1 + f_{21}\dfrac{T_s^2}{2} & T_s + f_{22}\dfrac{T_s^2}{2} \\ f_{21}T_s + f_{22}f_{21}\dfrac{T_s^2}{2} & 1 + f_{22}T_s + (f_{21} + f_{22}^2)\dfrac{T_s^2}{2} \end{bmatrix}$$

The four-term approximation to the fundamental matrix can be found from

$$\Phi_{k_{4\ \mathrm{Term}}} = \begin{bmatrix} 1 + f_{21}\dfrac{T_s^2}{2} & T_s + f_{22}\dfrac{T_s^2}{2} \\ f_{21}T_s + f_{22}f_{21}\dfrac{T_s^2}{2} & 1 + f_{22}T_s + (f_{21} + f_{22}^2)\dfrac{T_s^2}{2} \end{bmatrix} + \begin{bmatrix} f_{22}f_{21} & f_{21} + f_{22}^2 \\ f_{21}^2 + f_{22}^2 f_{21} & 2f_{22}f_{21} + f_{22}^3 \end{bmatrix} \frac{T_s^3}{6}$$

Listing 7.3 Second attempt at extended Kalman filter in which more terms are added to fundamental matrix calculation

```
C THE FIRST THREE STATEMENTS INVOKE THE ABSOFT RANDOM
  NUMBER GENERATOR ON THE MACINTOSH
      GLOBAL DEFINE
               INCLUDE 'quickdraw.inc'
      END
      IMPLICIT REAL*8 (A-H)
      IMPLICIT REAL*8 (O-Z)
      REAL*8 PHI(2,2),P(2,2),M(2,2),PHIP(2,2),PHIPPHIT(2,2),GAIN(2,1)
      REAL*8 Q(2,2),HMAT(1,2),HM(1,2),MHT(2,1)
      REAL*8 PHIT(2,2)
      REAL*8 HMHT(1,1),HT(2,1),KH(2,2),IDN(2,2),IKH(2,2)
      INTEGER ORDER
      ITERM=4
      SIGNOISE=25.
      X=200000.
      XD=-6000.
      BETA=500.
      XH=200025.
      XDH=-6150.
      OPEN(1,STATUS='UNKNOWN',FILE='DATFIL')
      OPEN(2,STATUS='UNKNOWN',FILE='COVFIL')
      ORDER=2
      TS=.1
      TF=30.
      PHIS=0./TF
      T=0.
      S=0.
      H=.001
      DO 1000 I=1,ORDER
      DO 1000 J=1,ORDER
               PHI(I,J)=0.
               P(I,J)=0.
               Q(I,J)=0.
               IDN(I,J)=0.
1000  CONTINUE
      IDN(1,1)=1.
      IDN(2,2)=1.
      P(1,1)=SIGNOISE*SIGNOISE
      P(2,2)=20000.
      DO 1100 I=1,ORDER
               HMAT(1,I)=0.
               HT(I,1)=0.
1100  CONTINUE
      HMAT(1,1)=1.
      HT(1,1)=1.
      WHILE(T<=TF)
            XOLD=X
            XDOLD=XD
            XDD=.0034*32.2*XD*XD*EXP(-X/22000.)/(2.*BETA)-32.2
```

(*continued*)

