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

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

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

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

where the derivative is obtained from

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

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]

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

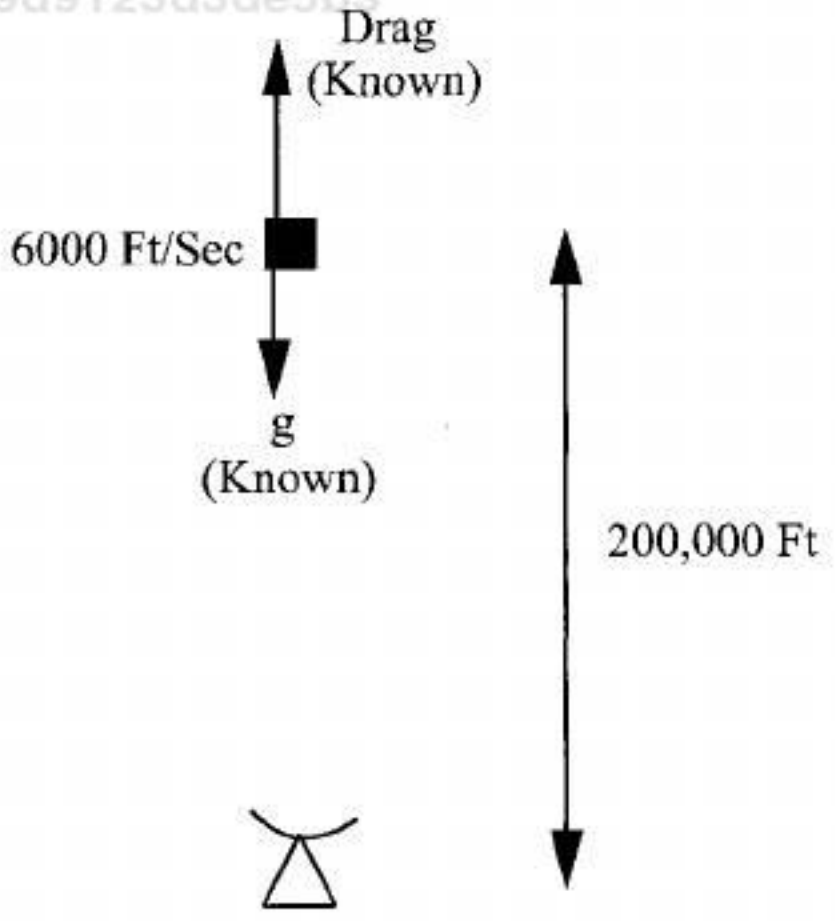

Fig. 7.1 Radar tracking falling object in presence of drag.

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
```


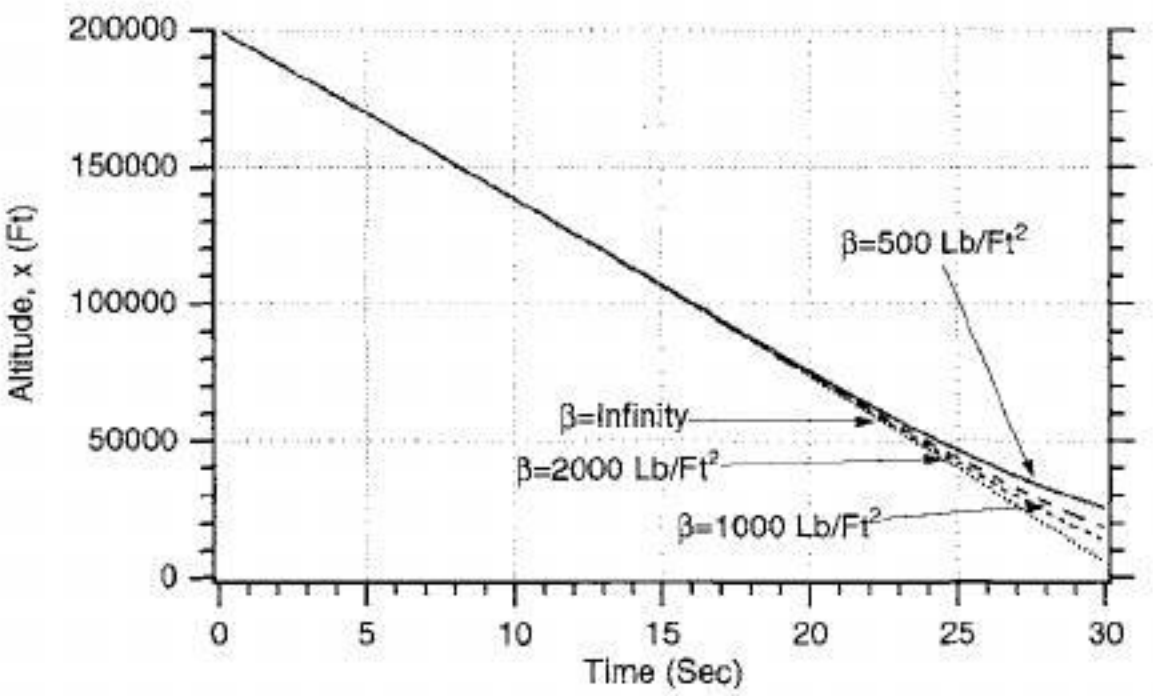

Fig. 7.2 Increasing drag decreases lowest altitude that can be reached in 30 s.

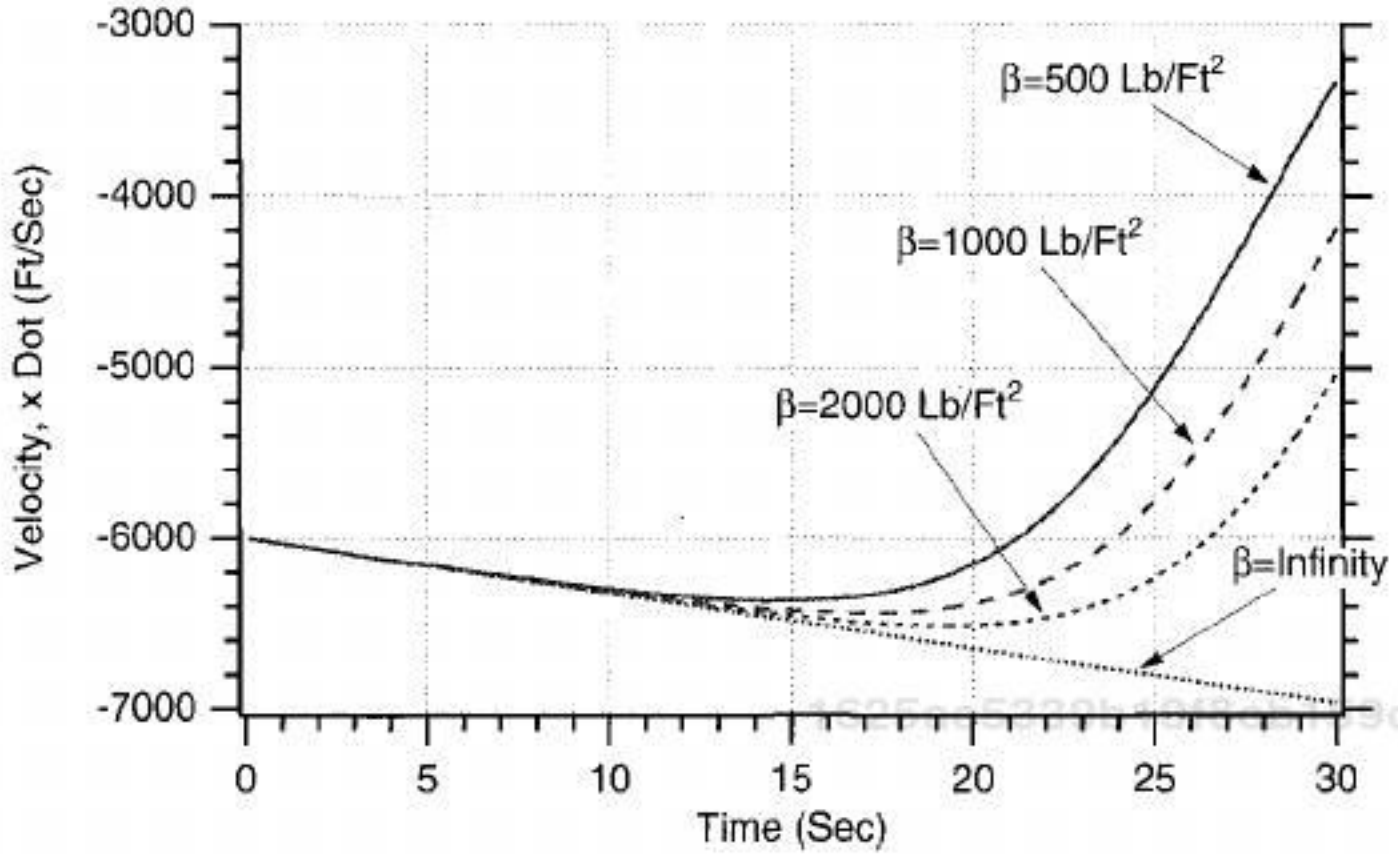

Fig. 7.3 Increasing drag reduces velocity at lower altitudes.

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

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

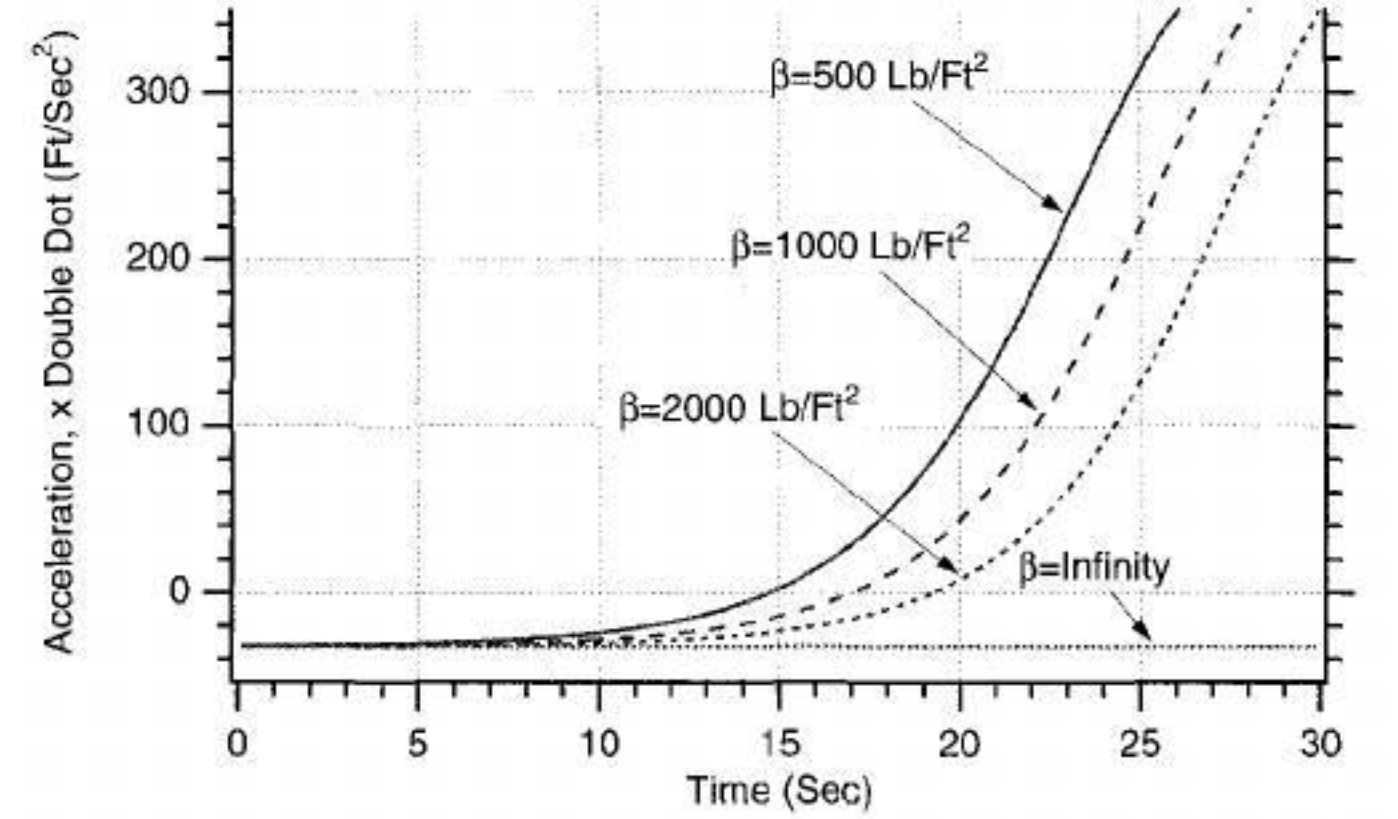

Fig. 7.4 Drag causes high decelerations.

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

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

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,
     1  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,
     1  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

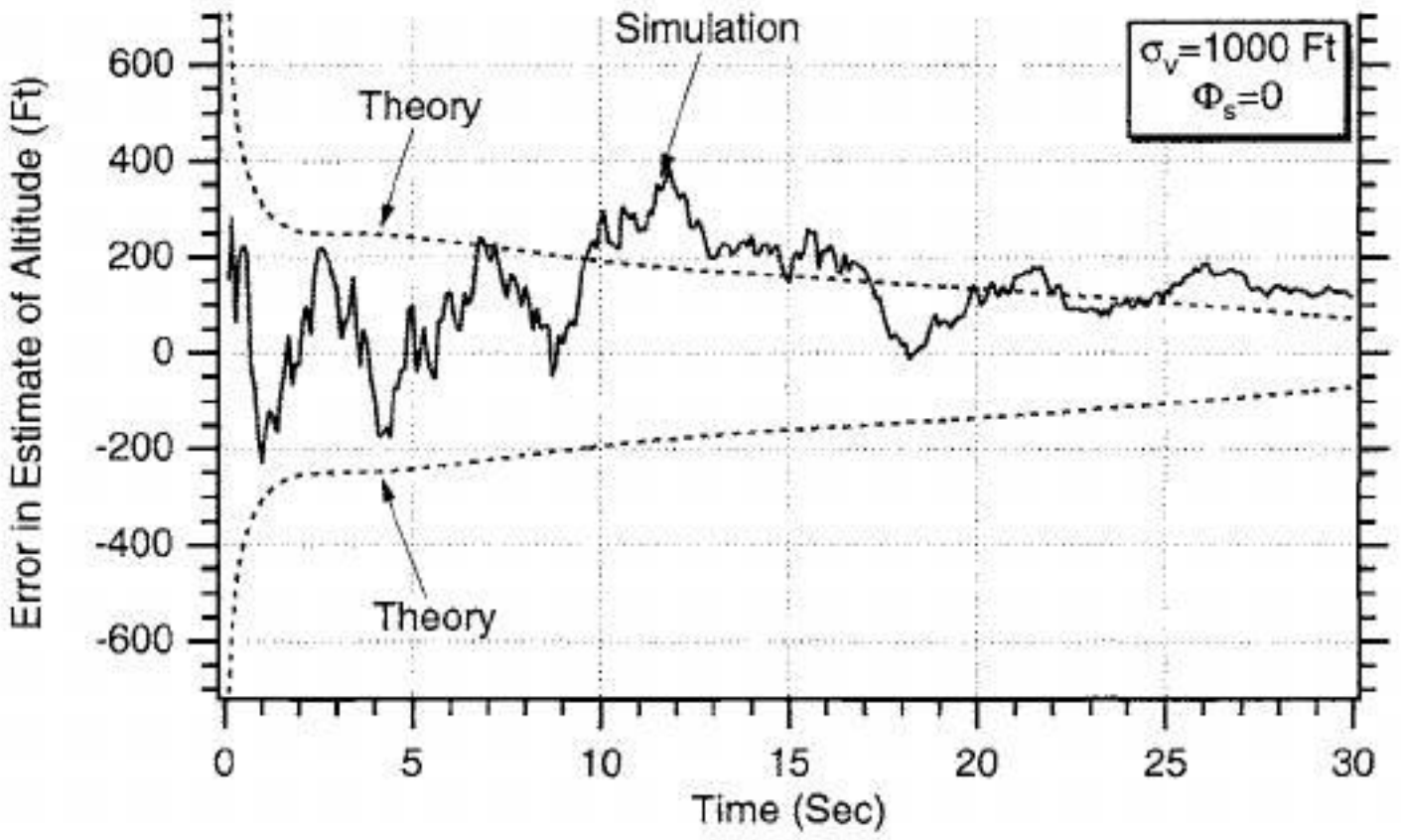

Fig. 7.5 Error in the estimate of position appears to be within the theoretical bounds.

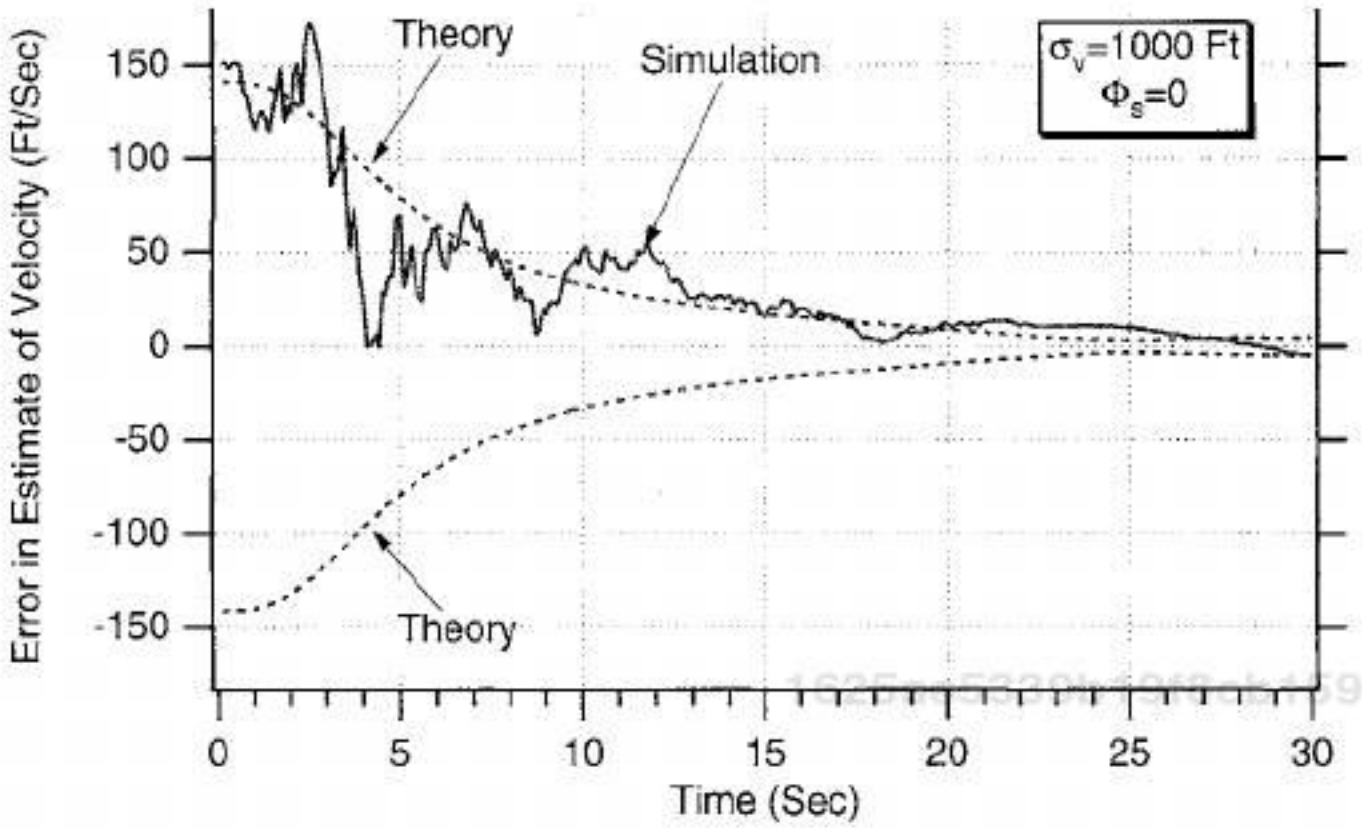

Fig. 7.6 Error in the estimate of velocity appears to be within the theoretical bounds.

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

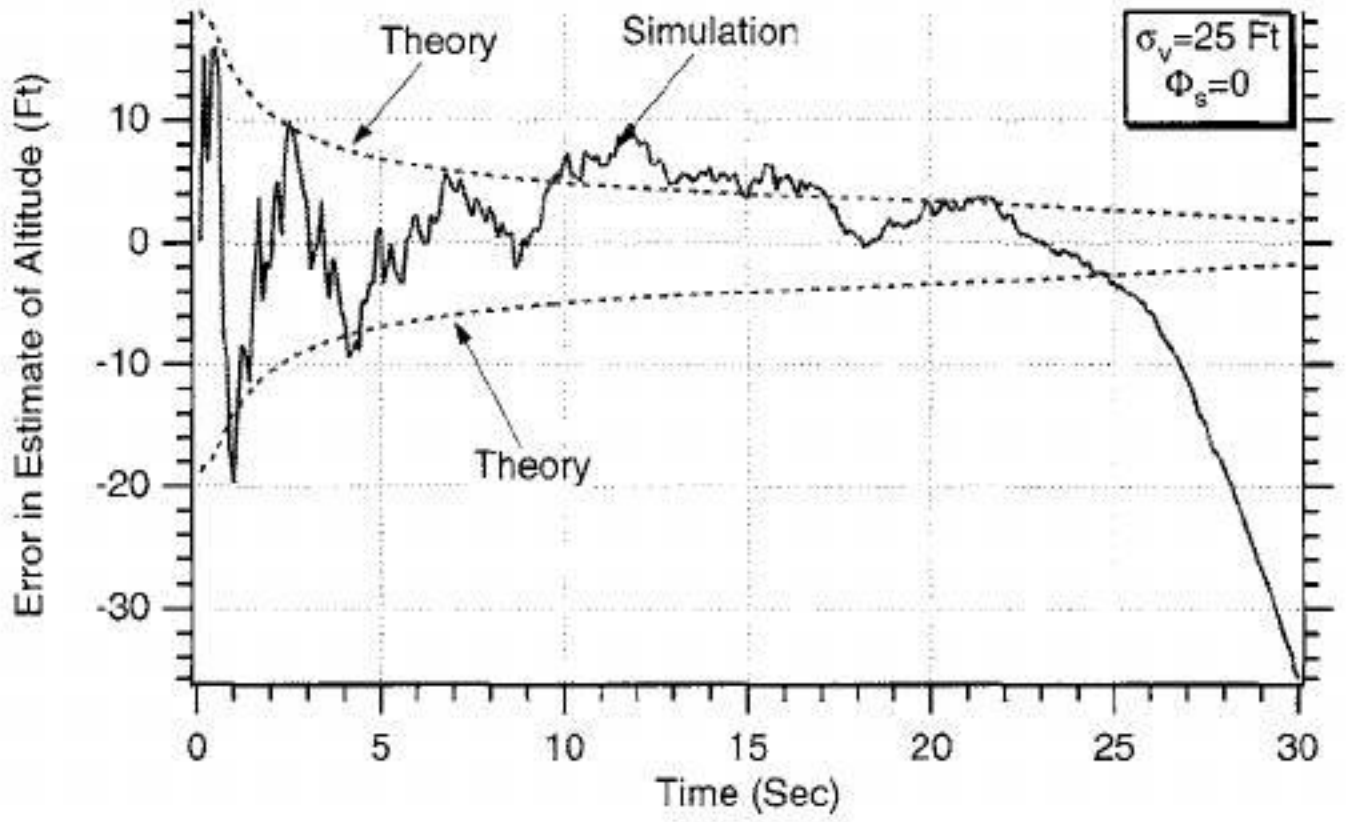

Fig. 7.7 Error in estimate of position appears to be diverging.

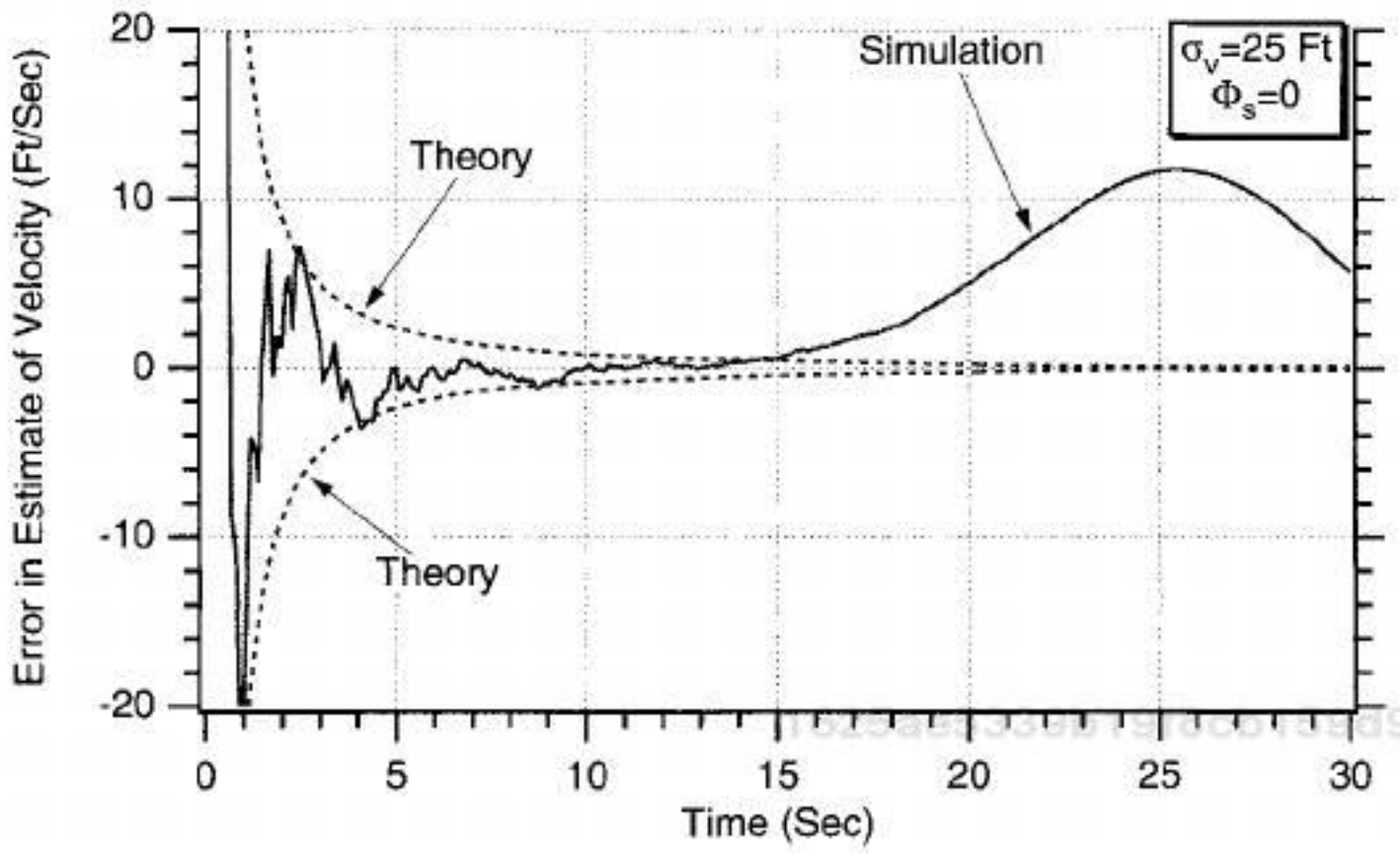

Figure 7.8 Error in estimate of velocity appears to be growing for most of the flight.

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

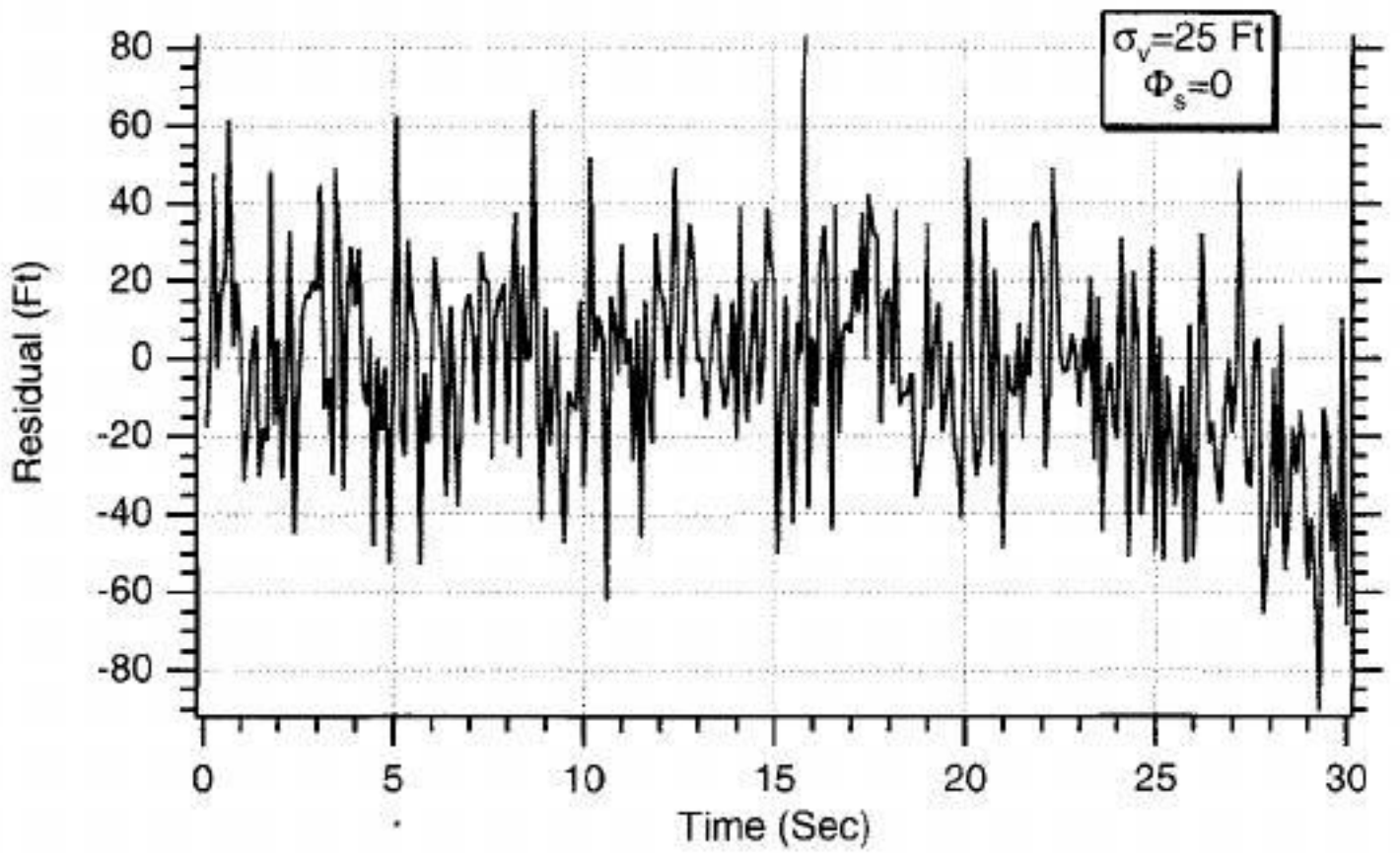

Fig. 7.9 Residual drifts from zero after 20 s.

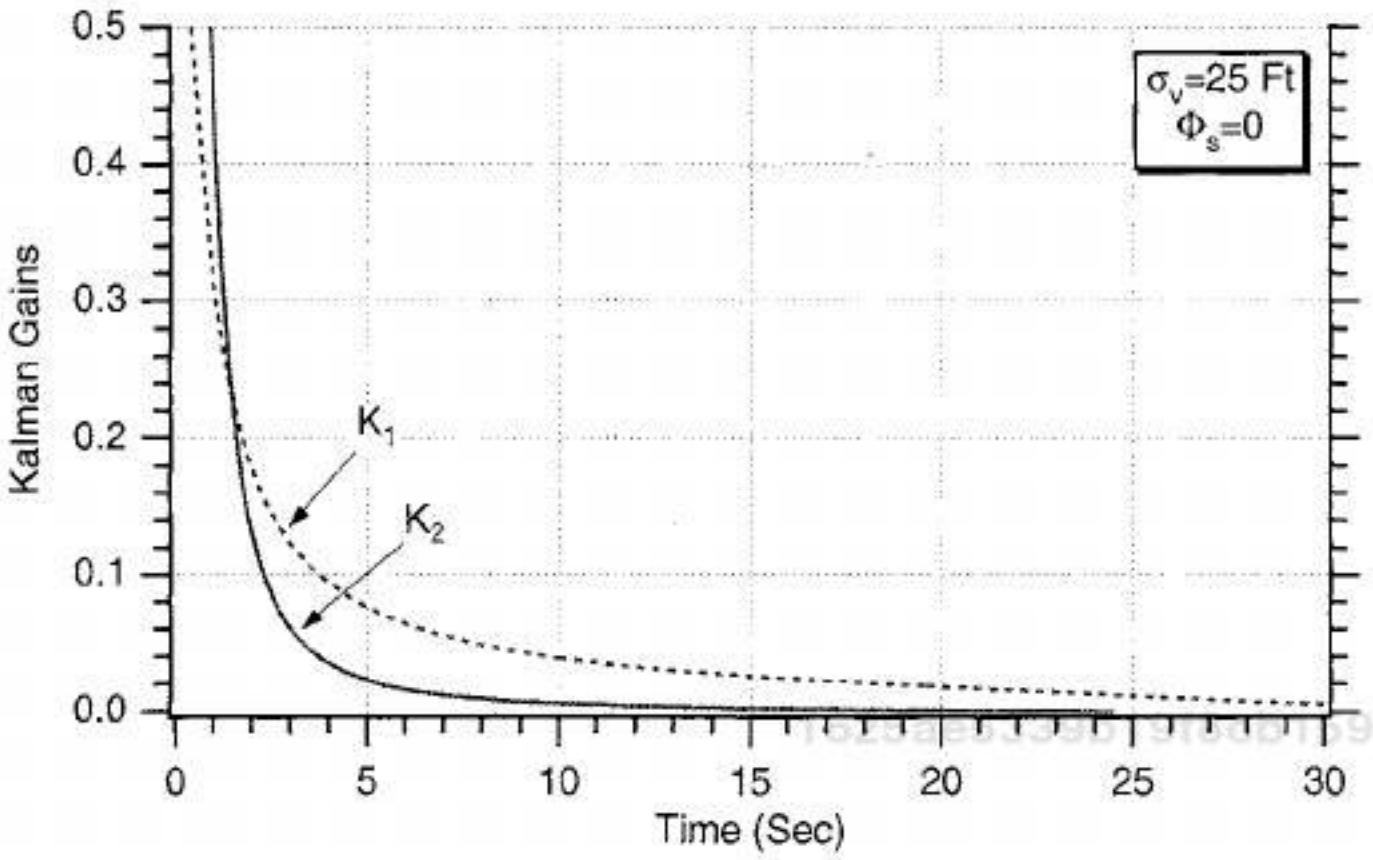

Fig. 7.10 Both Kalman gains approach zero.

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

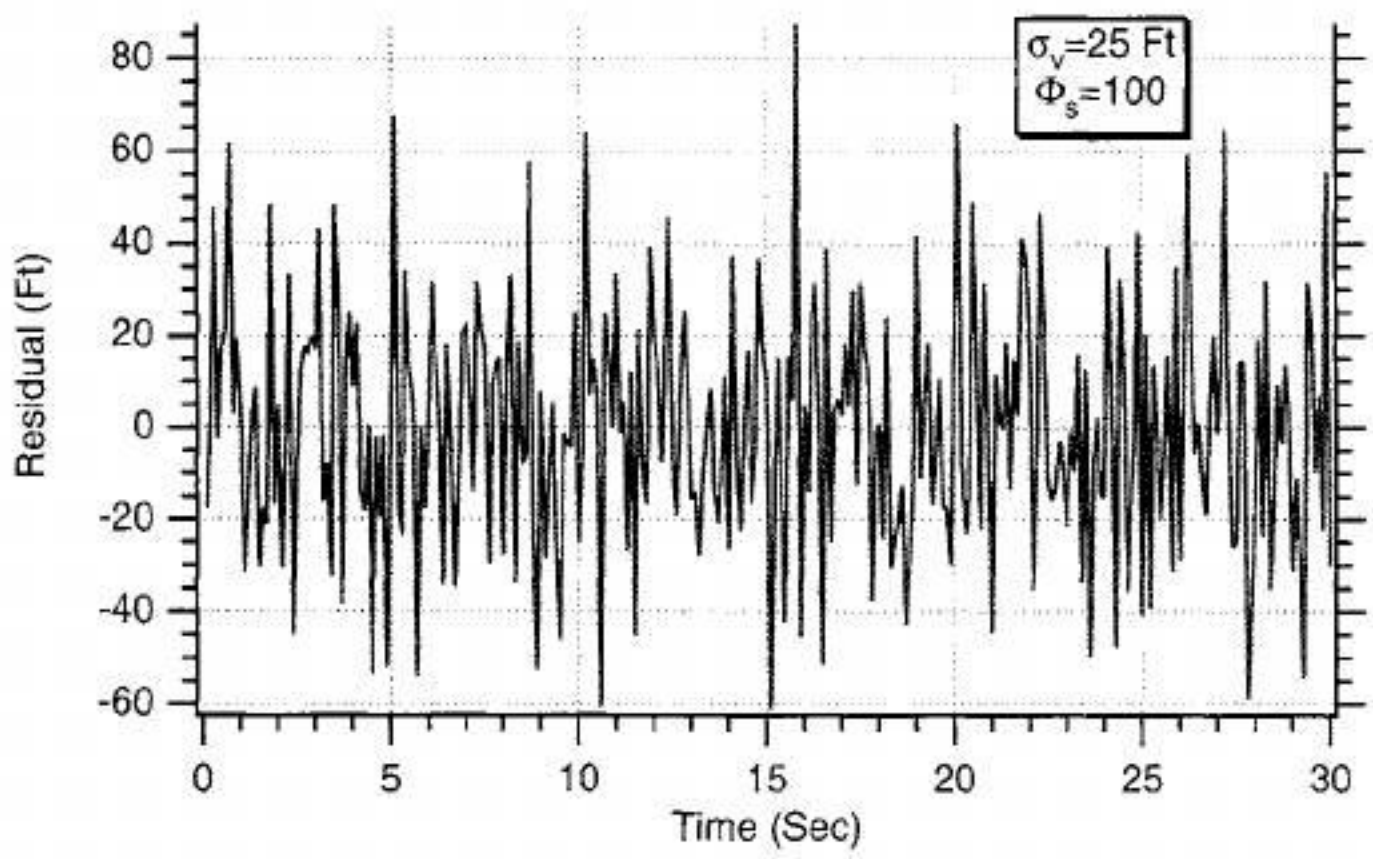

Fig. 7.11 Adding process noise prevents residual from drifting away from zero.

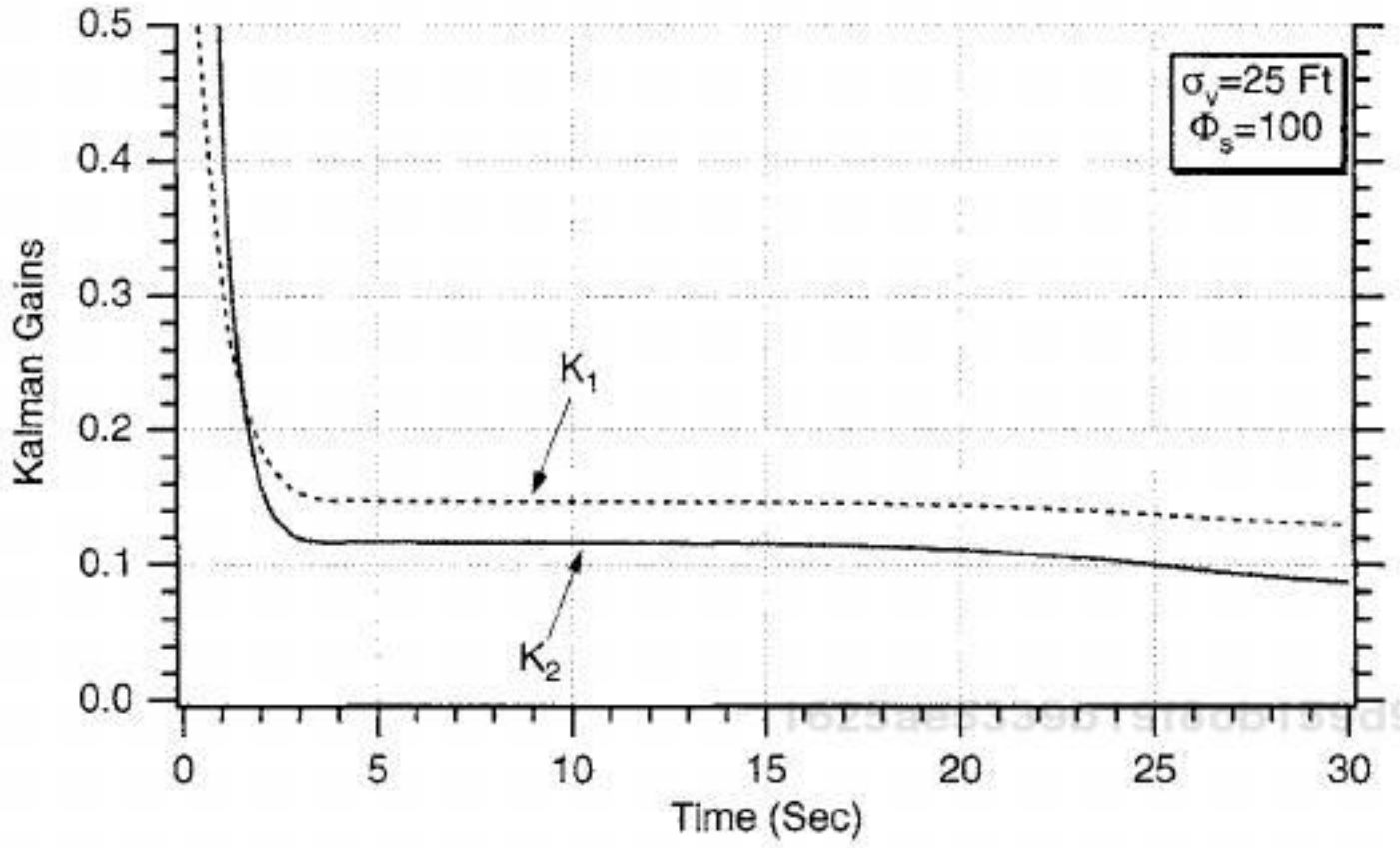

Fig. 7.12 Process noise prevents Kalman gains from going to zero.

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

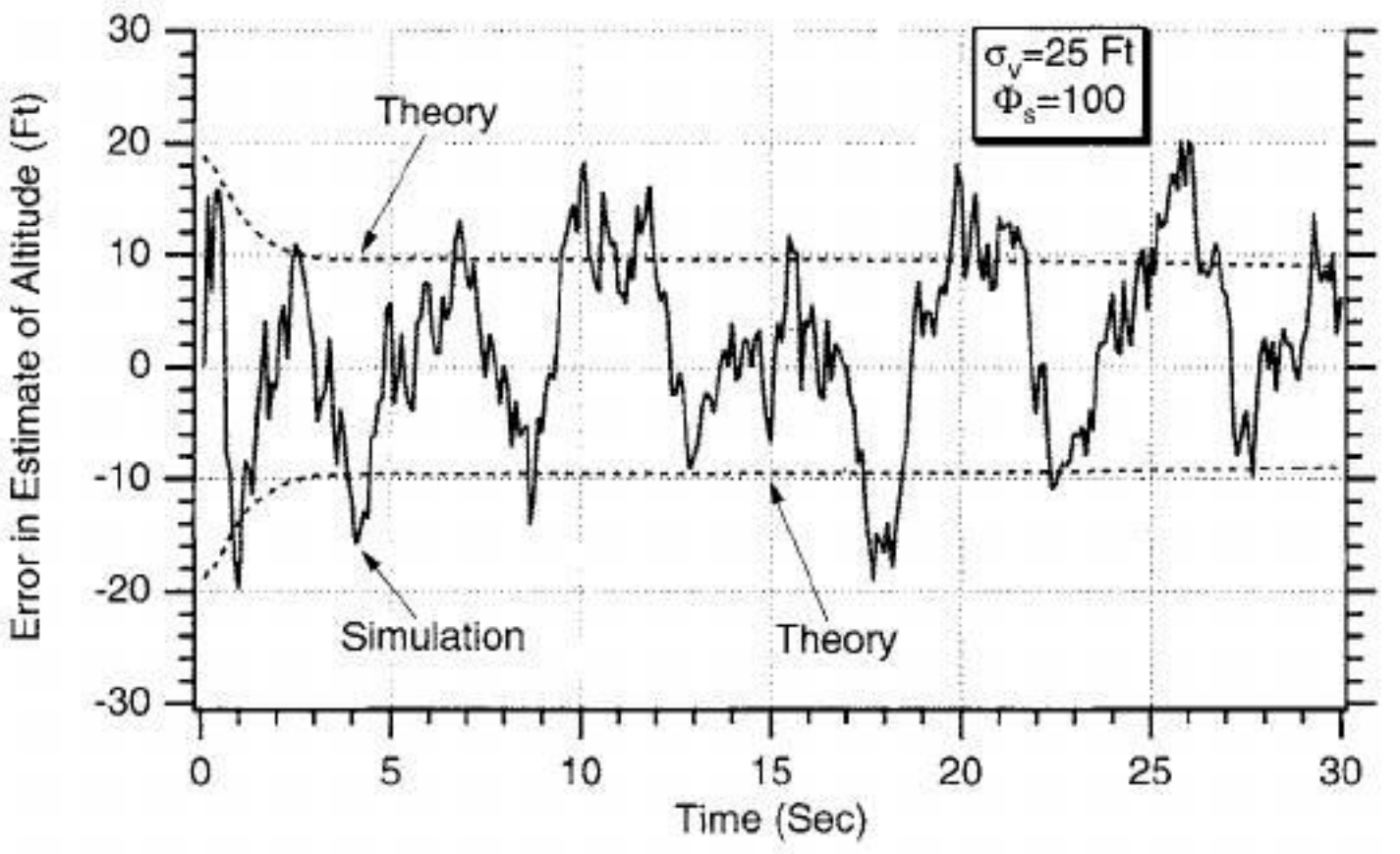

Fig. 7.13 Adding process noise eliminates filter divergence in position.

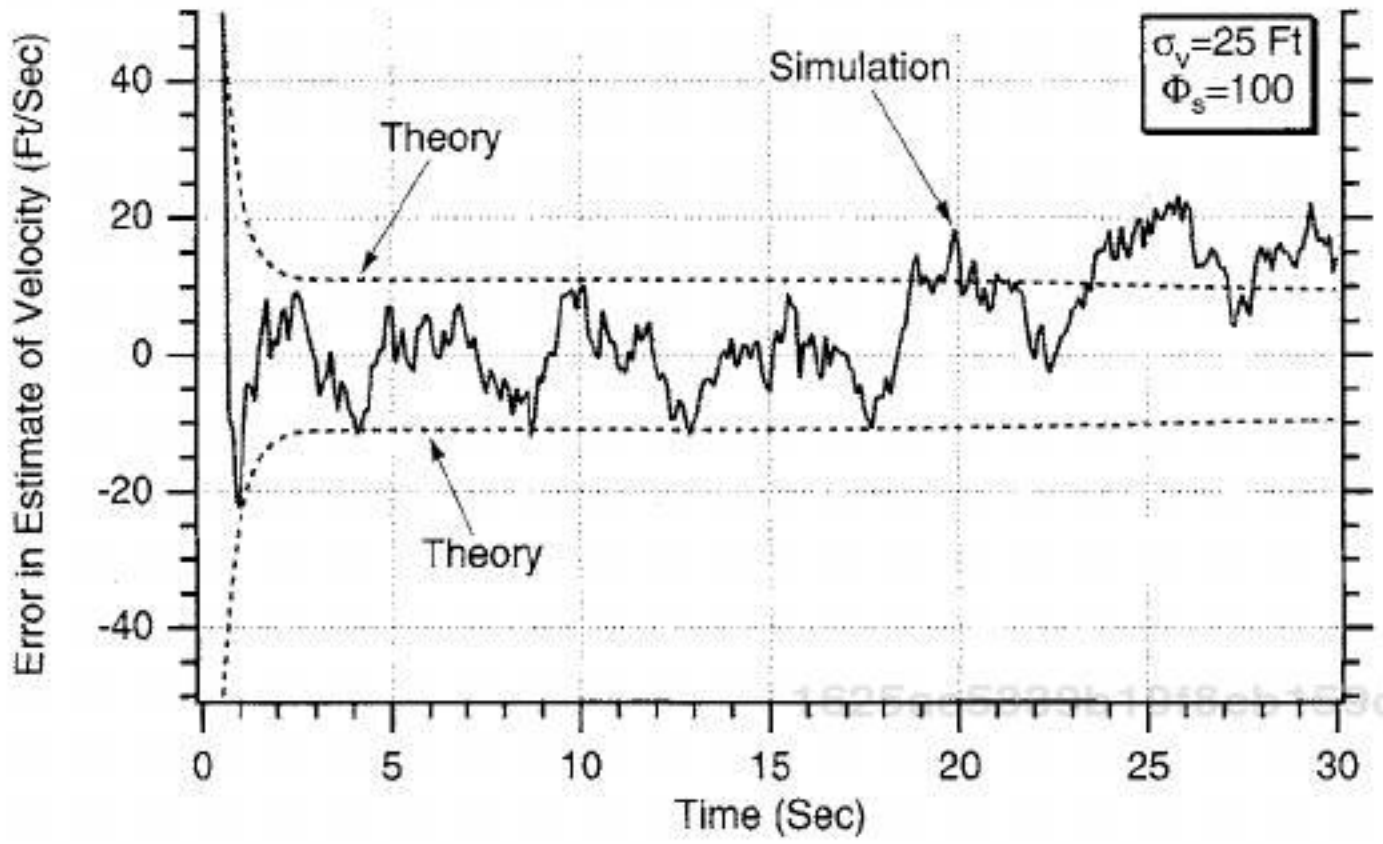

Fig. 7.14 Adding process noise eliminates filter divergence in velocity.

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

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

or more simply

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

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

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

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,
     1   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
```