Listing 7.3 (*Continued*)

```
                    X=X+H*XD
                    XD=XD+H*XDD
                    T=T+H
                    XDD=.0034*32.2*XD*XD*EXP(-X/22000.)/(2.*BETA)-32.2
                    X=.5*(XOLD+X+H*XD)
                    XD=.5*(XDOLD+XD+H*XDD)
                    S=S+H
                    IF(S>=(TS-.00001))THEN
                    S=0.
                    RHOH=.0034*EXP(-XH/22000.)
                    F21=-32.2*RHOH*XDH*XDH/(44000.*BETA)
                    F22=RHOH*32.2*XDH/BETA
                    RHOH=.0034*EXP(-XH/22000.)
                    F21=-32.2*RHOH*XDH*XDH/(44000.*BETA)
                    F22=RHOH*32.2*XDH/BETA
                    IF(ITERM.EQ.2)THEN
                        PHI(1,1)=1.
                        PHI(1,2)=TS
                        PHI(2,1)=F21*TS
                        PHI(2,2)=1.+F22*TS
                    ELSEIF(ITERM.EQ.3)THEN
                        PHI(1,1)=1.+.5*TS*TS*F21
                        PHI(1,2)=TS+.5*TS*TS*F22
                        PHI(2,1)=F21*TS+.5*TS*TS*F21*F22
                        PHI(2,2)=1.+F22*TS+.5*TS*TS*(F21+F22*F22)
                    ELSE
                        PHI(1,1)=1.+.5*TS*TS*F21+TS*TS*TS*F22*F21/6.

                        PHI(1,2)=TS+.5*TS*TS*F22+TS*TS*TS*
  1                          (F21+F22*F22)/6.
                        PHI(2,1)=F21*TS+.5*TS*TS*F21*F22+TS*TS
  1                          *TS*(F21*F21+F22*F22*F21)/6.
                        PHI(2,2)=1.+F22*TS+.5*TS*TS*(F21+F22
  1                          *F22)+TS*TS*TS*(2.*F21*F22
  2                           +F22**3)/6.
                    ENDIF
C Q Matrix Only Valid For Two Term Expansion For Fundamental Matrix
                    Q(1,1)=PHIS*TS*TS*TS/3.
                    Q(1,2)=PHIS*(TS*TS/2.+F22*TS*TS*TS/3.)
                    Q(2,1)=Q(1,2)
                    Q(2,2)=PHIS*(TS+F22*TS*TS+F22*F22*TS*TS*TS/3.)
                    CALL MATTRN(PHI,ORDER,ORDER,PHIT)
                    CALL MATMUL(PHI,ORDER,ORDER,P,ORDER,ORDER,
                       PHIP)
                    CALL MATMUL(PHIP,ORDER,ORDER,PHIT,ORDER,
  1                    ORDER,PHIPPHIT)
                    CALL MATADD(PHIPPHIT,ORDER,ORDER,Q,M)
                    CALL MATMUL(HMAT,1,ORDER,M,ORDER,ORDER,HM)
                    CALL MATMUL(HM,1,ORDER,HT,ORDER,1,HMHT)
```

(*continued*)

Listing 7.3 (*Continued*)

```
                    HMHTR=HMHT(1,1)+SIGNOISE*SIGNOISE
                    HMHTRINV=1./HMHTR
                    CALL MATMUL(M,ORDER,ORDER,HT,ORDER,1,MHT)
                    DO 150 I=1,ORDER
                       GAIN(I,1)=MHT(I,1)*HMHTRINV
 150                CONTINUE
                    CALL MATMUL(GAIN,ORDER,1,HMAT,1,ORDER,KH)
                    CALL MATSUB(IDN,ORDER,ORDER,KH,IKH)
                    CALL MATMUL(IKH,ORDER,ORDER,M,ORDER,
                      ORDER,P)
                    CALL GAUSS(XNOISE,SIGNOISE)
                    XDDB=.0034*32.2*XDH*XDH*EXP(-XH/22000.)/
   1                       (2.*BETA)-32.2
                    XDB=XDH+XDDB*TS
                    XB=XH+TS*XDB
                    RES=X+XNOISE-XB
                    XH=XB+GAIN(1,1)*RES
                    XDH=XDB+GAIN(2,1)*RES
                    ERRX=X-XH
                    SP11=SQRT(P(1,1))
                    ERRXD=XD-XDH
                    SP22=SQRT(P(2,2))
                    WRITE(9,*)T,X,XH,XD,XDH
                    WRITE(1,*)T,X,XH,XD,XDH
                    WRITE(2,*)T,ERRX,SP11,-SP11,ERRXD,SP22,-SP22
               ENDIF
       END DO
       PAUSE
       CLOSE(1)
       END

C SUBROUTINE GAUSS IS SHOWN IN LISTING 1.8
C SUBROUTINE MATTRN IS SHOWN IN LISTING 1.3
C SUBROUTINE MATMUL IS SHOWN IN LISTING 1.4
C SUBROUTINE MATADD IS SHOWN IN LISTING 1.1
C SUBROUTINE MATSUB IS SHOWN IN LISTING 1.2
```

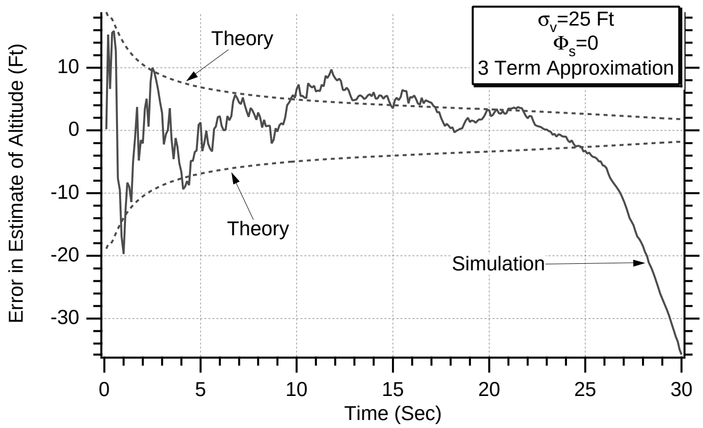

Fig. 7.15 Having three-term approximation for fundamental matrix does not remove filter divergence when there is no process noise.