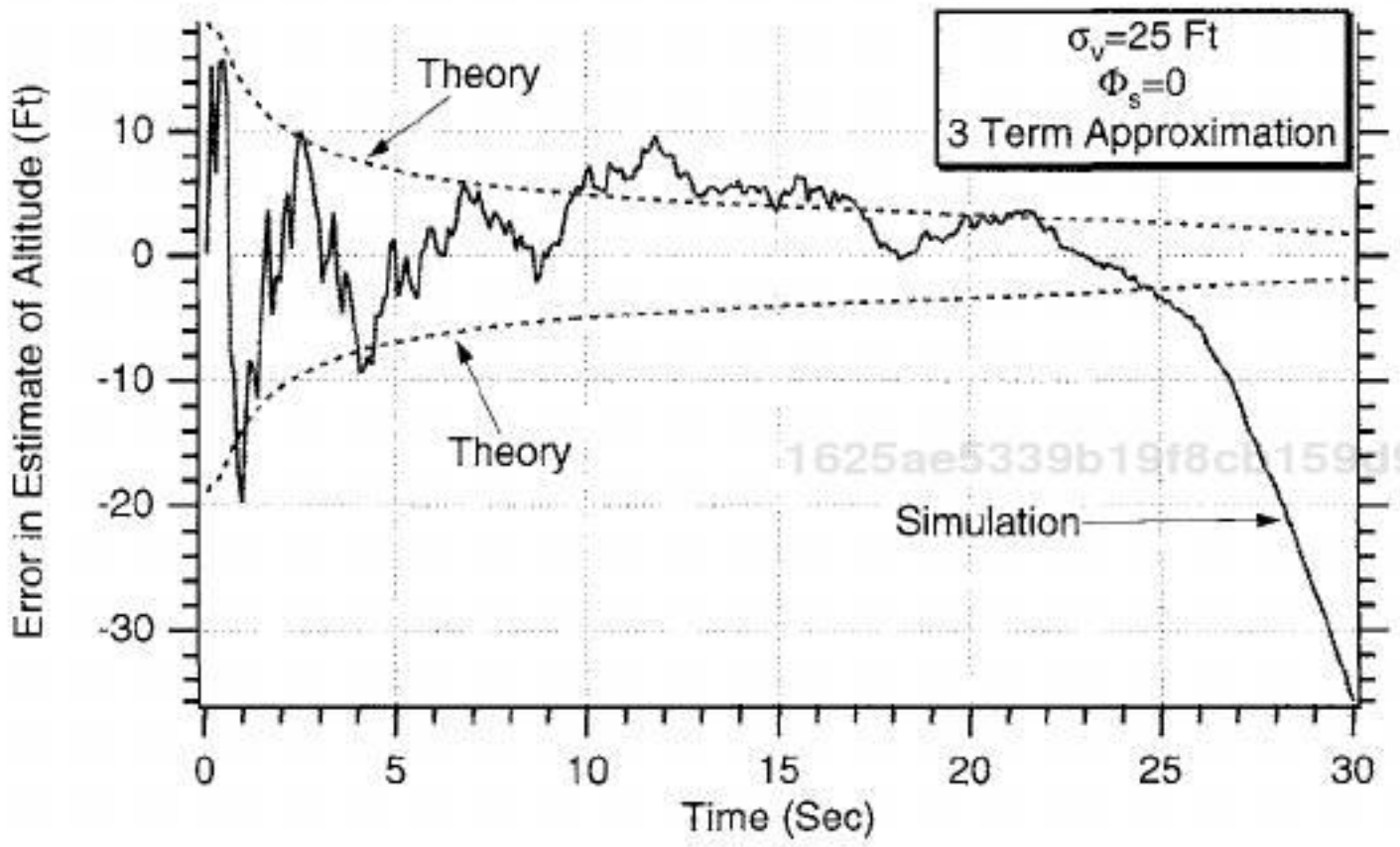

Fig. 7.15 Having three-term approximation for fundamental matrix does not remove filter divergence when there is no process noise.

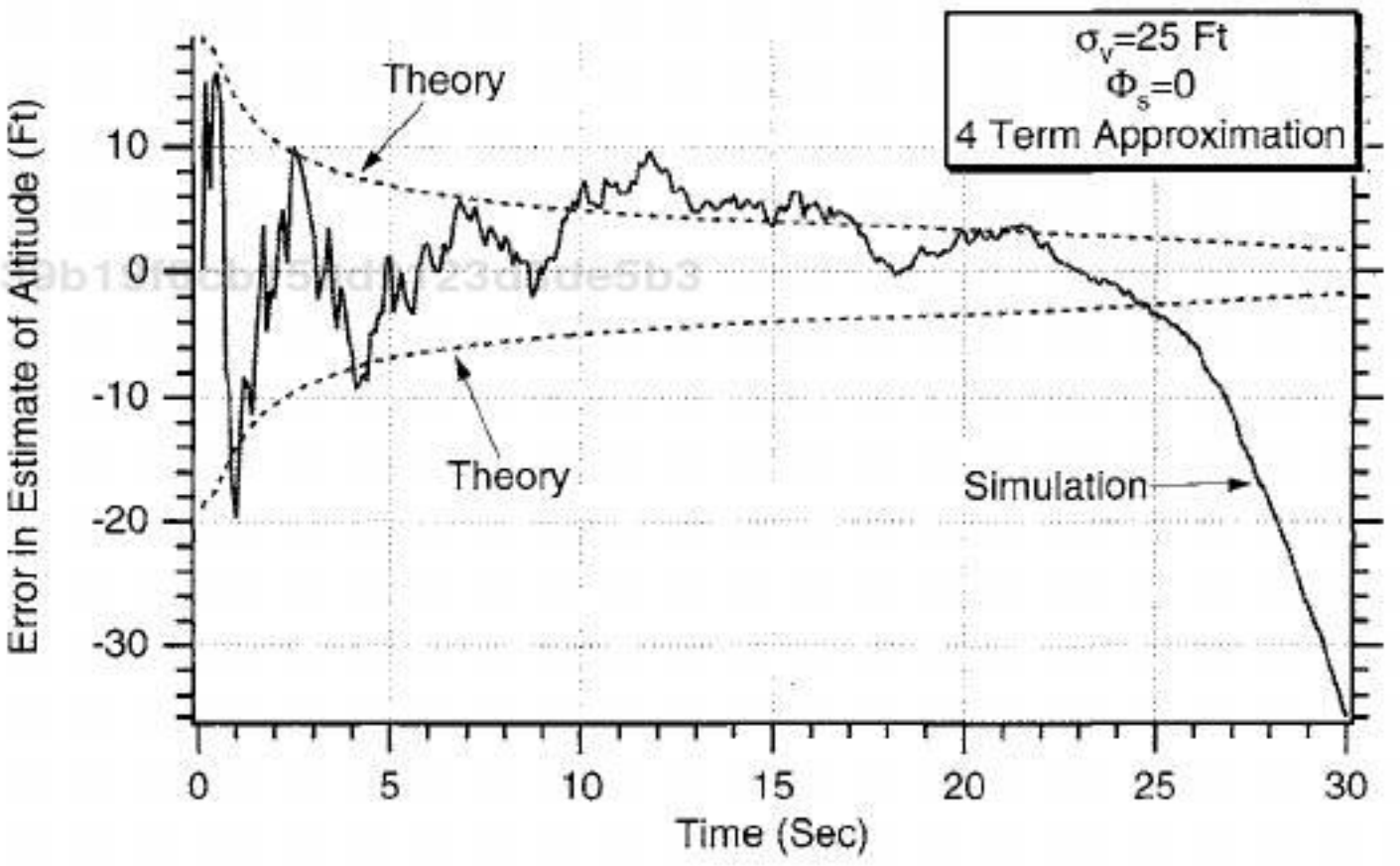

Fig. 7.16 Having four-term approximation for fundamental matrix does not remove filter divergence when there is no process noise.

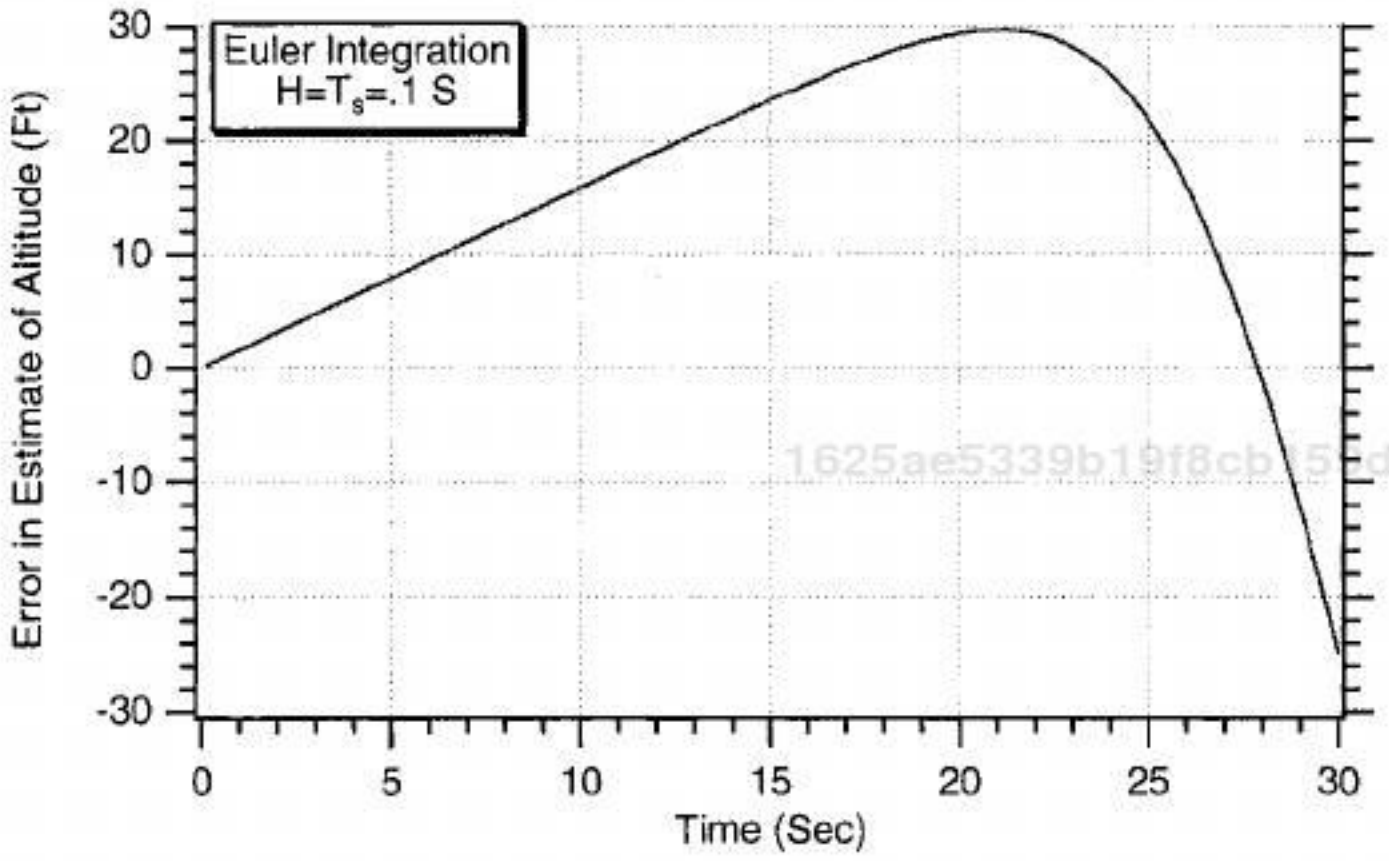

Fig. 7.17 Euler integration with an integration interval of 0.1 s is not adequate for eliminating altitude errors.

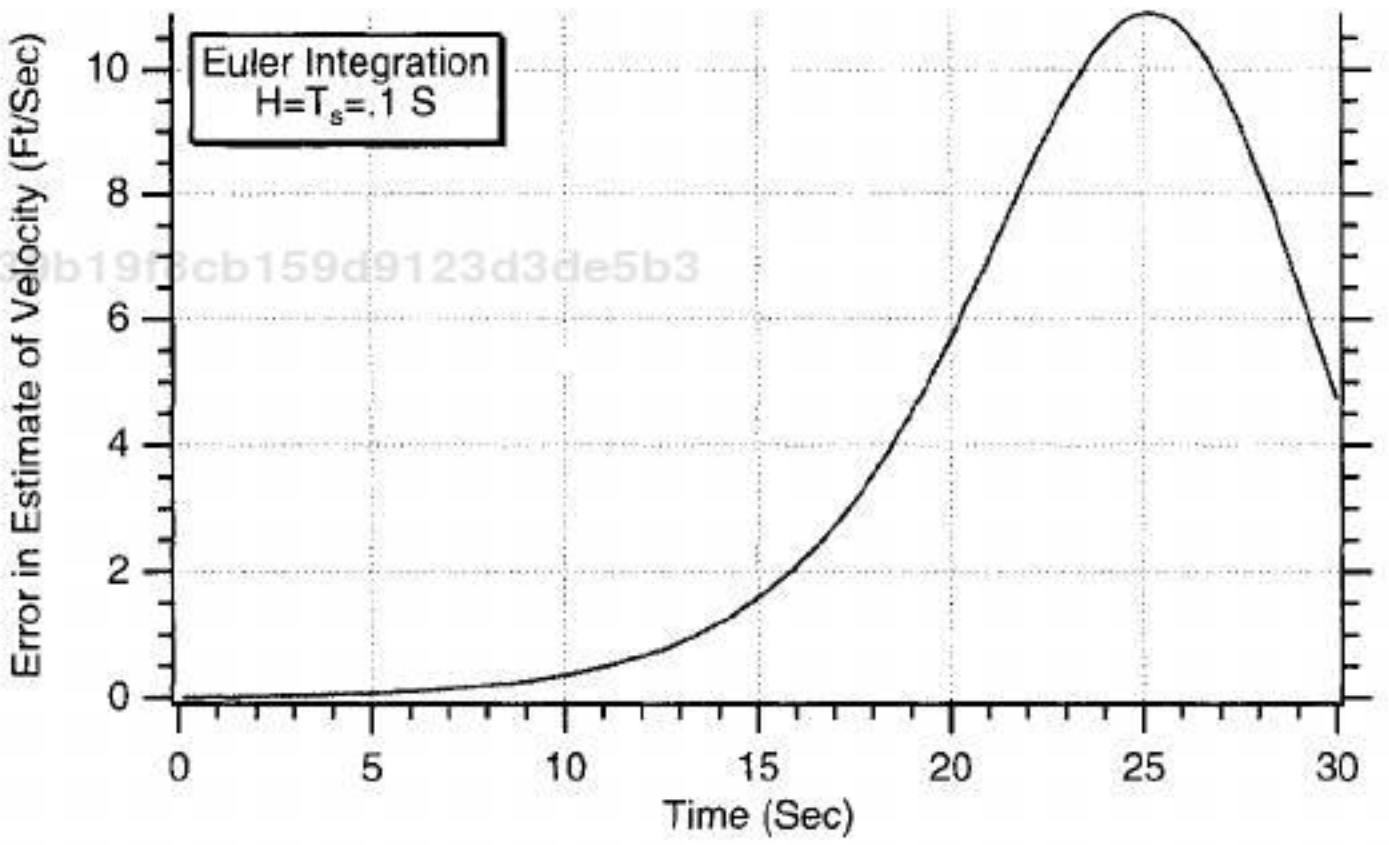

Fig. 7.18 Euler integration with an integration interval of .1 s is not adequate for eliminating velocity errors.

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

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

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

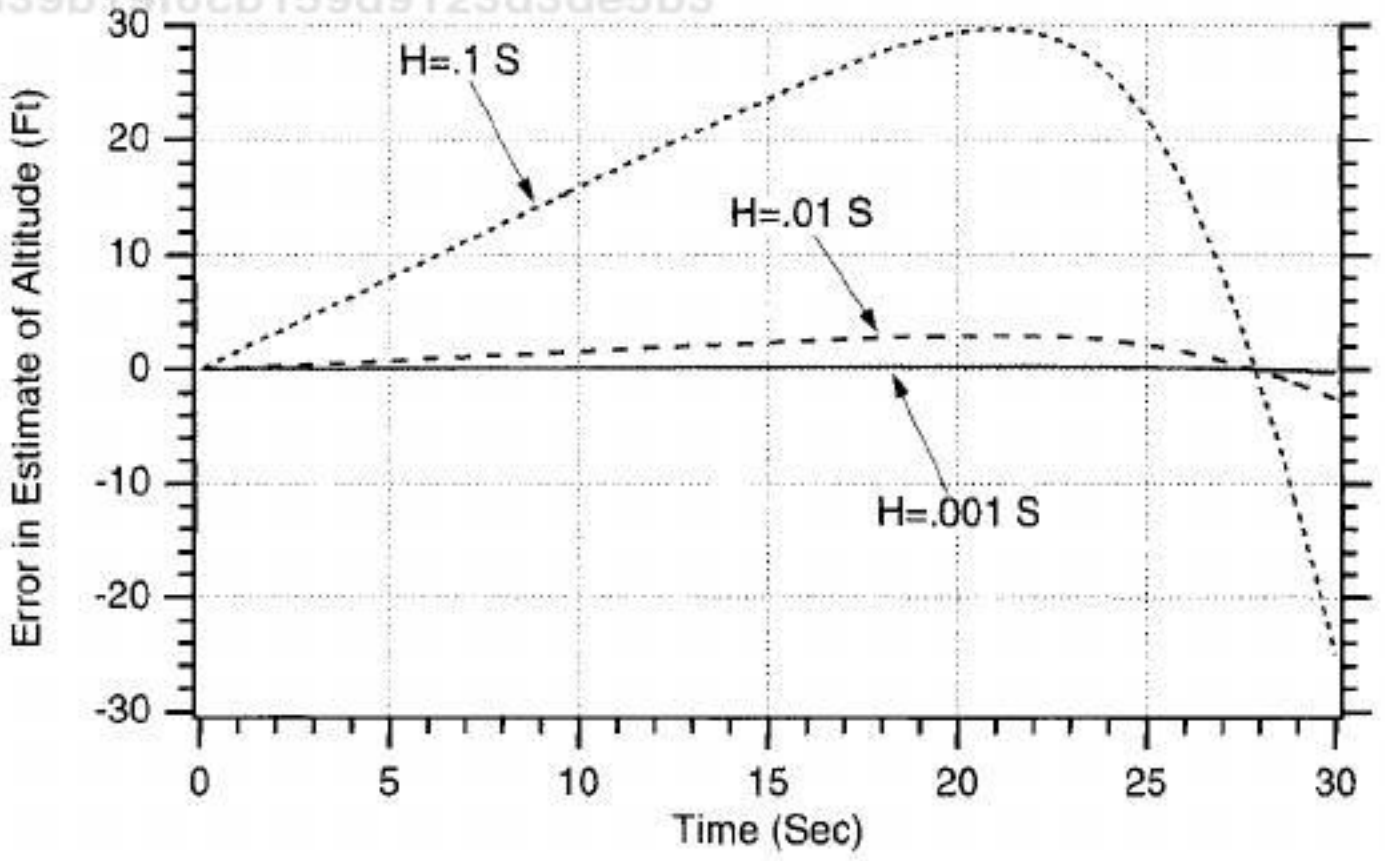

Fig. 7.19 Integration step size in propagation subroutine must be reduced to 0.001 s to keep errors in estimate of altitude near zero.

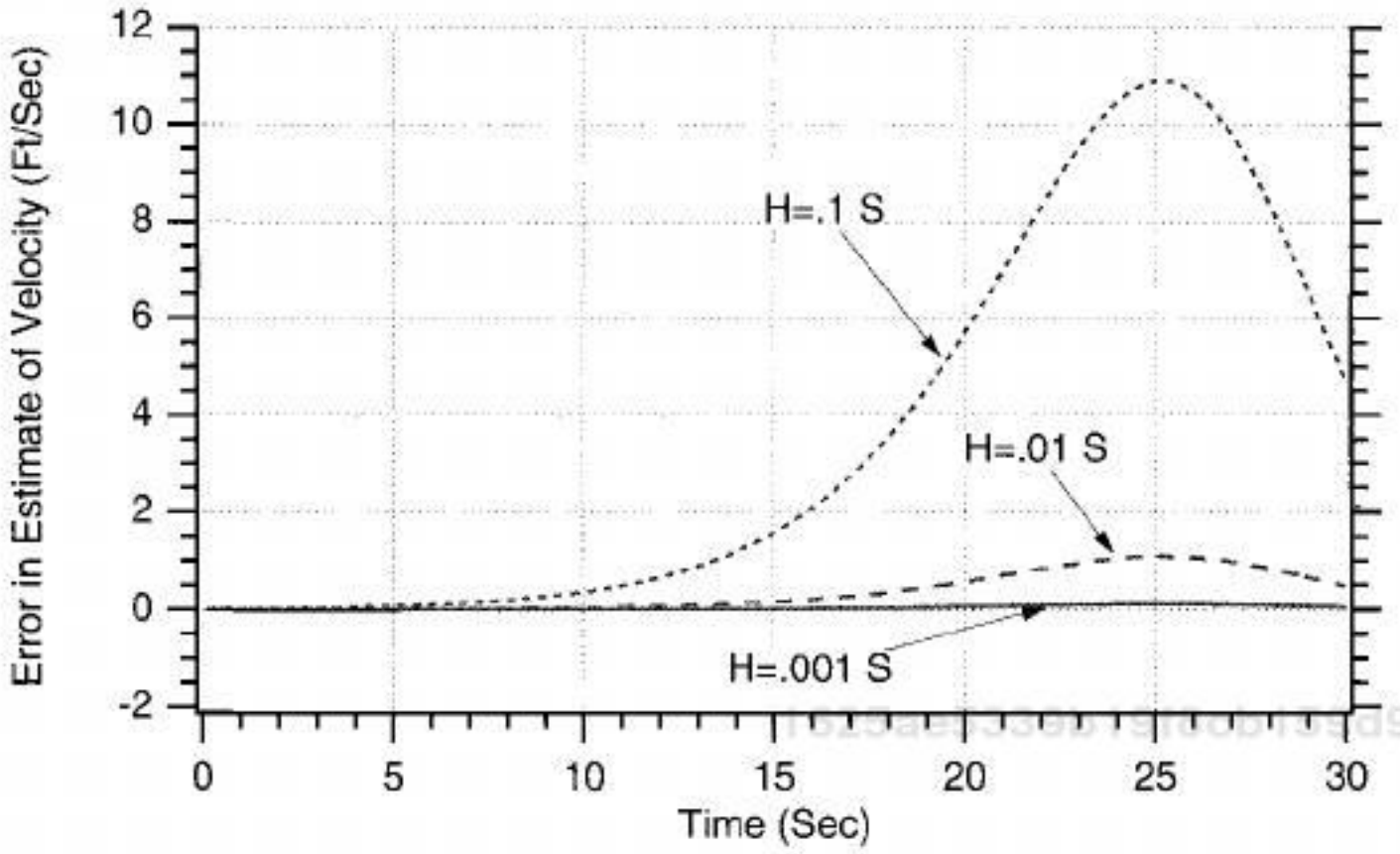

Fig. 7.20 Integration step size in propagation subroutine must be reduced to 0.001 s to keep errors in estimate of velocity near zero.