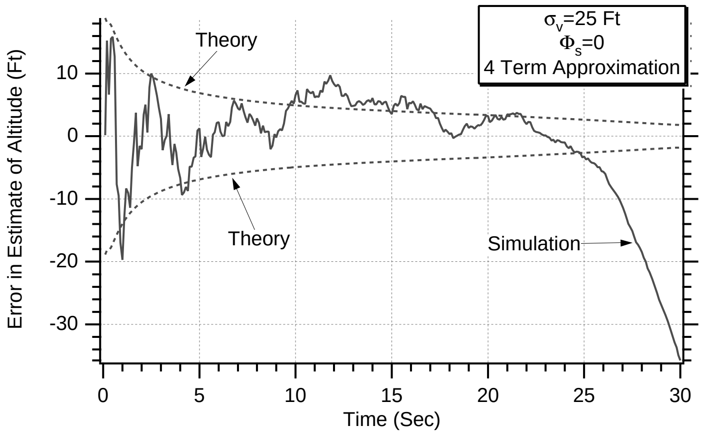

Fig. 7.16 Having four-term approximation for fundamental matrix does not remove filter divergence when there is no process noise.

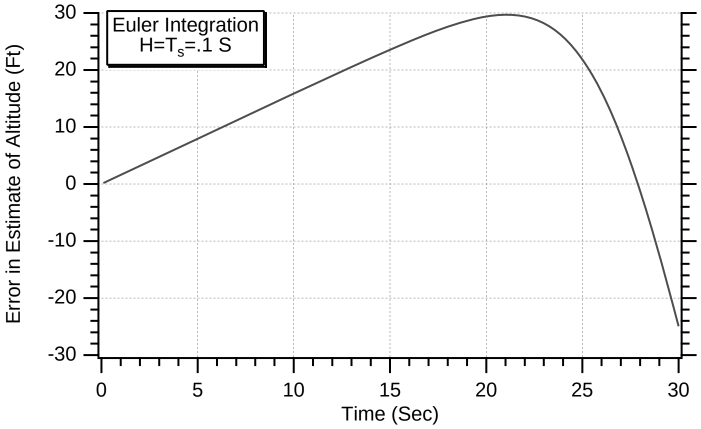

Fig. 7.17 Euler integration with an integration interval of 0.1 s is not adequate for eliminating altitude errors.

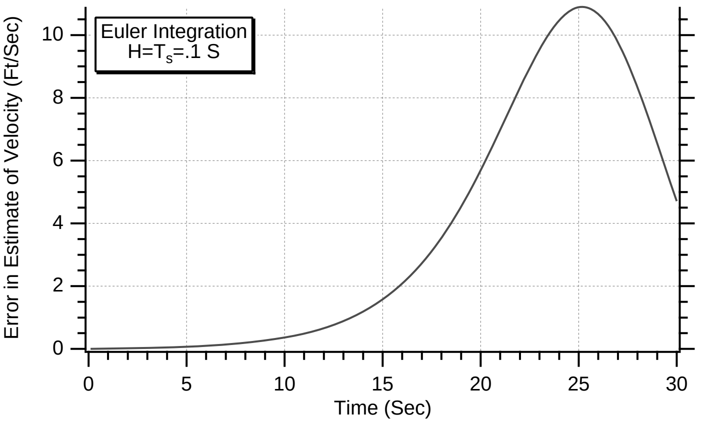

Fig. 7.18 Euler integration with an integration interval of .1 s is not adequate for eliminating velocity errors.

Listing 7.4 Simulation to test state propagation

```
IMPLICIT REAL*8 (A-H)
IMPLICIT REAL*8 (O-Z)
X=200000.
XD=-6000.
BETA=500.
XH=X
XDH=XD
OPEN(1,STATUS='UNKNOWN',FILE='DATFIL')
TS=.1
TF=30.
T=0.
S=0.
H=.001
HP=.1
WHILE(T<=TF)
        XOLD=X
        XDOLD=XD
        XDD=.0034*32.2*XD*XD*EXP(-X/22000.)/(2.*BETA)-32.2
        X=X+H*XD
        XD=XD+H*XDD
        T=T+H
        XDD=.0034*32.2*XD*XD*EXP(-X/22000.)/(2.*BETA)-32.2
        X=.5*(XOLD+X+H*XD)
        XD=.5*(XDOLD+XD+H*XDD)
        S=S+H
        IF(S>=(TS-.00001))THEN
                S=0.
                CALL PROJECT(T,TS,XH,XDH,BETA,XB,XDB,XDDB,HP)
                XH=XB
                XDH=XDB
                ERRX=X-XH
                ERRXD=XD-XDH
                WRITE(9,*)T,ERRX,ERRXD
                WRITE(1,*)T,ERRX,ERRXD
        ENDIF
END DO
PAUSE
CLOSE(1)
END

SUBROUTINE PROJECT(TP,TS,XP,XDP,BETA,XH,XDH,XDDH,HP)
IMPLICIT REAL*8 (A-H)
IMPLICIT REAL*8 (O-Z)
T=0.
X=XP
XD=XDP
H=HP
WHILE(T<=(TS-.0001))
        XDD=.0034*32.2*XD*XD*EXP(-X/22000.)/(2.*BETA)-32.2
        XD=XD+H*XDD
```

(*continued*)

Listing 7.4 (*Continued*)

```
          X=X+H*XD
          T=T+H
END  DO
XH=X
XDH=XD
XDDH=XDD
RETURN
END
```

or more simply

$$\Phi_{k_{4\mathrm{Term}}} = \begin{bmatrix} 1 + f_{21}\dfrac{T_s^2}{2} + f_{22}f_{21}\dfrac{T_s^3}{6} & T_s + f_{22}\dfrac{T_s^2}{2} + (f_{21} + f_{22}^2)\dfrac{T_s^3}{6} \\ f_{21}T_s + f_{22}f_{21}\dfrac{T_s^2}{2} + (f_{21}^2 + f_{22}^2 f_{21})\dfrac{T_s^3}{6} & 1 + f_{22}T_s + (f_{21} + f_{22}^2)\dfrac{T_s^2}{2} + (2f_{22}f_{21} + f_{22}^3)\dfrac{T_s^3}{6} \end{bmatrix}$$

Listing 7.2 was modified to account for the fact that we might want to have more terms in the Taylor-series expansion to approximate the fundamental matrix, and the resultant simulation appears in Listing 7.3. All changes from the original simulation are highlighted in bold. Because we are only going to run with zero

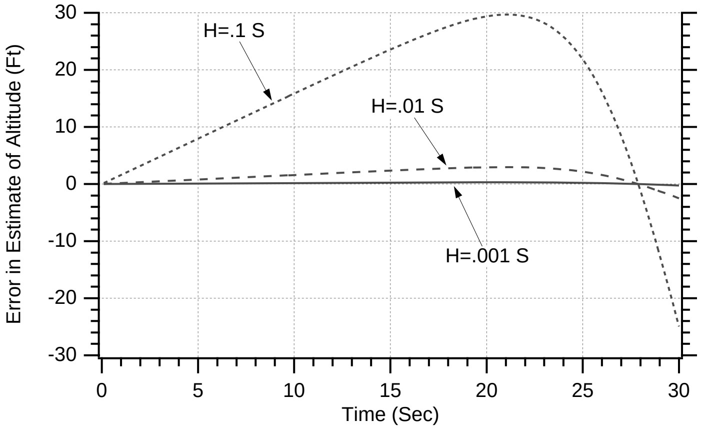

Fig. 7.19 Integration step size in propagation subroutine must be reduced to 0.001 s to keep errors in estimate of altitude near zero.

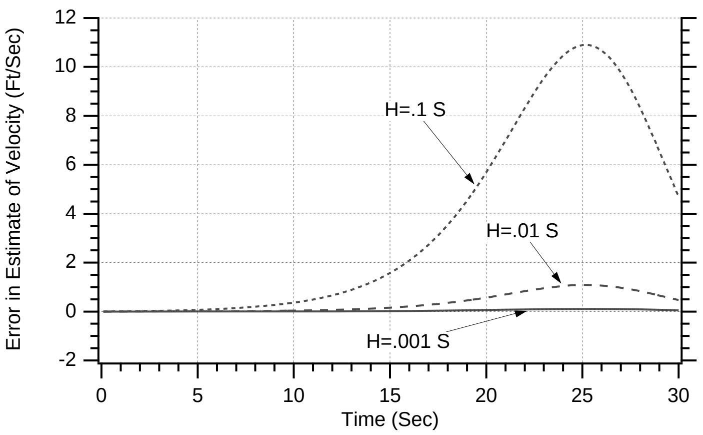

Fig. 7.20 Integration step size in propagation subroutine must be reduced to 0.001 s to keep errors in estimate of velocity near zero.

process noise, the discrete process matrix was not modified in this experiment. If ITERM = 2, we have a two-term approximation; if ITERM = 3, we have a three-term approximation; and, finally, if ITERM = 4, we have a four-term approximation to the fundamental matrix.