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
```


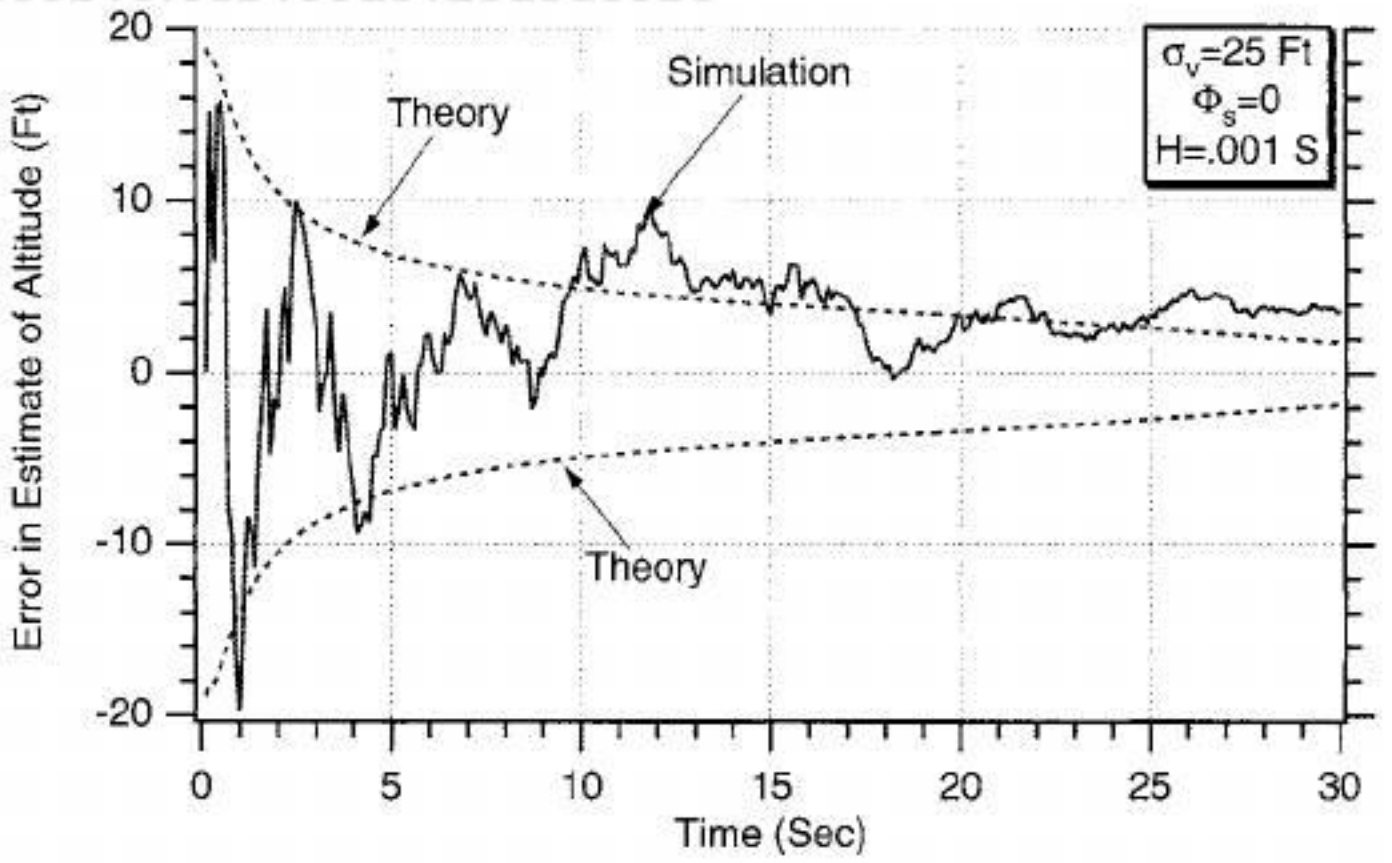

Fig. 7.21 Divergence has been eliminated in altitude estimate by use of more accurate state propagation methods.

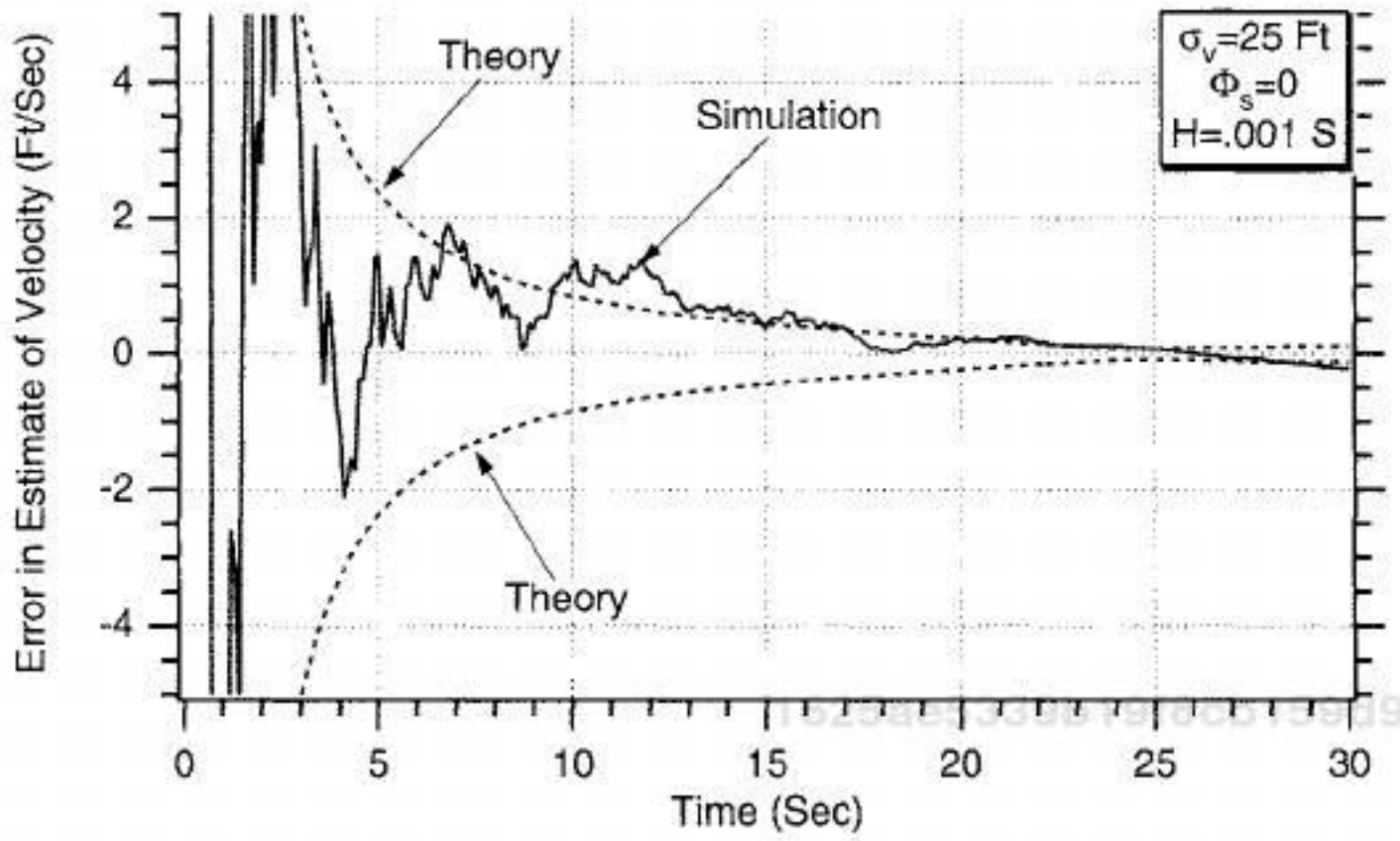

Fig. 7.22 Divergence has been eliminated in velocity estimate by use of more accurate state propagation methods.

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.

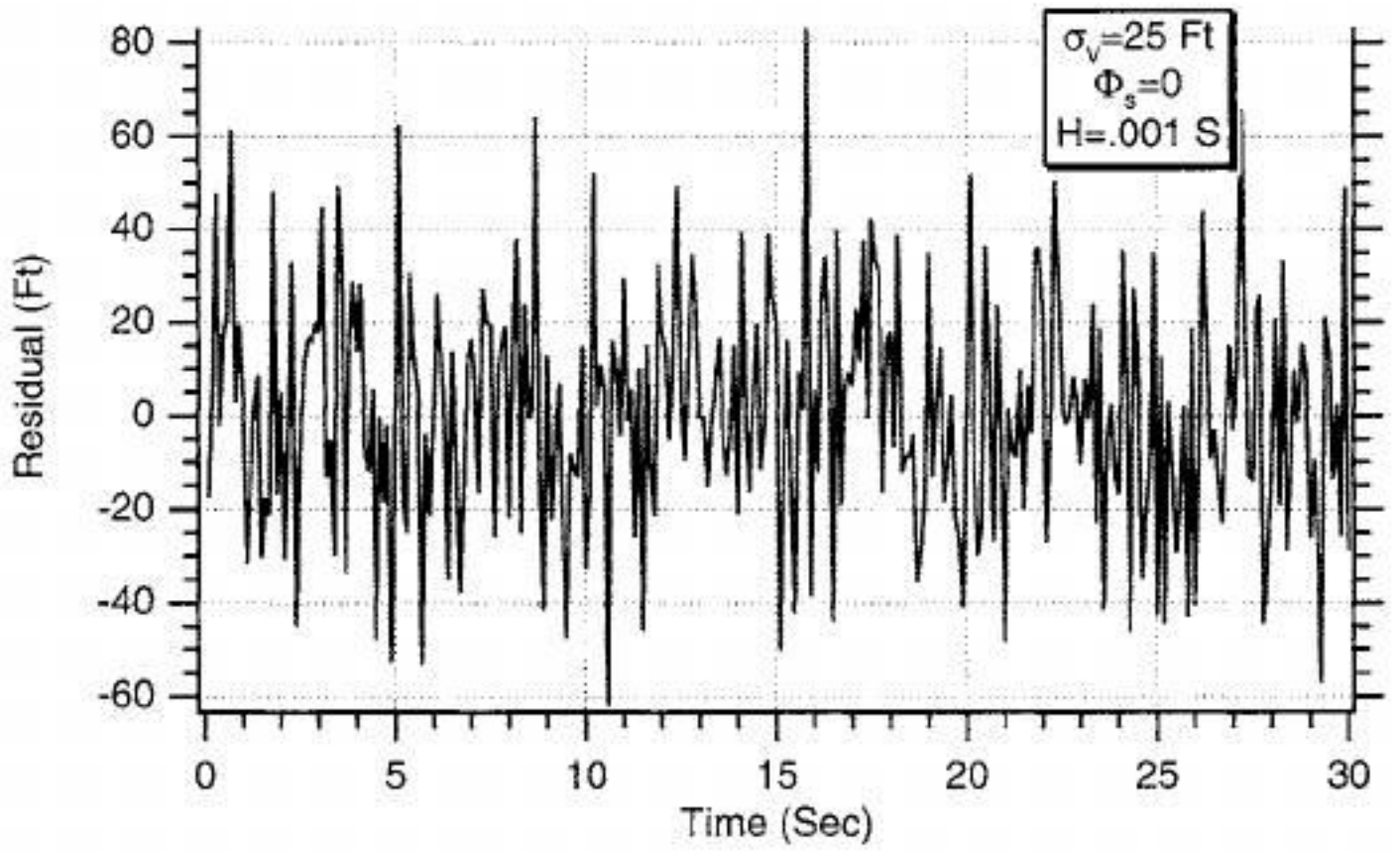

Fig. 7.23 More accurate state propagation ensures residual has zero mean even though Kalman gains approach zero.

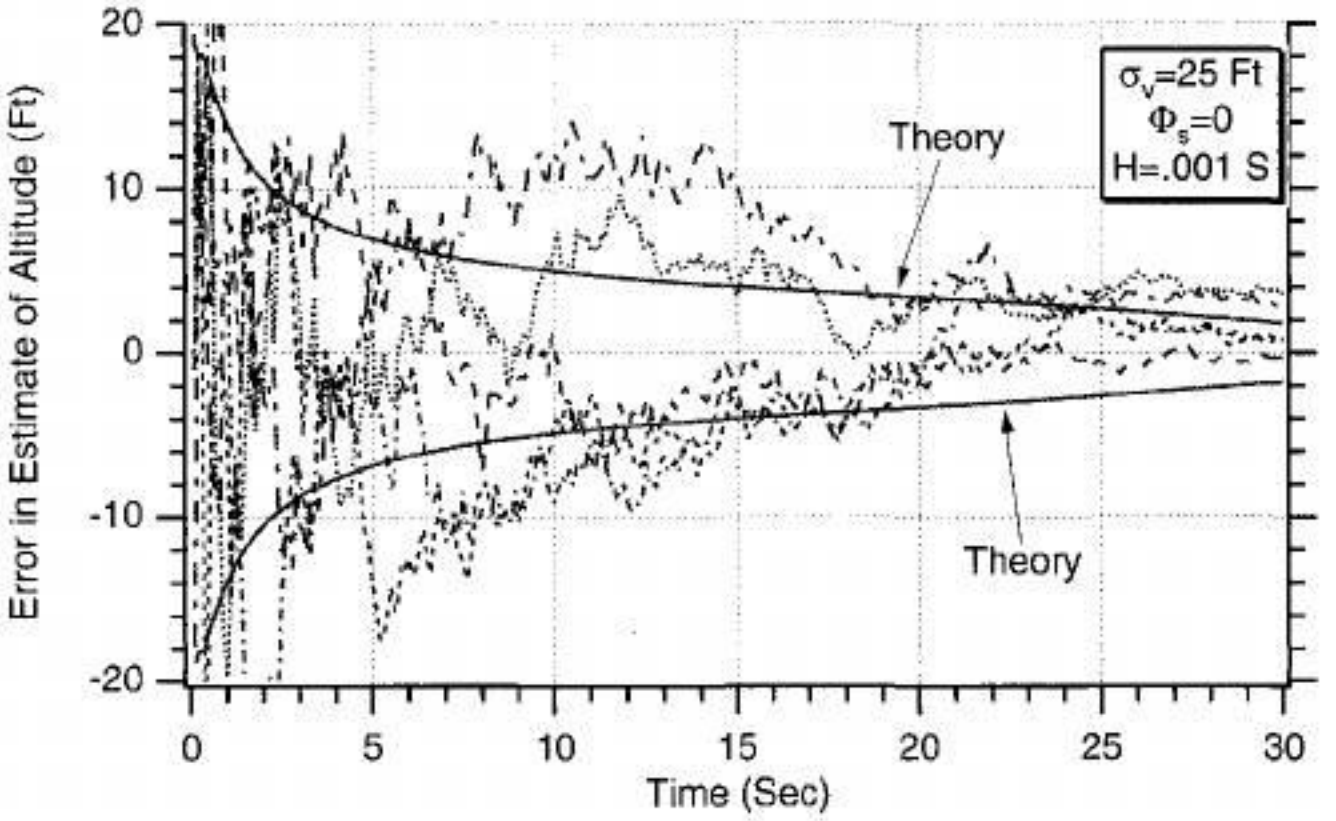

Fig. 7.24 Monte Carlo results are within theoretical bounds for error in estimate of position.

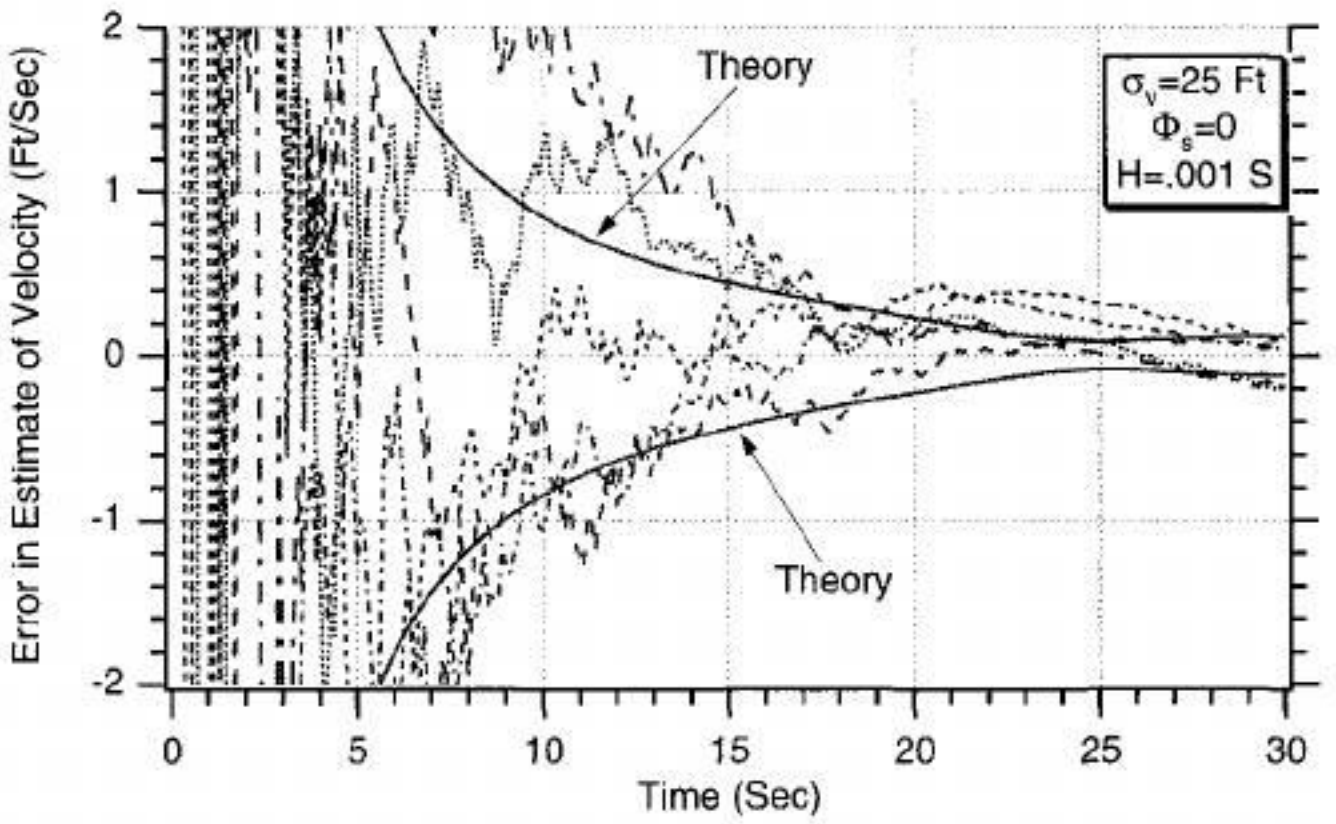

Fig. 7.25 Monte Carlo results are within theoretical bounds for error in estimate of velocity.

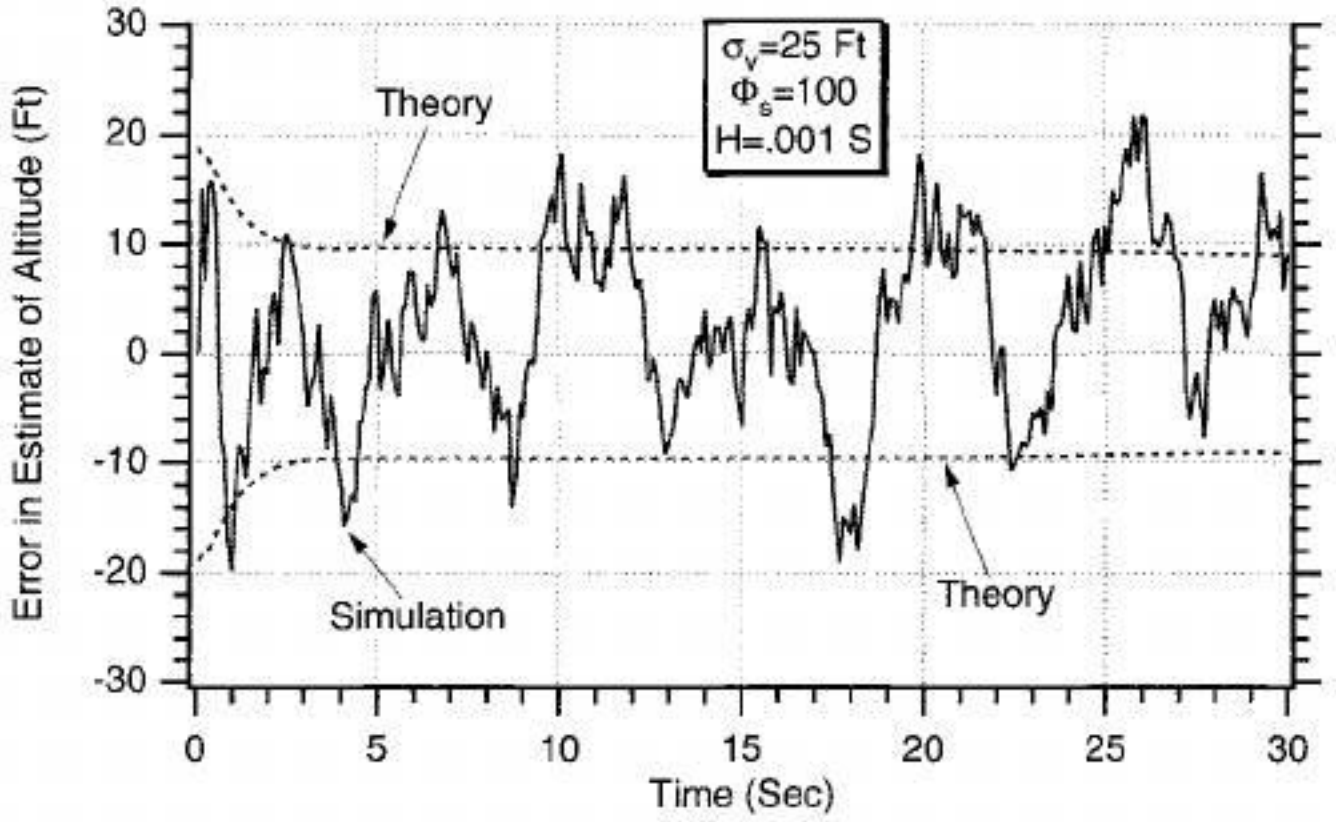

Fig. 7.26 Adding process noise increases error in estimate of position.

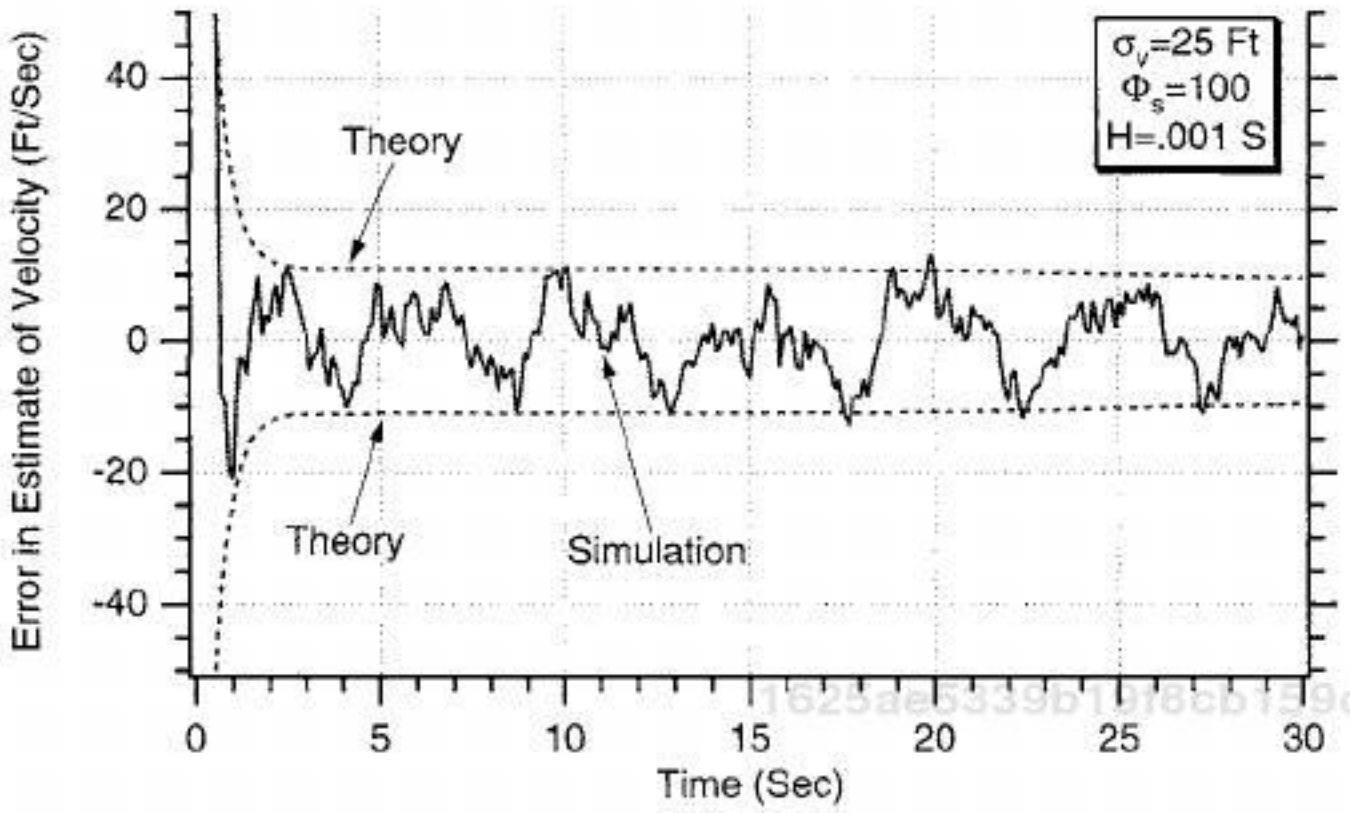

Fig. 7.27 Adding process noise increases error in estimate of velocity.

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.

This page intentionally left blank

Chapter 8

Drag and Falling Object

Introduction

IN THE preceding chapter we introduced extended Kalman filtering by trying to estimate the altitude and velocity of a falling object under the influence of drag. It was assumed in Chapter 7 that the amount of drag acting on the object was known in advance via knowledge of the object's ballistic coefficient. In that example we derived a two-state extended Kalman filter in which it was assumed that there were noisy measurements of the object's altitude and that the altitude and velocity of the object had to be estimated. In this chapter we will add further complexity to the problem by assuming that the drag or ballistic coefficient is unknown. Therefore, in this problem we desire to build another extended Kalman filter that will estimate the object's altitude, velocity, and ballistic coefficient based on noisy measurements of the object's altitude.

Problem Setup

Let us again reconsider the one-dimensional example of an object falling on a tracking radar, as was shown in Fig. 7.1 but is now redrawn for convenience in Fig. 8.1. This time the problem is different because we are assuming that the drag is unknown, which means that we have to estimate the ballistic coefficient in order to get a good estimate of the position and velocity of the object. Recall that the object was initially at 200,000 ft above the radar and had a velocity of 6000 ft/s toward the radar, which is located on the surface of a flat Earth. As was the case in Chapters 4 and 7, let us pretend that the radar measures the range from the radar to the object (i.e., altitude of the object) with a 25-ft standard deviation measurement accuracy. The radar takes measurements 10 times a second for 30 s. We would like to build a filter to estimate the altitude, velocity, and ballistic coefficient of the object.

As was the case in Chapter 7, if x is the distance from the radar to the object, then the acceleration acting on the object consists of gravity and drag or

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

where β is the ballistic coefficient of the object. Recall that the ballistic coefficient, which is considered to be a constant in this problem, is a term

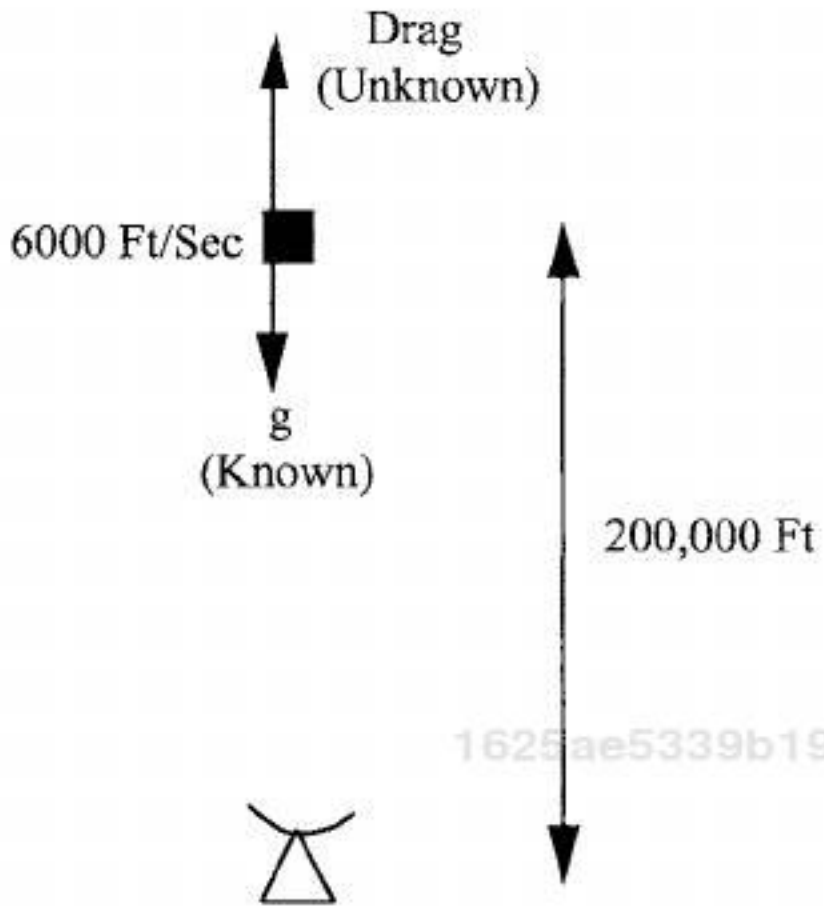

Fig. 8.1 Radar tracking falling object in presence of unknown drag.

describing the amount of drag acting on the object. In Chapter 7 we assumed that β was known in advance. The dynamic pressure Q_p in the preceding equation is given by

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

where ρ is the air density. As was the case in Chapter 7, we can still assume that the air density is an exponential function of altitude or

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

Therefore, the acceleration acting on the object can be expressed as the nonlinear second-order differential equation

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

Because the ballistic coefficient of the object we are tracking is unknown, we can make it a state. Under these circumstances it is necessary to estimate the altitude, velocity, and ballistic coefficient of the object in order to track it.[1,2] Thus, the states for the proposed filter are given by

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

Therefore, the acceleration acting on the object is a nonlinear function of the states according to

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

In addition, if we believe that the ballistic coefficient is constant, that means its derivative must be zero or

$$\dot{\beta} = 0$$

However, to make the resultant filter more robust we can add process noise to the derivative of the ballistic coefficient (i.e., the ballistic coefficient may not be a constant). Therefore, the equation for the derivative of the ballistic coefficient becomes

$$\dot{\beta} = u_s$$

where u_s is white process noise. We can linearize the preceding two differential equations by saying that

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

We can see from the preceding state-space differential equation that the systems dynamics matrix is the matrix of partial derivatives or

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

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, we can say that for this example the continuous process-noise matrix is

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

where Φ_s is the spectral density of the white noise source assumed to be on the derivative of the ballistic coefficient. To evaluate the systems dynamics matrix, we must take the necessary partial derivatives of the nonlinear state-space equation. The first row of partial derivatives yields

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

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

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

Whereas the second row of partial derivatives yields

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

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

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

and, finally, the last row of partial derivatives yields

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

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

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

Therefore, the systems dynamics matrix turns out to be

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

where gravity g is assumed to be known perfectly, but this time the ballistic coefficient β must be estimated. The estimated air density in the preceding expression depends on the estimated altitude and is given by

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

The fundamental matrix can be found by a Taylor-series approximation, yielding

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

If we define

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

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

$$f_{23} = \frac{-\hat{\rho} g \hat{\dot{x}}^2}{2\hat{\beta}^2}$$

then the systems dynamics matrix can be written as

$$\boldsymbol{F}(t) = \begin{bmatrix} 0 & 1 & 0 \\ f_{21} & f_{22} & f_{23} \\ 0 & 0 & 0 \end{bmatrix}$$

Because the fundamental matrix will only be used in the Riccati equations but will not be used in state propagation, we will only take the first two terms of the Taylor-series expansion for the fundamental matrix and obtain

$$\Phi(t) \approx \boldsymbol{I} + \boldsymbol{F}t = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 \\ f_{21} & f_{22} & f_{23} \\ 0 & 0 & 0 \end{bmatrix} t$$

$$= \begin{bmatrix} 1 & t & 0 \\ f_{21}t & 1 + f_{22}t & f_{23}t \\ 0 & 0 & 1 \end{bmatrix}$$

or, more simply,

$$\Phi(t) \approx \begin{bmatrix} 1 & t & 0 \\ f_{21}t & 1 + f_{22}t & f_{23}t \\ 0 & 0 & 1 \end{bmatrix}$$

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

$$\Phi_k \approx \begin{bmatrix} 1 & T_s & 0 \\ f_{21}T_s & 1 + f_{22}T_s & f_{23}T_s \\ 0 & 0 & 1 \end{bmatrix}$$

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

$$x^* = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \beta \end{bmatrix} + v$$

where v is the measurement noise. From the preceding equation the measurement noise matrix can be written by inspection as

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

Because the discrete measurement noise matrix is by definition

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

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

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

where σ_v^2 is the variance of the altitude noise. Recall that the continuous process-noise matrix is

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

Therefore, the discrete process-noise matrix can be derived from the continuous process-noise matrix according to

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

Substitution of the appropriate matrices into the preceding expression yields

$$\boldsymbol{Q}_k = \Phi_s \int_0^{T_s} \begin{bmatrix} 1 & \tau & 0 \\ f_{21}\tau & 1+f_{22}\tau & f_{23}\tau \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & f_{21}\tau & 0 \\ \tau & 1+f_{22}\tau & 0 \\ 0 & f_{23}\tau & 1 \end{bmatrix} \mathrm{d}\tau$$

After some algebra we obtain

$$\boldsymbol{Q}_k = \Phi_s \int_0^{T_s} \begin{bmatrix} 0 & 0 & 0 \\ 0 & f_{23}^2\tau^2 & f_{23}\tau \\ 0 & f_{23}\tau & 1 \end{bmatrix} \mathrm{d}\tau$$

Finally, after integration we get the final expression for the discrete process-noise matrix:

$$\boldsymbol{Q}_k = \Phi_s \begin{bmatrix} 0 & 0 & 0 \\ 0 & f_{23}^2 \dfrac{T_s^3}{3} & f_{23} \dfrac{T_s^2}{2} \\ 0 & f_{23} \dfrac{T_s^2}{2} & T_s \end{bmatrix}$$

We now have defined all of the matrices required to solve the 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. We learned in Chapter 7 that the best way to do this is to integrate, using Euler integration with a small step size, the nonlinear differential equations until the next update. The results of the integration will yield the projected position and velocity. In this example the nonlinear differential equation to be integrated is

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

Therefore, the Kalman-filtering equations can be written as

$$\hat{x}_k = \bar{x}_k + K_{1_k}(x_k^* - \bar{x}_k)$$
$$\hat{\dot{x}}_k = \bar{\dot{x}}_k + K_{2_k}(x_k^* - \bar{x}_k)$$
$$\hat{\beta}_k = \hat{\beta}_{k-1} + K_{3_k}(x_k^* - \bar{x}_k)$$

where x_k^* is the noisy measurement of altitude and the barred quantities are to be obtained by using Euler numerical integration with a small integration step size.

The preceding equations for the three-state extended Kalman filter and Riccati equations were programmed and are shown in Listing 8.1, along with a simulation of the real world. We can see that the process-noise matrix is set to zero in this listing (i.e., PHIS = 0) and that the third diagonal element of the initial covariance matrix has been set to 300^2 to reflect the uncertainty in our initial guess at the estimated ballistic coefficient. We have initialized the states of the filter close to the true values. The position states are in error by 25 ft, and the velocity states are in error by 150 ft/s. The initial covariance matrix reflects those errors. As was the case near the end of Chapter 7, we now have the subroutine PROJECT to numerically integrate the nonlinear acceleration equation over the sampling interval. We can see that this subroutine uses Euler integration with an integration step size of 0.001 s (i.e., HP = 0.001).

A nominal case was run with Listing 8.1 in which the process noise was zero and the single-run simulation results for the errors in the estimates of altitude, velocity, and ballistic coefficient appear in Figs. 8.2–8.4. We can see from Fig. 8.2 that the error in the estimate of altitude is within the theoretical bounds.

Listing 8.1 Simulation of extended Kalman filter to estimate ballistic coefficient

```
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(3,3),P(3,3),M(3,3),PHIP(3,3),PHIPPHIT(3,3),GAIN(3,1)
      REAL*8 Q(3,3),HMAT(1,3),HM(1,3),MHT(3,1)
      REAL*8 PHIT(3,3)
      REAL*8 HMHT(1,1),HT(3,1),KH(3,3),IDN(3,3),IKH(3,3)
      INTEGER ORDER
      ITERM=1
      SIGNOISE=25.
      X=200000.
      XD=-6000.
      BETA=500.
      XH=200025.
      XDH=-6150.
      BETAH=800.
      OPEN(2,STATUS='UNKNOWN',FILE='COVFIL')
      OPEN(1,STATUS='UNKNOWN',FILE='DATFIL')
      ORDER=3
      TS=.1
      TF=30.
      PHIS=0.
      T=0.
      S=0.
      H=.001
      HP=.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.
      IDN(3,3)=1.
      P(1,1)=SIGNOISE*SIGNOISE
      P(2,2)=20000.
      P(3,3)=300.**2
      DO 1100 I=1,ORDER
              HMAT(1,I)=0.
              HT(I,1)=0.
1100  CONTINUE
      HMAT(1,1)=1.
      HT(1,1)=1.
```

(*continued*)

Listing 8.1 (*Continued*)

```
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.
        RHOH=.0034*EXP(-XH/22000.)
        F21=-32.2*RHOH*XDH*XDH/(44000.*BETAH)
        F22=RHOH*32.2*XDH/BETAH
        F23=-RHOH*32.2*XDH*XDH/(2.*BETAH*BETAH)
        IF(ITERM.EQ.1)THEN
                PHI(1,1)=1.
                PHI(1,2)=TS
                PHI(2,1)=F21*TS
                PHI(2,2)=1.+F22*TS
                PHI(2,3)=F23*TS
                PHI(3,3)=1.
        ELSE
                PHI(1,1)=1.+.5*TS*TS*F21
                PHI(1,2)=TS+.5*TS*TS*F22
                PHI(1,3)=.5*TS*TS*F23
                PHI(2,1)=F21*TS+.5*TS*TS*F22*F21
                PHI(2,2)=1.+F22*TS+.5*TS*TS*(F21+F22*F22)
                PHI(2,3)=F23*TS+.5*TS*TS*F22*F23
                PHI(3,3)=1.
        ENDIF
C THIS PROCESS NOISEMATRIX ONLY VALID FOR TWO TERM
  APPROXIMATION TO FUNDAMNETAL MATRIX
        Q(2,2)=F23*F23*PHIS*TS*TS*TS/3.
        Q(2,3)=F23*PHIS*TS*TS/2.
        Q(3,2)=Q(2,3)
        Q(3,3)=PHIS*TS
        CALL  MATTRN(PHI,ORDER,ORDER,PHIT)
        CALL  MATMUL(PHI,ORDER,ORDER,P,ORDER,ORDER,
     1    PHIP)
        CALL  MATMUL(PHIP,ORDER,ORDER,PHIT,ORDER,ORDER,
     1    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
```

(*continued*)

Listing 8.1 (*Continued*)

```
                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,BETAH,XB,XDB,XDDB,HP)
                RES=X+XNOISE-XB
                XH=XB+GAIN(1,1)*RES
                XDH=XDB+GAIN(2,1)*RES
                BETAH=BETAH+GAIN(3,1)*RES
                ERRX=X-XH
                SP11=SQRT(P(1,1))
                ERRXD=XD-XDH
                SP22=SQRT(P(2,2))
                ERRBETA=BETA-BETAH
                SP33=SQRT(P(3,3))
                WRITE(9,*)T,X,XH,XD,XDH,BETA,BETAH
                WRITE(1,*)T,X,XH,XD,XDH,BETA,BETAH
                WRITE(2,*)T,ERRX,SP11,-SP11,ERRXD,SP22,-SP22,
1                 ERRBETA,SP33,-SP33
             ENDIF
          END DO
          PAUSE
          CLOSE(1)
          END

          SUBROUTINE PROJECT(TP,TS,XP,XDP,BETAP,XH,XDH,XDDH,HP)
          IMPLICIT REAL*8 (A-H)
          IMPLICIT REAL*8 (O-Z)
          T=0.
          X=XP
          XD=XDP
          BETA=BETAP
          H=HP
          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
```

(*continued*)

Listing 8.1 *(Continued)*

```
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
```

However, Figs. 8.3 and 8.4 indicate that there appear to be hangoff errors in the error in the estimates of velocity and ballistic coefficient. This is not a filter divergence problem because the error is not growing with time. However, we can see that there is a difference between what the covariance matrix is indicating and what is actually being achieved. Theory says that if there is no process noise, the errors in the estimates should continually decrease as more measurements are taken. We can see from Figs. 8.3 and 8.4 that there appears to be a lower limit on how small we can make the errors in the estimates. In addition, Fig. 8.4 indicates that it takes approximately 10 s before the errors in the estimates of ballistic coefficient start to decrease. The reason for this is that at the high altitudes (i.e., 200 kft) there is very little drag making the ballistic coefficient unobservable. Only as the object descends in altitude (i.e., time is increasing) and there is more drag are we able to get more accurate estimates of the ballistic coefficient.

Figure 8.5 expands the scales of Fig. 8.4 to get a more detailed understanding of how the errors in the estimate of ballistic coefficient reduce as time increases (i.e., altitude decreases). We can see from Fig. 8.5 that, although the errors reduce, there is also definitely hangoff error in this state (i.e., error does not go to zero). We can see that there is a hangoff error of approximately 3 lb/ft^2 in the ballistic coefficient estimate.

An alternative hypothesis is that perhaps the reason for the hangoff error in the velocity and ballistic coefficient estimates was that there were an insufficient

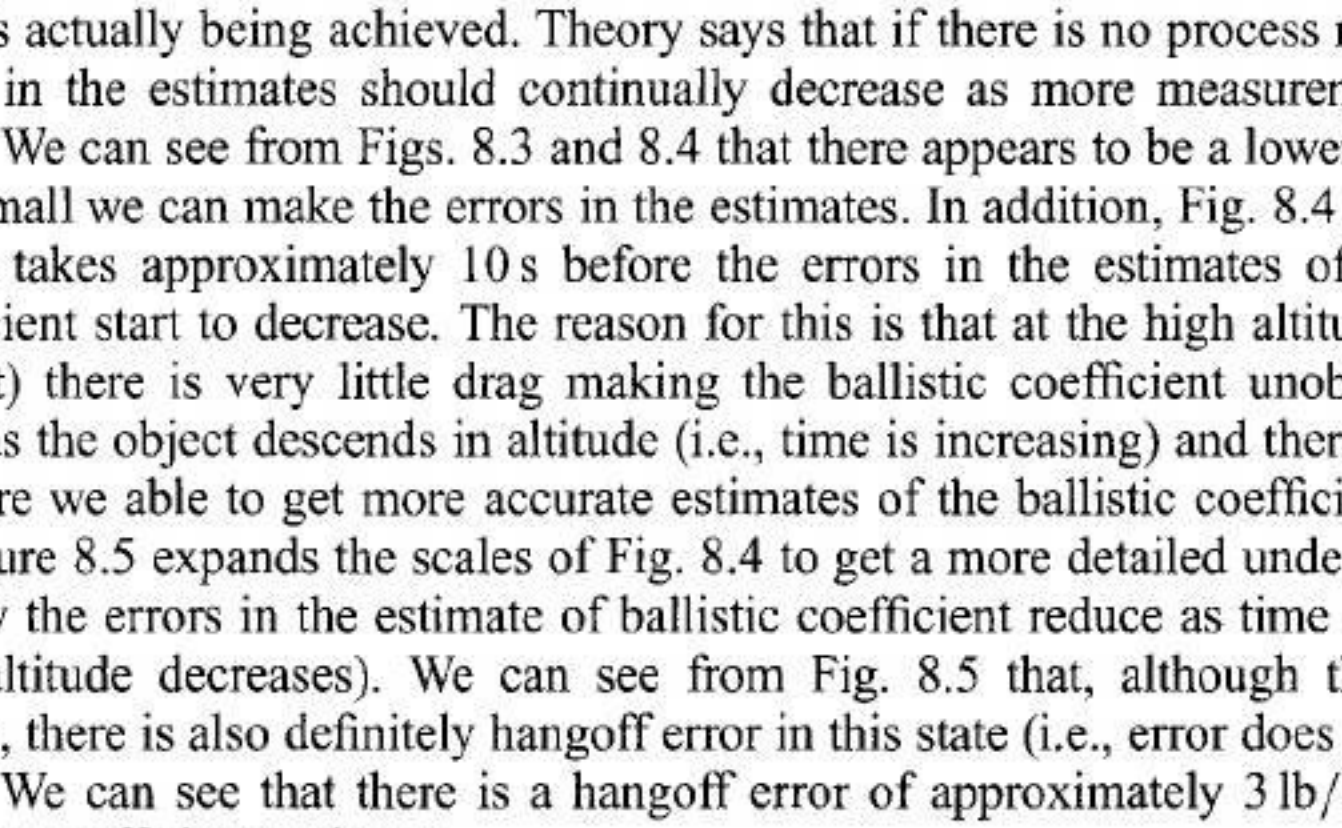

Fig. 8.2 Error in estimate of altitude is within theoretical bounds.

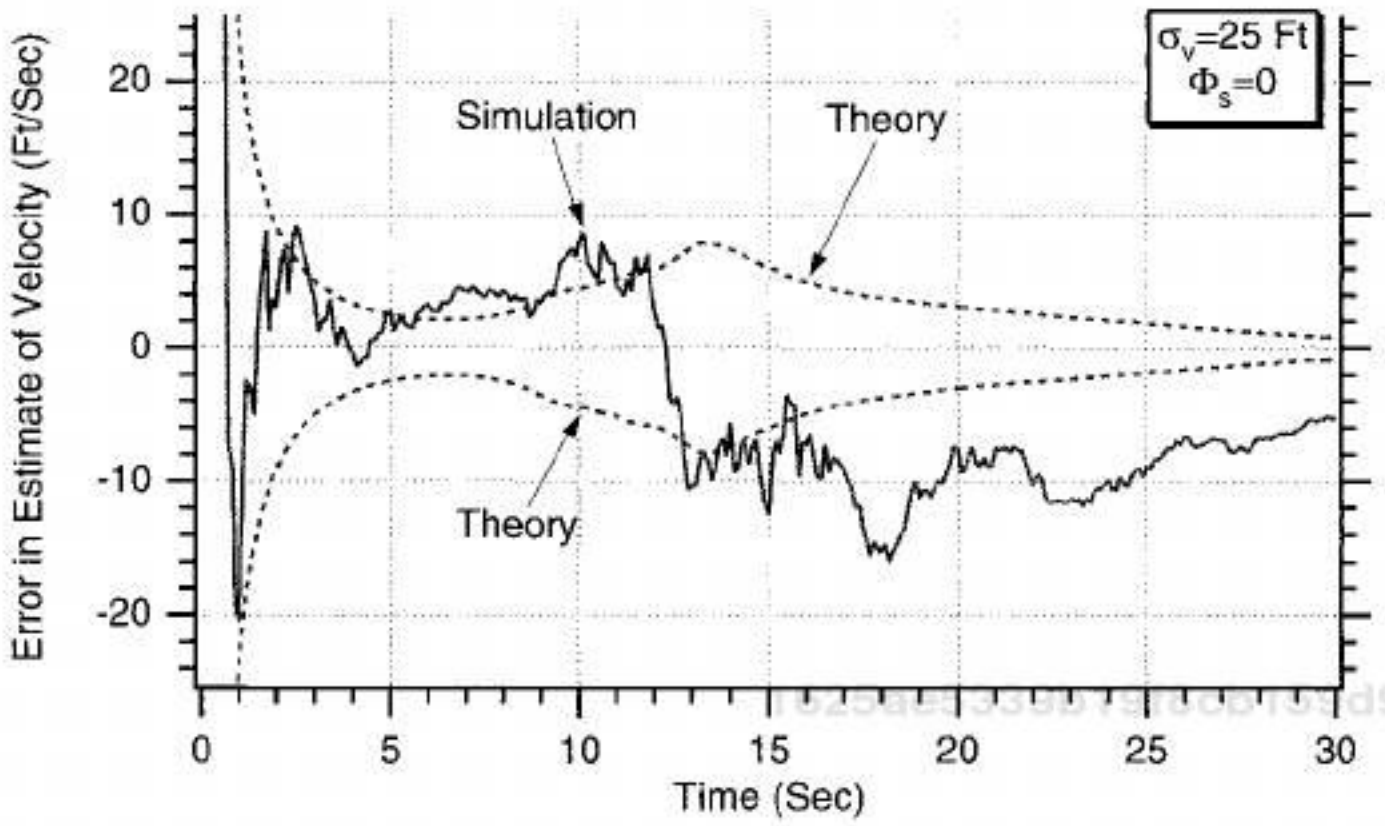

Fig. 8.3 Error in estimate of velocity appears to have hangoff error.

number of terms in the Taylor-series expansion used in obtaining the fundamental matrix. Recall that the systems dynamics matrix was given by

$$\boldsymbol{F}(t) = \begin{bmatrix} 0 & 1 & 0 \\ f_{21} & f_{22} & f_{23} \\ 0 & 0 & 0 \end{bmatrix}$$

where the terms f_{21}, f_{22}, and f_{23} have already been defined. If we now use the first three terms of a Taylor series involving the systems dynamics matrix to define the fundamental matrix, we get

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

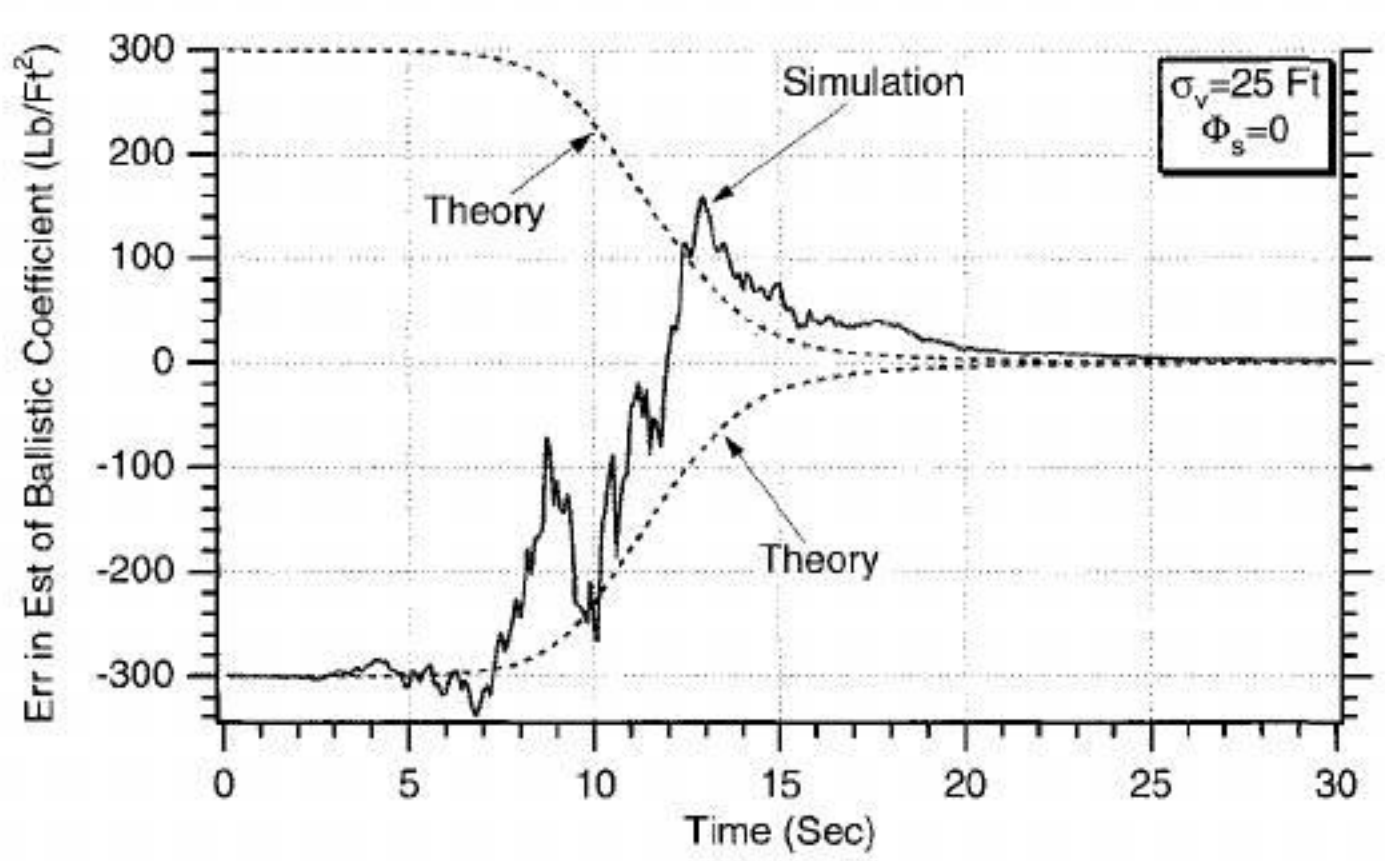

Fig. 8.4 It takes more than 10 s to reduce errors in estimating ballistic coefficient.

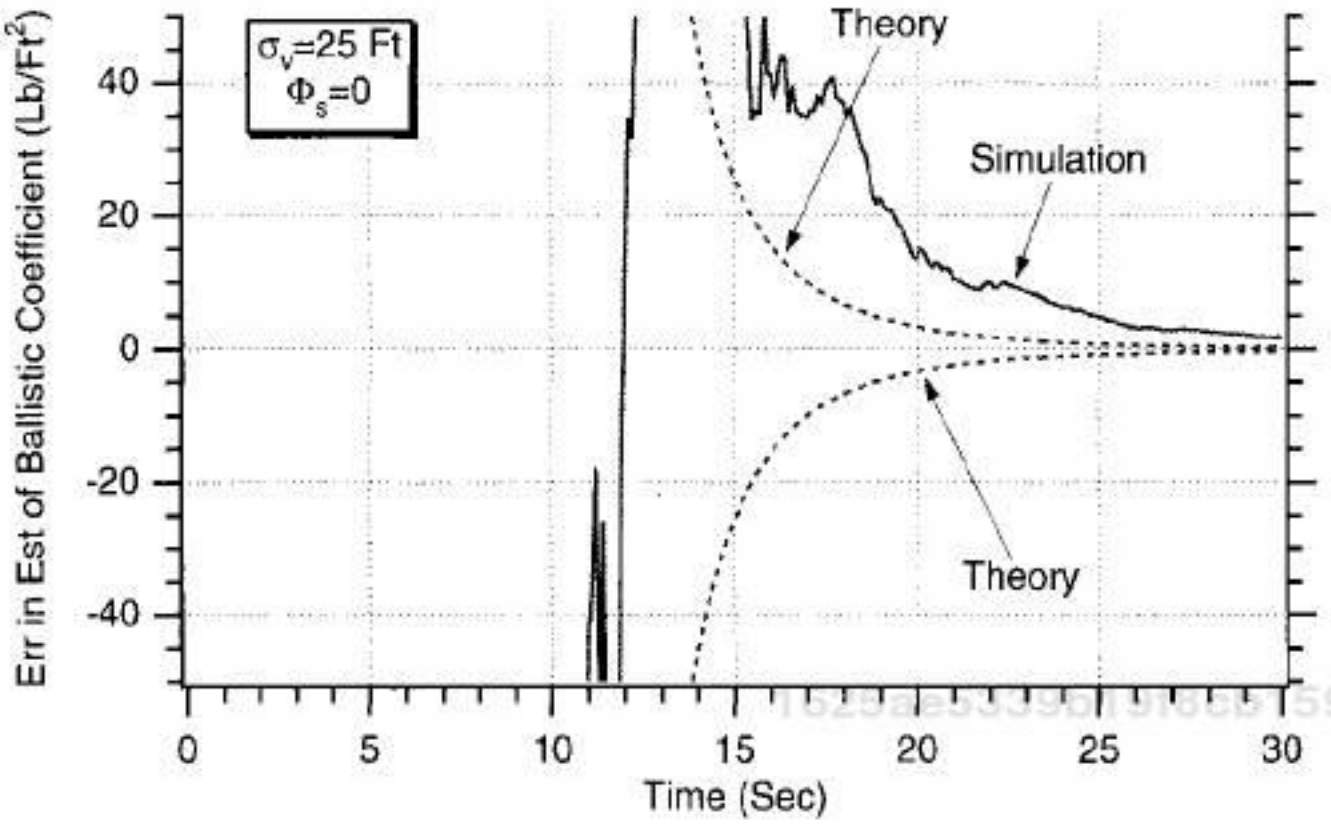

Fig. 8.5 There is also hangoff error in estimating ballistic coefficient.

If we define the two-term fundamental matrix as Φ_2, then the three-term fundamental matrix can be written as

$$\Phi_3 = \Phi_2 + \frac{\boldsymbol{F}^2 t^2}{2}$$

or

$$\Phi_3 = \begin{bmatrix} 1 & t & 0 \\ f_{21}t & 1+f_{22}t & f_{23}t \\ 0 & 0 & 1 \end{bmatrix} + \frac{\boldsymbol{F}^2 t^2}{2}$$

We can find the square of the systems dynamics matrix from multiplication as

$$\boldsymbol{F}^2 = \begin{bmatrix} 0 & 1 & 0 \\ f_{21} & f_{22} & f_{23} \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ f_{21} & f_{22} & f_{23} \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} f_{21} & f_{22} & f_{23} \\ f_{22}f_{21} & f_{21}+f_{22}^2 & f_{22}f_{23} \\ 0 & 0 & 0 \end{bmatrix}$$

Substitution of the preceding expression into the definition of the three-term Taylor-series expansion yields

$$\Phi_3 = \begin{bmatrix} 1 & t & 0 \\ f_{21}t & 1+f_{22}t & f_{23}t \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} f_{21}\dfrac{t^2}{2} & f_{22}\dfrac{t^2}{2} & f_{23}\dfrac{t^2}{2} \\ f_{22}f_{21}\dfrac{t^2}{2} & (f_{21}+f_{22}^2)\dfrac{t^2}{2} & f_{22}f_{23}\dfrac{t^2}{2} \\ 0 & 0 & 0 \end{bmatrix}$$

After simplification we get

$$\Phi_3 = \begin{bmatrix} 1 + f_{21}\dfrac{t^2}{2} & t + f_{22}\dfrac{t^2}{2} & f_{23}\dfrac{t^2}{2} \\ f_{21}t + f_{22}f_{21}\dfrac{t^2}{2} & 1 + f_{22}t + (f_{21} + f_{22}^2)\dfrac{t^2}{2} & f_{23}t + f_{22}f_{23}\dfrac{t^2}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

To find the new discrete fundamental matrix, we simply substitute the sampling time for time or

$$\Phi_{3_k} = \begin{bmatrix} 1 + f_{21}\dfrac{T_s^2}{2} & T_s + f_{22}\dfrac{T_s^2}{2} & f_{23}\dfrac{T_s^2}{2} \\ f_{21}T_s + f_{22}f_{21}\dfrac{T_s^2}{2} & 1 + f_{22}T_s + (f_{21} + f_{22}^2)\dfrac{T_s^2}{2} & f_{23}T_s + f_{22}f_{23}\dfrac{T_s^2}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

In theory we should also modify the discrete process-noise matrix because the fundamental matrix has changed. However, because we will run with zero process noise, this will not be necessary. A case was run with the new fundamental matrix by setting ITERM = 2 in Listing 8.1. Figure 8.6 displays the resultant error in the estimate of velocity as a function of time. We can see that adding an extra term to the approximation for the fundamental matrix had no influence on removing the hangoff error.

It was then hypothesized that perhaps the integration interval used to project the states forward was not small enough. The integration step size HP was reduced in subroutine PROJECT from 0.001 to 0.0001 s in Listing 8.1. We see from Fig. 8.7 that reducing the integration interval for the method of state propagation also has no effect in reducing the hangoff error.

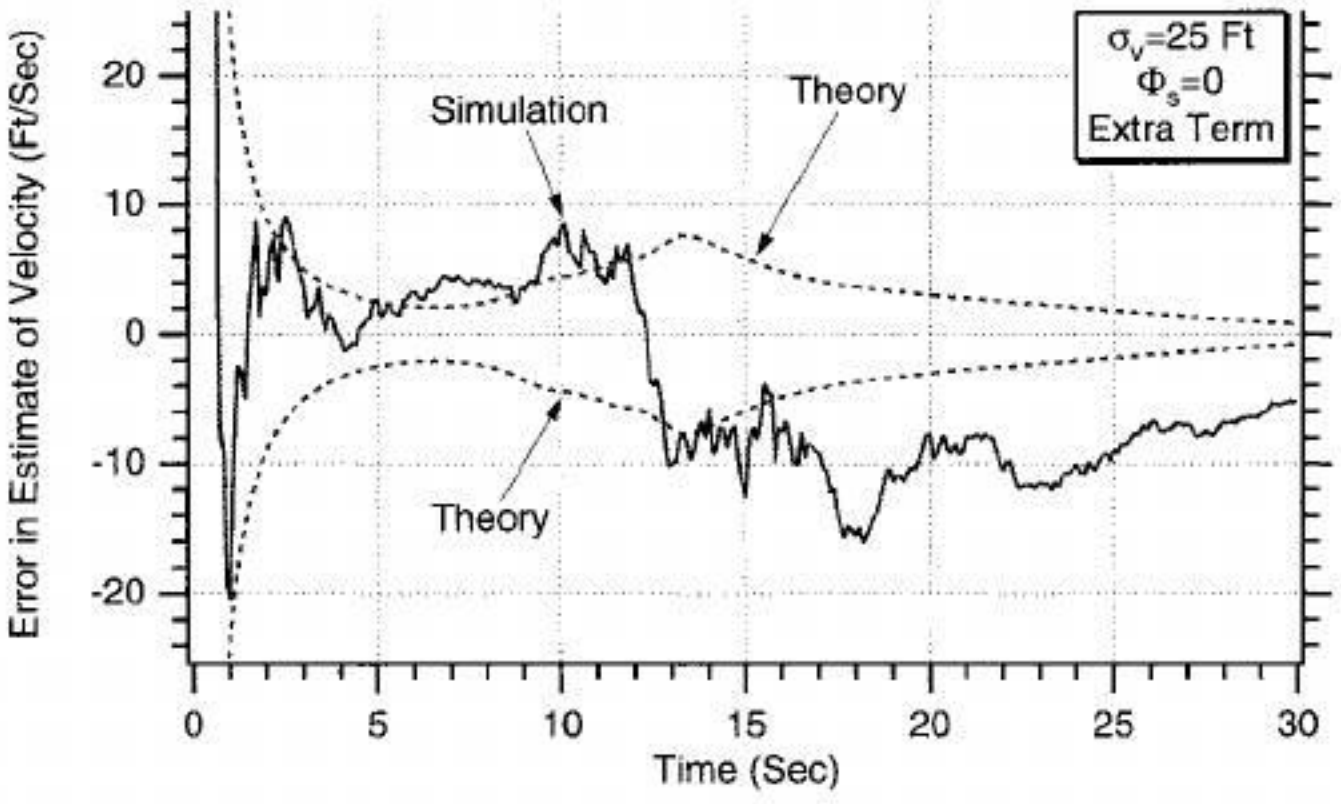

Fig. 8.6 Adding extra term to fundamental matrix does not remove hangoff error.

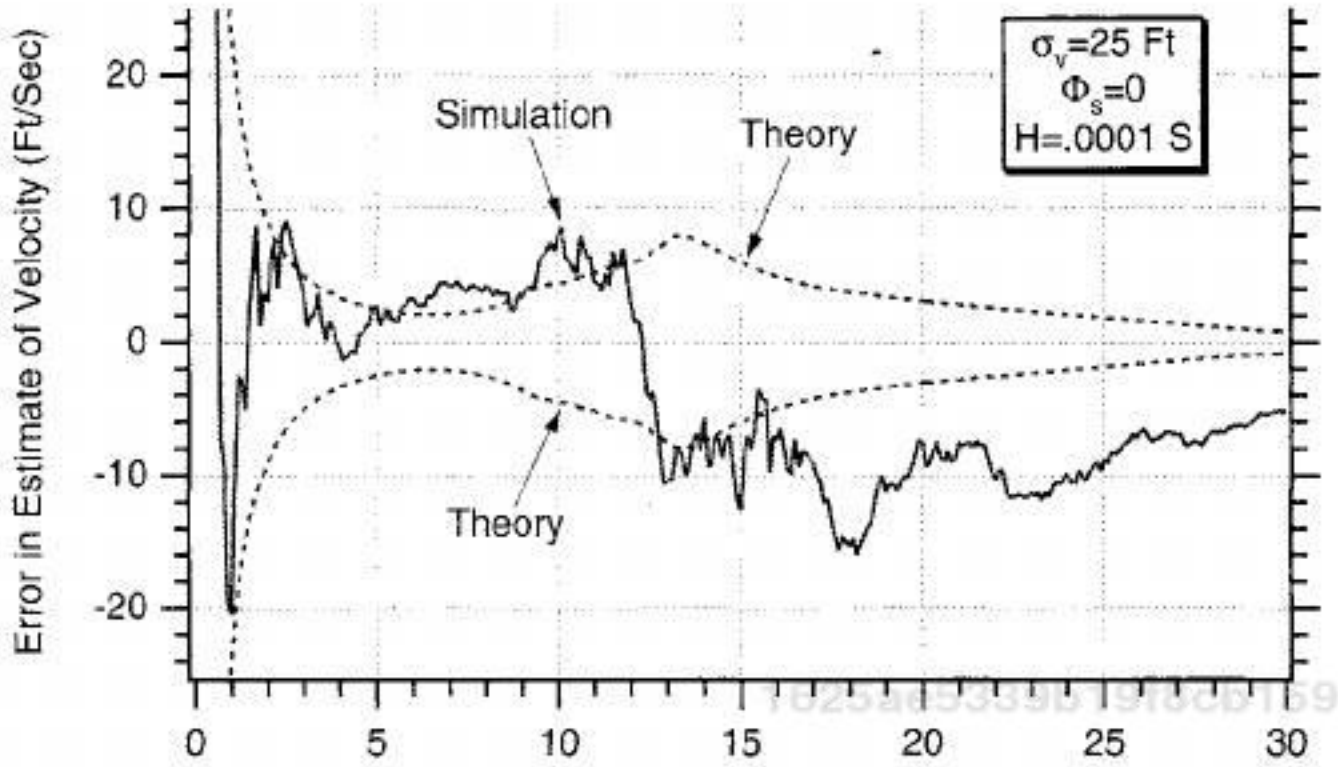

Fig. 8.7 Making integration interval 10 times smaller in PROJECT subroutine does not remove hangoff error in velocity estimate.

As the object descends in altitude, there is more drag, and the object becomes more observable from a filtering point of view. However, because there is no process noise the filter gains will go to zero. This means that the filter will stop paying attention to the measurements (i.e., when the ballistic coefficient is most observable) and hangoff error will result, as can be seen in Fig. 8.5. Finally, process noise was added to the extended Kalman filter. Figures 8.8–8.10 show how the errors in the estimates change with the addition of process noise. For example, we can see from Fig. 8.8 that using a value of 3000 (i.e., $\Phi_s = 3000$) increases slightly the error in the estimate of altitude. Figures 8.9 and 8.10 show how the addition of process noise eliminates the hangoff error in the velocity and

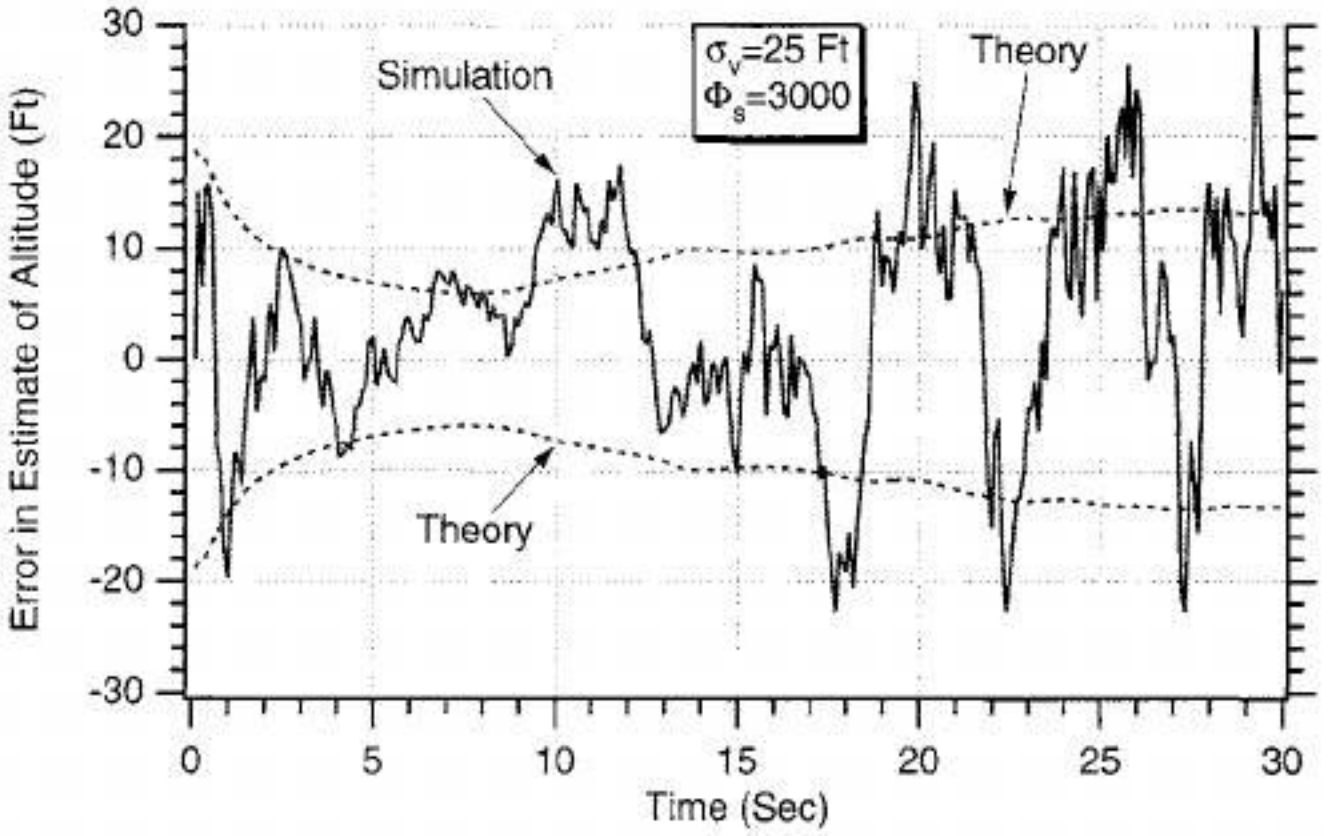

Fig. 8.8 Adding process noise increases errors in estimate of altitude.

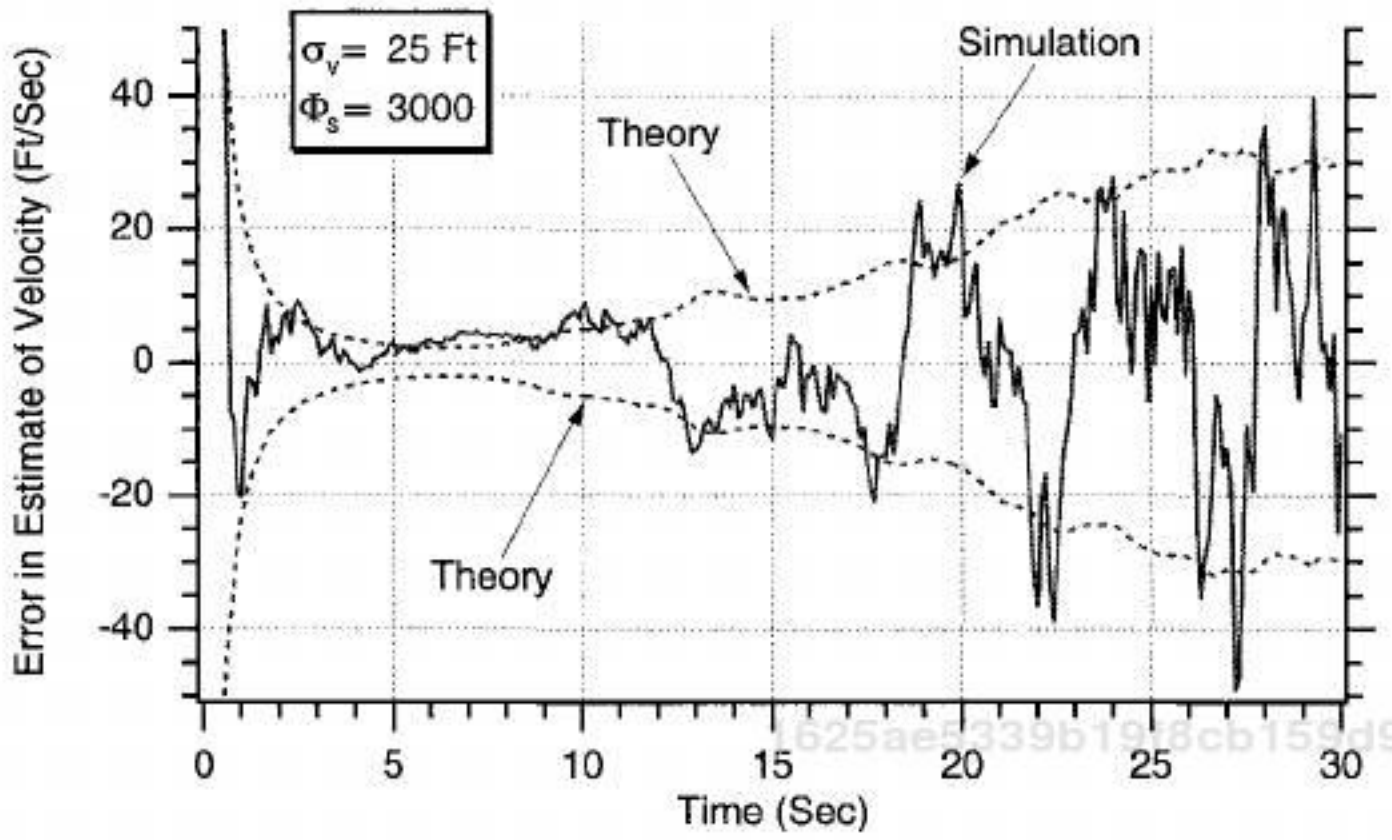

Fig. 8.9 Adding process noise removes hangoff error at expense of increasing errors in estimate of velocity.

ballistic coefficient estimate. By comparing Figs. 8.7 with 8.9, we see that the price we paid for adding the process noise was slightly higher errors in the estimates of velocity. By comparing Figs. 8.4 and 8.10, we can see that adding process noise removed the hangoff error in the ballistic coefficient at the expense of increased errors in that estimate. Adding process noise is the normal engineering fix to situations in which the errors in the estimates do not quite match theory. We are simply accounting for our imperfect knowledge of the real world. In general, adding process noise will make a filter more robust by giving it a wider bandwidth and is thus the prudent thing to do.

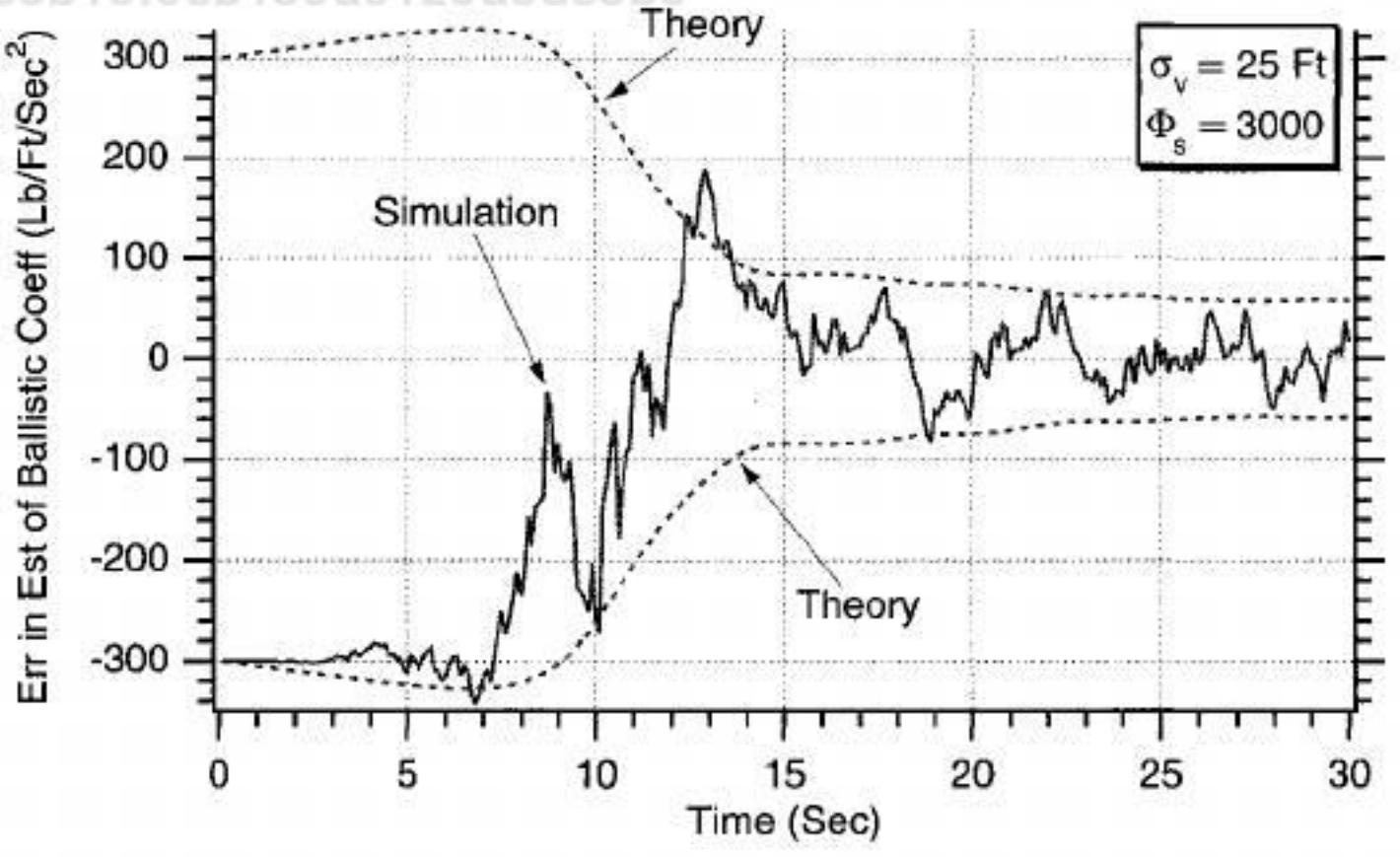

Fig. 8.10 Adding process noise removes hangoff error at expense of increasing errors in estimate of ballistic coefficient.

Changing Filter States

The choice of states for this problem is not unique. Unlike a linear Kalman filter, where the choice of states should make no difference provided that the new states are linear functions of the old states, the choice of states in an extended Kalman filter can make a major difference. Recall that the nonlinear differential equation describing the acceleration on the object is given by

$$\ddot{x} = \frac{0.0034ge^{-x/22000}\dot{x}^2}{2\beta} - g = \left(\frac{1}{\beta}\right)\frac{0.0034ge^{-x/22000}\dot{x}^2}{2} - g$$

Because the preceding equation is inversely proportional to the ballistic coefficient, it seems that a reasonable alternate set of states would be

$$\boldsymbol{x} = \begin{bmatrix} x \\ \dot{x} \\ 1/\beta \end{bmatrix}$$

because the original differential equation looks less nonlinear. In addition, if we believe that the ballistic coefficient is constant that means its derivative must be zero. Therefore, the derivative of the inverse ballistic coefficient must also be zero or

$$\dot{(1/\beta)} = 0$$

We can linearize the preceding differential equations by saying that

$$\begin{bmatrix} \Delta\dot{x} \\ \Delta\ddot{x} \\ \Delta\dot{(1/\beta)} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial\dot{x}}{\partial x} & \dfrac{\partial\dot{x}}{\partial\dot{x}} & \dfrac{\partial\dot{x}}{\partial(1/\beta)} \\ \dfrac{\partial\ddot{x}}{\partial x} & \dfrac{\partial\ddot{x}}{\partial\dot{x}} & \dfrac{\partial\ddot{x}}{\partial(1/\beta)} \\ \dfrac{\partial\dot{(1/\beta)}}{\partial x} & \dfrac{\partial\dot{(1/\beta)}}{\partial\dot{x}} & \dfrac{\partial\dot{(1/\beta)}}{\partial(1/\beta)} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\dot{x} \\ \Delta(1/\beta \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ u_s \end{bmatrix}$$

where u_s is white process noise that has been added to the derivative of the inverse ballistic coefficient rate equation for possible future protection (i.e., the ballistic coefficient may not be a constant). We can see from the preceding state-space differential equation that the systems dynamics matrix is the matrix of partial derivatives or

$$\boldsymbol{F} = \left[\begin{array}{ccc} \dfrac{\partial\dot{x}}{\partial x} & \dfrac{\partial\dot{x}}{\partial\dot{x}} & \dfrac{\partial\dot{x}}{\partial(1/\beta)} \\ \dfrac{\partial\ddot{x}}{\partial x} & \dfrac{\partial\ddot{x}}{\partial\dot{x}} & \dfrac{\partial\ddot{x}}{\partial(1/\beta)} \\ \dfrac{\partial\dot{(1/\beta)}}{\partial x} & \dfrac{\partial\dot{(1/\beta)}}{\partial\dot{x}} & \dfrac{\partial\dot{(1/\beta)}}{\partial(1/\beta)} \end{array}\right]_{x=\hat{x}}$$

Note that in the preceding systems dynamics matrix the partial derivatives are evaluated at the current state estimates. To evaluate the systems dynamics matrix, we must take the necessary partial derivatives. The first row of partial derivatives yields

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

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

$$\frac{\partial \dot{x}}{\partial (1/\beta)} = 0$$

while the second row of partial derivatives becomes

$$\frac{\partial \ddot{x}}{\partial x} = \frac{-0.0034e^{-x/22000}\dot{x}^2 g}{2\beta(22{,}000)} = \frac{-\rho g \dot{x}^2}{44{,}000\beta} = \left(\frac{1}{\beta}\right)\left(\frac{-\rho g \dot{x}^2}{44{,}000}\right)$$

$$\frac{\partial \ddot{x}}{\partial \dot{x}} = \frac{2*0.0034e^{-x/22000}\dot{x}g}{2\beta} = \frac{\rho g \dot{x}}{\beta} = \left(\frac{1}{\beta}\right)\rho g \dot{x}$$

$$\frac{\partial \ddot{x}}{\partial (1/\beta)} = \frac{0.0034e^{-x/22000}\dot{x}^2 g}{2} = \frac{\rho g \dot{x}^2}{2}$$

Finally, the last row of partial derivatives yields

$$\frac{\partial (1/\beta)^{\cdot}}{\partial x} = 0$$

$$\frac{\partial (1/\beta)^{\cdot}}{\partial \dot{x}} = 0$$

$$\frac{\partial (1/\beta)^{\cdot}}{\partial (1/\beta)} = 0$$

Therefore, the systems dynamics matrix turns out to be

$$\boldsymbol{F}(t) = \begin{bmatrix} 0 & 1 & 0 \\ f_{21} & f_{22} & f_{23} \\ 0 & 0 & 0 \end{bmatrix}$$

where

$$f_{21} = \frac{-\hat{\rho} g \hat{\dot{x}}^2}{44{,}000} \widehat{(1/\beta)}$$

$$f_{22} = \hat{\rho} \hat{\dot{x}} g \widehat{(1/\beta)}$$

$$f_{23} = \frac{-\hat{\rho} g \hat{\dot{x}}^2}{2}$$

As was the case before, the two-term approximation to the fundamental is given by

$$\Phi_k = \begin{bmatrix} 1 & T_s & 0 \\ f_{21} T_s & 1 + f_{22} T_s & f_{23} T_s \\ 0 & 0 & 1 \end{bmatrix}$$

The simulation of the extended Kalman filter, for which $1/\beta$ is a state, appears in Listing 8.2. We can see that the inputs are identical to the other extended Kalman filter, which appears in Listing 8.1. The simulation is set up so that nominally there is no process noise (i.e., PHIS = 0).

The nominal case of Listing 8.2 was run when there was no process noise. We can see from Fig. 8.11 that the error in the estimate of altitude is approximately the same as it was for the other extended Kalman filter (see Fig. 8.2). However, we also can see from Fig. 8.12 that there is no longer any hangoff error in the error in the estimate of velocity when there is no process noise.

Figure 8.13 seems also to be saying that there is also no hangoff error in the error of the estimate for the inverse ballistic coefficient. We also can see that the inverse ballistic coefficient is not observable for about the same period of time as the other extended Kalman filter (see Fig. 8.4). However, Fig. 8.14, which simply expands the scales of the preceding plot for more clarity, also says that there is no hangoff error.

Because most people do not have a feeling for the inverse ballistic coefficient, the actual estimate of the ballistic coefficient (i.e., reciprocal of inverse ballistic coefficient) appears in Fig. 8.15 along with the actual ballistic coefficient. We can see that after 10 s we get extremely accurate estimates.

Thus, we can see that the hangoff error of the original extended Kalman filter was eliminated by changing the filter states. The choice of $1/\beta$ as a state made the state-space equations appear less strongly nonlinear, thus eliminating the need for process noise to correct matters. In principle, using smaller amounts of process noise (or none at all) will reduce the errors in the estimates as long as the filter model is perfectly matched to the real world.

Why Process Noise Is Required

It would appear from the work presented in this chapter that it is desirable to set the process noise to zero because it reduces the effect of the measurement noise. We have seen that if the errors in the estimate (i.e., truth minus estimate)

Listing 8.2 Simulation of extended Kalman filter with inverse ballistic coefficient as state