Cases were rerun with Listing 7.3 in which there was no process noise and 25 ft of measurement noise. We can see from Figs. 7.15 and 7.16 that having more terms in the approximation for the fundamental matrix has absolutely no influence on the filter divergence in the error in the estimate on altitude. In fact these curves are virtually identical to the results of the two-term approximation to the fundamental matrix (see Fig. 7.7). Therefore, we can conclude for this problem that the number of terms used in the series approximation for the fundamental matrix is not important and that something else must be causing the filter estimates to diverge.

Third Attempt at Extended Kalman Filter

Another possibility for the filter divergence is that perhaps our method of numerical integration in propagating the states forward from the nonlinear differential equation is not sufficiently accurate. The original program of Listing 7.2 was modified so that the initial state estimates were perfect and the Kalman gains were set to zero. Under these conditions the filter estimates should be perfect if a perfect method of integration was used because there are no errors. Therefore, the errors in the state estimates should be zero. However, we can see from Figs. 7.17 and 7.18 that the errors in the estimates of altitude and velocity are not zero under these circumstances. Therefore, the method of integration or state propagation needs improvement.

Another program was written in which the integration interval of the propagation method could be varied, and it is shown in Listing 7.4. We can see that Euler integration is used in subroutine PROJECT, which integrates the

Listing 7.5 Extended Kalman filter with accurate propagation subroutine

```
C THE FIRST THREE STATEMENTS INVOKE THE ABSOFT RANDOM
  NUMBER GENERATOR ON THE MACINTOSH
      GLOBAL DEFINE
            INCLUDE 'quickdraw.inc'
      END
      IMPLICIT REAL*8 (A-H)
      IMPLICIT REAL*8 (O-Z)
      REAL*8 PHI(2,2),P(2,2),M(2,2),PHIP(2,2),PHIPPHIT(2,2),GAIN(2,1)
      REAL*8 Q(2,2),HMAT(1,2),HM(1,2),MHT(2,1)
      REAL*8 PHIT(2,2)
      REAL*8 HMHT(1,1),HT(2,1),KH(2,2),IDN(2,2),IKH(2,2)
      INTEGER ORDER
      SIGNOISE=25.
      X=200000.
      XD=-6000.
      BETA=500.
      XH=200025.
      XDH=-6150.
      OPEN(1,STATUS='UNKNOWN',FILE='DATFIL')
      OPEN(2,STATUS='UNKNOWN',FILE='COVFIL')
      ORDER=2
      TS=.1
      TF=30.
      PHIS=0.
      T=0.
      S=0.
      H=.001

      DO 1000 I=1,ORDER
      DO 1000 J=1,ORDER
            PHI(I,J)=0.
            P(I,J)=0.
            Q(I,J)=0.
            IDN(I,J)=0.
1000  CONTINUE
      IDN(1,1)=1.
      IDN(2,2)=1.
      P(1,1)=SIGNOISE*SIGNOISE
      P(2,2)=20000.
      DO 1100 I=1,ORDER
            HMAT(1,I)=0.
            HT(I,1)=0.
1100  CONTINUE
      HMAT(1,1)=1.
      HT(1,1)=1.
      WHILE(T<=TF)
           XOLD=X
           XDOLD=XD
           XDD=.0034*32.2*XD*XD*EXP(-X/22000.)/(2.*BETA)-32.2
           X=X+H*XD
```

(*continued*)

Listing 7.5 (*Continued*)

```
          XD=XD+H*XDD
          T=T+H
          XDD=.0034*32.2*XD*XD*EXP(-X/22000.)/(2.*BETA)-32.2
          X=.5*(XOLD+X+H*XD)
          XD=.5*(XDOLD+XD+H*XDD)
          S=S+H
          IF(S>=(TS-.00001))THEN
               S=0.
               RHOH=.0034*EXP(-XH/22000.)
               F21=-32.2*RHOH*XDH*XDH/(44000.*BETA)
               F22=RHOH*32.2*XDH/BETA
               PHI(1,1)=1.
               PHI(1,2)=TS
               PHI(2,1)=F21*TS
               PHI(2,2)=1.+F22*TS
               Q(1,1)=PHIS*TS*TS*TS/3.
               Q(1,2)=PHIS*(TS*TS/2.+F22*TS*TS*TS/3.)
               Q(2,1)=Q(1,2)
               Q(2,2)=PHIS*(TS+F22*TS*TS+F22*F22*TS*TS*TS/3.)
               CALL MATTRN(PHI,ORDER,ORDER,PHIT)
               CALL MATMUL(PHI,ORDER,ORDER,P,ORDER,ORDER,
               PHIP)
               CALL MATMUL(PHIP,ORDER,ORDER,PHIT,ORDER,
  1              ORDER,PHIPPHIT)
               CALL MATADD(PHIPPHIT,ORDER,ORDER,Q,M)
               CALL MATMUL(HMAT,1,ORDER,M,ORDER,ORDER,HM)
               CALL MATMUL(HM,1,ORDER,HT,ORDER,1,HMHT)
               HMHTR=HMHT(1,1)+SIGNOISE*SIGNOISE
               HMHTRINV=1./HMHTR
               CALL MATMUL(M,ORDER,ORDER,HT,ORDER,1,MHT)
               DO 150 I=1,ORDER
                    GAIN(I,1)=MHT(I,1)*HMHTRINV
150            CONTINUE
               CALL MATMUL(GAIN,ORDER,1,HMAT,1,ORDER,KH)
               CALL MATSUB(IDN,ORDER,ORDER,KH,IKH)
               CALL MATMUL(IKH,ORDER,ORDER,M,ORDER,
                 ORDER,P)
               CALL GAUSS(XNOISE,SIGNOISE)
               CALL PROJECT(T,TS,XH,XDH,BETA,XB,XDB,XDDB)
               RES=X+XNOISE-XB
               XH=XB+GAIN(1,1)*RES
               XDH=XDB+GAIN(2,1)*RES
               ERRX=X-XH
               SP11=SQRT(P(1,1))
               ERRXD=XD-XDH
               SP22=SQRT(P(2,2))
               WRITE(9,*)T,X,XH,XD,XDH
               WRITE(1,*)T,X,XH,XD,XDH
               WRITE(2,*)T,ERRX,SP11,-SP11,ERRXD,SP22,-SP22
          ENDIF
```

(*continued*)

Listing 7.5 (*Continued*)

```
        END DO
        PAUSE
        CLOSE(1)
        END

        SUBROUTINE PROJECT(TP,TS,XP,XDP,BETA,XH,XDH,XDDH)
        IMPLICIT REAL*8 (A-H)
        IMPLICIT REAL*8 (O-Z)
        T=0.
        X=XP
        XD=XDP
        H=.001
        WHILE(T<=(TS-.0001))
                XDD=.0034*32.2*XD*XD*EXP(-X/22000.)/(2.*BETA)-32.2
                XD=XD+H*XDD
                X=X+H*XD
                T=T+H
        END DO
        XH=X
        XDH=XD
        XDDH=XDD
        RETURN
        END
C SUBROUTINE GAUSS IS SHOWN IN LISTING 1.8
C SUBROUTINE MATTRN IS SHOWN IN LISTING 1.3
C SUBROUTINE MATMUL IS SHOWN IN LISTING 1.4
C SUBROUTINE MATADD IS SHOWN IN LISTING 1.1
C SUBROUTINE MATSUB IS SHOWN IN LISTING 1.2
```

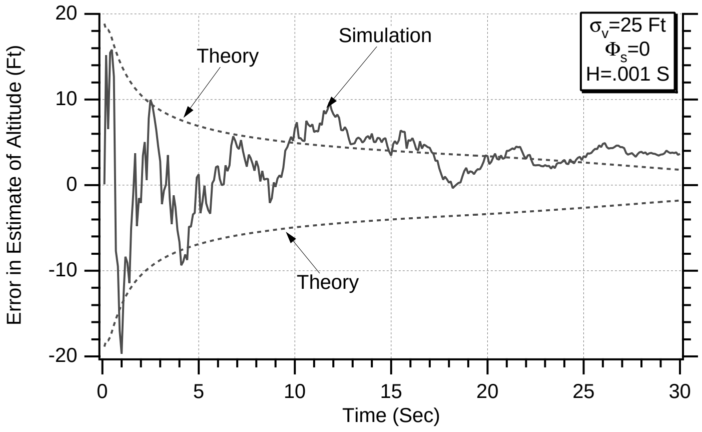

Fig. 7.21 Divergence has been eliminated in altitude estimate by use of more accurate state propagation methods.