```
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(3,3),P(3,3),M(3,3),PHIP(3,3),PHIPPHIT(3,3),GAIN(3,1)
      REAL*8 Q(3,3),HMAT(1,3),HM(1,3),MHT(3,1)
      REAL*8 PHIT(3,3)
      REAL*8 HMHT(1,1),HT(3,1),KH(3,3),IDN(3,3),IKH(3,3)
      INTEGER ORDER
      ITERM=1
      G=32.2
      SIGNOISE=25.
      X=200000.
      XD=-6000.
      BETA=500.
      XH=200025.
      XDH=-6150.
      BETAH=800.
      BETAINV=1./BETA
      BETAINVH=1./BETAH
      OPEN(2,STATUS='UNKNOWN',FILE='COVFIL')
      OPEN(1,STATUS='UNKNOWN',FILE='DATFIL')
      ORDER=3
      TS=.1
      TF=30.
      PHIS=0.
      T=0.
      S=0.
      H=.001
      HP=.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.
      IDN(3,3)=1.
      P(1,1)=SIGNOISE*SIGNOISE
      P(2,2)=20000.
      P(3,3)=(BETAINV-BETAINVH)**2
      DO  1100  I=1,ORDER
              HMAT(1,I)=0.
              HT(I,1)=0.
```

(*continued*)

Listing 8.2 (*Continued*)

```
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
               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=-G*RHOH*XDH*XDH*BETAINVH/44000.
                         F22=RHOH*G*XDH*BETAINVH
                         F23=.5*RHOH*XDH*XDH*G
                         PHI(1,1)=1.
                         PHI(1,2)=TS
                         PHI(2,1)=F21*TS
                         PHI(2,2)=1.+F22*TS
                         PHI(2,3)=F23*TS
                         PHI(3,3)=1.
                         Q(2,2)=F23*F23*PHIS*TS*TS*TS/3.
                         Q(2,3)=F23*PHIS*TS*TS/2.
                         Q(3,2)=Q(2,3)
                         Q(3,3)=PHIS*TS
                         CALL MATTRN(PHI,ORDER,ORDER,PHIT)
                         CALL MATMUL(PHI,ORDER,ORDER,P,ORDER,ORDER,
                           PHIP)
                         CALL MATMUL(PHIP,ORDER,ORDER,PHIT,ORDER,
                           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)
                         BETAH=1./BETAINVH
```

(*continued*)

Listing 8.2 (*Continued*)

```
                CALL PROJECT(T,TS,XH,XDH,BETAH,XB,XDB,
                  XDDB,HP)
                RES=X+XNOISE-XB
                XH=XB+GAIN(1,1)*RES
                XDH=XDB+GAIN(2,1)*RES
                BETAINVH=BETAINVH+GAIN(3,1)*RES

                ERRX=X-XH
                SP11=SQRT(P(1,1))
                ERRXD=XD-XDH
                SP22=SQRT(P(2,2))
                ERRBETAINV=1./BETA-BETAINVH
                SP33=SQRT(P(3,3))
                BETAH=1./BETAINVH
                WRITE(9,*)T,X,XH,XD,XDH,BETA,BETAH
                WRITE(1,*)T,X,XH,XD,XDH,BETA,BETAH
                WRITE(2,*)T,ERRX,SP11,-SP11,ERRXD,SP22,-SP22,
1                 ERRBETAINV,SP33,-SP33
             ENDIF
          END DO
          PAUSE
          CLOSE(1)
          END

          SUBROUTINE PROJECT(TP,TS,XP,XDP,BETAP,XH,XDH,XDDH,HP)
          IMPLICIT REAL*8 (A-H)
          IMPLICIT REAL*8 (O-Z)
          T=0.
          X=XP
          XD=XDP
          BETA=BETAP
          H=HP
          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
```

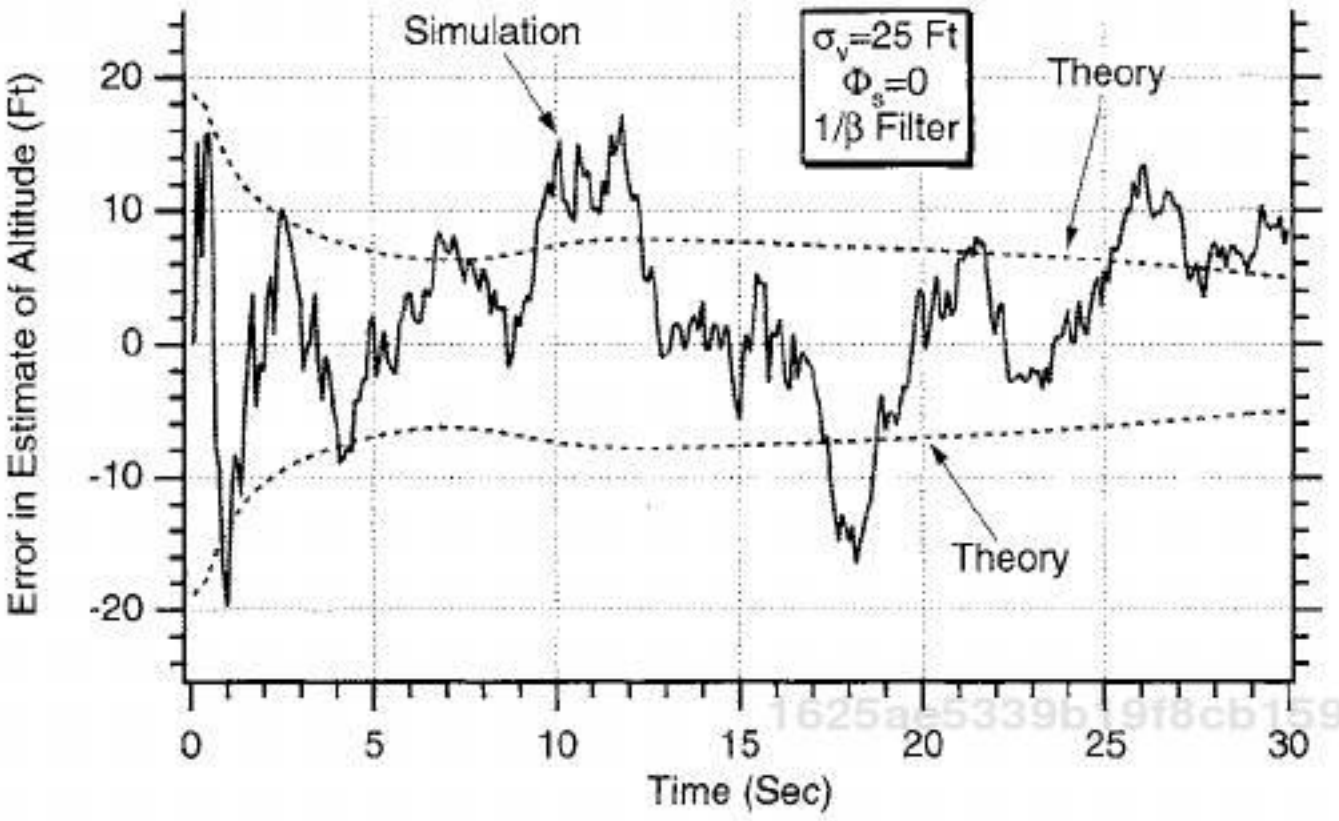

Fig. 8.11 Error in the estimate of altitude is approximately the same for both extended Kalman filters.

have no hangoff error, then having the process noise set to zero enables the error in the estimate to approach zero as more measurements are taken. Although increasing the process noise often removes hangoff errors, we have also seen that the errors in the estimates increase with increasing process noise because of the higher Kalman gains, which admit more measurement noise. The real purpose of the process noise is to account for the fact that the filter's knowledge of the real world is often in error.

Consider the case in which we are still tracking the same object, but after 25 s of tracking the object breaks in half. This causes the ballistic coefficient to reduce by a factor of two (i.e., from 500 to 250 lb/ft^2). By using zero process noise, the extended Kalman-filter model was assuming that the actual ballistic coefficient

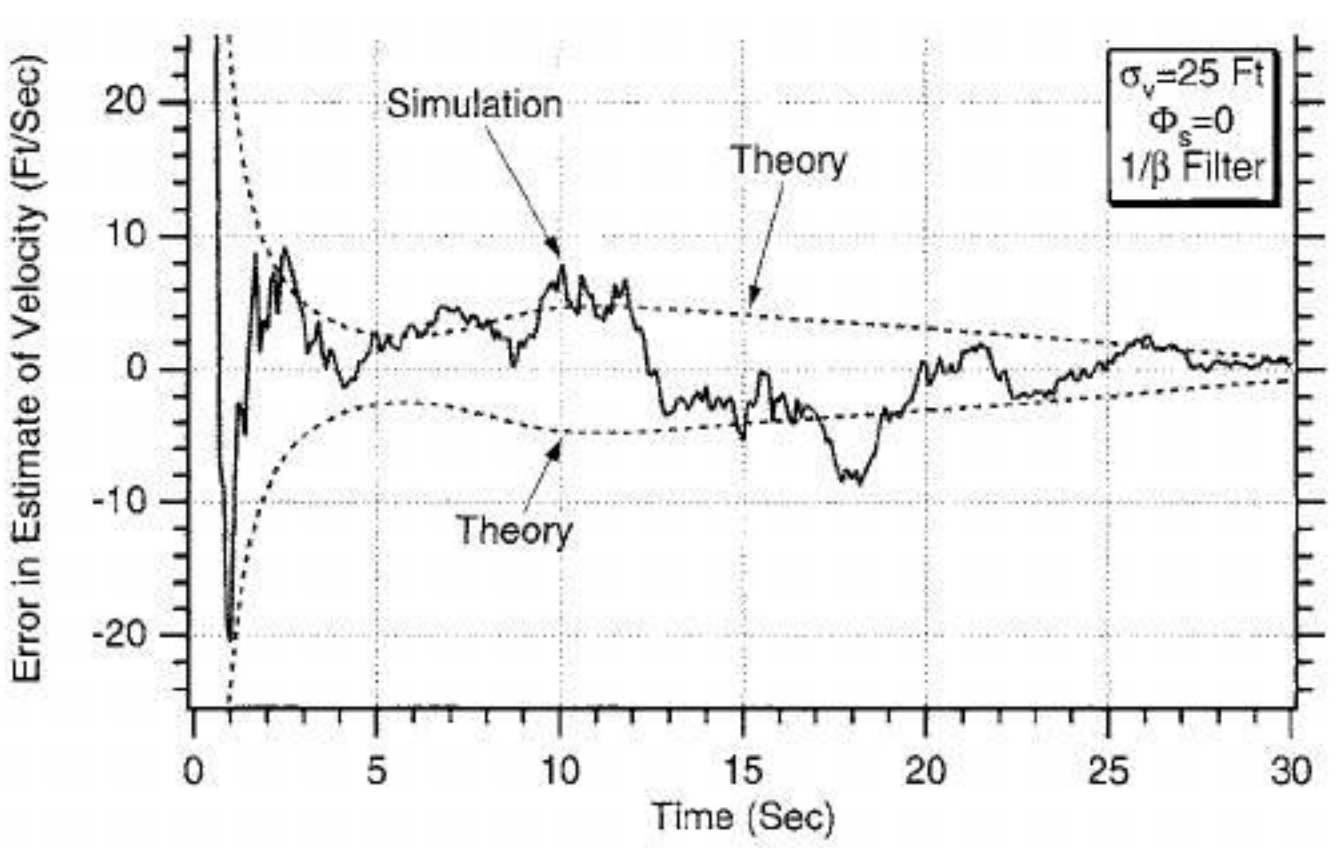

Fig. 8.12 Hangoff error has been eliminated with new extended Kalman filter.

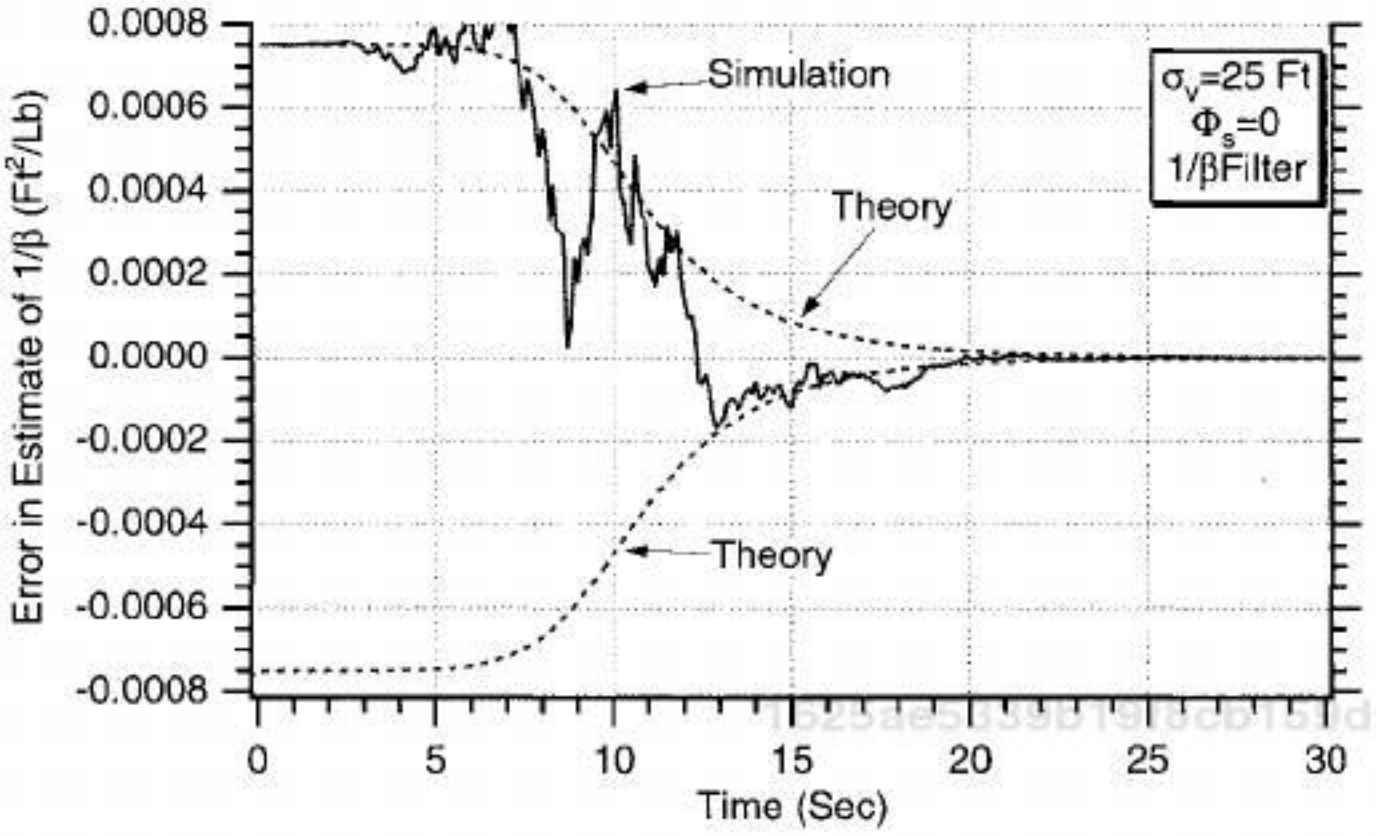

Fig. 8.13 It still takes approximately 10 s to reduce error in estimate of inverse ballistic coefficient.

was a constant. Figure 8.16 shows that before the object breaks in half (i.e., first 25 s of simulation) the extended Kalman filter's estimate of the ballistic coefficient was excellent. However, after the actual ballistic coefficient changes by a factor of two (i.e., object breaks in half), the filter's estimate of the ballistic coefficient considerably lags the actual ballistic coefficient. This is because setting the process noise to zero eventually causes the filter to stop paying attention to new measurements (i.e., filter bandwidth is reduced or the filter gain is too small for incorporating more measurements). In other words, a zero process noise extended Kalman filter eventually goes to sleep!

Although the amount of process noise to use is often determined experimentally, a good starting point is that the amount of process noise that a Kalman filter

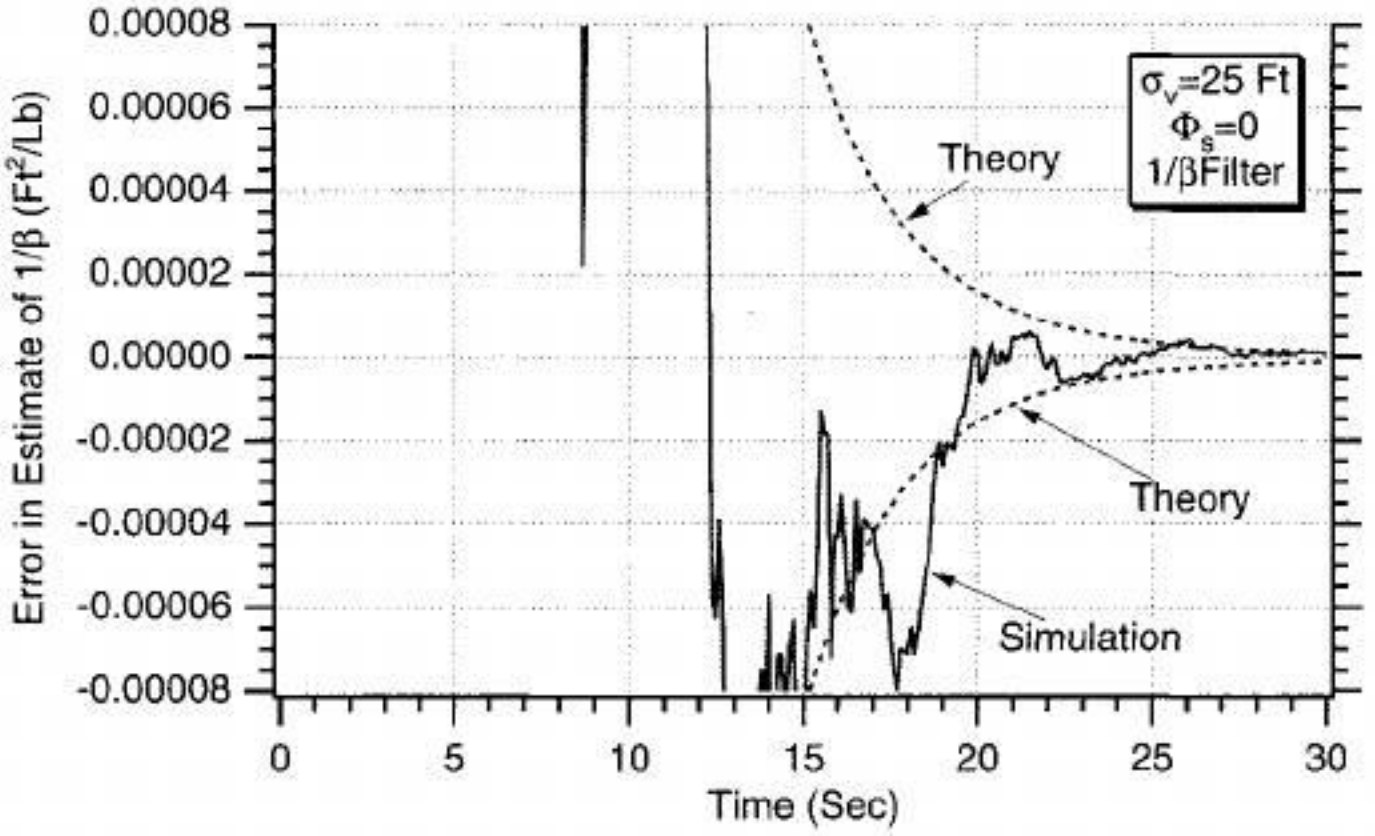

Fig. 8.14 Hangoff error has been eliminated with new extended Kalman filter.

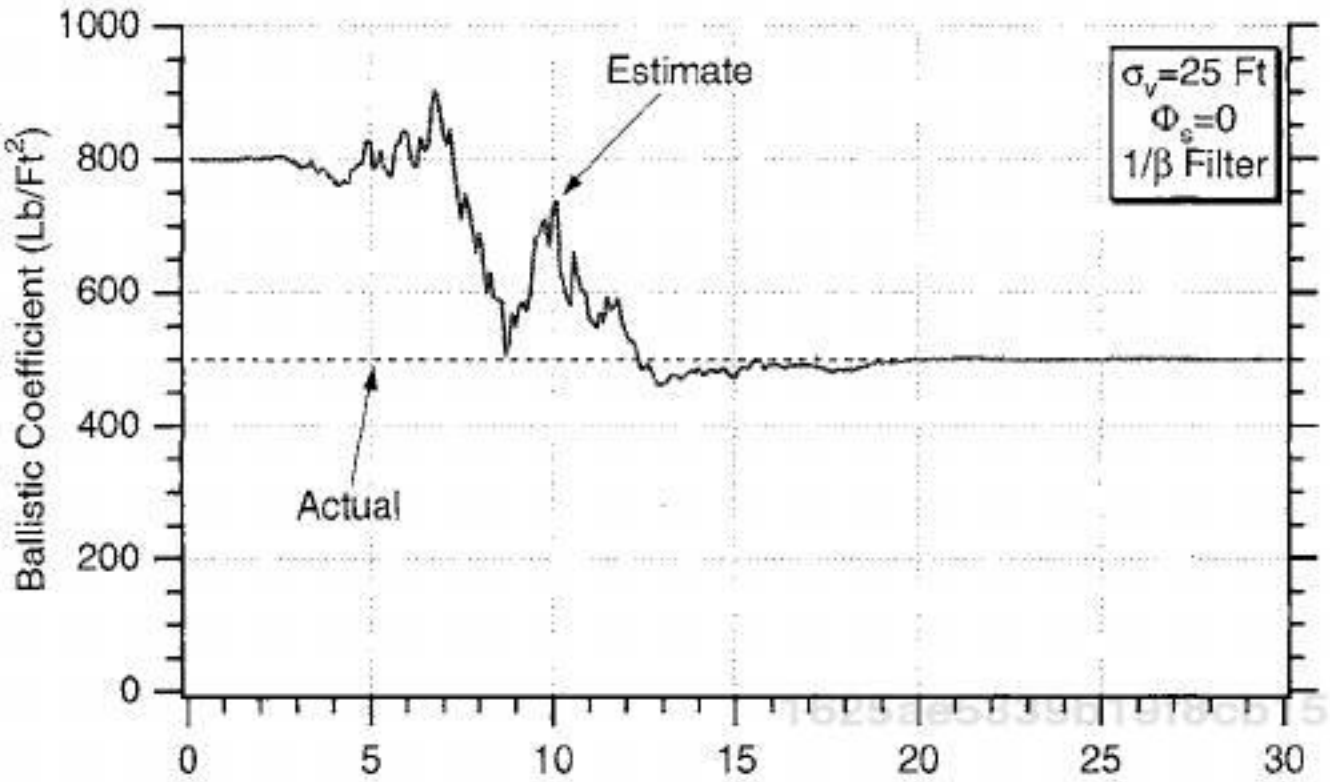

Fig. 8.15 It takes 10 s to get accurate estimates of ballistic coefficient.

requires should reflect our estimate of our lack of knowledge of the real world. One method for determining process noise is to square the uncertainty in our expected initial error in the estimate and divide by the amount of filtering time.[2] For example, because we initially thought that the ballistic coefficient was 800 lb/ft^2 rather than 500 lb/ft^2 and the amount of filtering time was 40 s (i.e., see Fig. 8.16) we could say that a good value of the process noise for the inverse ballistic coefficient filter should be

$$\Phi_s = \frac{(1/500 - 1/800)^2}{40}$$

Figure 8.17 shows that when the process noise is set to the value of the preceding equation the track quality of the extended Kalman filter is still excellent

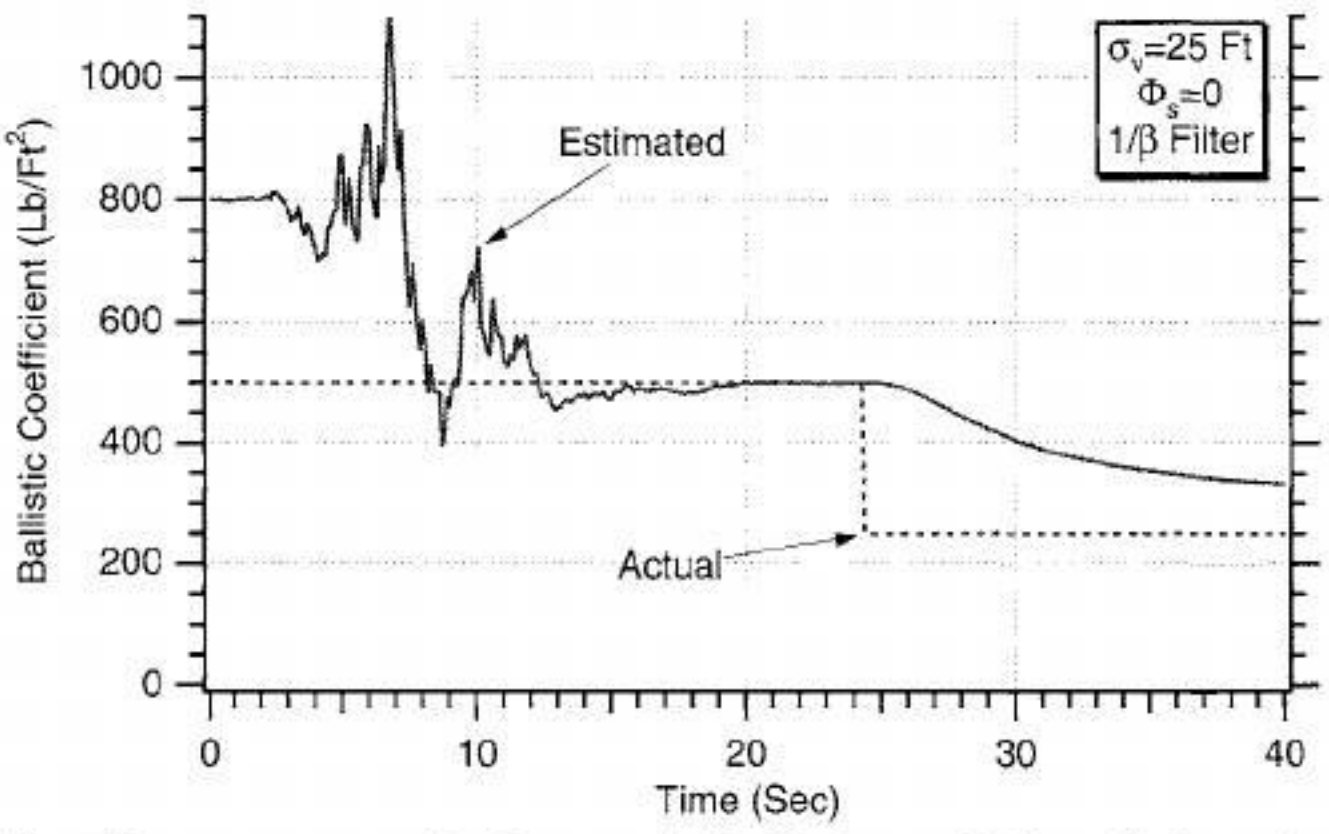

Fig. 8.16 Filter cannot track changes in ballistic coefficient if there is no process noise.

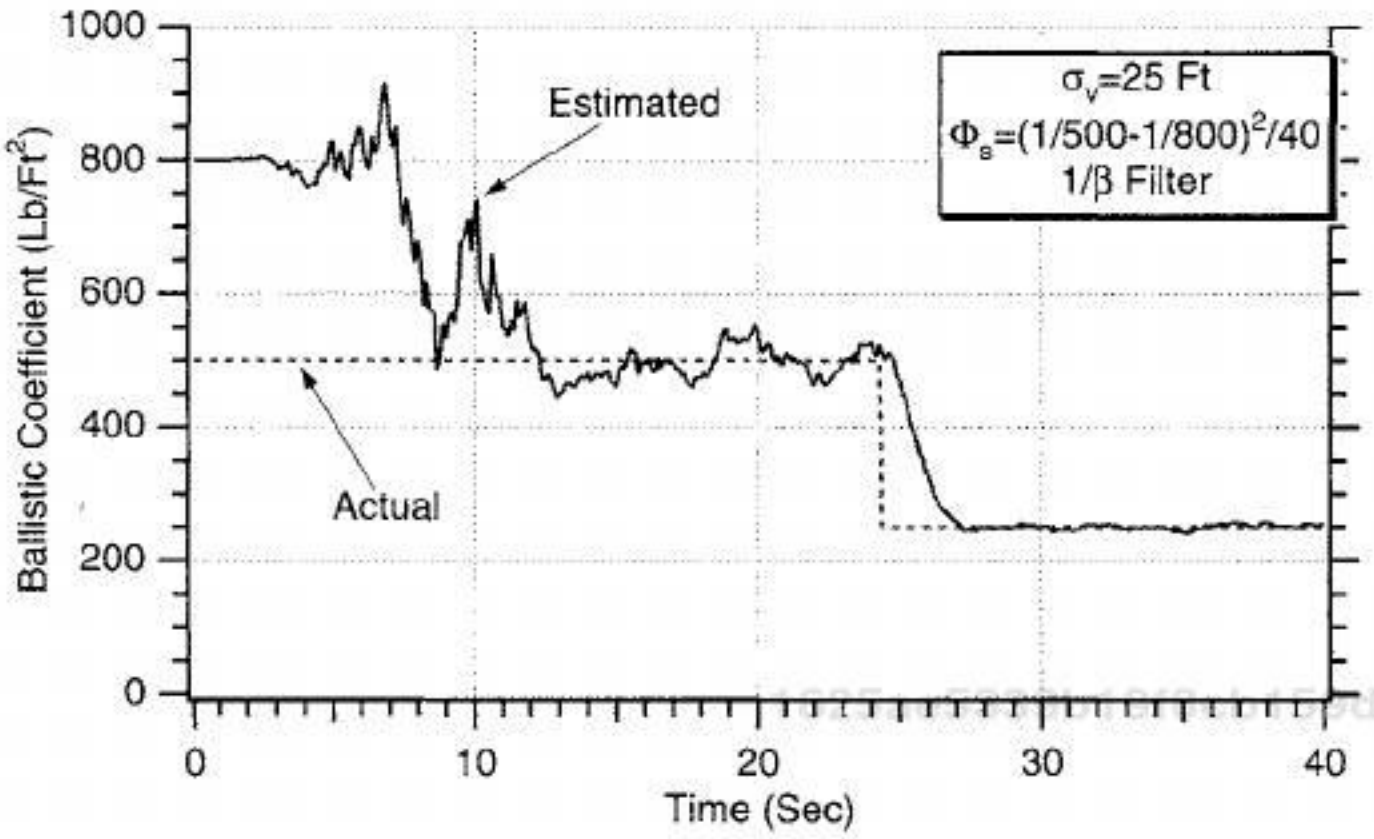

Fig. 8.17 Adding process noise enables filter to track changes in the ballistic coefficient.

for the first 25 s (i.e., where the ballistic coefficient is constant) but somewhat noisier than the case in which there was no process noise. More importantly, when the ballistic coefficient changes, the extended Kalman filter is now able to keep track of the motion.

We have now seen that it is probably prudent to include process noise in an extended Kalman filter trying to estimate the ballistic coefficient or its inverse. However, there is also a price that must be paid when process noise is included. We have just observed from Fig. 8.17 that the estimates will be noisier than they were without the inclusion of process noise. Figures 8.18–8.20 show how the errors in the estimates change when process noise is included. Figure 8.18

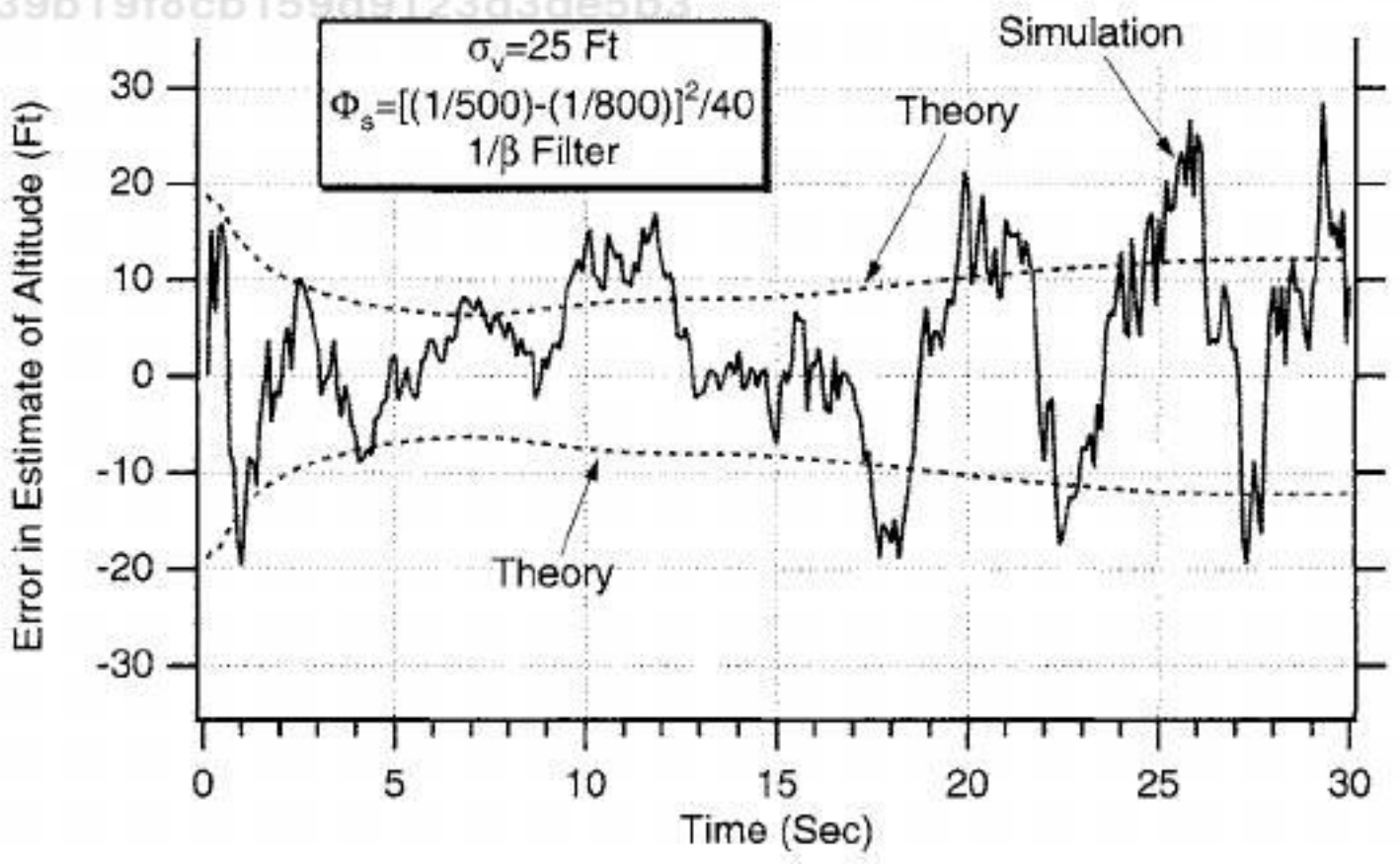

Fig. 8.18 Nominal errors in estimate of altitude deteriorate slightly when process noise is added.

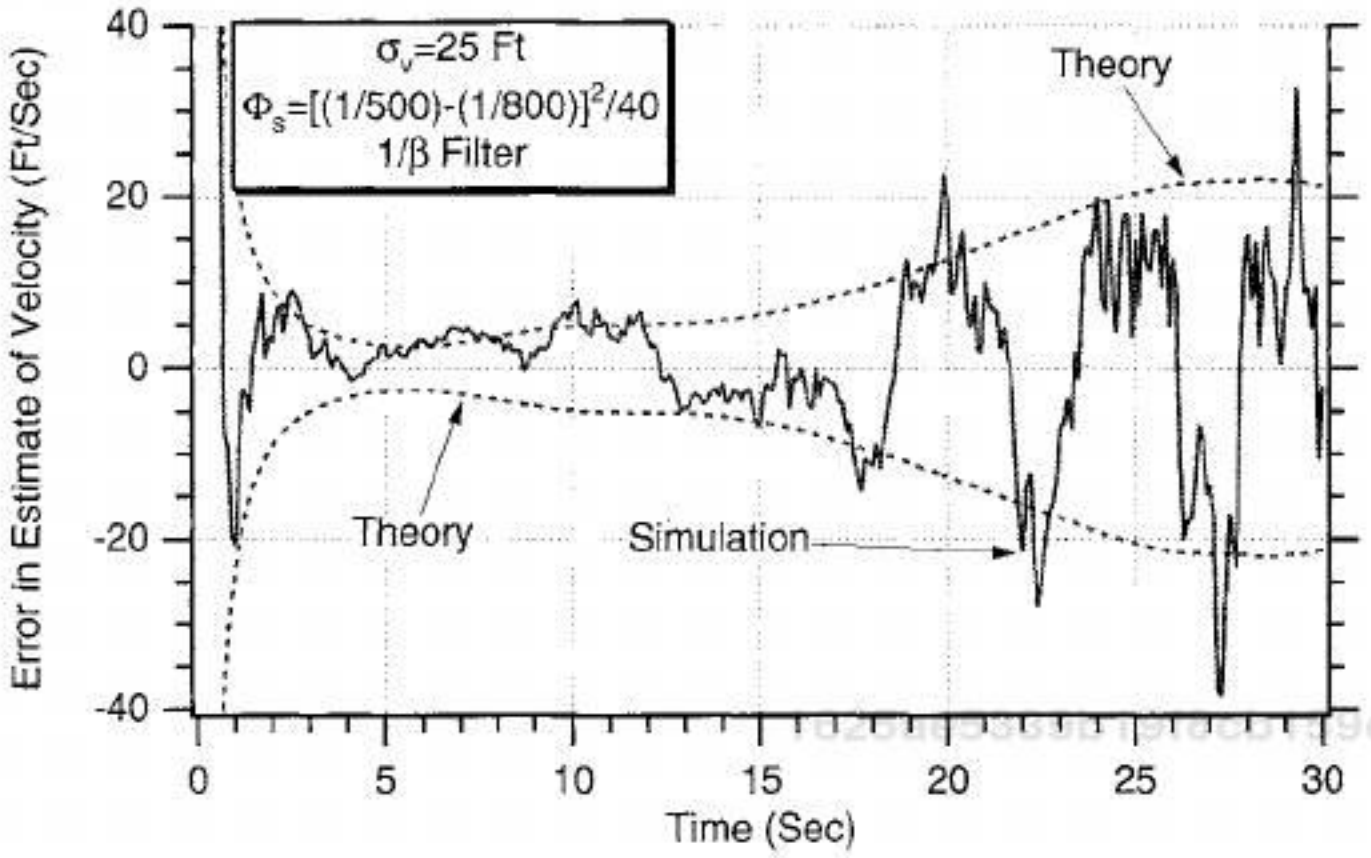

Fig. 8.19 Nominal errors in estimate of velocity deteriorate slightly when process noise is added.

presents the single-run simulation results and theoretical errors in the estimate of altitude for the inverse ballistic coefficient extended Kalman filter. By comparing Fig. 8.18 with Fig. 8.11, we can see that after 30 s the filter with process-noise yields errors in the estimate of altitude that are approximately twice as large as when the process noise is zero (i.e. 10 vs 5 ft). By comparing Fig. 8.19 with Fig. 8.12, we can see that after 30 s, the filter with process noise yields errors in the estimate of velocity that are approximately an order of magnitude larger than when the process noise is zero (i.e., 20 vs. 2 ft/s). We can see from Fig. 8.20 that when process noise is included the error in the estimate of the inverse ballistic coefficient no longer goes to zero as it did in Fig. 8.13, but it now approaches a

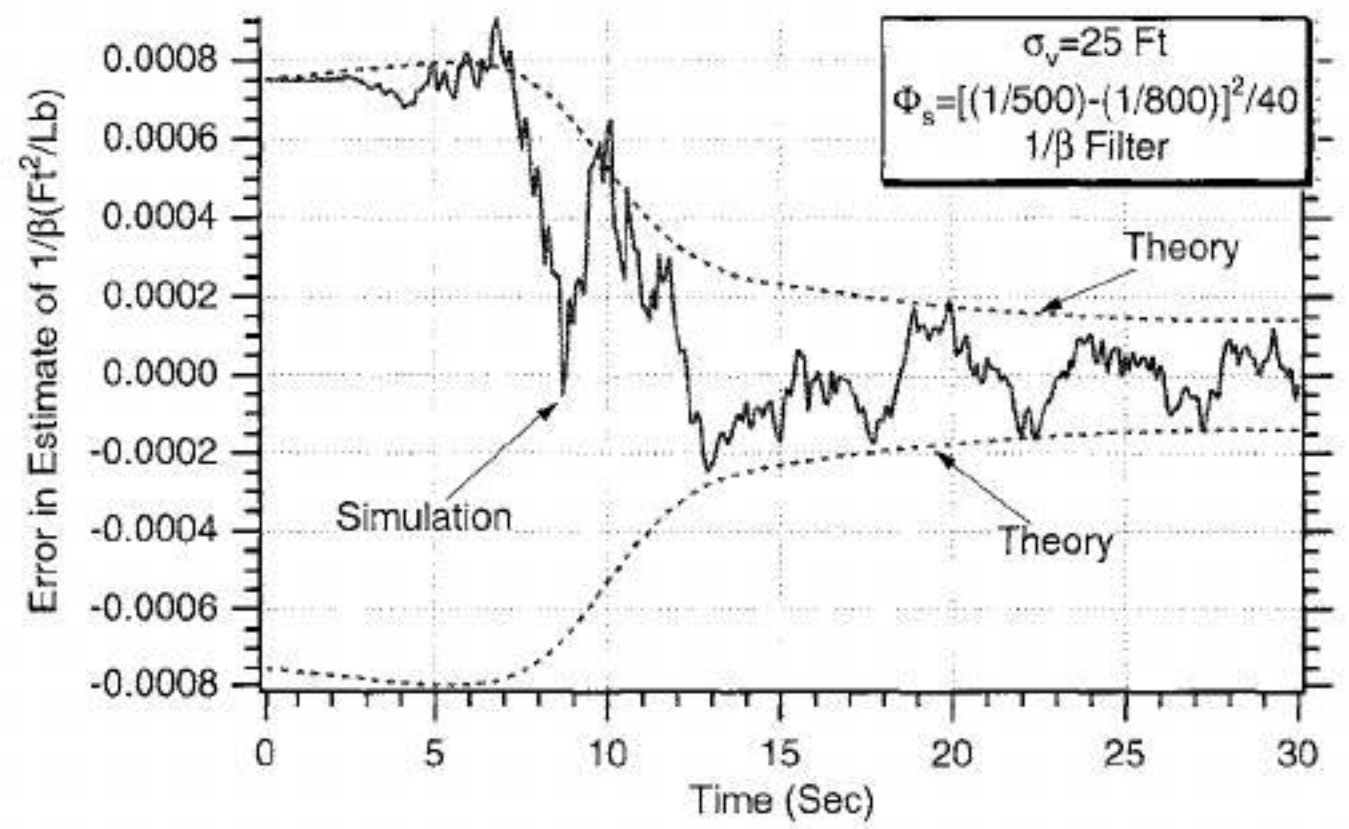

Fig. 8.20 Nominal errors in estimate of ballistic coefficient deteriorate slightly when process noise is added.

steady-state value. This is not a hangoff error because the error in the estimate of the inverse ballistic coefficient appears to have zero mean.

Linear Polynomial Kalman Filter

We have seen so far in this chapter that an extended Kalman filter could be used to track a rapidly decelerating falling object by estimating its ballistic coefficient. It is reasonable to ask how a linear polynomial Kalman filter would perform for the same task. Because the object is decelerating, at least a second-order or three-state linear polynomial Kalman filter will be required to track the object without building up excessive truncation error. For a linear polynomial Kalman-filter design we will not make use of any a priori information concerning the object's dynamics. We will simply assume that because the object's acceleration is not constant we can say that the derivative of acceleration or jerk is equal to white noise or

$$\dddot{x} = u_s$$

If we assume the states are altitude, velocity, and acceleration, the state-space equation, upon which the linear Kalman filter will be designed, can be written as

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ u_s \end{bmatrix}$$

where u_s is white noise with spectral density Φ_s. The continuous process-noise matrix for this example can be obtained from the preceding equation as

$$\boldsymbol{Q} = E\left[\begin{bmatrix} 0 \\ 0 \\ u_s \end{bmatrix} \begin{bmatrix} 0 & 0 & u_s \end{bmatrix}\right] = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \Phi_s \end{bmatrix}$$

whereas the systems dynamics matrix can be written by inspection as

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

In this example, we can assume that the measurement x^* is linearly related to the states according to

$$x^* = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix} + v$$

where v is white measurement noise. From the preceding equation we can see that the measurement matrix is given by

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

For the second-order or three-state linear polynomial Kalman filter, we summarized in Chapter 4 (see Table 4.1) the discrete form of the fundamental and measurement noise matrices for the three-state linear polynomial Kalman filter, which are given by

$$\Phi_k = \begin{bmatrix} 1 & T_s & 0.5T_s^2 \\ 0 & 1 & T_s \\ 0 & 0 & 1 \end{bmatrix}$$

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

where T_s is the sampling time and σ_n^2 is the variance of the measurement noise v. The discrete process-noise matrix for the second-order or three-state linear polynomial Kalman filter was also shown in Table 4.2 of Chapter 4 to be

$$\boldsymbol{Q}_k = \Phi_s \begin{bmatrix} \frac{T_s^5}{20} & \frac{T_s^4}{8} & \frac{T_s^3}{6} \\ \frac{T_s^4}{8} & \frac{T_s^3}{3} & \frac{T_s^2}{2} \\ \frac{T_s^3}{6} & \frac{T_s^2}{2} & T_s \end{bmatrix}$$

We now have enough information to build the linear polynomial Kalman filter to estimate the altitude, velocity, and acceleration of the falling object based on noisy measurements of the object's altitude. However, if we would also like to estimate the object's ballistic coefficient, which is not required for tracking the object, then a little more work must be done.

At the beginning of this chapter, we showed that the acceleration of the object could be expressed in terms of the drag and gravity acting on the object as

$$\ddot{x} = \frac{g\rho\dot{x}^2}{2\beta} - g$$

We can invert the preceding expression to solve for the ballistic coefficient to obtain

$$\beta = \frac{g\rho\dot{x}^2}{2(\ddot{x} + g)}$$

where the air density is a function of altitude according to

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

Therefore, we can use the altitude, velocity, and acceleration estimates from the three-state linear polynomial Kalman filter to estimate the ballistic coefficient according to

$$\hat{\beta} = \frac{g\hat{\rho}\hat{\dot{x}}^2}{2(\hat{\ddot{x}} + g)}$$

where the estimated air density is a function of the estimated altitude from

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

Listing 8.3 is a simulation of the falling object and the three-state linear polynomial Kalman filter for estimating the object's altitude, velocity, acceleration, and ballistic coefficient. We can see from Listing 8.3 that the initial conditions for the state estimates and covariance matrix are the same as they were in Listing 8.2. This filter has process noise, whose spectral density is given by

$$\Phi_s = \frac{322^2}{30}$$

to account for the fact that the object's deceleration might be as large as 10 g (i.e., 322 ft/s^2) over a 30-s flight. The third diagonal element of the initial covariance matrix also has an initial value to reflect the uncertainty in the object's deceleration (i.e., $P_{33} = 322*322$). We can see from Listing 8.3 that the object's acceleration state of the filter has initially been set to zero, which means we are not assuming any a priori information regarding the object's initial acceleration.

The nominal case of Listing 8.3 was run, and the error in the estimate results are displayed in Figs. 8.21–8.23. First, we can see from Figs. 8.21–8.23 that because the single-run errors in the estimates are within the theoretical bounds the linear second-order polynomial Kalman filter appears to be working correctly. We can also see from Fig. 8.21 that the error in the estimate of altitude is approximately 15 ft, which is slightly larger than the results of the extended Kalman filter with the inverse ballistic coefficient (see Fig. 8.18, where the error in the estimate of altitude is approximately 10 ft when process noise is present). Using smaller values for the process noise in the linear filter probably would have yielded nearly identical results to the extended filter. We can see from Fig. 8.22 that the error in the estimate of velocity is approximately 30 ft/s, which is slightly larger than the results of the extended Kalman filter with the inverse ballistic coefficient (see Fig. 8.20, where the error in the estimate of velocity is approximately 20 ft/s when process noise is present). Again, using smaller values for the process noise in the linear filter probably would have yielded nearly identical results to the extended filter. We can see from Fig. 8.23 that the

Listing 8.3 Using three-state linear polynomial Kalman filter to track falling object under the influence of drag

```
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(3,3),P(3,3),M(3,3),PHIP(3,3),PHIPPHIT(3,3),GAIN(3,1)
      REAL*8 Q(3,3),HMAT(1,3),HM(1,3),MHT(3,1)
      REAL*8 PHIT(3,3)
      REAL*8 HMHT(1,1),HT(3,1),KH(3,3),IDN(3,3),IKH(3,3)
      INTEGER ORDER
      ITERM=1
      G=32.2
      SIGNOISE=25.
      X=200000.
      XD=-6000.
      BETA=500.
      XH=200025.
      XDH=-6150.
      XDDH=0.
      XNT=322.
      OPEN(2,STATUS='UNKNOWN',FILE='COVFIL')
      OPEN(1,STATUS='UNKNOWN',FILE='DATFIL')
      ORDER=3
      TS=.1
      TF=30.
      PHIS=XNT*XNT/TF
      T=0.
      S=0.
      H=.001
      HP=.001
      TS2=TS*TS
      TS3=TS2*TS
      TS4=TS3*TS
      TS5=TS4*TS
      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
      PHI(1,1)=1.
      PHI(1,2)=TS
      PHI(1,3)=.5*TS*TS
      PHI(2,2)=1.
      PHI(2,3)=TS
```

(*continued*)

Listing 8.3 (*Continued*)

```
      PHI(3,3)=1.
      CALL  MATTRN(PHI,ORDER,ORDER,PHIT)
      Q(1,1)=TS5*PHIS/20.
      Q(1,2)=TS4*PHIS/8.
      Q(1,3)=PHIS*TS3/6.
      Q(2,1)=Q(1,2)
      Q(2,2)=PHIS*TS3/3.
      Q(2,3)=.5*TS2*PHIS
      Q(3,1)=Q(1,3)
      Q(3,2)=Q(2,3)
      Q(3,3)=PHIS*TS
      IDN(1,1)=1.
      IDN(2,2)=1.
      IDN(3,3)=1.
      P(1,1)=SIGNOISE*SIGNOISE
      P(2,2)=20000.
      P(3,3)=XNT*XNT
      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
                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  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)
```

(*continued*)

Listing 8.3 (*Continued*)