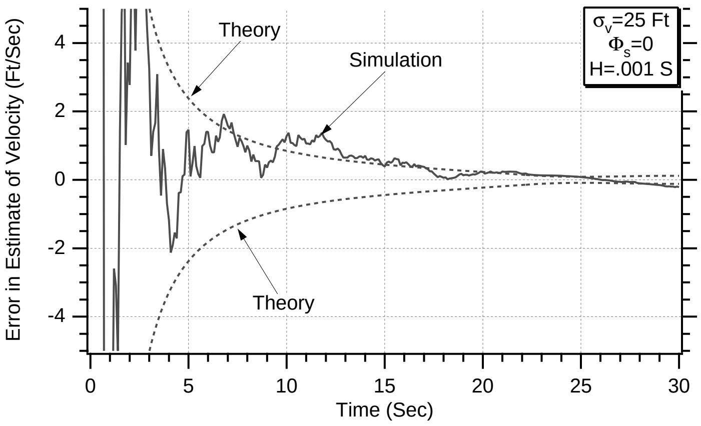

Fig. 7.22 Divergence has been eliminated in velocity estimate by use of more accurate state propagation methods.

nonlinear differential equations forward one sampling interval to produce the estimates. Second-order Runge–Kutta numerical integration with an integration step size of 0.001 s is used to propagate the actual states forward. The estimates are initially set equal to the actual states so that a perfect propagation subroutine would yield perfect state estimates. The parameter HP determines the integration interval used by the subroutine. When the integration interval HP is small enough the errors in the estimates of altitude ERRX and velocity ERRXD should go to zero.

Cases were run with Listing 7.4 in which the subroutine integration interval HP was made a parameter and varied from 0.1 to 0.001 s. We can see from Figs. 7.19 and 7.20 that a value of 0.001 s must be used for the subroutine integration interval to keep the errors in the estimates of altitude and velocity near zero.

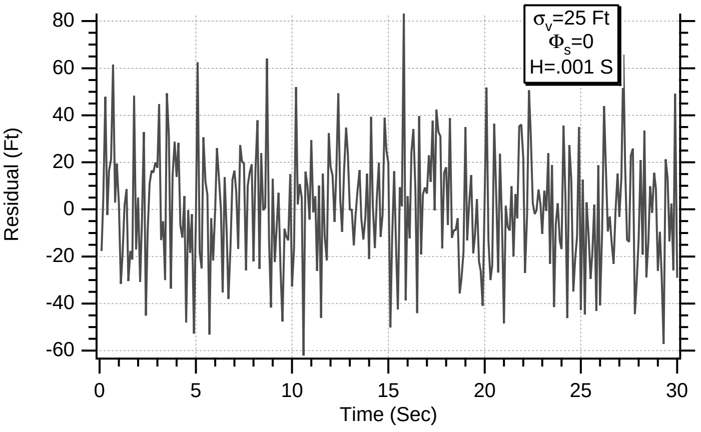

Fig. 7.23 More accurate state propagation ensures residual has zero mean even though Kalman gains approach zero.

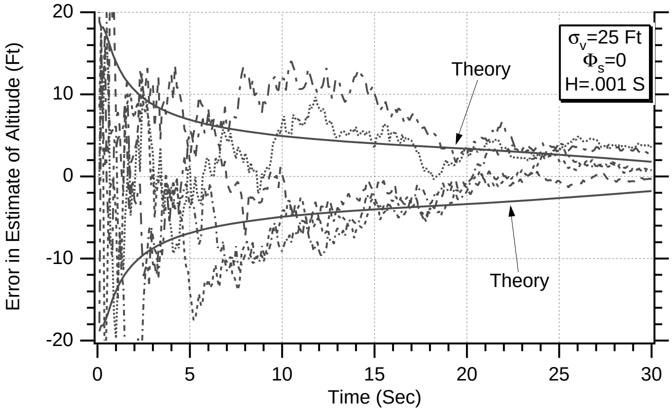

Fig. 7.24 Monte Carlo results are within theoretical bounds for error in estimate of position.

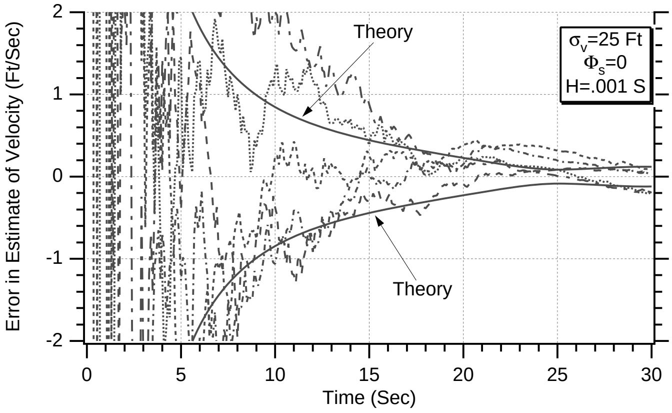

Fig. 7.25 Monte Carlo results are within theoretical bounds for error in estimate of velocity.