```
      CALL MATSUB(IDN,ORDER,ORDER,KH,IKH)
      CALL MATMUL(IKH,ORDER,ORDER,M,ORDER,
        ORDER,P)
      CALL GAUSS(XNOISE,SIGNOISE)
      RES=X+XNOISE-XH-TS*XDH-.5*TS*TS*XDDH
      XH=XH+TS*XDH+.5*TS*TS*XDDH+GAIN(1,1)*RES
      XDH=XDH+TS*XDDH+GAIN(2,1)*RES
      XDDH=XDDH+GAIN(3,1)*RES
      RHOH=.0034*EXP(-XH/22000.)
      BETAH=16.1*RHOH*XDH*XDH/(XDDH+32.2)
      ERRX=X-XH
      SP11=SQRT(P(1,1))
      ERRXD=XD-XDH
      SP22=SQRT(P(2,2))
      ERRXDD=XDD-XDDH
      SP33=SQRT(P(3,3))
      WRITE(9,*)T,X,XH,XD,XDH,XDD,XDDH,BETA,BETAH
      WRITE(1,*)T,X,XH,XD,XDH,XDD,XDDH,BETA,BETAH
      WRITE(2,*)T,ERRX,SP11,-SP11,ERRXD,SP22,-SP22,
     1   ERRXDD,SP33,-SP33
        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
```


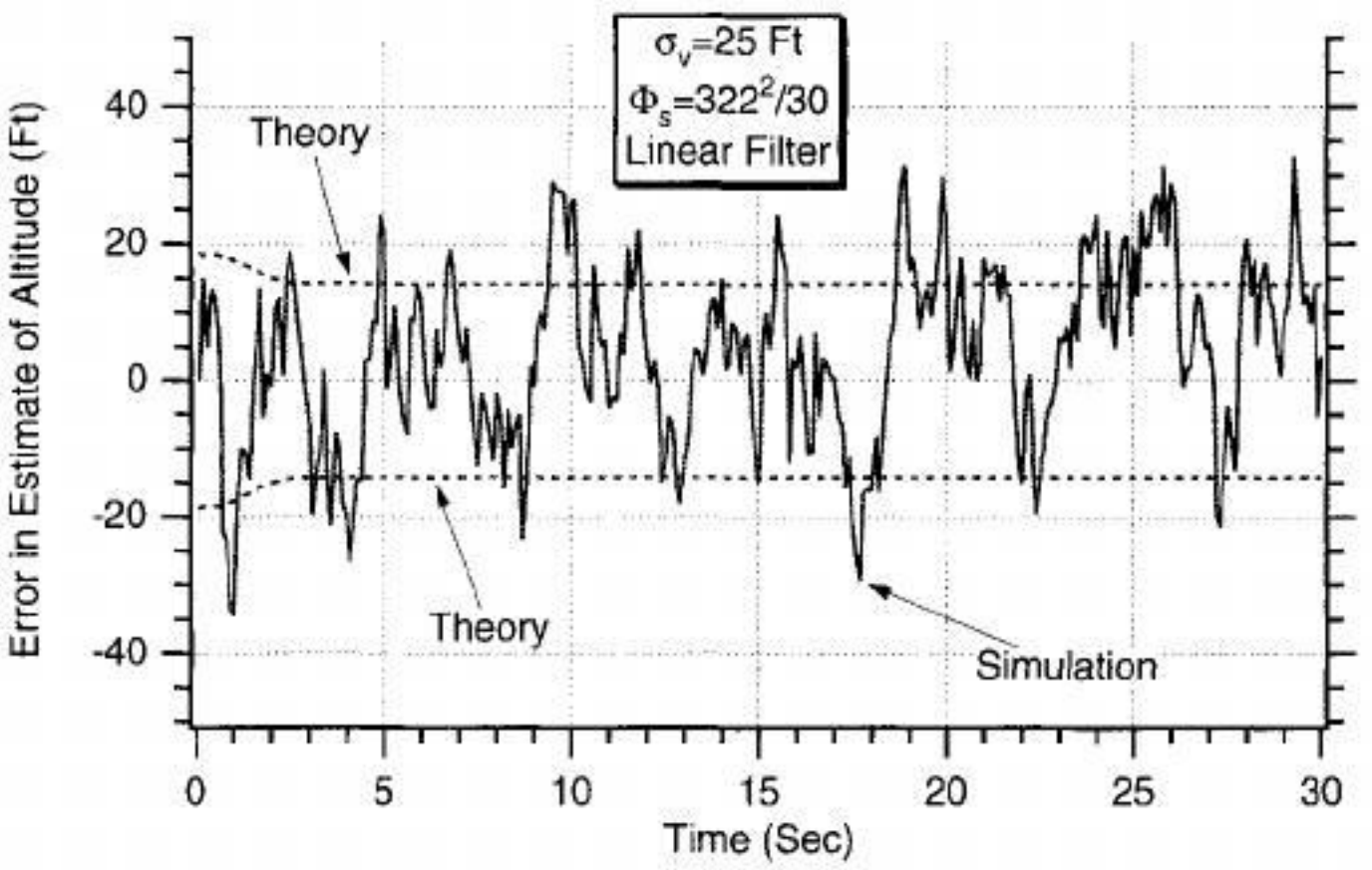

Fig. 8.21 Altitude estimates are slightly worse with linear filter.

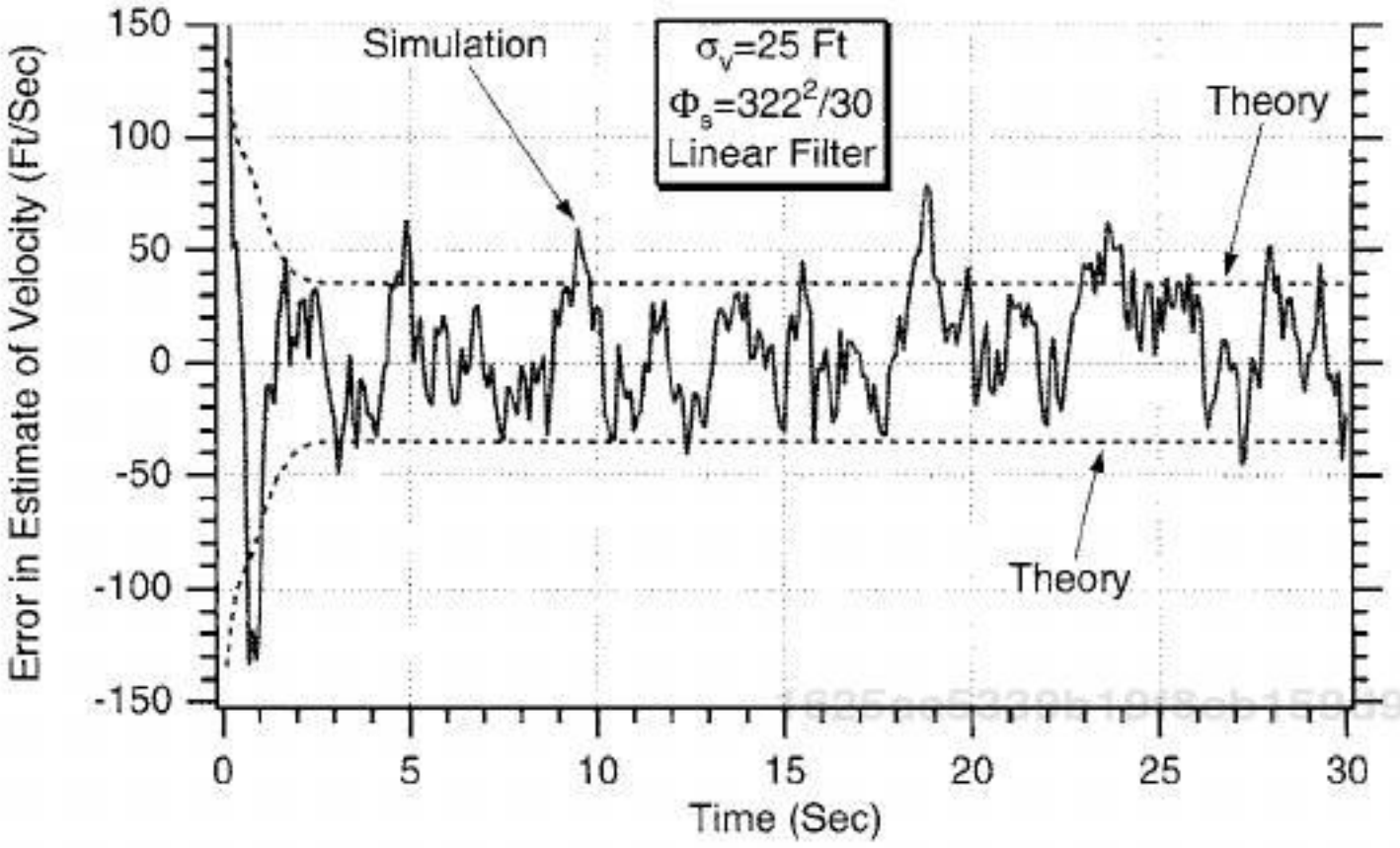

Fig. 8.22 Velocity estimates slightly worse with linear filter.

error in the estimate of acceleration is approximately 50 ft/s^2 in the steady state. We cannot compare these results to the extended filter results because the extended Kalman filter that estimated the inverse ballistic coefficient did not estimate the object's acceleration directly.

To clarify the filtering results even further, Fig. 8.24 compares the acceleration estimate of the three-state linear polynomial Kalman filter to the actual acceleration of the object. We can see that the linear filter is able to track the highly changing acceleration characteristics of the object. There does not appear to be any significant lag between the actual acceleration and the estimated acceleration. In addition, we can see from Fig. 8.24 that the estimates are not too noisy, indicating that just about the right amount of process noise was used.

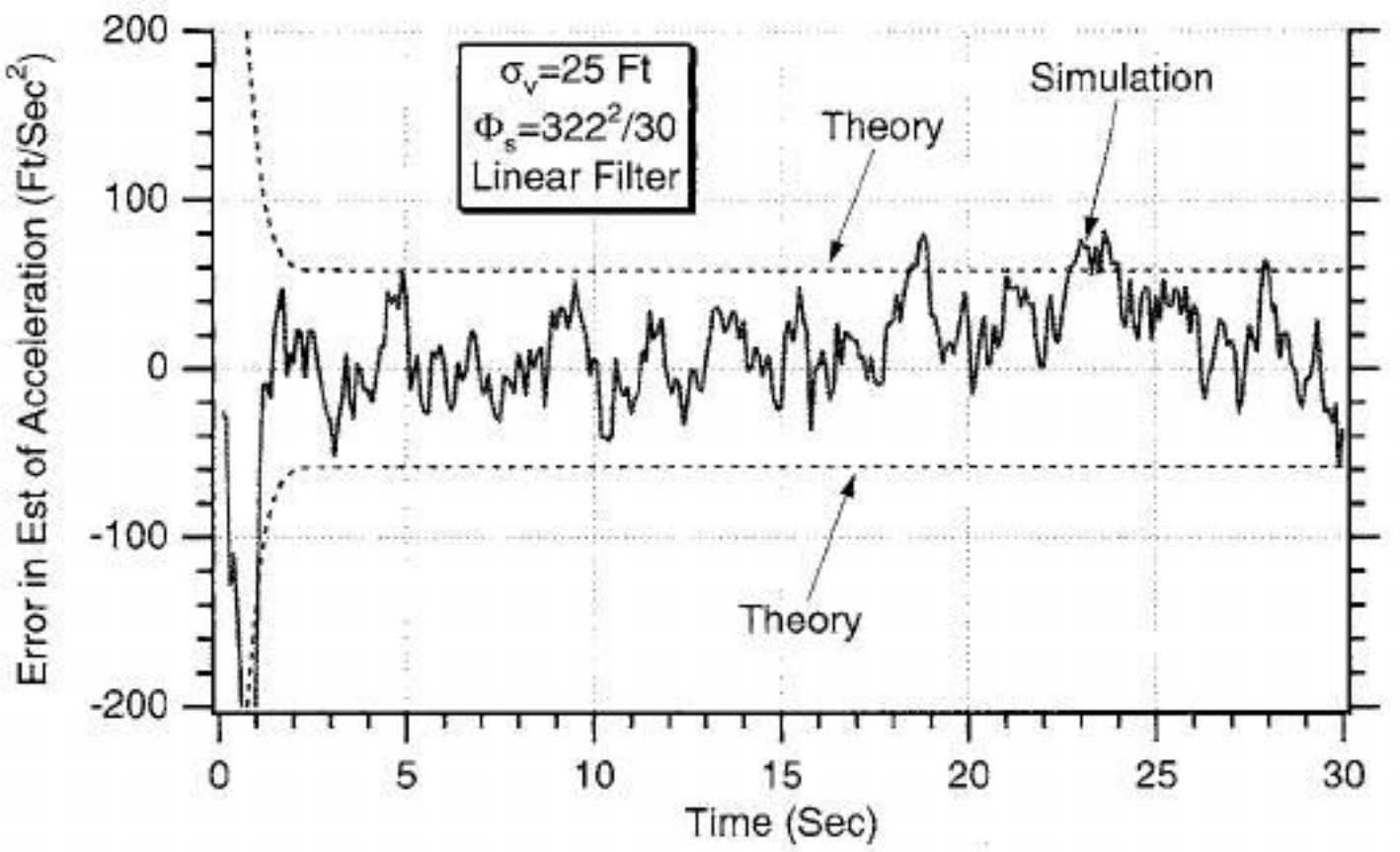

Fig. 8.23 Second-order linear polynomial Kalman filter is able to track object's acceleration.

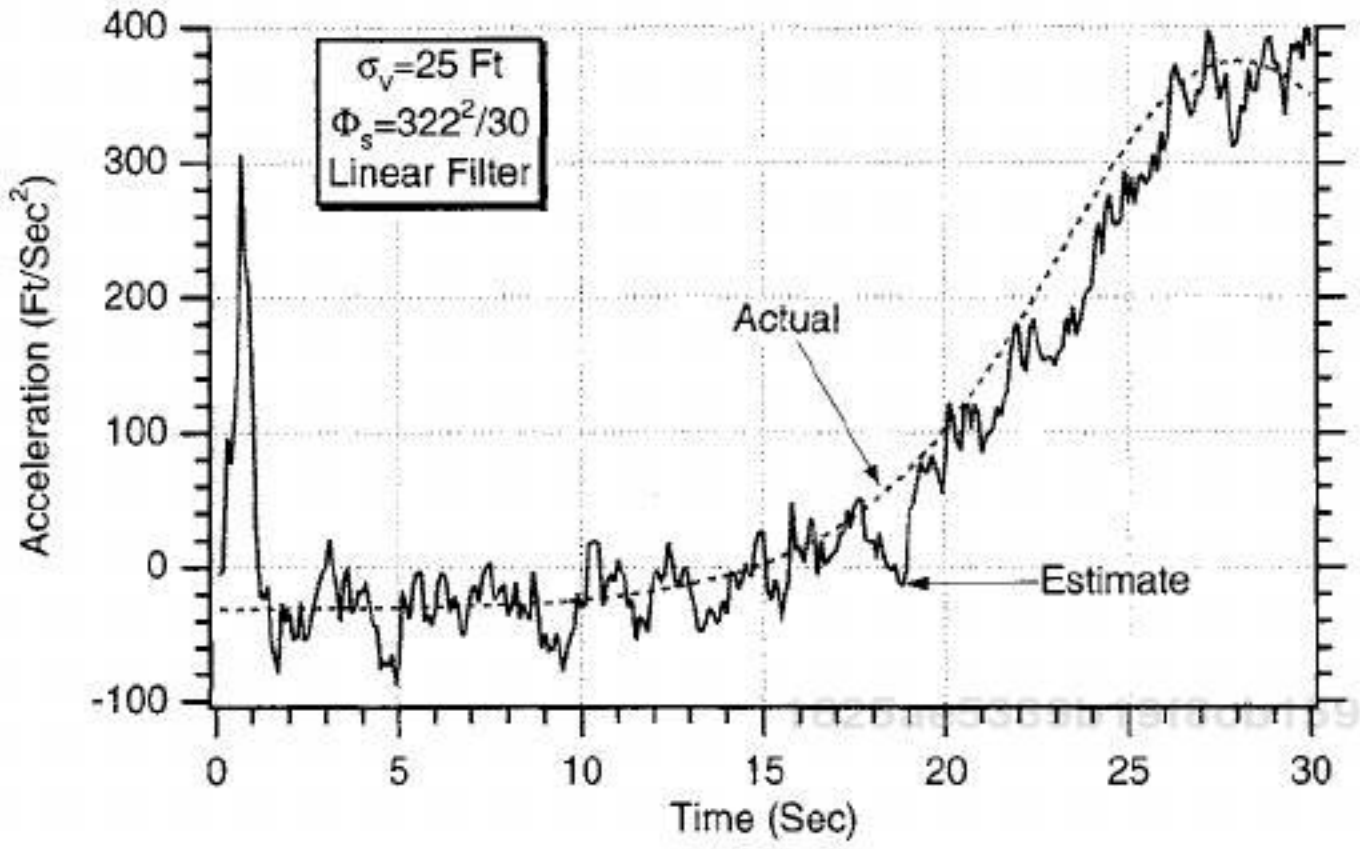

Fig. 8.24 Second-order linear polynomial Kalman filter is able to track object's acceleration without too much lag or noise propagation.

From the same computer run where the acceleration estimates of the second-order linear polynomial Kalman filter were fairly decent, an attempt was also made to estimate the object's ballistic coefficient using the formulas already derived

$$\hat{\beta} = \frac{g\hat{\rho}\hat{\dot{x}}^2}{2(\hat{\ddot{x}} + g)}$$

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

We can see from Fig. 8.25 that using the linear filter to estimate ballistic coefficient is not a good idea because the estimate is far too noisy for most of the

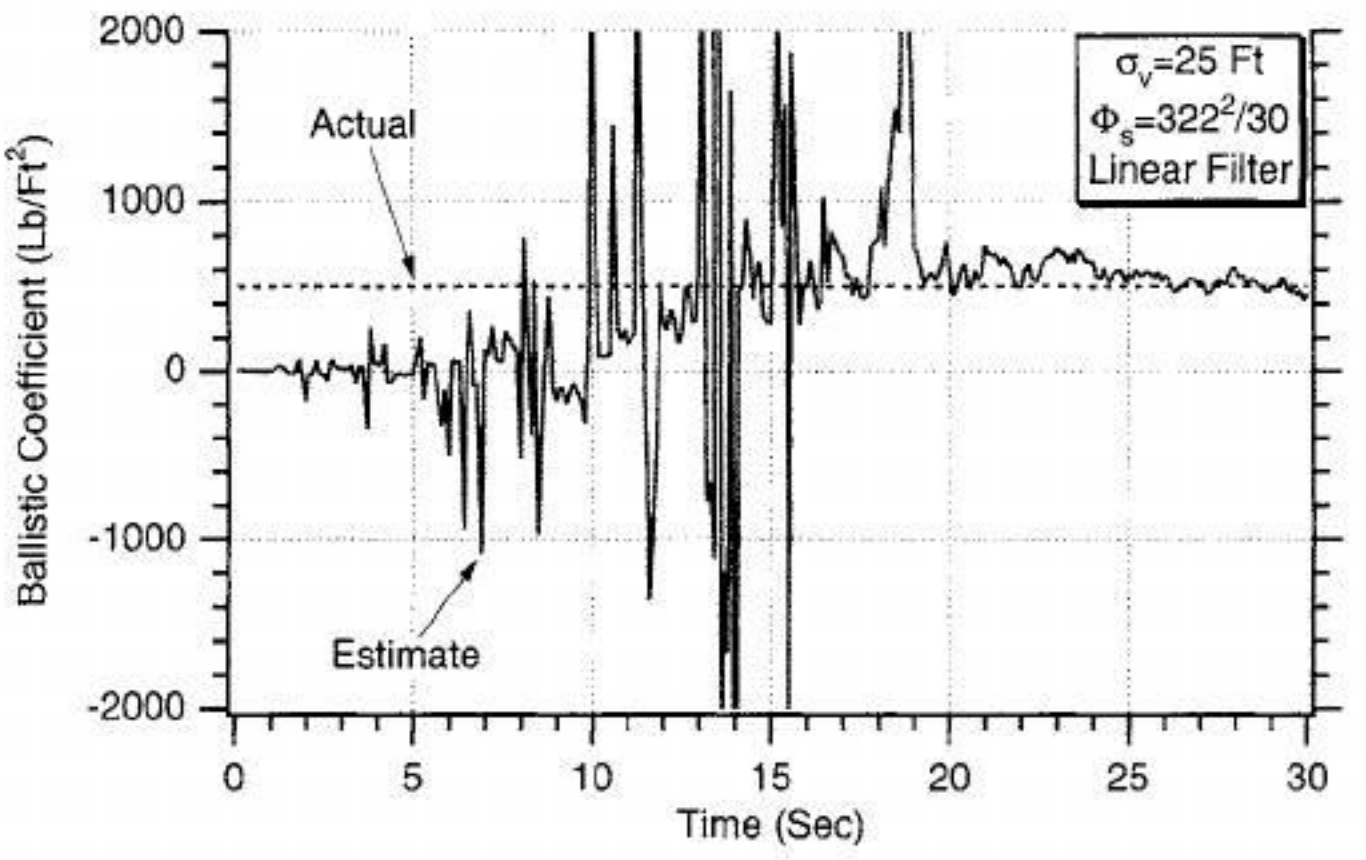

Fig. 8.25 It takes approximately 20 s for the second-order linear polynomial Kalman filter to accurately estimate the ballistic coefficient.

flight. Only after about 20 s are we able to estimate the ballistic coefficient. The accuracy to which the ballistic coefficient can be estimated with the linear polynomial Kalman filter is far worse than the accuracy results of both extended Kalman filters. Of course, an estimate of the ballistic coefficient is not required for the second-order linear polynomial Kalman filter to estimate the altitude, velocity, and acceleration of the falling object, and so this type of filter may be perfectly suitable for this application.

To understand why the ballistic coefficient estimate of the linear filter was so terrible, another experiment was conducted. Because the ballistic coefficient is inversely proportional to the acceleration, it was hypothesized that the division by the noisy acceleration estimate effectively multiplied the noise. To check the hypothesis, Listing 8.3 was modified so that the ballistic coefficient estimate made use of perfect acceleration information, rather than estimated acceleration from the filter, or

$$\hat{\beta} = \frac{g\hat{\rho}\hat{\dot{x}}^2}{2(\ddot{x}+g)}$$

We can now see from Fig. 8.26 that the linear filter's ballistic coefficient estimate is now near perfect, indicating that there was too much noise on the acceleration estimate to accurately provide an estimate of the ballistic coefficient for most of the flight.

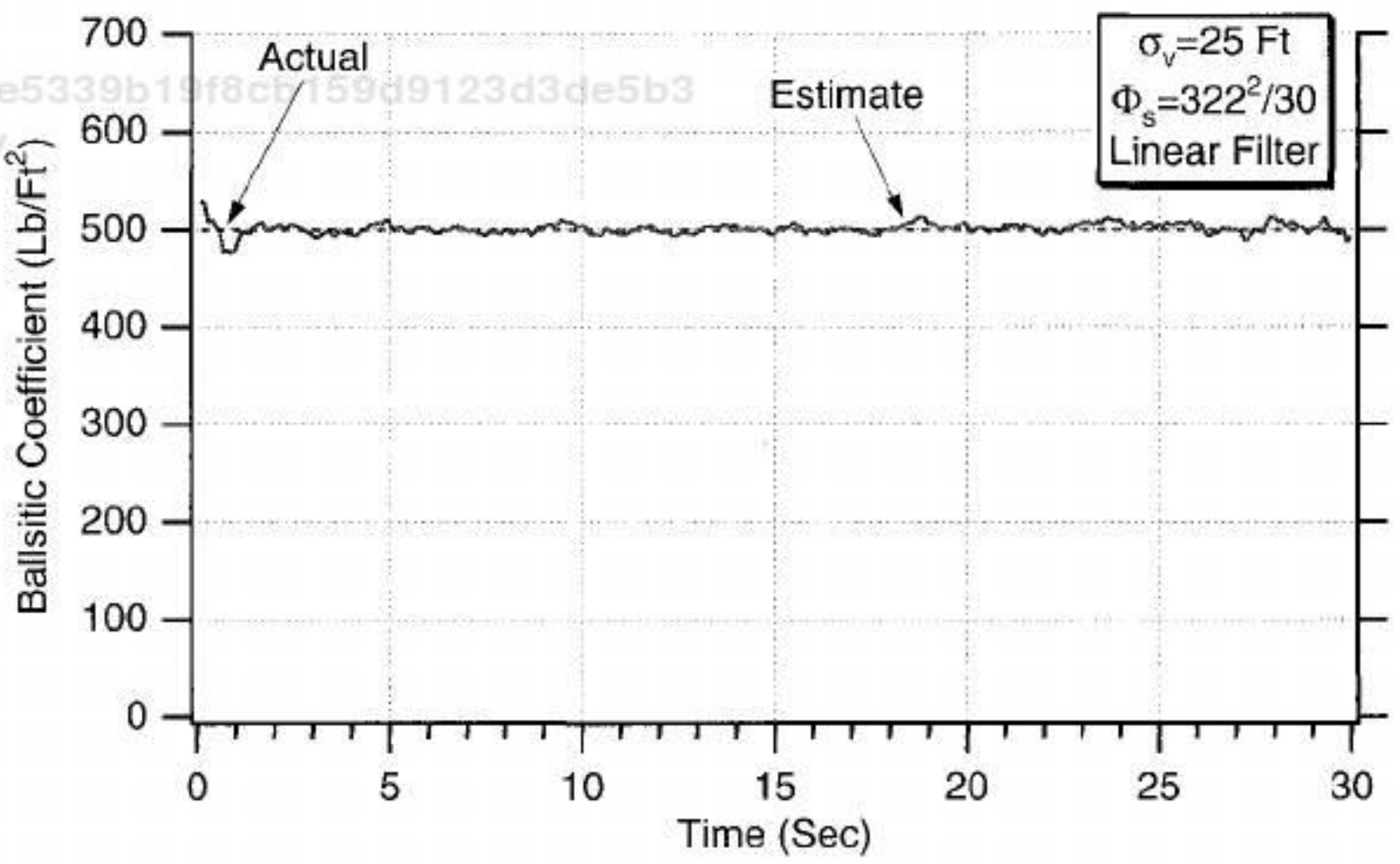

Fig. 8.26 If acceleration is known exactly, ballistic coefficient estimate is near perfect.

Summary

In this chapter we have worked on several extended Kalman filters for another version of the falling-object problem. We have again seen the utility of adding process noise to the extended Kalman filter to eliminate hangoff errors in the estimates and to make the filter more responsive to unanticipated changes in the object's dynamics. We have also seen that changing the states of an extended Kalman filter can sometimes reduce estimation errors because certain states make the problem look less nonlinear. Finally, we also have seen that for this application a second-order linear polynomial Kalman filter could be made to work if the only goal was to track the object and estimate altitude, velocity, and acceleration. On the other hand, we also saw that the linear filter had certain deficiencies if we attempted to also estimate the object's ballistic coefficient.

References

[1]Gelb., A., *Applied Optimal Estimation*, Massachusetts Inst. of Technology Press, Cambridge, MA, 1974, pp. 194–198.

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

This page intentionally left blank

Chapter 9

Cannon-Launched Projectile Tracking Problem

Introduction

IN ALL of the applications so far, there was only one sensor measurement from which we built our Kalman filters. In this chapter we will attempt to track a cannon-launched projectile with a sensor that measures both the range and angle to the projectile. We will first see how the extra sensor measurement is incorporated into an extended Kalman filter. In this example the real world is linear in the Cartesian coordinate system, but the measurements are nonlinear. The extended Kalman filter will initially be designed in the Cartesian coordinate system to establish a benchmark for tracking performance. We will then see if performance can be improved by redesigning the filter in the polar coordinate system, where the measurements are linear but the model of the real world is nonlinear. Finally, we will also see if linear coupled and decoupled polynomial Kalman filters also can be made to work for this problem.

Problem Statement

Consider another simple example in which a radar tracks a cannon-launched projectile traveling in a two-dimensional, drag-free environment, as shown in Fig. 9.1. After the projectile is launched at an initial velocity, only gravity acts on the projectile. The tracking radar, which is located at coordinates x_R, y_R of Fig. 9.1, measures the range r and angle θ from the radar to the projectile.

In this example the projectile is launched at an initial velocity of 3000 ft/s at a 45-deg angle with respect to the horizontal axis in a zero-drag environment (i.e., atmosphere is neglected). The radar is located on the ground and is 100,000 ft downrange from where the projectile is initially launched. The radar is considered to have an angular measurement accuracy of 0.01 rad and a range measurement accuracy of 100 ft.

Before we can proceed with the development of a filter for this application, it is important to first get a numerical and analytical feel for all aspects of the problem. Because only gravity acts on the projectile, we can say that in the downrange or x direction there is no acceleration, whereas in the altitude or y direction there is the acceleration of gravity g or[1]

$$\ddot{x}_T = 0$$

$$\ddot{y}_T = -g$$

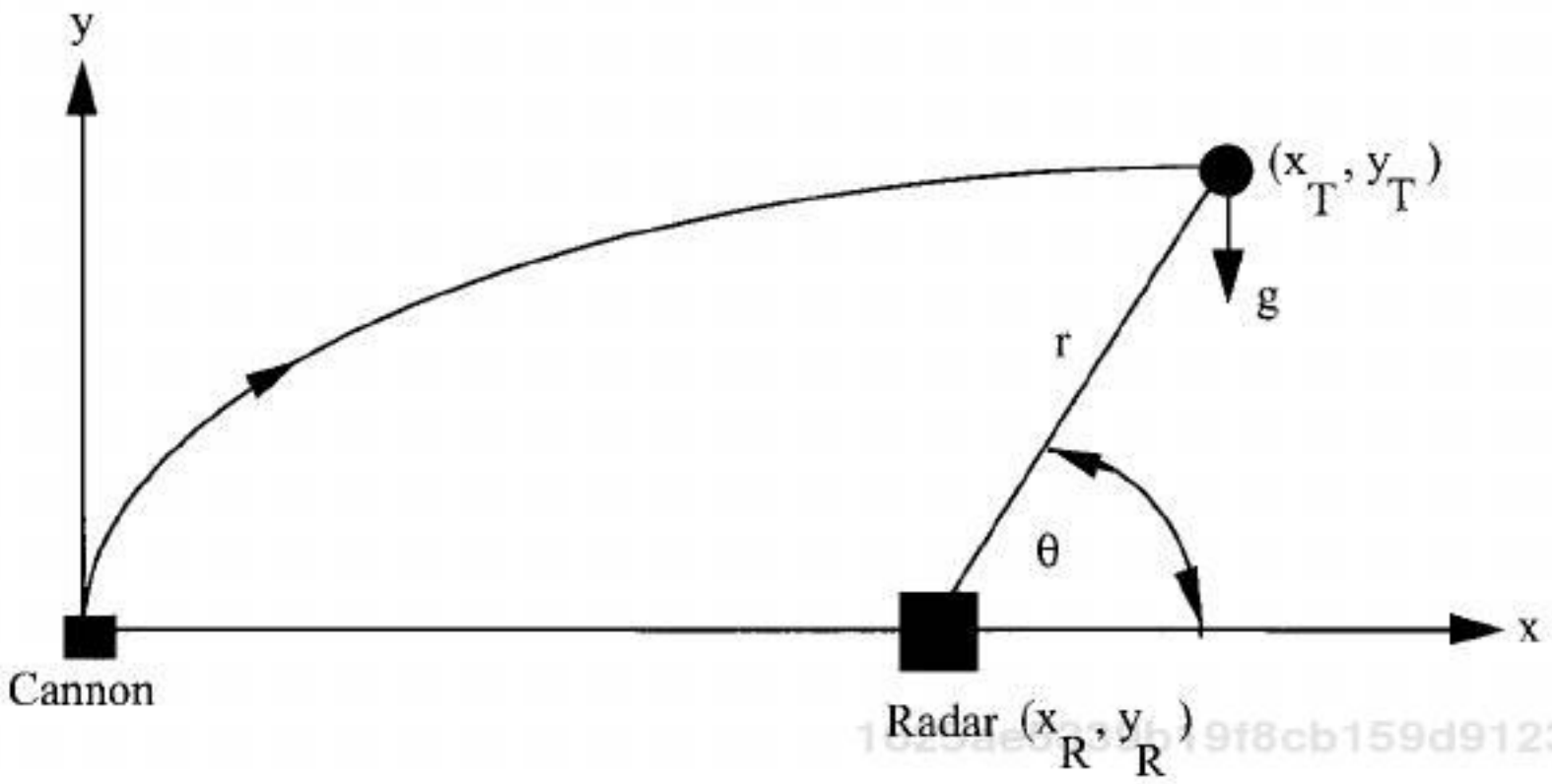

Fig. 9.1 Radar tracking cannon-launched projectile.

where the acceleration of gravity has a value of 32.2 ft/s^2 in the English system of units. Using trigonometry, the angle and range from the radar to the projectile can be obtained by inspection of Fig. 9.1 to be

$$\theta = \tan^{-1}\left(\frac{y_T - y_R}{x_T - x_R}\right)$$

$$r = \sqrt{(x_T - x_R)^2 + (y_T - y_R)^2}$$

Before we build an extended Kalman filter to track the projectile and estimate its position and velocity at all times based on noisy angle and range measurements, let us see what can be done without any filtering at all. By inspection of Fig. 9.1, we can see that we can express the location of the cannon-launched projectile (i.e., x_T, y_T) in terms of the radar range and angle as

$$x_T = r\cos\theta + x_R$$

$$y_T = r\sin\theta + y_R$$

If the radar coordinates are known precisely and r^* and θ^* are the noisy radar range and angle measurements, then the location of the projectile at any time can be calculated directly or estimated without any filtering at all as

$$\hat{x}_T = r^*\cos\theta^* + x_R$$

$$\hat{y}_T = r^*\sin\theta^* + y_R$$

The estimates of the velocity components of the projectile can be obtained from the noisy position estimates by using the definition of a derivative from calculus. In other words, we simply subtract the old position estimate from the

new position estimate and divide by the sampling time or the time between measurements to get the estimated projectile velocity components or

$$\hat{\dot{x}}_{T_k} = \frac{\hat{x}_{T_k} - \hat{x}_{t_{k-1}}}{T_s}$$

$$\hat{\dot{y}}_{T_k} = \frac{\hat{y}_{T_k} - \hat{y}_{t_{k-1}}}{T_s}$$

Figure 9.2 displays the true projectile trajectory and the estimated projectile trajectory based on the raw radar measurements. We can see that the estimated trajectory appears to be quite close to the actual trajectory, so one might wonder why filtering techniques are required in this example. On the other hand, Figs. 9.3 and 9.4 show that when the raw measurements are used to estimate the projectile velocity the results are very poor. For example, the actual downrange velocity is constant and is approximately 2000 ft/s. However, we can see from Fig. 9.3 that the estimated downrange velocity varies from 500 to 5000 ft/s. Similarly, Fig. 9.4 shows that the actual altitude velocity varies from 2000 ft/s at cannon ball launch to −2000 ft/s near impact. However, the figure also shows that the estimated altitude velocity is often in error by several thousand feet per second. Clearly, filtering is required if we desire a better way of estimating the projectile's velocity.

It might appear from Fig. 9.2 that the position estimates of the projectile were satisfactory because they appeared to be so close to the actual trajectory. We have seen in the earlier filtering examples of this text that we often compared the error in the estimates to the theoretical predictions of the covariance matrix. Although in this example there is no covariance matrix, we can look at the errors in the estimates of downrange and altitude. For this problem the initial downrange and altitude estimates of the projectile were in error by 1000 ft. We can see from Figs. 9.5 and 9.6 that the errors in the estimates of the projectile's downrange and altitude are never substantially below 1000 ft. In other words, we are not reducing

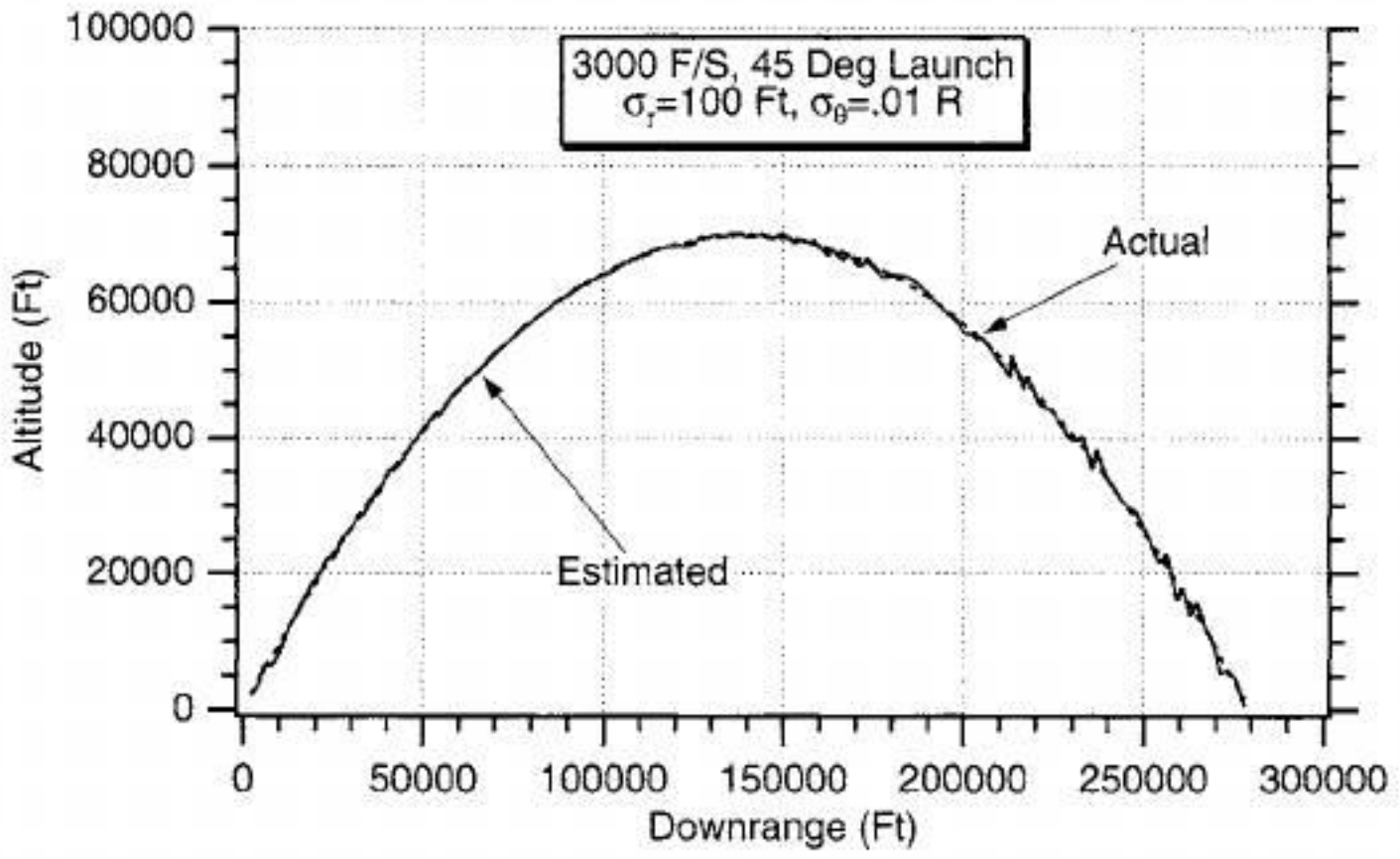

Fig. 9.2 Using raw measurements to estimate trajectory appears to be satisfactory.

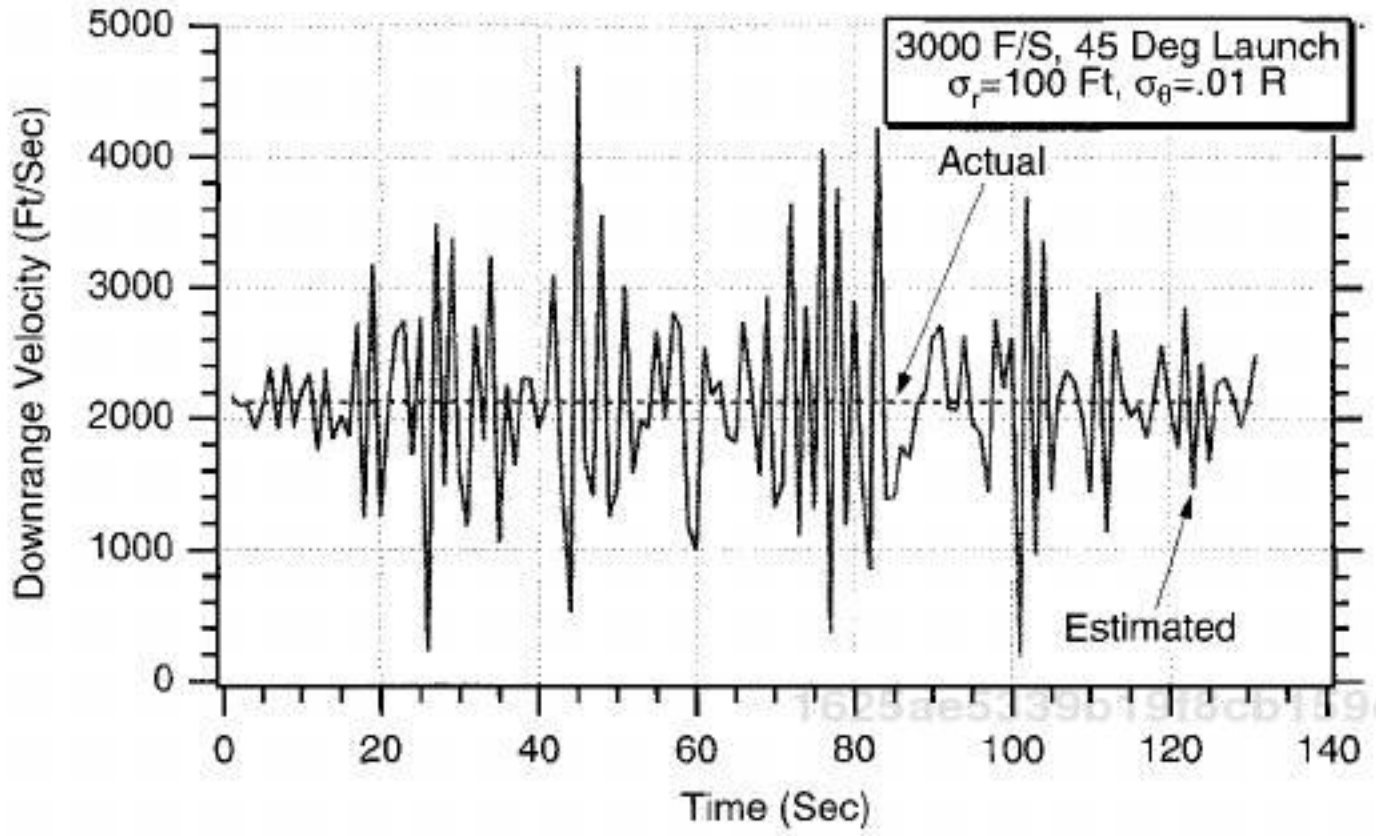

Fig. 9.3 Using raw measurements yields terrible downrange velocity estimates.

the initial errors. This is to be expected because we are not really doing any filtering but are simply using the raw measurements to obtain the location of the projectile.

Extended Cartesian Kalman Filter

If we desire to keep the model describing the real world linear in the cannon-launched projectile tracking problem, we can choose as states projectile location and velocity in the downrange or x direction and projectile location and velocity in the altitude or y direction. Thus, the proposed states are given by

$$\boldsymbol{x} = \begin{bmatrix} x_T \\ \dot{x}_T \\ y_T \\ \dot{y}_T \end{bmatrix}$$

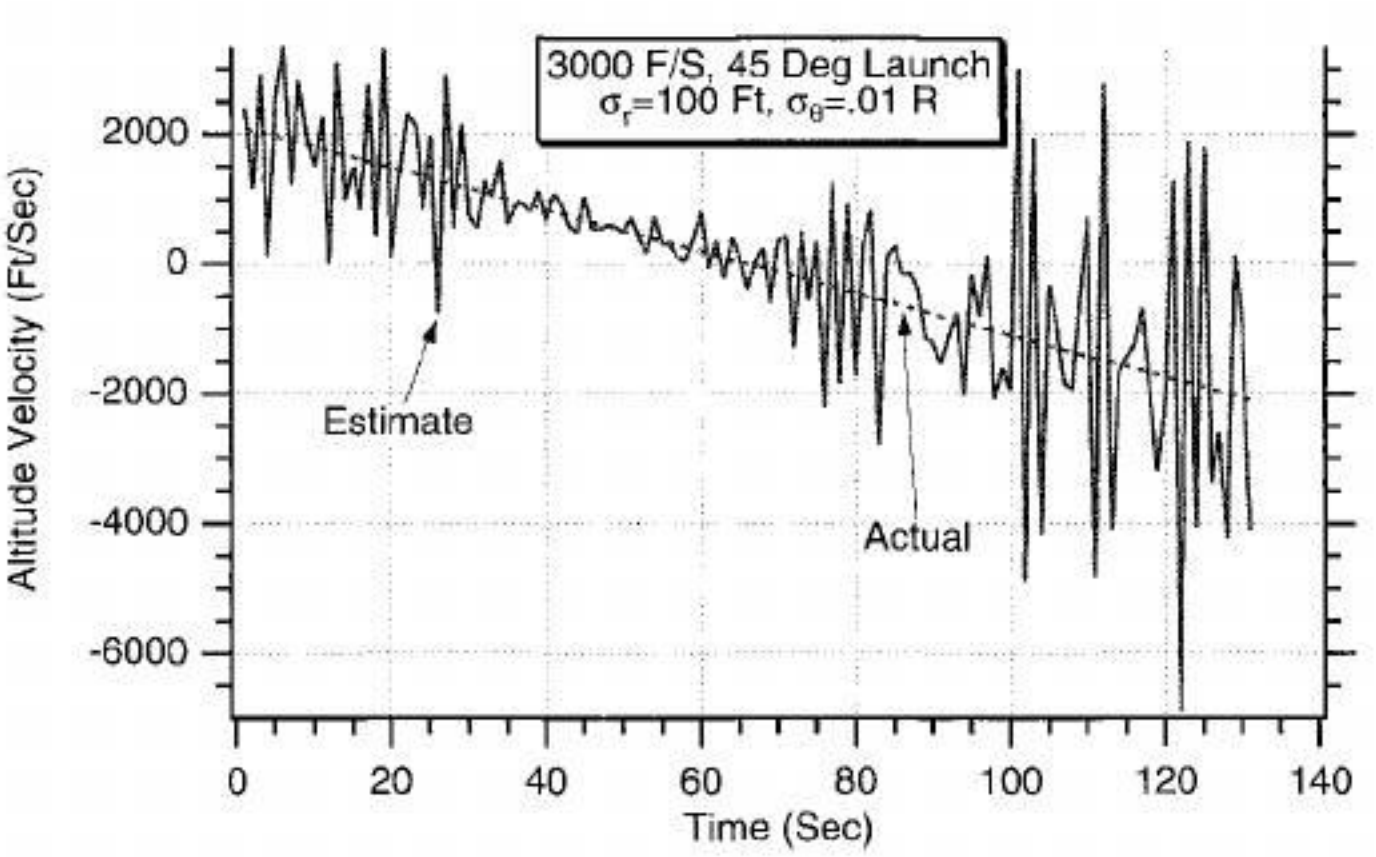

Fig. 9.4 Using raw measurements also yields very poor altitude velocity estimates.

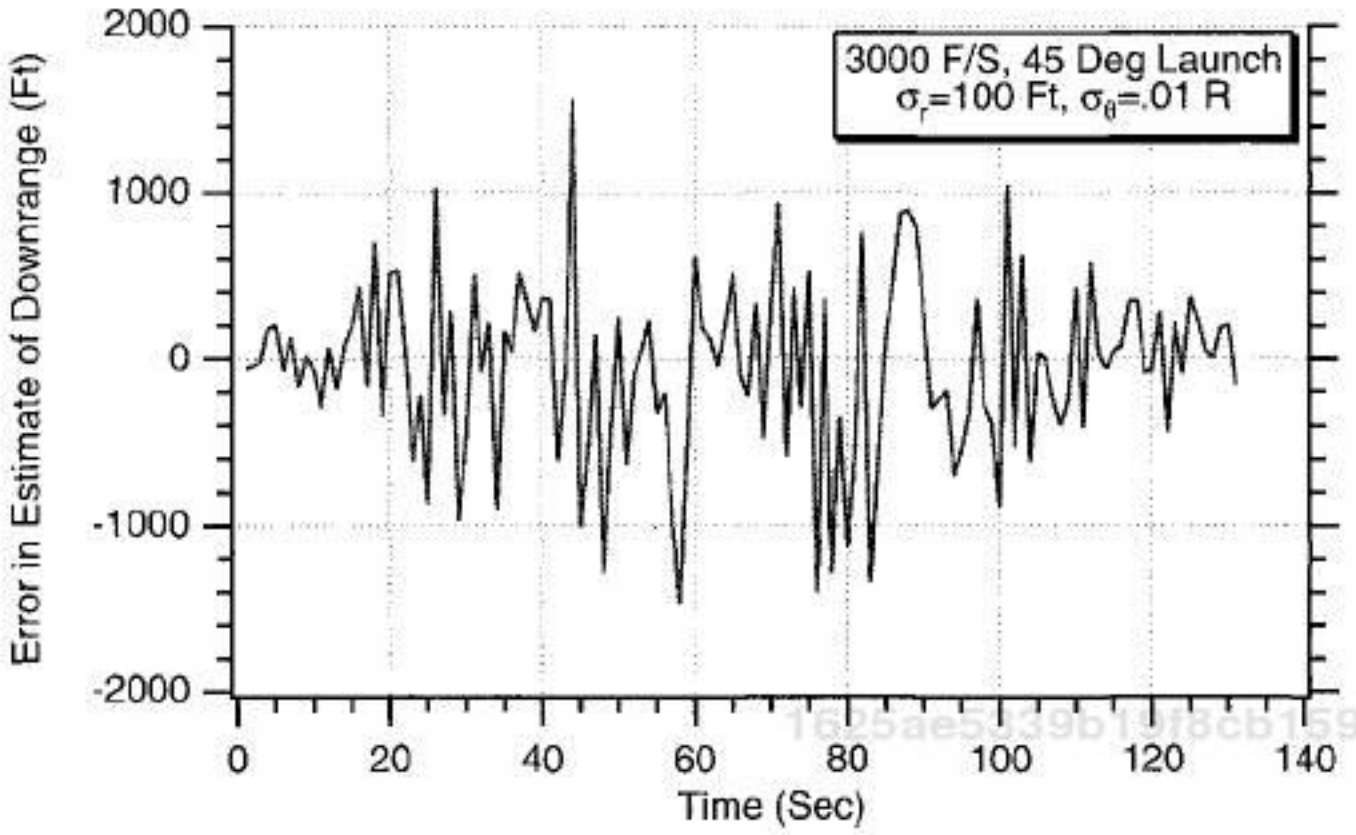

Fig. 9.5 Error in estimate of downrange is often greater than 1000 ft.

Therefore, when the preceding Cartesian states are chosen the state-space differential equation describing projectile motion becomes

$$\begin{bmatrix} \dot{x}_T \\ \ddot{x}_T \\ \dot{y}_T \\ \ddot{y}_T \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_T \\ \dot{x}_T \\ y_T \\ \dot{y}_T \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ -g \end{bmatrix} + \begin{bmatrix} \begin{bmatrix} 0 \\ u_s \\ 0 \\ u_s \end{bmatrix} \end{bmatrix}$$

Notice that in the preceding equation gravity g is not a state that has to be estimated but is assumed to be known in advance. We have also added process noise u_s to the acceleration portion of the equations as protection for effects that may not be considered by the Kalman filter. However, in our initial experiments

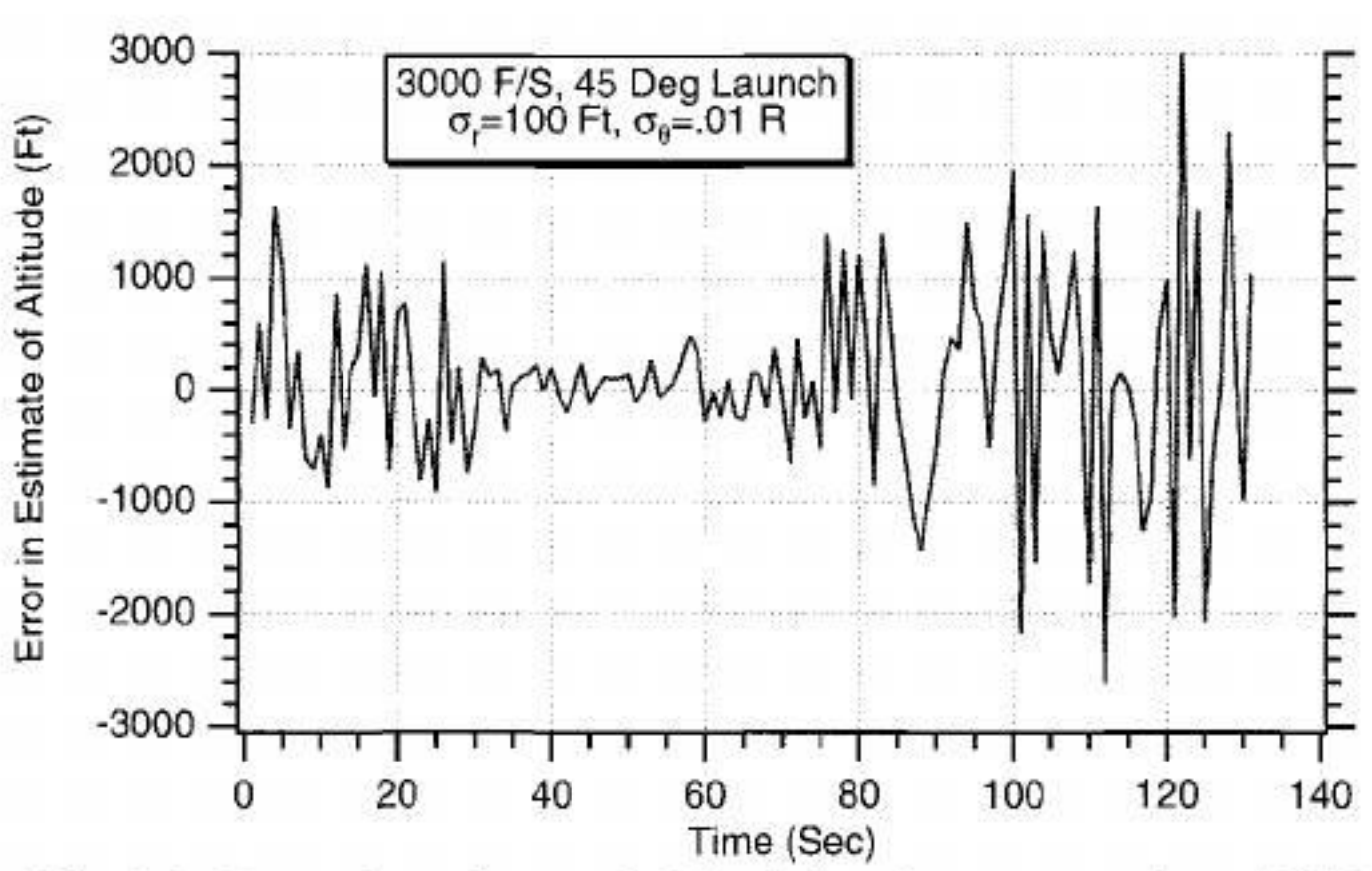

Fig. 9.6 Error in estimate of altitude is often greater than 1000 ft.

we will assume the process noise to be zero because our state-space equations upon which the Kalman filter is based will be a perfect match to the real world.

From the preceding state-space equation we can see that systems dynamics matrix is given by

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

Because the fundamental matrix for a time-invariant system is given by

$$\Phi(t) = \mathcal{L}^{-1}[(s\boldsymbol{I} - \boldsymbol{F})^{-1}]$$

we could find the fundamental matrix from the preceding expression. However, the required inversion of a 4×4 matrix might prove very tedious. A faster way to compute the fundamental matrix in this example is to simply use the Taylor-series approximation, which is given by

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

Because

$$\boldsymbol{F}^2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

all of the higher-order terms of the Taylor-series expansion must be zero, and the fundamental matrix becomes

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

or, more simply,

$$\Phi(t) = \begin{bmatrix} 0 & t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & t \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

$$\Phi_k = \begin{bmatrix} 0 & T_s & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T_s \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Because our states have been chosen to be Cartesian, the radar measurements r and θ will automatically be nonlinear functions of those states. Therefore, we must write the linearized measurement equation as

$$\begin{bmatrix} \Delta\theta^* \\ \Delta r^* \end{bmatrix} = \begin{bmatrix} \dfrac{\partial\theta}{\partial x_T} & \dfrac{\partial\theta}{\partial \dot{x}_T} & \dfrac{\partial\theta}{\partial y_T} & \dfrac{\partial\theta}{\partial \dot{y}_T} \\ \dfrac{\partial r}{\partial x_T} & \dfrac{\partial r}{\partial \dot{x}_T} & \dfrac{\partial r}{\partial y_T} & \dfrac{\partial r}{\partial \dot{y}_T} \end{bmatrix} \begin{bmatrix} \Delta x_T \\ \Delta \dot{x}_T \\ \Delta y_T \\ \Delta \dot{y}_T \end{bmatrix} + \begin{bmatrix} v_\theta \\ v_r \end{bmatrix}$$

where v_θ and v_r represent the measurement noise on angle and range, respectively. Because the angle from the radar to the projectile is given by

$$\theta = \tan^{-1}\left(\frac{y_T - y_R}{x_T - x_R}\right)$$

the four partial derivatives of the angle with respect to each of the states can be computed as

$$\frac{\partial\theta}{\partial x_T} = \frac{1}{1 + [(y_T - y_R)^2/(x_T - x_R)^2]} \frac{(x_T - x_R)*0 - (y_T - y_R)*1}{(x_T - x_R)^2} = \frac{-(y_T - y_R)}{r^2}$$

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

$$\frac{\partial\theta}{\partial y_T} = \frac{1}{1 + [(y_T - y_R)^2/(x_T - x_R)^2]} \frac{(x_T - x_R)*1 - (y_T - y_R)*0}{(x_T - x_R)^2} = \frac{(x_T - x_R)}{r^2}$$

$$\frac{\partial\theta}{\partial \dot{y}_T} = 0$$

Similarly, because the range from the radar to the projectile is given by

$$r = \sqrt{(x_T - x_R)^2 + (y_T - y_R)^2}$$

the four partial derivatives of the range with respect to each of the states can be computed as

$$\frac{\partial r}{\partial x_T} = \frac{1}{2}[(x_T - x_R)^2 + (y_T - y_R)^2]^{-1/2} 2(x_T - x_R) = \frac{(x_T - x_R)}{r}$$

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

$$\frac{\partial r}{\partial y_T} = \frac{1}{2}[(x_T - x_R)^2 + (y_T - y_R)^2]^{-1/2} 2(y_T - y_R) = \frac{(y_T - y_R)}{r}$$

$$\frac{\partial r}{\partial \dot{y}_T} = 0$$

Because the linearized measurement matrix is given by

$$\boldsymbol{H} = \begin{bmatrix} \frac{\partial \theta}{\partial x_T} & \frac{\partial \theta}{\partial \dot{x}_T} & \frac{\partial \theta}{\partial y_T} & \frac{\partial \theta}{\partial \dot{y}_T} \\ \frac{\partial r}{\partial x_T} & \frac{\partial r}{\partial \dot{x}_T} & \frac{\partial r}{\partial y_T} & \frac{\partial r}{\partial \dot{y}_T} \end{bmatrix}$$

we can use substitution of the already derived partial derivatives to yield the linearized measurement matrix for this example to be

$$\boldsymbol{H} = \begin{bmatrix} \frac{-(y_T - y_R)}{r^2} & 0 & \frac{x_T - x_R}{r^2} & 0 \\ \frac{x_T - x_R}{r} & 0 & \frac{y_T - y_R}{r} & 0 \end{bmatrix}$$

For this problem it is assumed that we know where the radar is so that x_R and y_R are known and do not have to be estimated. The states required for the discrete linearized measurement matrix will be based on the projected state estimate or

$$\boldsymbol{H}_k = \begin{bmatrix} \frac{-(\bar{y}_{T_k} - y_R)}{\bar{r}_k^2} & 0 & \frac{\bar{x}_{T_k} - x_R}{\bar{r}^2} & 0 \\ \frac{\bar{x}_{T_k} - x_R}{\bar{r}_k} & 0 & \frac{\bar{y}_{T_k} - y_R}{\bar{r}_k} & 0 \end{bmatrix}$$

Because the discrete measurement noise matrix is given by

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

we can say that for this problem the discrete measurement noise matrix is

$$\boldsymbol{R}_k = \begin{bmatrix} \sigma_\theta^2 & 0 \\ 0 & \sigma_r^2 \end{bmatrix}$$

where σ_θ^2 and σ_r^2 are the variances of the angle noise and range noise measurements, respectively. Similarly because the continuous process-noise matrix is given by

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

we can easily show from the linear state-space equation for this example that the continuous process-noise matrix is

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

where Φ_s is the spectral density of the white noise sources assumed to be on the downrange and altitude accelerations acting on the projectile. 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 expression yields

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

After some algebra we get

$$\boldsymbol{Q}_k = \int_0^{T_s} \begin{bmatrix} \tau^2\Phi_s & \tau\Phi_s & 0 & 0 \\ \tau\Phi_s & \Phi_s & 0 & 0 \\ 0 & 0 & \tau^2\Phi_s & \tau\Phi_s \\ 0 & 0 & \tau\Phi_s & \Phi_s \end{bmatrix} \mathrm{d}\tau$$

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

$$\boldsymbol{Q}_k = \begin{bmatrix} \frac{T_s^3\Phi_s}{3} & \frac{T_s^2\Phi_s}{2} & 0 & 0 \\ \frac{T_s^2\Phi_s}{2} & T_s\Phi_s & 0 & 0 \\ 0 & 0 & \frac{T_s^3\Phi_s}{2} & \frac{T_s^2\Phi_s}{2} \\ 0 & 0 & \frac{T_s^2\Phi_s}{2} & T_s\Phi_s \end{bmatrix}$$

We now have defined all of the matrices required to solve the 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 can be done in this example precisely with the fundamental matrix because the fundamental matrix is exact. In other nonlinear examples, where the fundamental matrix was approximate, we have already shown that the numerical integration of the nonlinear differential equations until the next update (i.e., using Euler integration) was required to eliminate potential filtering problems (i.e., hangoff error or even filter divergence).

Recall that for the linear filtering problem the real world was represented by the state-space equation

$$\dot{\boldsymbol{x}} = \boldsymbol{F}\boldsymbol{x} + \boldsymbol{G}\boldsymbol{u} + \boldsymbol{w}$$

where $\boldsymbol{G}$ is a matrix multiplying a known disturbance or control vector $\boldsymbol{u}$ that does not have to be estimated. We can show that the discrete linear Kalman-filtering equation is given by

$$\hat{\boldsymbol{x}}_k = \Phi_k\hat{\boldsymbol{x}}_{k-1} + \boldsymbol{G}_k\boldsymbol{u}_{k-1} + \boldsymbol{K}_k(\boldsymbol{z}_k - \boldsymbol{H}\Phi_k\hat{x}_{k-1} - \boldsymbol{H}\boldsymbol{G}_k\boldsymbol{u}_{k-1})$$

where $\boldsymbol{G}_k$ is obtained from

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

If we forget about the gain times the residual portion of the filtering equation, we can see that the projected state is simply

$$\bar{\boldsymbol{x}}_k = \Phi_k\hat{\boldsymbol{x}}_{k-1} + \boldsymbol{G}_k\boldsymbol{u}_{k-1}$$

For this problem

$$\boldsymbol{G} = \boldsymbol{G}\boldsymbol{u} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -g \end{bmatrix}$$

Therefore, the $\boldsymbol{G}_k$ becomes

$$\boldsymbol{G}_k = \int_0^{T_s} \begin{bmatrix} 1 & \tau & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \tau \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ -g \end{bmatrix} \mathrm{d}\tau = \begin{bmatrix} 0 \\ 0 \\ -\dfrac{gT_s^2}{2} \\ -gT_s \end{bmatrix}$$

and our projected state is determined from

$$\bar{\boldsymbol{x}}_k = \begin{bmatrix} 1 & T_s & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T_s \\ 0 & 0 & 0 & 1 \end{bmatrix} \hat{\boldsymbol{x}}_{k-1} + \begin{bmatrix} 0 \\ 0 \\ -\dfrac{gT_s^2}{2} \\ -gT_s \end{bmatrix}$$

When we convert the preceding matrix equation for the projected states to four scalar equations, we obtain

$$\bar{x}_{T_k} = \hat{x}_{T_{k-1}} + T_s \hat{\dot{x}}_{T_{k-1}}$$

$$\overline{\dot{x}_{T_k}} = \hat{\dot{x}}_{T_{k-1}}$$

$$\bar{y}_{T_k} = \hat{y}_{T_{k-1}} + T_s \hat{\dot{y}}_{T_{k-1}} - 0.5gT_s^2$$

$$\overline{\dot{y}_{T_k}} = \hat{\dot{y}}_{T_{k-1}} - gT_s$$

The next portion of the Kalman filter uses gains times residuals. Because the measurements are nonlinear, the residuals are simply the measurements minus the projected values of the measurements (i.e., we do not want to use the linearized measurement matrix). Therefore, the projected value of the angle and range from the radar to the projectile must be based upon the projected state estimates or

$$\bar{\theta}_k = \tan^{-1}\left(\frac{\bar{y}_{T_{k-1}} - y_R}{\bar{x}_{T_{k-1}} - x_R}\right)$$

$$\bar{r}_k = \sqrt{(\bar{x}_{T_{k-1}} - x_R)^2 + (\bar{y}_{T_{k-1}} - y_R)^2}$$

Now the extended Kalman-filtering equations can be written simply as

$$\hat{x}_{T_k} = \bar{x}_{T_k} + K_{11_k}(\theta_k^* - \bar{\theta}_k) + K_{12_k}(r_k^* - \bar{r}_k)$$

$$\hat{\dot{x}}_{T_k} = \bar{\dot{x}}_{T_k} + K_{21_k}(\theta_k^* - \bar{\theta}_k) + K_{22_k}(r_k^* - \bar{r}_k)$$

$$\hat{y}_{T_k} = \bar{y}_{T_k} + K_{31_k}(\theta_k^* - \bar{\theta}_k) + K_{32_k}(r_k^* - \bar{r}_k)$$

$$\hat{\dot{y}}_{T_k} = \bar{\dot{y}}_{T_k} + K_{41_k}(\theta_k^* - \bar{\theta}_k) + K_{42_k}(r_k^* - \bar{r}_k)$$

where θ_k^* and r_k^* are the noisy measurements of radar angle and range. Again, notice that we are using the actual nonlinear measurement equations in the extended Kalman filter.

The preceding equations for the Kalman filter and Riccati equations were programmed and are shown in Listing 9.1, 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., $\Phi_s = 0$). In addition, we have initialized the states of the filter close to the true values. The filter's position states are in error by 1000 ft, and the velocity states are in error by 100 ft/s. The initial covariance matrix reflects those errors. Also, because we have two independent measurements, the Riccati equation requires the inverse of a 2×2 matrix. This inverse is done exactly using the matrix inverse formula for a 2×2 matrix from Chapter 1.

The extended Kalman filter of Listing 9.1 was run for the nominal conditions described. Figure 9.7 compares the error in the estimate of the projectile's

Listing 9.1 Cartesian extended Kalman-filter simulation for projectile tracking problem