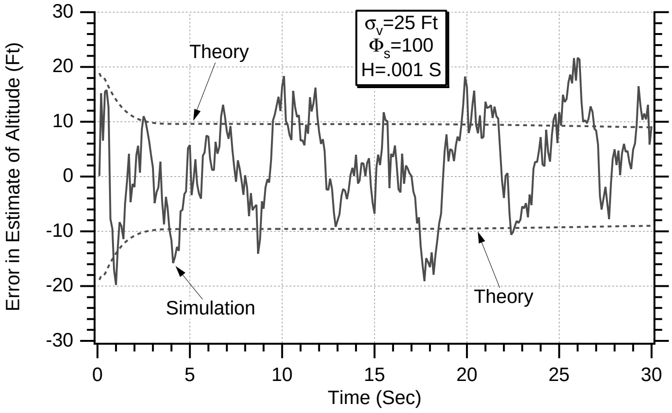

Fig. 7.26 Adding process noise increases error in estimate of position.

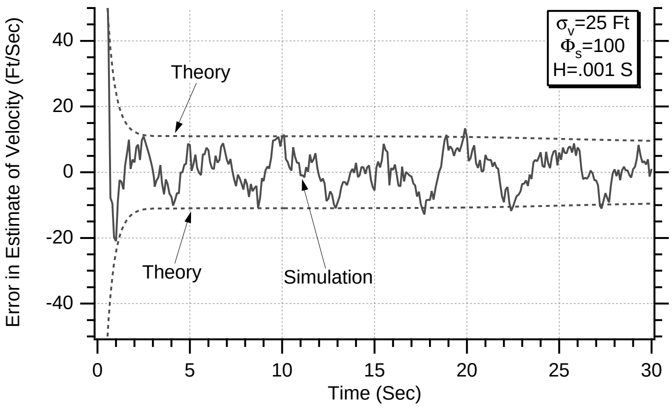

Fig. 7.27 Adding process noise increases error in estimate of velocity.

Therefore, to avoid additional errors in our extended Kalman filter, we will use a value of 0.001 s for the integration interval in the propagation subroutine.

The extended Kalman filter simulation of Listing 7.2 was modified to account for the more accurate method of integration required to propagate the states between measurements and appears in Listing 7.5. The changes to the original simulation are highlighted in bold.

The nominal case of Listing 7.5 was run in which there was 25 ft of measurement noise and no process noise. The integration interval was set to 0.001 s in the integration method for projecting the states (see subroutine PROJECT). We can see from Figs. 7.21–7.22 that the preceding divergence problem in the errors in the estimates of altitude and velocity has now been eliminated. This simple experiment shows that the accuracy of the propagation method for the filter states is extremely important when there is little or no process noise.

We can also see from Fig. 7.23 that the more accurate state propagation ensures that the residual has an average value of zero. This is now true even though the Kalman gains approach zero (see Fig. 7.10).

Although it was clear (at least to the authors) in Figs. 7.21 and 7.22 that the divergence problem disappeared when accurate state propagation was used, some might not be convinced from those single-run results. Listing 7.5 was modified so that it could be run in the Monte Carlo mode for this particular example. The results of Figs. 7.21 and 7.22 were repeated for five different runs, and the error in the estimate results appears in Figs. 7.24 and 7.25. From these results it is clearer that the simulated errors in the estimates of altitude and velocity are within the theoretical bounds approximately 68% of the time. In particular, note that the scale on Fig. 7.25 has been expanded to see how close the error in the estimate of velocity is to the theoretical values near the end of the measurements at 30 s.

Another case was run to see how the new filter with the more accurate state propagation methods performed in the presence of process noise. The process noise used was $\Phi_s = 100$, which is identical to what was used before to solve the divergence problem. We can see from Figs. 7.26 and 7.27 that the errors in the estimates of altitude and velocity are approximately what they were before (see Figs. 7.13 and 7.14) when process noise was used to solve the divergence problem. However, decreasing the integration step size from 0.1 to 0.001 s for the state propagation is considerably more costly in terms of computer throughput than simply using a step size of 0.1 s with process noise.

Summary

The extended Kalman-filtering equations were presented in this chapter, and a numerical example was used to illustrate the operation and characteristics of the filter. The extended Kalman filter was intentionally designed and tested in different ways to illustrate things that can go wrong. A divergence problem was presented, and it was shown that either more accurate integration in the state propagation or the addition of process noise were legitimate methods for improving filter performance. Also, adding more terms to the Taylor-series expansion for the fundamental matrix not only did not solve the divergence problem but, for this example, hardly had any influence on the results.

References

[1]Gelb., A., *Applied Optimal Estimation*, Massachusetts Inst. of Technology Press, Cambridge, MA, 1974, pp. 180–228.

[2]Zarchan, P., *Tactical and Strategic Missile Guidance*, 3rd ed., Progress in Astronautics and Aeronautics, AIAA, Reston, VA, 1998, pp. 373–387.