```
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,O-Z)
      REAL*8  P(4,4),Q(4,4),M(4,4),PHI(4,4),HMAT(2,4),HT(4,2),PHIT(4,4)
      REAL*8  RMAT(2,2),IDN(4,4),PHIP(4,4),PHIPPHIT(4,4),HM(2,4)
      REAL*8  HMHT(2,2),HMHTR(2,2),HMHTRINV(2,2),MHT(4,2),K(4,2)
      REAL*8  KH(4,4),IKH(4,4)
      INTEGER  ORDER
      TS=1.
      ORDER=4
      PHIS=0.
      SIGTH=.01
      SIGR=100.
      VT=3000.
      GAMDEG=45.
      G=32.2
      XT=0.
      YT=0.
```

(continued)

Listing 9.1 (*Continued*)

```
      XTD=VT*COS(GAMDEG/57.3)
      YTD=VT*SIN(GAMDEG/57.3)
      XR=100000.
      YR=0.
      OPEN(1,STATUS='UNKNOWN',FILE='DATFIL')
      OPEN(2,STATUS='UNKNOWN',FILE='COVFIL')
      T=0.
      S=0.
      H=.001
      DO  14  I=1,ORDER
      DO  14  J=1,ORDER
      PHI(I,J)=0.
      P(I,J)=0.
      Q(I,J)=0.
      IDN(I,J)=0.
14    CONTINUE
      TS2=TS*TS
      TS3=TS2*TS
      Q(1,1)=PHIS*TS3/3.
      Q(1,2)=PHIS*TS2/2.
      Q(2,1)=Q(1,2)
      Q(2,2)=PHIS*TS
      Q(3,3)=Q(1,1)
      Q(3,4)=Q(1,2)
      Q(4,3)=Q(3,4)
      Q(4,4)=Q(2,2)
      PHI(1,1)=1.
      PHI(1,2)=TS
      PHI(2,2)=1.
      PHI(3,3)=1.
      PHI(3,4)=TS
      PHI(4,4)=1.
      CALL  MATTRN(PHI,ORDER,ORDER,PHIT)
      RMAT(1,1)=SIGTH**2
      RMAT(1,2)=0.
      RMAT(2,1)=0.
      RMAT(2,2)=SIGR**2
      IDN(1,1)=1.
      IDN(2,2)=1.
      IDN(3,3)=1.
      IDN(4,4)=1.
      P(1,1)=1000.**2
      P(2,2)=100.**2
      P(3,3)=1000.**2
      P(4,4)=100.**2
      XTH=XT+1000.
      XTDH=XTD-100.
      YTH=YT-1000.
      YTDH=YTD+100.
```

(*continued*)

Listing 9.1 (*Continued*)

```
      WHILE(YT>=0.)
            XTOLD=XT
            XTDOLD=XTD
            YTOLD=YT
            YTDOLD=YTD
            XTDD=0.
            YTDD=-G
            XT=XT+H*XTD
            XTD=XTD+H*XTDD
            YT=YT+H*YTD
            YTD=YTD+H*YTDD
            T=T+H
            XTDD=0.
            YTDD=-G
            XT=.5*(XTOLD+XT+H*XTD)
            XTD=.5*(XTDOLD+XTD+H*XTDD)
            YT=.5*(YTOLD+YT+H*YTD)
            YTD=.5*(YTDOLD+YTD+H*YTDD)
            S=S+H
            IF(S>=(TS-.00001))THEN
                  S=0.
                  XTB=XTH+TS*XTDH
                  XTDB=XTDH
                  YTB=YTH+TS*YTDH-.5*G*TS*TS
                  YTDB=YTDH-G*TS
                  RTB=SQRT((XTB-XR)**2+(YTB-YR)**2)
                  HMAT(1,1)=-(YTB-YR)/RTB**2
                  HMAT(1,2)=0.
                  HMAT(1,3)=(XTB-XR)/RTB**2
                  HMAT(1,4)=0.
                  HMAT(2,1)=(XTB-XR)/RTB
                  HMAT(2,2)=0.
                  HMAT(2,3)=(YTB-YR)/RTB
                  HMAT(2,4)=0.
                  CALL MATTRN(HMAT,2,ORDER,HT)
                  CALL MATMUL(PHI,ORDER,ORDER,P,ORDER,
                    ORDER,PHIP)
                  CALL MATMUL(PHIP,ORDER,ORDER,PHIT,ORDER,
                    ORDER,PHIPPHIT)
                  CALL MATADD(PHIPPHIT,ORDER,ORDER,Q,M)
                  CALL MATMUL(HMAT,2,ORDER,M,ORDER,
                    ORDER,HM)
                  CALL MATMUL(HM,2,ORDER,HT,ORDER,2,HMHT)
                  CALL MATADD(HMHT,ORDER,ORDER,RMAT,
                    HMHTR)
                  DET=HMHTR(1,1)*HMHTR(2,2)-HMHTR(1,2)*
                    HMHTR(2,1)
                  HMHTRINV(1,1)=HMHTR(2,2)/DET
                  HMHTRINV(1,2)=-HMHTR(1,2)/DET
```

(*continued*)

Listing 9.1 (*Continued*)

```
      HMHTRINV(2,1)=-HMHTR(2,1)/DET
      HMHTRINV(2,2)=HMHTR(1,1)/DET
      CALL  MATMUL(M,ORDER,ORDER,HT,ORDER,2,MHT)
      CALL  MATMUL(MHT,ORDER,2,HMHTRINV,2,2,K)
      CALL  MATMUL(K,ORDER,2,HMAT,2,ORDER,KH)
      CALL  MATSUB(IDN,ORDER,ORDER,KH,IKH)
      CALL  MATMUL(IKH,ORDER,ORDER,M,ORDER,
     ORDER,P)
      CALL  GAUSS(THETNOISE,SIGTH)
      CALL  GAUSS(RTNOISE,SIGR)
      THET=ATAN2((YT-YR),(XT-XR))
      RT=SQRT((XT-XR)**2+(YT-YR)**2)
      THETMEAS=THET+THETNOISE
      RTMEAS=RT+RTNOISE
      THETB=ATAN2((YTB-YR),(XTB-XR))
      RTB=SQRT((XTB-XR)**2+(YTB-YR)**2)
      RES1=THETMEAS-THETB
      RES2=RTMEAS-RTB
      XTH=XTB+K(1,1)*RES1+K(1,2)*RES2
      XTDH=XTDB+K(2,1)*RES1+K(2,2)*RES2
      YTH=YTB+K(3,1)*RES1+K(3,2)*RES2
      YTDH=YTDB+K(4,1)*RES1+K(4,2)*RES2
      ERRX=XT-XTH
      SP11=SQRT(P(1,1))
      ERRXD=XTD-XTDH
      SP22=SQRT(P(2,2))
      ERRY=YT-YTH
      SP33=SQRT(P(3,3))
      ERRYD=YTD-YTDH
      SP44=SQRT(P(4,4))
      WRITE(9,*)T,XT,YT,THET*57.3,RT
      WRITE(1,*)T,XT,XTH,XTD,XTDH,YT,YTH,YTD,YTDH
      WRITE(2,*)T,ERRX,SP11,-SP11,ERRXD,SP22,-SP22,
     1   ERRY,SP33,-SP33,ERRYD,SP44,-SP44
        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
```

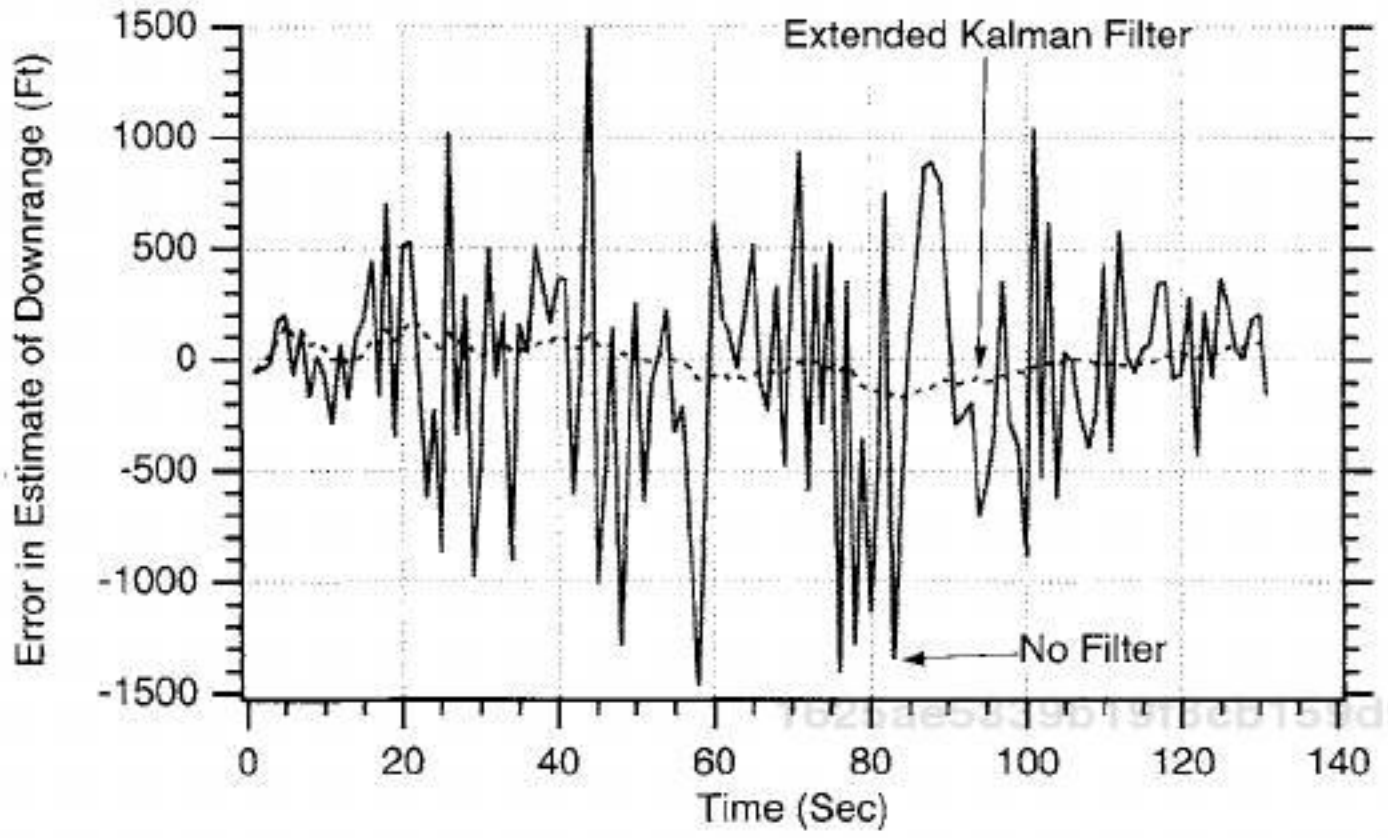

Fig. 9.7 Filtering dramatically reduces position error over using raw measurements.

downrange position for a single-run in which the extended Kalman filter is used in the preceding example in which there was no filtering at all (i.e., projectile's position reconstructed directly from range and angle measurements). The solid curve represents the case in which no filtering is used, whereas the dashed curve represents the error in the estimate when the extended Kalman filter is used. We can see that filtering dramatically reduces the error in the estimate of the projectile's position from more than 1000 ft to less than 100 ft.

To get a better intuitive feel for how the extended Kalman filter is performing, it is best to compare errors in the estimate of the projectile's position and velocity to the theoretical predictions of the covariance matrix. Figures 9.8 and 9.9 show how the single simulation run errors in the estimates of downrange position (i.e.,

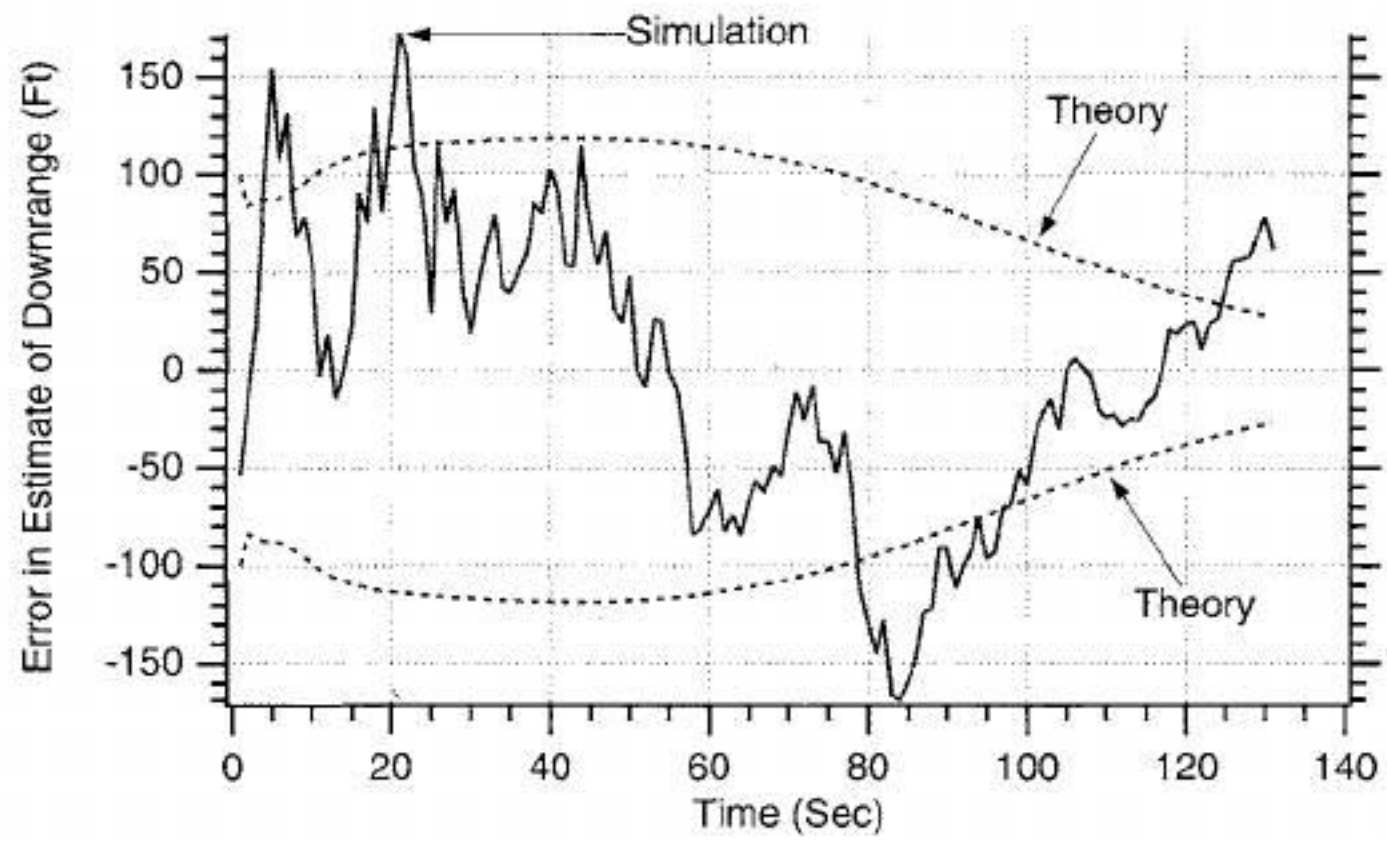

Fig. 9.8 Extended Cartesian Kalman filter's projectile's downrange estimates appear to agree with theory.

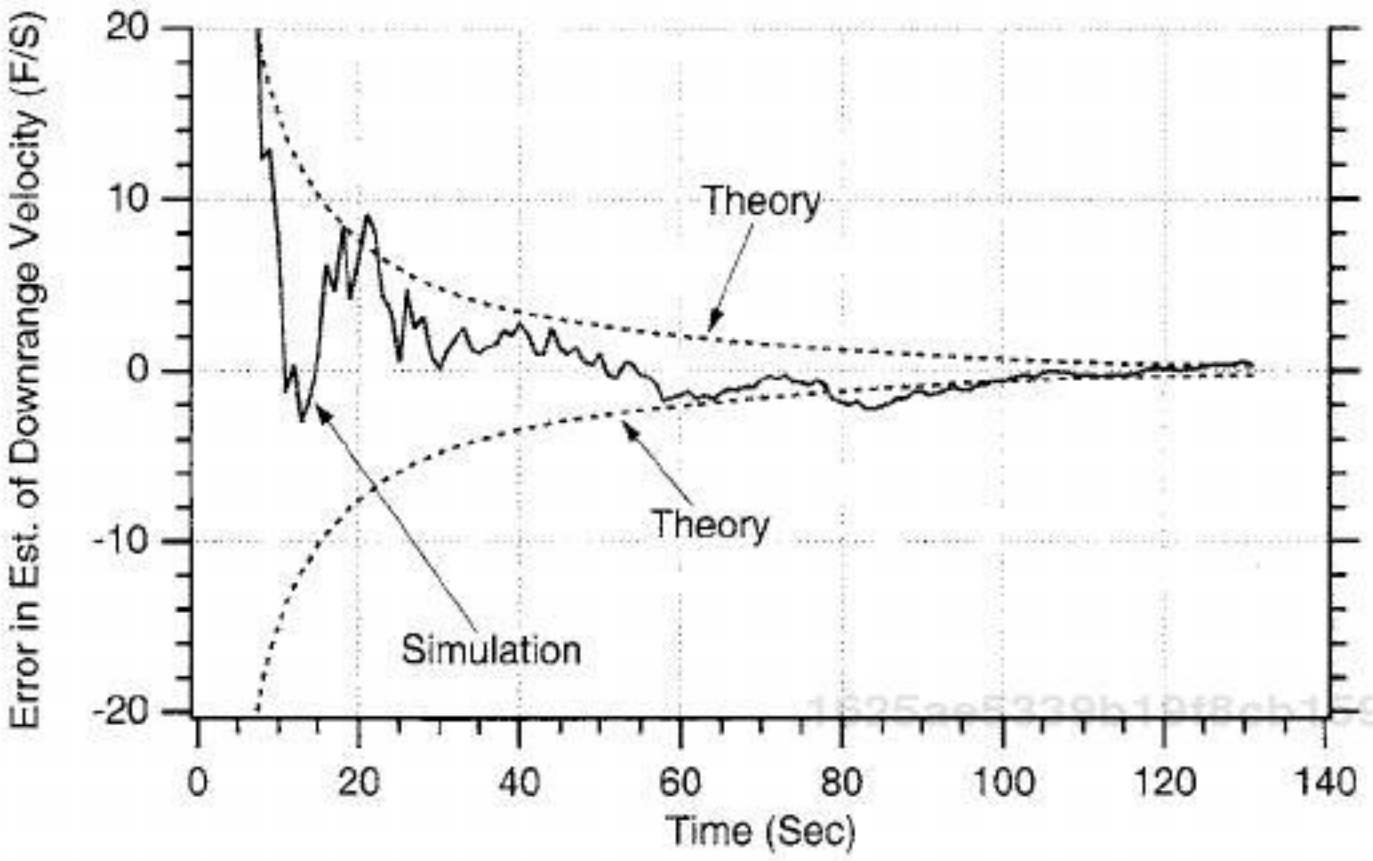

Fig. 9.9 Extended Cartesian Kalman filter's projectile's downrange velocity estimates appear to agree with theory.

x_T) and velocity (i.e., $\dot{x}_T$) compare with the theoretical predictions of the Riccati equation covariance matrix (i.e., square root of $P11$ and $P22$, respectively). We can see that the single flight simulation results lie within the theoretical bounds approximately 68% of the time, giving us a good feeling that the extended Cartesian Kalman filter is performing properly. Because there is no process noise in this example, the errors in the estimates continue to get smaller as more measurements are taken.

Similarly, Figs. 9.10 and 9.11 show how the single-run simulation errors in the estimates of the projectile's altitude (i.e., y_T) and altitude velocity (i.e., $\dot{y}_T$) compare with the theoretical predictions of the covariance matrix (i.e., square root of $P33$ and $P44$). Again, we can see that the single-run simulation results lie

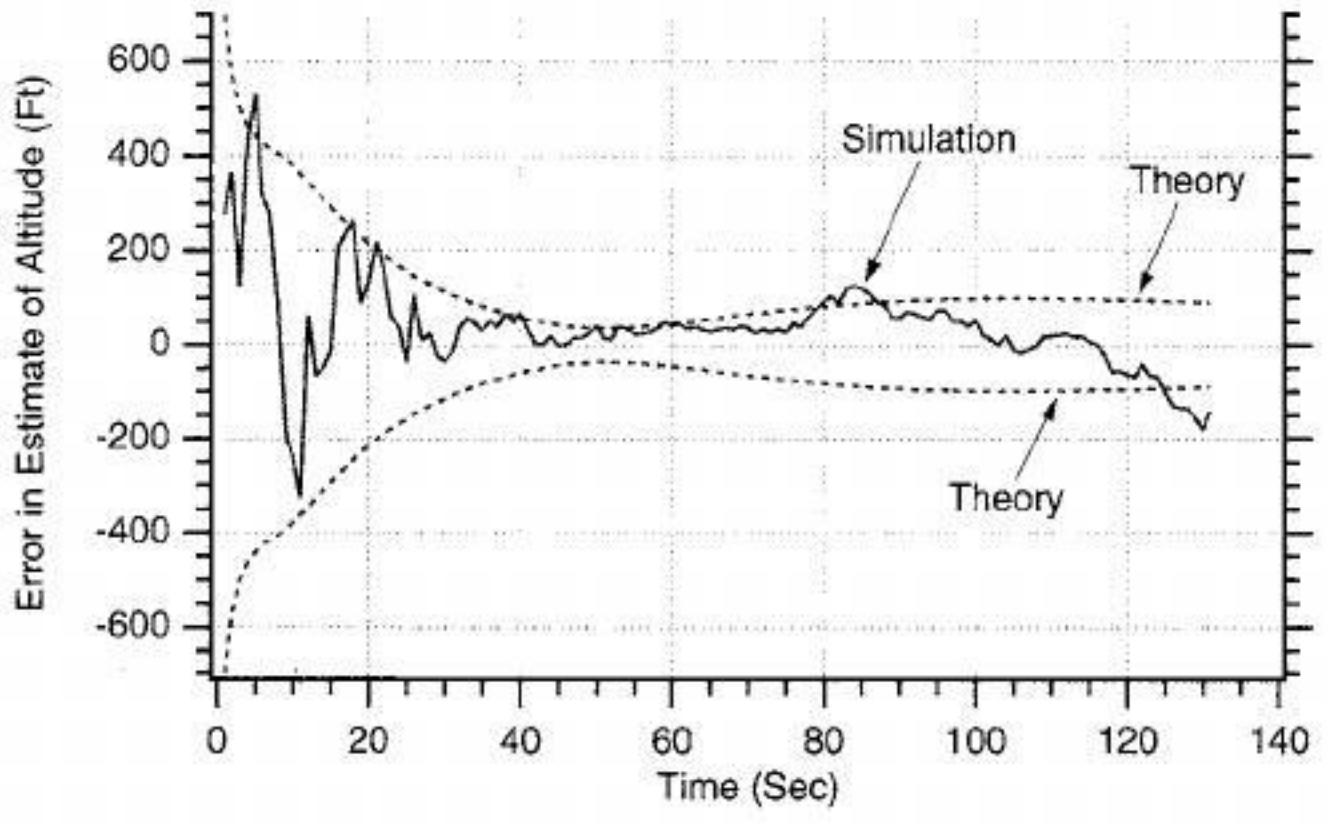

Fig. 9.10 Extended Cartesian Kalman filter's projectile's altitude estimates appear to agree with theory.

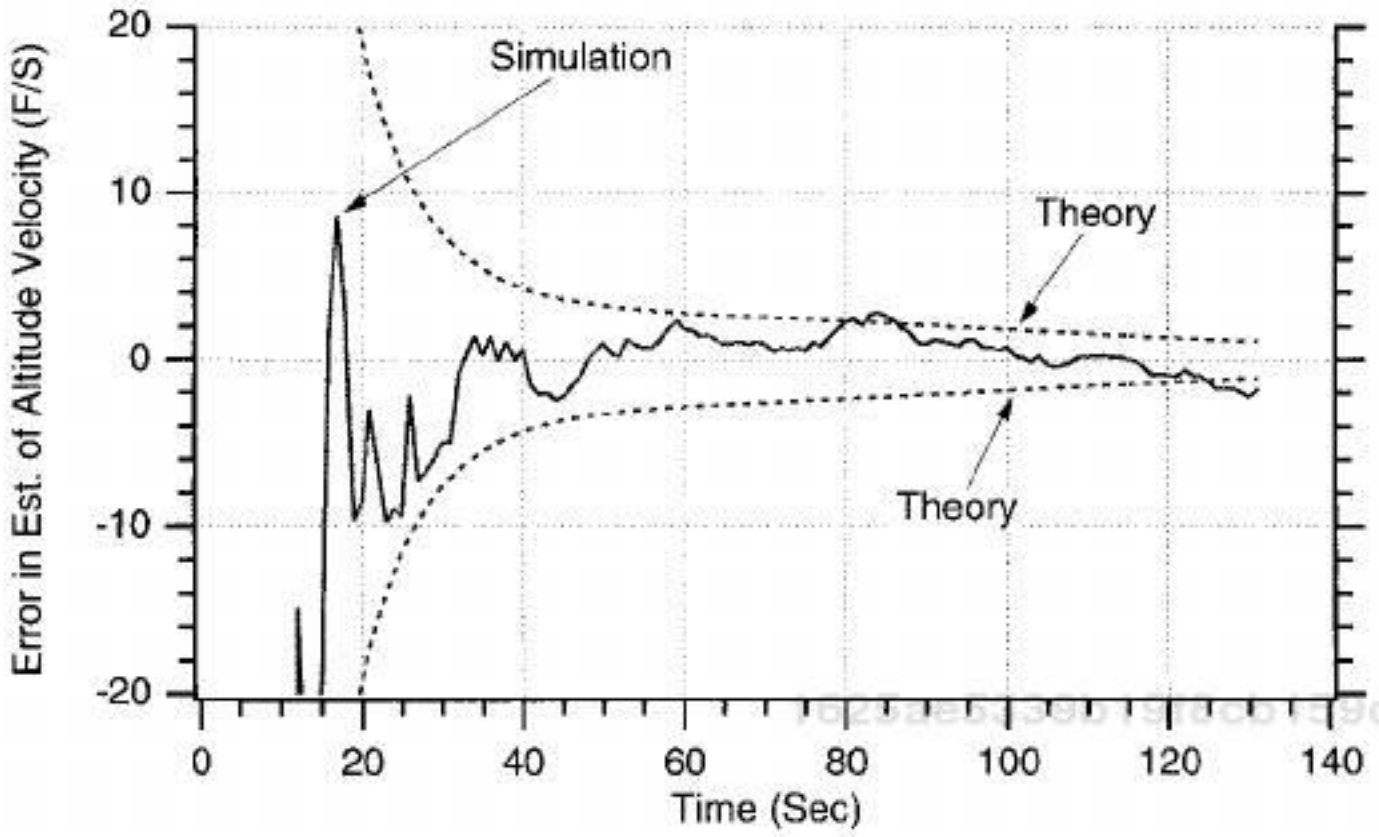

Fig. 9.11 Extended Cartesian Kalman filter's projectile's altitude velocity estimates appear to agree with theory.

within the theoretical bounds 68% of the time, giving us a good indication that the filter is performing properly.

In summary, we can say that for this example both the projectile's downrange and altitude position estimates appeared to be good to within 100 ft. The projectile's downrange and altitude velocity estimates appear to be good to within a few feet per second of the actual velocity. The results were for the case in which there was zero process noise. Adding process noise for practical reasons (i.e., prevent filter divergence when projectile does something that is not anticipated) will make the errors in the estimates larger. Recall that the process-noise matrix for the Cartesian extended Kalman filter of this example is given by

$$\boldsymbol{Q}_k = \begin{bmatrix} \frac{T_s^3\Phi_s}{3} & \frac{T_s^2\Phi_s}{2} & 0 & 0 \\ \frac{T_s^2\Phi_s}{2} & T_s\Phi_s & 0 & 0 \\ 0 & 0 & \frac{T_s^3\Phi_s}{3} & \frac{T_s^2\Phi_s}{2} \\ 0 & 0 & \frac{T_s^2\Phi_s}{2} & T_s\Phi_s \end{bmatrix}$$

Therefore, we can see that the amount of process noise is determined by Φ_s. Several runs were made in which the amount of process noise was varied, and the square root of the second diagonal element of the covariance matrix was saved. This covariance matrix element represents the theoretical error in the estimate of the projectile's downrange velocity. We can see from Fig. 9.12 that when there is no process noise (i.e., $\Phi_s = 0$) the projectile's velocity errors get smaller as time goes on (i.e., more measurements are taken). However, with process noise the

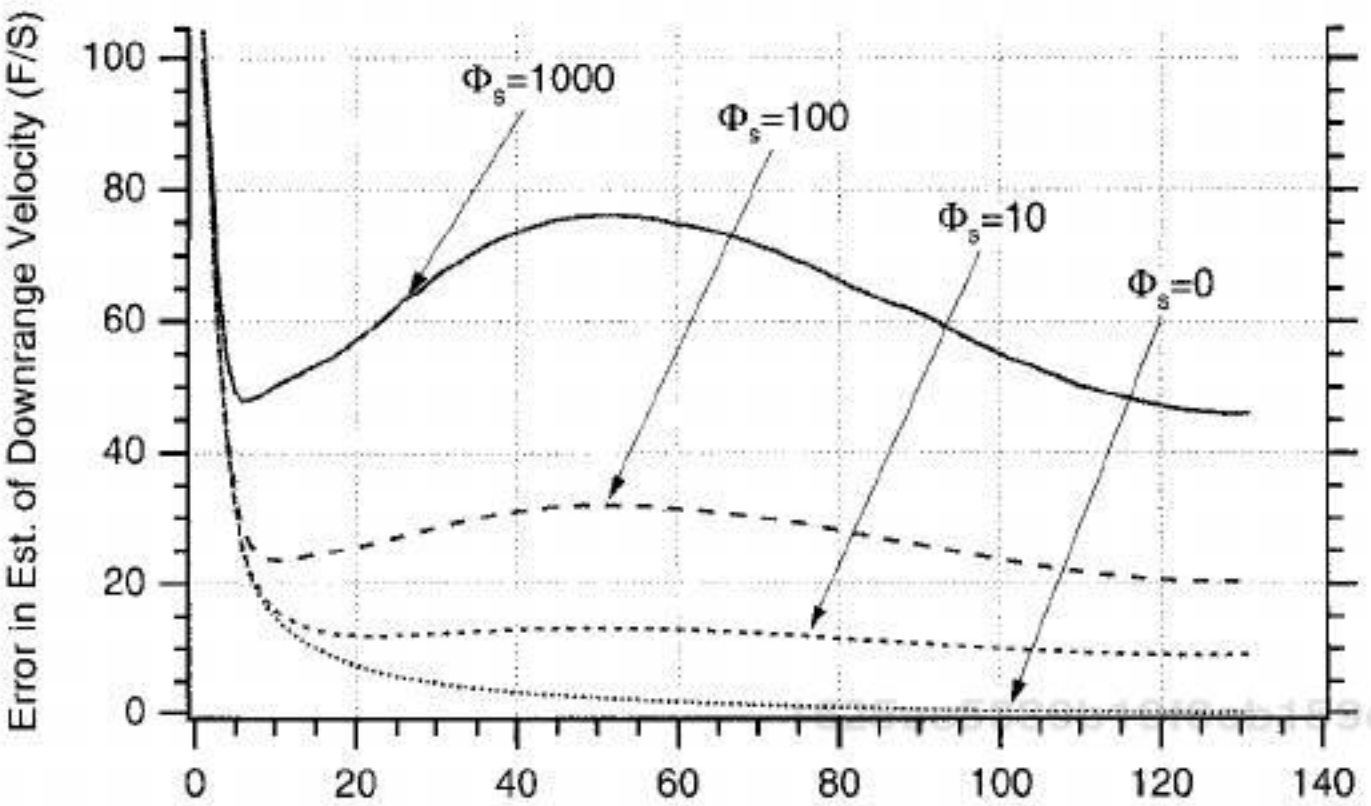

Fig. 9.12 Errors in estimate of velocity increase with increasing process noise.

error in the estimate of the projectile's downrange velocity approaches a steady state. In other words, errors in the estimate of the projectile's velocity do not decrease as more measurements are taken. We can see that the steady-state value of the error in the estimate of the projectile's velocity increases with increasing values of the process noise Φ_s.

Polar Coordinate System

In the preceding section we derived an extended Kalman filter, whose states were in a Cartesian coordinate system and whose measurements were in a polar coordinate system. With this methodology the model of the real world could easily be represented by linear differential equations, while the measurements were nonlinear functions of the states. In this section we will derive an extended Kalman filter based on a different set of states. These states will be for a polar coordinate system, which in turn will lead to measurements that are linear functions of the states. However, the resulting polar differential equations representing the real world will now be more complex and nonlinear. Eventually we want to see if an extended polar Kalman filter, based on this alternative representation, will perform better than the extended Cartesian Kalman filter. To accomplish this task, we must first derive and verify the polar equations for the cannon-launched projectile tracking problem.

Recall from Fig. 9.1 that the angle and range from the radar to the cannon ball is given by

$$\theta = \tan^{-1}\left(\frac{y_T - y_R}{x_T - x_R}\right)$$

$$r = \sqrt{(x_T - x_R)^2 + (y_T - y_R)^2}$$

Taking the first derivative of the angle yields

$$\dot{\theta} = \frac{1}{1 + (y_T - y_R)^2/(x_T - x_R)^2} \frac{(x_T - x_R)\dot{y}_T - (y_T - y_R)\dot{x}_T}{(x_T - x_R)^2}$$

After some simplification we can substitute the formula for range into the preceding expression to obtain

$$\dot{\theta} = \frac{(x_T - x_R)\dot{y}_T - (y_T - y_R)\dot{x}_T}{r^2}$$

Using calculus, we can also take the first derivative of range

$$\dot{r} = \frac{1}{2}[(x_T - x_R)^2 + (y_7 - y_R)^2]^{-1/2}[2(x_T - x_R)\dot{x}_T + 2(y_T - y_R)\dot{y}_T]$$

which simplifies to

$$\dot{r} = \frac{(x_T - x_R)\dot{x}_T + (y_T - y_R)\dot{y}_T}{r}$$

We can also find the angular acceleration by taking the derivative of the angular velocity or

$$\ddot{\theta} = \frac{r^2[\dot{x}_T\dot{y}_T + (x_T - x_R)\ddot{y}_t - \dot{x}_T\dot{y}_T - (y_T - y_R)\ddot{x}_T] - [(x_T - x_R)\dot{y}_T - (y_T - y_R)\dot{y}_T]2r\dot{r}}{r^4}$$

After some simplification we obtain

$$\ddot{\theta} = \frac{(x_T - x_R)\ddot{y}_T - (y_T - y_R)\ddot{x}_T - 2r\dot{r}\dot{\theta}}{r^2}$$

Recognizing that

$$\ddot{x}_T = 0$$

$$\ddot{y}_T = -g$$

$$\cos\theta = \frac{x_T - x_R}{r}$$

$$\sin\theta = \frac{y_T - y_R}{r}$$

the formula for angular acceleration simplifies even further to

$$\ddot{\theta} = \frac{-g\cos\theta - 2\dot{r}\dot{\theta}}{r}$$

Finally, the second derivative of range yields

$$\dot{r} = \frac{r[\dot{x}_T\dot{x}_T + (x_T - x_R)\ddot{x}_T + \dot{y}_T\dot{y}_T + (y_T - y_R)\ddot{y}_T] - [(x_T - x_R)\dot{x}_T + (y_T - y_R)\dot{y}_T]\dot{r}}{r^2}$$

After a great deal of algebraic manipulation and simplification, we obtain

$$\dot{r} = \frac{r^2\dot{\theta}^2 - gr\sin\theta}{r}$$

In summary, we are saying that the nonlinear polar differential equations representing the cannon-launched projectile are given by

$$\ddot{\theta} = \frac{-g\cos\theta - 2\dot{r}\dot{\theta}}{r}$$

$$\ddot{r} = \frac{r^2\dot{\theta}^2 - gr\sin\theta}{r}$$

The preceding two nonlinear differential equations are completely equivalent to the linear Cartesian differential equations

$$\ddot{x}_T = 0$$

$$\ddot{y}_T = -g$$

if they both have the same initial conditions. For example, if the initial conditions in the Cartesian equations are denoted

$$x_T(0), \quad y_T(0), \quad \dot{x}_T(0), \quad \dot{y}_T(0)$$

then the initial conditions on the polar equations must be

$$\theta(0) = \tan^{-1}\left[\frac{y_T(0) - y_R}{x_T(0) - x_R}\right]$$

$$r(0) = \sqrt{[x_T(0) - x_R]^2 + [y_T(0) - y_R]^2}$$

$$\dot{\theta}(0) = \frac{[x_T(0) - x_R]\dot{y}_T(0) - [y_T(0) - y_R]\dot{x}_T(0)}{r(0)^2}$$

$$\dot{r}(0) = \frac{[x_T(0) - x_R]\dot{x}_T(0) + [y_T(0) - y_R]\dot{y}_T(0)}{r(0)}$$

When we have integrated the polar equations, we must also then find a way to convert the resultant answers back to Cartesian coordinates in order to perform a

direct comparison with the Cartesian differential equations. The conversion can be easily accomplished by recognizing that

$$\hat{x}_T = r\cos\theta + x_R$$

$$\hat{y}_T = r\sin\theta + y_R$$

where the caret denotes the estimates based on the conversion. Taking the derivative of the preceding two equations yields the estimated velocity components, which are also based on the conversion or

$$\hat{\dot{x}}_T = \dot{r}\cos\theta - r\dot{\theta}\sin\theta$$

$$\hat{\dot{y}}_T = \dot{r}\sin\theta + r\dot{\theta}\cos\theta$$

The simulation that compares the flight of the cannon-launched projectile in both the polar and Cartesian coordinate systems appears in Listing 9.2. We can see that the simulation consists of both Cartesian and polar differential equations for the projectile. Both sets of differential equations have the same initial conditions and integration step size. The location and velocity of the projectile, based on the Cartesian and polar differential equations, are printed out every second.

The nominal case of Listing 9.2 was run, and we can see from Fig. 9.13 that the projectile locations for both the Cartesian and polar formulation of the

Listing 9.2 Simulation that compares polar and Cartesian differential equations of cannon-launched projectile

```
IMPLICIT REAL*8(A-H,O-Z)
VT=3000.
GAMDEG=45.
G=32.2
TS=1.
XT=0.
YT=0.
XTD=VT*COS(GAMDEG/57.3)
YTD=VT*SIN(GAMDEG/57.3)
XR=100000.
YR=0.
OPEN(1,STATUS='UNKNOWN',FILE='DATFIL')
T=0.
S=0.
H=.001
THET=ATAN2((YT-YR),(XT-XR))
RT=SQRT((XT-XR)**2+(YT-YR)**2)
THETD=((XT-XR)*YTD-(YT-YR)*XTD)/RT**2
RTD=((XT-XR)*XTD+(YT-YR)*YTD)/RT
WHILE(YT>=0.)
        XTOLD=XT
        XTDOLD=XTD
        YTOLD=YT
        YTDOLD=YTD
        THETOLD=THET
```

(*continued*)

Listing 9.2 (*Continued*)

```
THETDOLD=THETD
RTOLD=RT
RTDOLD=RTD
XTDD=0.
YTDD=-G
THETDD=(-G*COS(THET)-2.*RTD*THETD)/RT
RTDD=((RT*THETD)**2-G*RT*SIN(THET))/RT
XT=XT+H*XTD
XTD=XTD+H*XTDD
YT=YT+H*YTD
YTD=YTD+H*YTDD
THET=THET+H*THETD
THETD=THETD+H*THETDD
RT=RT+H*RTD
RTD=RTD+H*RTDD
T=T+H
XTDD=0.
YTDD=-G
THETDD=(-G*COS(THET)-2.*RTD*THETD)/RT
RTDD=((RT*THETD)**2-G*RT*SIN(THET))/RT
XT=.5*(XTOLD+XT+H*XTD)
XTD=.5*(XTDOLD+XTD+H*XTDD)
YT=.5*(YTOLD+YT+H*YTD)
YTD=.5*(YTDOLD+YTD+H*YTDD)
THET=.5*(THETOLD+THET+H*THETD)
THETD=.5*(THETDOLD+THETD+H*THETDD)
RT=.5*(RTOLD+RT+H*RTD)
RTD=.5*(RTDOLD+RTD+H*RTDD)
S=S+H
IF(S>=(TS-.00001))THEN
        S=0.
        XTH=RT*COS(THET)+XR
        YTH=RT*SIN(THET)+YR
        XTDH=RTD*COS(THET)-RT*SIN(THET)*THETD
        YTDH=RTD*SIN(THET)+RT*COS(THET)*THETD
        WRITE(9,*)T,XT,XTH,YT,YTH,XTD,XTDH,YTD,YTDH
        WRITE(1,*)T,XT,XTH,YT,YTH,XTD,XTDH,YTD,YTDH
    ENDIF
END DO
PAUSE
CLOSE(1)
END
```

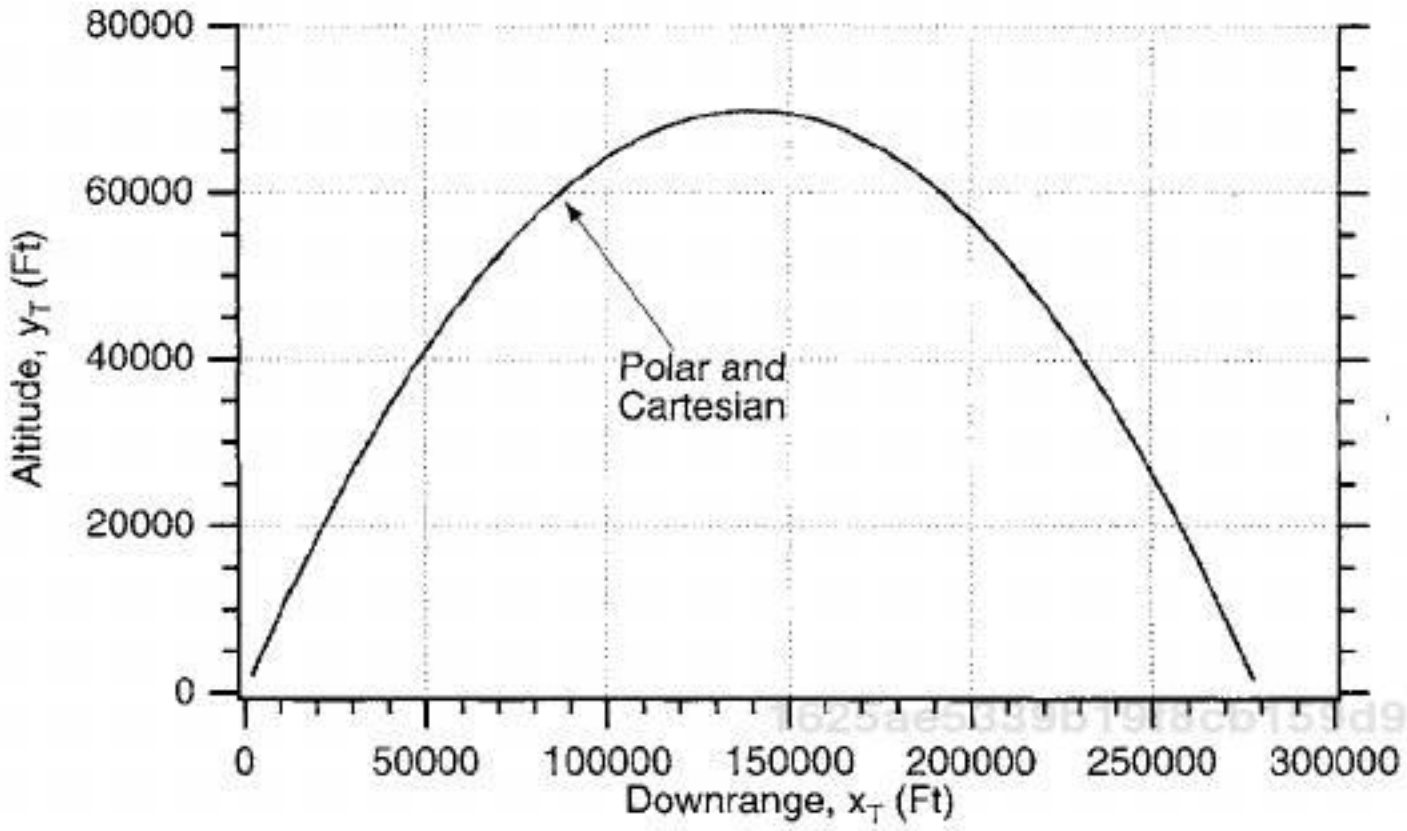

Fig. 9.13 Polar and Cartesian differential equations are identical in position.

differential equations are identical. In addition, Fig. 9.14 shows that the projectile component velocities are also identical in both coordinate systems, indicating the polar nonlinear differential equations have been derived correctly and are completely equivalent to the Cartesian linear differential equations.

Extended Polar Kalman Filter

We have already derived and validated the nonlinear polar differential equations for the cannon-launched projectile, which are given by

$$\ddot{\theta} = \frac{-g\cos\theta - 2\dot{r}\dot{\theta}}{r}$$

$$\ddot{r} = \frac{r^2\dot{\theta}^2 - gr\sin\theta}{r}$$

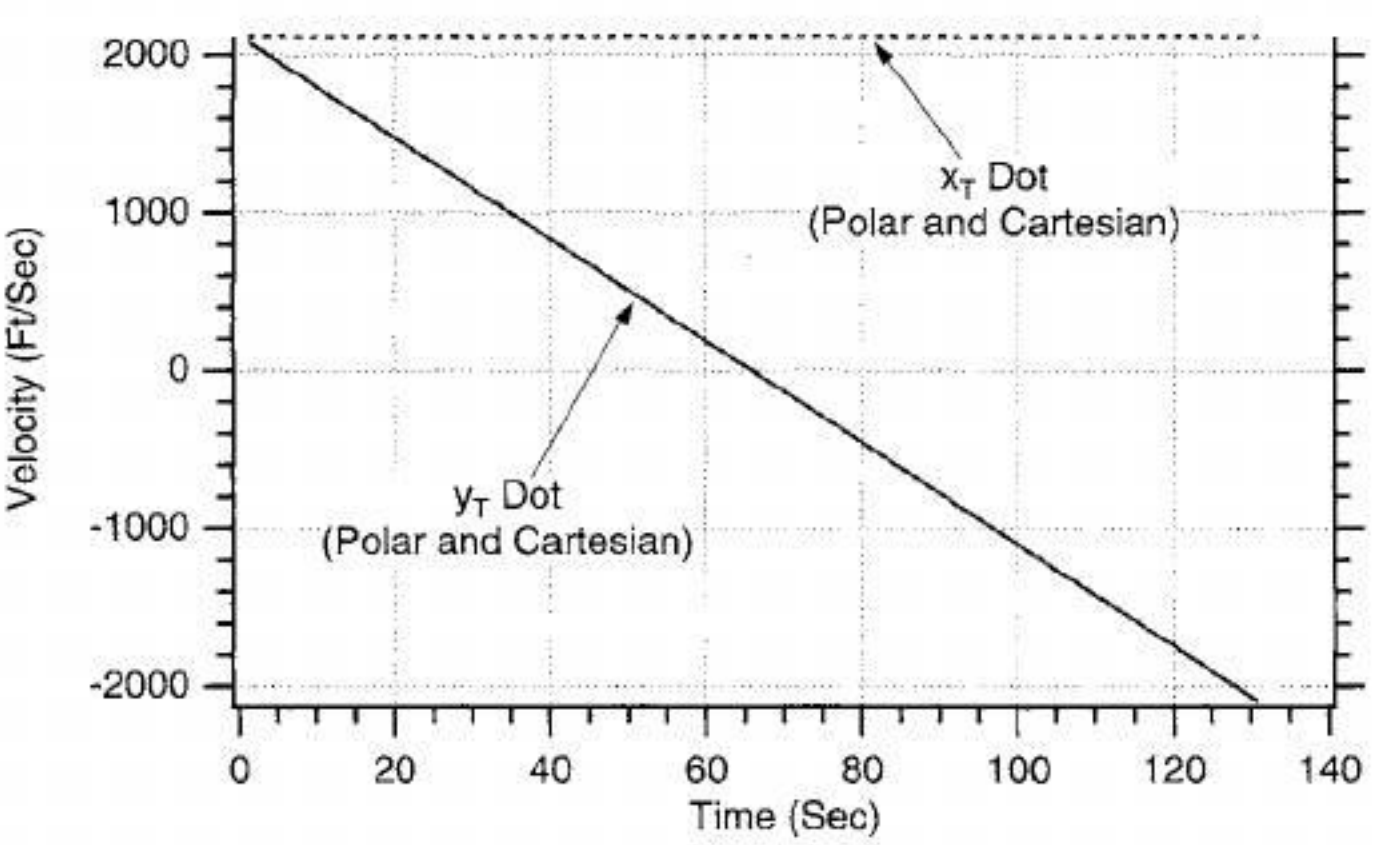

Fig. 9.14 Polar and Cartesian differential equations are identical in velocity.

In addition, we have already seen that, for this example, the extended Cartesian Kalman filter performed well without a process-noise matrix because the filter model of the real world was perfect. Let us assume, for purposes of comparison, that we also can neglect the process noise in polar coordinates. Under these circumstances we can express the nonlinear polar differential equations describing projectile motion in state-space form as

$$\begin{bmatrix} \Delta\dot{\theta} \\ \Delta\ddot{\theta} \\ \Delta\dot{r} \\ \Delta\ddot{r} \end{bmatrix} = \begin{bmatrix} \frac{\partial\dot{\theta}}{\partial\theta} & \frac{\partial\dot{\theta}}{\partial\dot{\theta}} & \frac{\partial\dot{\theta}}{\partial r} & \frac{\partial\dot{\theta}}{\partial\dot{r}} \\ \frac{\partial\ddot{\theta}}{\partial\theta} & \frac{\partial\ddot{\theta}}{\partial\dot{\theta}} & \frac{\partial\ddot{\theta}}{\partial r} & \frac{\partial\ddot{\theta}}{\partial\dot{r}} \\ \frac{\partial\dot{r}}{\partial\theta} & \frac{\partial\dot{r}}{\partial\dot{\theta}} & \frac{\partial\dot{r}}{\partial r} & \frac{\partial\dot{r}}{\partial\dot{r}} \\ \frac{\partial\ddot{r}}{\partial\theta} & \frac{\partial\ddot{r}}{\partial\dot{\theta}} & \frac{\partial\ddot{r}}{\partial r} & \frac{\partial\ddot{r}}{\partial\dot{r}} \end{bmatrix} \begin{bmatrix} \Delta\theta \\ \Delta\dot{\theta} \\ \Delta r \\ \Delta\dot{r} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{\partial\ddot{\theta}}{\partial\theta} & \frac{\partial\ddot{\theta}}{\partial\dot{\theta}} & \frac{\partial\ddot{\theta}}{\partial r} & \frac{\partial\ddot{\theta}}{\partial\dot{r}} \\ 0 & 0 & 0 & 1 \\ \frac{\partial\ddot{r}}{\partial\theta} & \frac{\partial\ddot{r}}{\partial\dot{\theta}} & \frac{\partial\ddot{r}}{\partial r} & \frac{\partial\ddot{r}}{\partial\dot{r}} \end{bmatrix} \begin{bmatrix} \Delta\theta \\ \Delta\dot{\theta} \\ \Delta r \\ \Delta\dot{r} \end{bmatrix}$$

In this case the system dynamics matrix, which is the preceding matrix of partial derivatives, can be written by inspection as

$$\boldsymbol{F} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{g\sin\theta}{r} & \frac{-2\dot{r}}{r} & \frac{g\cos\theta + 2\dot{\theta}\dot{r}}{r^2} & \frac{-2\dot{\theta}}{r} \\ 0 & 0 & 0 & 1 \\ -g\cos\theta & 2r\dot{\theta} & \dot{\theta}^2 & 0 \end{bmatrix}$$

Therefore, if we use a two-term Taylor-series approximation for the fundamental matrix, we get

$$\Phi(t) = \boldsymbol{I} + \boldsymbol{F}t = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & t & 0 & 0 \\ \frac{g\sin\theta}{r}t & \frac{-2\dot{r}}{r}t & \frac{g\cos\theta + 2\dot{\theta}\dot{r}}{r^2}t & \frac{-2\dot{\theta}}{r}t \\ 0 & 0 & 0 & t \\ -g\cos\theta t & 2r\dot{\theta}t & \dot{\theta}^2 t & 0 \end{bmatrix}$$

After some simplification we finally obtain

$$\Phi(t) = \begin{bmatrix} 1 & t & 0 & 0 \\ \frac{g\sin\theta}{r}t & 1 - \frac{2\dot{r}}{r}t & \frac{g\cos\theta + 2\dot{\theta}\dot{r}}{r^2}t & \frac{-2\dot{\theta}}{r}t \\ 0 & 0 & 1 & t \\ -g\cos\theta t & 2r\dot{\theta}t & \dot{\theta}^2 t & 1 \end{bmatrix}$$

The discrete fundamental matrix can be found by substituting T_s for t in the preceding expression, thus obtaining

$$\Phi_k = \begin{bmatrix} 1 & T_s & 0 & 0 \\ \dfrac{g\sin\theta}{r}T_s & 1-\dfrac{2\dot{r}}{r}T_s & \dfrac{g\cos\theta+2\dot{\theta}\dot{r}}{r^2}T_s & \dfrac{-2\dot{\theta}}{r}T_s \\ 0 & 0 & 1 & T_s \\ -g\cos\theta T_s & 2r\theta\dot{T}_s & \dot{\theta}^2 T_s & 1 \end{bmatrix}$$

Because the states are polar, the measurement equation is linear and can be expressed as

$$\begin{bmatrix} \theta^* \\ r^* \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \\ r \\ \dot{r} \end{bmatrix} + \begin{bmatrix} v_\theta \\ v_r \end{bmatrix}$$

where the measurement matrix is

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

Because the discrete measurement noise matrix is given by

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

we can say that for this problem the discrete measurement noise matrix turns out to be

$$\boldsymbol{R}_k = \begin{bmatrix} \sigma_\theta^2 & 0 \\ 0 & \sigma_r^2 \end{bmatrix}$$

where σ_θ^2 and σ_r^2 are the variances of the angle noise and range noise measurements, respectively.

Because we are assuming zero process noise, we now have defined all of the matrices required to solve the Riccati equations. However, if we want to compare the resultant polar covariance matrices obtained from the Riccati equations with the preceding Cartesian covariance matrices, then we must have a way of going between both systems. Recall that we have shown in the preceding section that

the four equations relating the polar description of the projectile to the Cartesian description of the projectile are given by

$$x_T = r\cos\theta + x_R$$

$$y_T = r\sin\theta + y_R$$

$$\dot{x}_T = \dot{r}\cos\theta - r\dot{\theta}\sin\theta$$

$$\dot{y}_T = \dot{r}\sin\theta + r\dot{\theta}\cos\theta$$

We can use the chain rule from calculus to get the total differentials

$$\Delta x_T = \frac{\partial x_T}{\partial\theta}\Delta\theta + \frac{\partial x_T}{\partial\dot{\theta}}\Delta\dot{\theta} + \frac{\partial x_T}{\partial r}\Delta r + \frac{\partial x_T}{\partial\dot{r}}\Delta\dot{r}$$

$$\Delta\dot{x}_T = \frac{\partial\dot{x}_T}{\partial\theta}\Delta\theta + \frac{\partial\dot{x}_T}{\partial\dot{\theta}}\Delta\dot{\theta} + \frac{\partial\dot{x}_T}{\partial r}\Delta r + \frac{\partial\dot{x}_T}{\partial\dot{r}}\Delta\dot{r}$$

$$\Delta y_T = \frac{\partial y_T}{\partial\theta}\Delta\theta + \frac{\partial y_T}{\partial\dot{\theta}}\Delta\dot{\theta} + \frac{\partial y_T}{\partial r}\Delta r + \frac{\partial y_T}{\partial\dot{r}}\Delta\dot{r}$$

$$\Delta\dot{y}_T = \frac{\partial\dot{y}_T}{\partial\theta}\Delta\theta + \frac{\partial\dot{y}_T}{\partial\dot{\theta}}\Delta\dot{\theta} + \frac{\partial\dot{y}_T}{\partial r}\Delta r + \frac{\partial\dot{y}_T}{\partial\dot{r}}\Delta\dot{r}$$

Evaluating each of the partial derivatives yields the following expressions for each of the total differentials:

$$\Delta x_T = -r\sin\theta\Delta\theta + \cos\theta\Delta r$$

$$\Delta\dot{x}_T = (-\dot{r}\sin\theta - r\dot{\theta}\cos\theta)\Delta\theta - r\sin\theta\Delta\dot{\theta} - \dot{\theta}\sin\theta\Delta r + \cos\theta\Delta\dot{r}$$

$$\Delta y_T = r\cos\theta\Delta\theta + \sin\theta\Delta r$$

$$\Delta\dot{y}_T = (\dot{r}\cos\theta - r\dot{\theta}\sin\theta)\Delta\theta + r\cos\theta\Delta\dot{\theta} + \dot{\theta}\cos\theta\Delta r + \sin\theta\Delta\dot{r}$$

Placing the preceding set of equations into state-space form yields a more compact expression for the total differential

$$\begin{bmatrix}\Delta x_T\\ \Delta\dot{x}_T\\ \Delta y_T\\ \Delta\dot{y}_T\end{bmatrix} = \begin{bmatrix}-r\sin\theta & 0 & \cos\theta & 0\\ -\dot{r}\sin\theta - r\dot{\theta}\cos\theta & -r\sin\theta & -\dot{\theta}\sin\theta & \cos\theta\\ r\cos\theta & 0 & \sin\theta & 0\\ \dot{r}\cos\theta & r\cos\theta & \dot{\theta}\cos\theta & \sin\theta\end{bmatrix}\begin{bmatrix}\Delta\theta\\ \Delta\dot{\theta}\\ \Delta r\\ \Delta\dot{r}\end{bmatrix}$$

If we define the transformation matrix A relating the polar total differentials to the Cartesian total differentials, we get

$$A = \begin{bmatrix} -r\sin\theta & 0 & \cos\theta & 0 \\ -\dot{r}\sin\theta - r\dot{\theta}\cos\theta & -r\sin\theta & -\dot{\theta}\sin\theta & \cos\theta \\ r\cos\theta & 0 & \sin\theta & 0 \\ \dot{r}\cos\theta & r\cos\theta & \dot{\theta}\cos\theta & \sin\theta \end{bmatrix}$$

It is easy to show that the Cartesian covariance matrix P_{CART} is related to the polar covariance matrix P_{POL} according to

$$P_{CART} = AP_{POL}A^T$$

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 could be done exactly with the fundamental matrix when we were working in the Cartesian system because the fundamental matrix was exact. In the polar system the fundamental matrix is approximate, and so it is best to numerically integrate the nonlinear differential equations for a sampling interval to get the projected states. Now the Kalman-filtering equations can be written as

$$\hat{\theta}_k = \bar{\theta}_k + K_{11_k}(\theta_k^* - \bar{\theta}_k) + K_{12_k}(r_k^* - \bar{r}_k)$$

$$\hat{\dot{\theta}}_k = \bar{\dot{\theta}}_k + K_{21_k}(\theta_k^* - \bar{\theta}_k) + K_{22_k}(r_k^* - \bar{r}_k)$$

$$\hat{r}_k = \bar{r}_k + K_{31_k}(\theta_k^* - \bar{\theta}_k) + K_{32_k}(r_k^* - \bar{r}_k)$$

$$\hat{\dot{r}}_k = \bar{\dot{r}}_k + K_{41_k}(\theta_k^* - \bar{\theta}_k) + K_{42_k}(r_k^* - \bar{r}_k)$$

where θ_k^* and r_k^* are the noisy measurements of radar angle and range and the bar indicates the projected states that are obtained directly by numerical integration.

The preceding equations for the polar Kalman filter and Riccati equations were programmed and are shown in Listing 9.3, along with a simulation of the real world. There is no process-noise matrix in this formulation. We have initialized the states of the polar filter to have the same initialization of the Cartesian filter. The initial covariance matrix reflects the initial state estimate errors. The subroutine PROJECT handles the numerical integration of the nonlinear polar differential equations to get the projected value of the states T_s seconds in the future.

The nominal case of Listing 9.3 was run, and the polar and Cartesian errors in the estimates were computed. By comparing Fig. 9.15 with Fig. 9.8, we can see that the errors in the estimates of downrange are comparable for both filters. Initially, the error in the estimate of x is 100 ft, and after 130 s the error reduces to approximately 25 ft. Figure 9.15 also shows that the single-run simulation error in the estimate of x appears to be correct when compared to the theoretical

Listing 9.3 Polar extended Kalman filter simulation for projectile tracking problem

```
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 P(4,4),M(4,4),GAIN(4,2),PHI(4,4),PHIT(4,4),PHIP(4,4)
        REAL*8 Q(4,4),HMAT(2,4),HM(2,4),MHT(4,2),PHIPPHIT(4,4),AT(4,4)
        REAL*8 RMAT(2,2),HMHTR(2,2),HMHTRINV(2,2),A(4,4),PNEW(4,4)
        REAL*8 HMHT(2,2),HT(4,2),KH(4,4),IDN(4,4),IKH(4,4),FTS(4,4)
        REAL*8 AP(4,4)
        INTEGER ORDER
106     CONTINUE
        OPEN(1,STATUS='UNKNOWN',FILE='DATFIL')
        OPEN(2,STATUS='UNKNOWN',FILE='COVFIL')
        OPEN(3,STATUS='UNKNOWN',FILE='COVFIL2')
        SIGTH=.01
        SIGR=100.
        TS=1.
        G=32.2
        VT=3000.
        GAMDEG=45.
        XT=0.
        YT=0.
        XTD=VT*COS(GAMDEG/57.3)
        YTD=VT*SIN(GAMDEG/57.3)
        XR=100000.
        YR=0.
        XTH=XT+1000.
        YTH=YT-1000.
        XTDH=XTD-100.
        YTDH=YTD+100.
        TH=ATAN2((YT-YR),(XT-XR)+.001)
        R=SQRT((XT-XR)**2+(YT-YR)**2)
        THD=((XT-XR)*YTD-(YT-YR)*XTD)/R**2
        RD=((XT-XR)*XTD+(YT-YR)*YTD)/R
        THH=ATAN2((YTH-YR),(XTH-XR)+.001)
        RH=SQRT((XTH-XR)**2+(YTH-YR)**2)
        THDH=((XTH-XR)*YTDH-(YTH-YR)*XTDH)/RH**2
        RDH=((XTH-XR)*XTDH+(YTH-YR)*YTDH)/RH
        ORDER=4
        TF=100.
        T=0.
        S=0.
        H=.001
        HP=.001
        DO 1000 I=1,ORDER
        DO 1000 J=1,ORDER
```

(*continued*)

Listing 9.3 (*Continued*)

```
                P(I,J)=0.
                Q(I,J)=0.
                IDN(I,J)=0.
                PHI(I,J)=0.
                FTS(I,J)=0.
                A(I,J)=0.
1000        CONTINUE
            IDN(1,1)=1.
            IDN(2,2)=1.
            IDN(3,3)=1.
            IDN(4,4)=1.
            P(1,1)=(TH-THH)**2
            P(2,2)=(THD-THDH)**2
            P(3,3)=(R-RH)**2
            P(4,4)=(RD-RDH)**2
            RMAT(1,1)=SIGTH**2
            RMAT(1,2)=0.
            RMAT(2,1)=0.
            RMAT(2,2)=SIGR**2
            HMAT(1,1)=1.
            HMAT(1,2)=0.
            HMAT(1,3)=0.
            HMAT(1,4)=0.
            HMAT(2,1)=0.
            HMAT(2,2)=0.
            HMAT(2,3)=1.
            HMAT(2,4)=0.
            WHILE(YT>=0.)
                    THOLD=TH
                    THDOLD=THD
                    ROLD=R
                    RDOLD=RD
                    THDD=(-G*R*COS(TH)-2.*THD*R*RD)/R**2
                    RDD=(R*R*THD*THD-G*R*SIN(TH))/R
                    TH=TH+H*THD
                    THD=THD+H*THDD
                    R=R+H*RD
                    RD=RD+H*RDD
                    T=T+H
                    THDD=(-G*R*COS(TH)-2.*THD*R*RD)/R**2
                    RDD=(R*R*THD*THD-G*R*SIN(TH))/R
                    TH=.5*(THOLD+TH+H*THD)
                    THD=.5*(THDOLD+THD+H*THDD)
                    R=.5*(ROLD+R+H*RD)
                    RD=.5*(RDOLD+RD+H*RDD)
                    S=S+H
                    IF(S>=(TS-.00001))THEN
                            S=0.
                            FTS(1,2)=1.*TS
                            FTS(2,1)=G*SIN(THH)*TS/RH
```

(*continued*)

Listing 9.3 *(Continued)*

```
FTS(2,2)=-2.*RDH*TS/RH
FTS(2,3)=(G*COS(THH)+2.*THDH*RDH)*TS/RH**2
FTS(2,4)=-2.*THDH*TS/RH
FTS(3,4)=1.*TS
FTS(4,1)=-G*COS(THH)*TS
FTS(4,2)=2.*RH*THDH*TS
FTS(4,3)=(THDH**2)*TS
CALL  MATADD(FTS,ORDER,ORDER,IDN,PHI)
CALL  MATTRN(HMAT,2,ORDER,HT)
CALL  MATTRN(PHI,ORDER,ORDER,PHIT)
CALL  MATMUL(PHI,ORDER,ORDER,P,ORDER,
   ORDER,PHIP)
CALL  MATMUL(PHIP,ORDER,ORDER,PHIT,ORDER,
   ORDER,  PHIPPHIT)
CALL  MATADD(PHIPPHIT,ORDER,ORDER,Q,M)
CALL  MATMUL(HMAT,2,ORDER,M,ORDER,
   ORDER,HM)
CALL  MATMUL(HM,2,ORDER,HT,ORDER,2,HMHT)
CALL  MATADD(HMHT,2,2,RMAT,HMHTR)
DET=HMHTR(1,1)*HMHTR(2,2)-HMHTR(1,2)*
   HMHTR(2,1)
HMHTRINV(1,1)=HMHTR(2,2)/DET
HMHTRINV(1,2)=-HMHTR(1,2)/DET
HMHTRINV(2,1)=-HMHTR(2,1)/DET
HMHTRINV(2,2)=HMHTR(1,1)/DET
CALL  MATMUL(M,ORDER,ORDER,HT,ORDER,
   2,MHT)
CALL  MATMUL(MHT,ORDER,2,HMHTRINV,2,
   2,GAIN)
CALL  MATMUL(GAIN,ORDER,2,HMAT,2,ORDER,KH)
CALL  MATSUB(IDN,ORDER,ORDER,KH,IKH)
CALL  MATMUL(IKH,ORDER,ORDER,M,ORDER,
   ORDER,P)
CALL  GAUSS(THNOISE,SIGTH)
CALL  GAUSS(RNOISE,SIGR)
CALL  PROJECT(T,TS,THH,THDH,RH,RDH,THB,
   THDB,RB,RDB,HP)
RES1=TH+THNOISE-THB
RES2=R+RNOISE-RB
THH=THB+GAIN(1,1)*RES1+GAIN(1,2)*RES2
THDH=THDB+GAIN(2,1)*RES1+GAIN(2,2)*RES2
RH=RB+GAIN(3,1)*RES1+GAIN(3,2)*RES2
RDH=RDB+GAIN(4,1)*RES1+GAIN(4,2)*RES2
ERRTH=TH-THH
SP11=SQRT(P(1,1))
ERRTHD=THD-THDH
SP22=SQRT(P(2,2))
ERRR=R-RH
```

(continued)

Listing 9.3 *(Continued)*

```
      SP33=SQRT(P(3,3))
      ERRRD=RD-RDH
      SP44=SQRT(P(4,4))
      XT=R*COS(TH)+XR
      YT=R*SIN(TH)+YR
      XTD=RD*COS(TH)-R*THD*SIN(TH)
      YTD=RD*SIN(TH)+R*THD*COS(TH)
      XTH=RH*COS(THH)+XR
      YTH=RH*SIN(THH)+YR
      XTDH=RDH*COS(THH)-RH*THDH*SIN(THH)
      YTDH=RDH*SIN(THH)+RH*THDH*COS(THH)
      A(1,1)=-RH*SIN(THH)
      A(1,3)=COS(THH)
      A(2,1)=-RDH*SIN(THH)-RH*THDH*COS(THH)
      A(2,2)=-RH*SIN(THH)
      A(2,3)=-THDH*SIN(THH)
      A(2,4)=COS(THH)
      A(3,1)=RH*COS(THH)
      A(3,3)=SIN(THH)
      A(4,1)=RDH*COS(THH)-RH*SIN(THH)*THDH
      A(4,2)=RH*COS(THH)
      A(4,3)=THDH*COS(THH)
      A(4,4)=SIN(THH)
      CALL  MATTRN(A,ORDER,ORDER,AT)
      CALL  MATMUL(A,ORDER,ORDER,P,ORDER,
     1   ORDER,AP)
      CALL  MATMUL(AP,ORDER,ORDER,AT,ORDER,
     1   ORDER,PNEW)
      ERRXT=XT-XTH
      SP11P=SQRT(PNEW(1,1))
      ERRXTD=XTD-XTDH
      SP22P=SQRT(PNEW(2,2))
      ERRYT=YT-YTH
      SP33P=SQRT(PNEW(3,3))
      ERRYTD=YTD-YTDH
      SP44P=SQRT(PNEW(4,4))
      WRITE(9,*)T,R,RH,RD,RDH,TH,THH,THD,THDH
      WRITE(1,*)T,R,RH,RD,RDH,TH,THH,THD,THDH
      WRITE(2,*)T,ERRTH,SP11,-SP11,ERRTHD,SP22,-SP22,
     1   ERRR,SP33,-SP33,ERRRD,SP44,-SP44
      WRITE(3,*)T,ERRXT,SP11P,-SP11P,ERRXTD,SP22P,
     1   -SP22P,ERRYT,SP33P,-SP33P,ERRYTD,
     2   SP44P,-SP44P
            ENDIF
      END DO
      PAUSE
      CLOSE(1)
      CLOSE(2)
      CLOSE(3)
      END
```

(continued)

Listing 9.3 (*Continued*)

```
      SUBROUTINE PROJECT(TP,TS,THP,THDP,RP,RDP,THH,THDH,RH,
     1   RDH,HP)
      IMPLICIT REAL*8 (A-H)
      IMPLICIT REAL*8 (O-Z)
      T=0.
      G=32.2
      TH=THP
      THD=THDP
      R=RP
      RD=RDP
      H=HP
      WHILE(T<=(TS-.0001))
            THDD=(-G*R*COS(TH)-2.*THD*R*RD)/R**2
            RDD=(R*R*THD*THD-G*R*SIN(TH))/R
            THD=THD+H*THDD
            TH=TH+H*THD
            RD=RD+H*RDD
            R=R+H*RD
            T=T+H
      END DO
      RH=R
      RDH=RD
      THH=TH
      THDH=THD
      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
C SUBROUTINE MATSCA IS SHOWN IN LISTING 1.5
```

predictions from the transformed covariance matrix in the sense that it appears to be within the theoretical error bounds approximately 68% of the time.

Figure 9.16 displays the error in the estimate of downrange velocity. By comparing Fig. 9.16 with Fig. 9.9, we can see that the errors in the estimate of downrange velocity are comparable for both the polar and Cartesian extended Kalman filters. Initially, the error in the estimate of velocity is approximately 20 ft/s at 10 s and diminishes to approximately 1 ft/s after 130 s. Figure 9.16 also shows that the single flight error in the estimate of $\dot{x}$ appears to be correct when compared to the theoretical predictions from the transformed covariance matrix.

Figure 9.17 displays the error in the estimate of altitude. By comparing Fig. 9.17 with Fig. 9.10, we can see that the errors in the estimate of altitude are comparable for both the polar and Cartesian extended Kalman filters. Initially, the

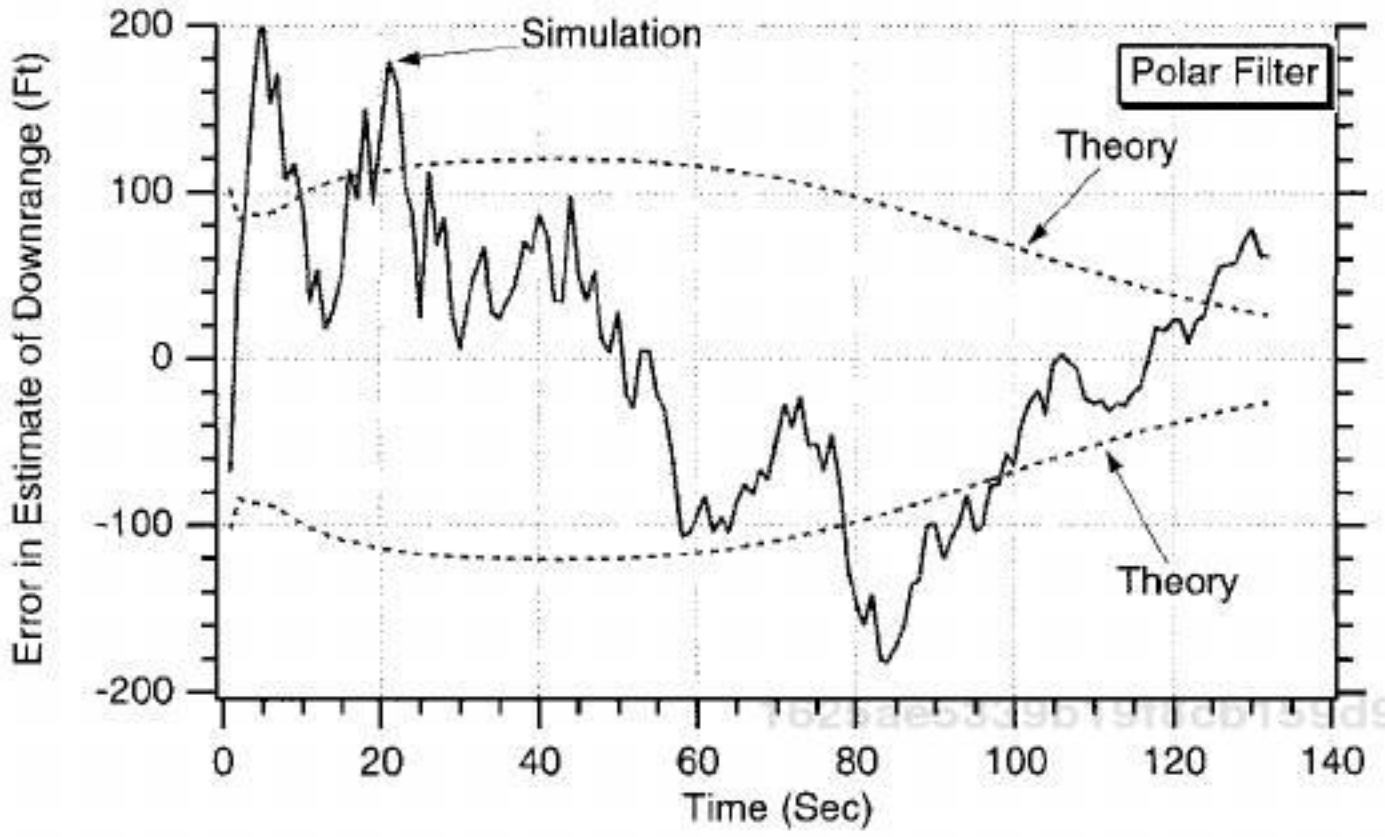

Fig. 9.15 Polar and Cartesian extended Kalman filters yield similar results for downrange estimates.

error in the estimate of altitude is approximately 600 ft and diminishes to approximately 100 ft after 130 s. Figure 9.17 also shows that the single-run simulation error in the estimate of y also appears to be correct when compared to the theoretical predictions from the transformed covariance matrix.

Figure 9.18 displays the error in the estimate of velocity in the altitude direction. By comparing Fig. 9.18 with Fig. 9.11, we can see that the errors in the estimate of the altitude velocity component are comparable for both the polar and Cartesian extended Kalman filters. Initially, the error in the estimate of altitude velocity is approximately 20 ft/s at 20 s and diminishes to approximately 2 ft/s after 130 s. Figure 9.18 also shows that the single-run simulation error in the estimate of $\dot{y}$ appears to be correct when compared to the theoretical predictions from the transformed covariance matrix.

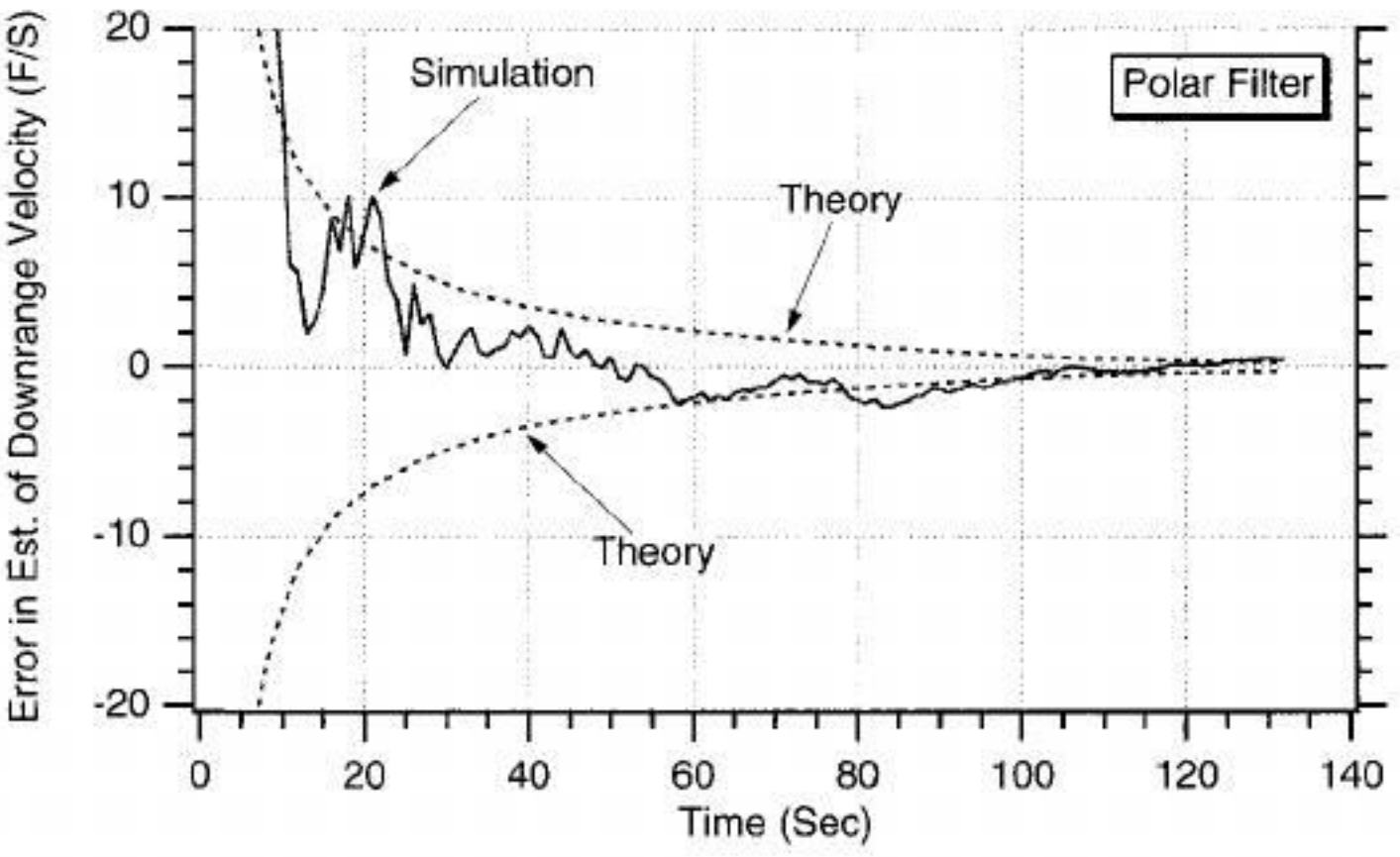

Fig. 9.16 Polar and Cartesian extended Kalman filters yield similar results for downrange velocity estimates.

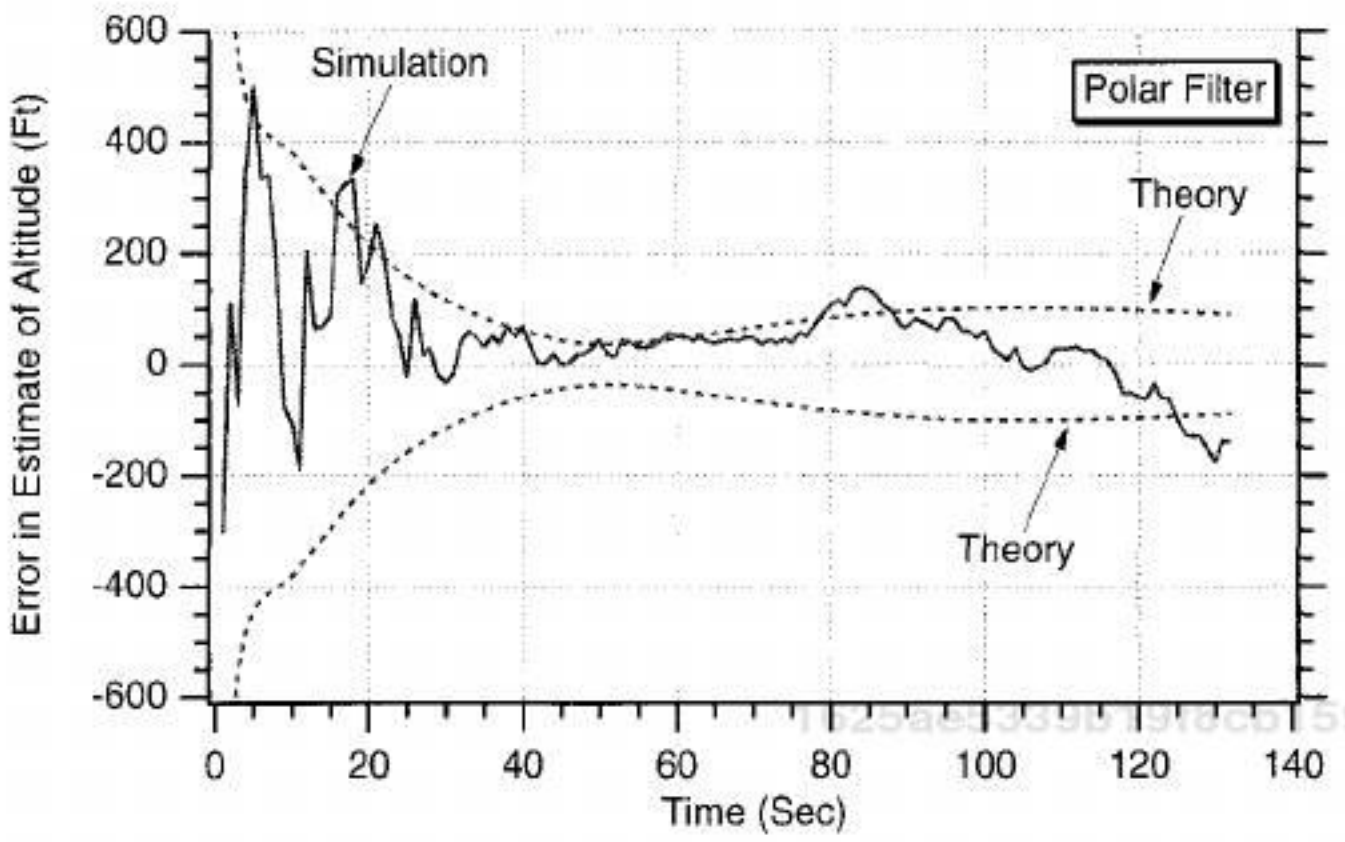

Fig. 9.17 Polar and Cartesian extended Kalman filters yield similar results for altitude estimates.

Recall that the polar filter states are angle, angle rate, distance, and distance rate or

$$\begin{bmatrix} \theta \\ \dot{\theta} \\ r \\ \dot{r} \end{bmatrix}$$

The covariance matrix resulting from the polar extended-Kalman-filter Riccati equations represents errors in the estimates of those states. Figures 9.19–9.22 display the single simulation run and theoretical (i.e., square root of appropriate diagonal elements of covariance matrix) error in the estimate of angle, angle rate,

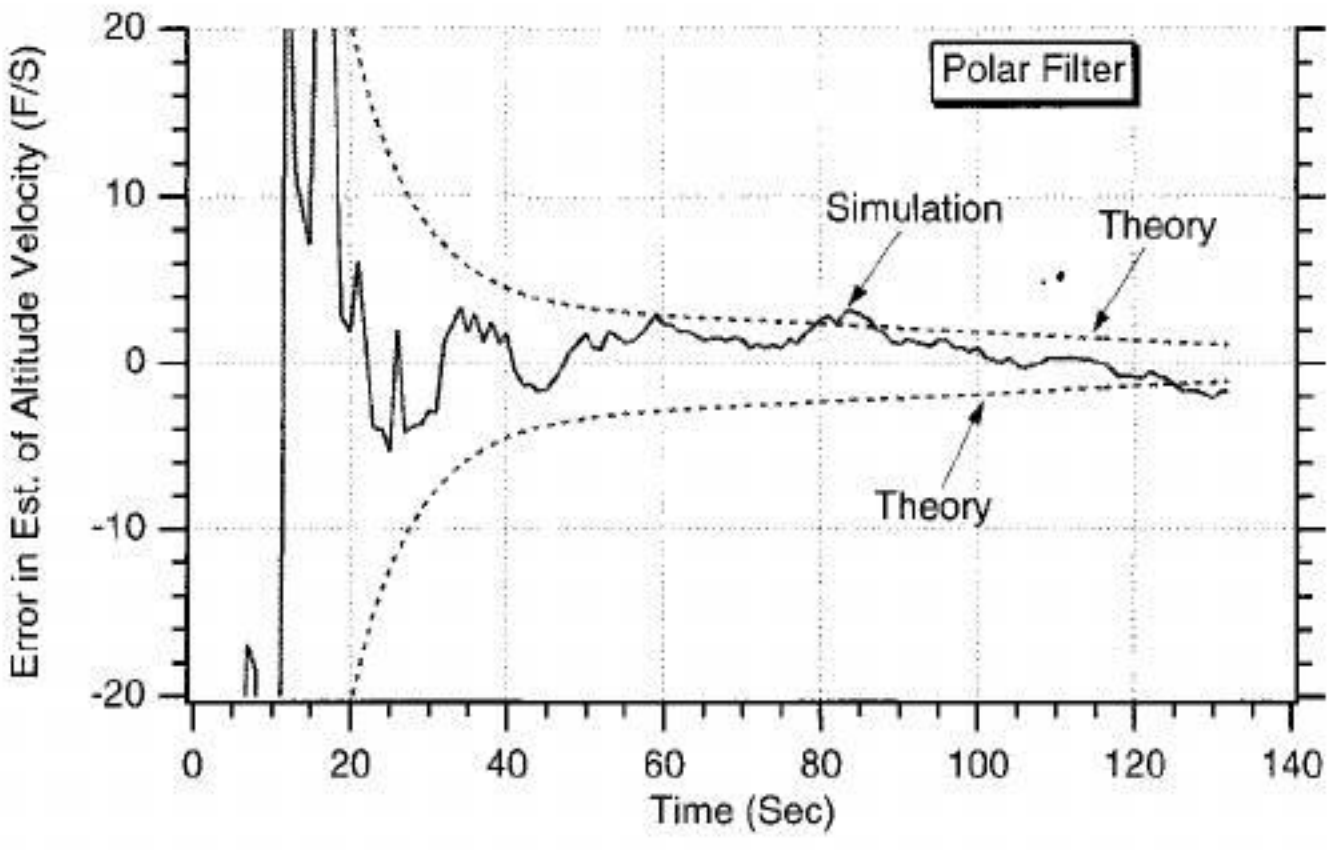

Fig. 9.18 Polar and Cartesian extended Kalman filters yield similar results for altitude velocity estimates.

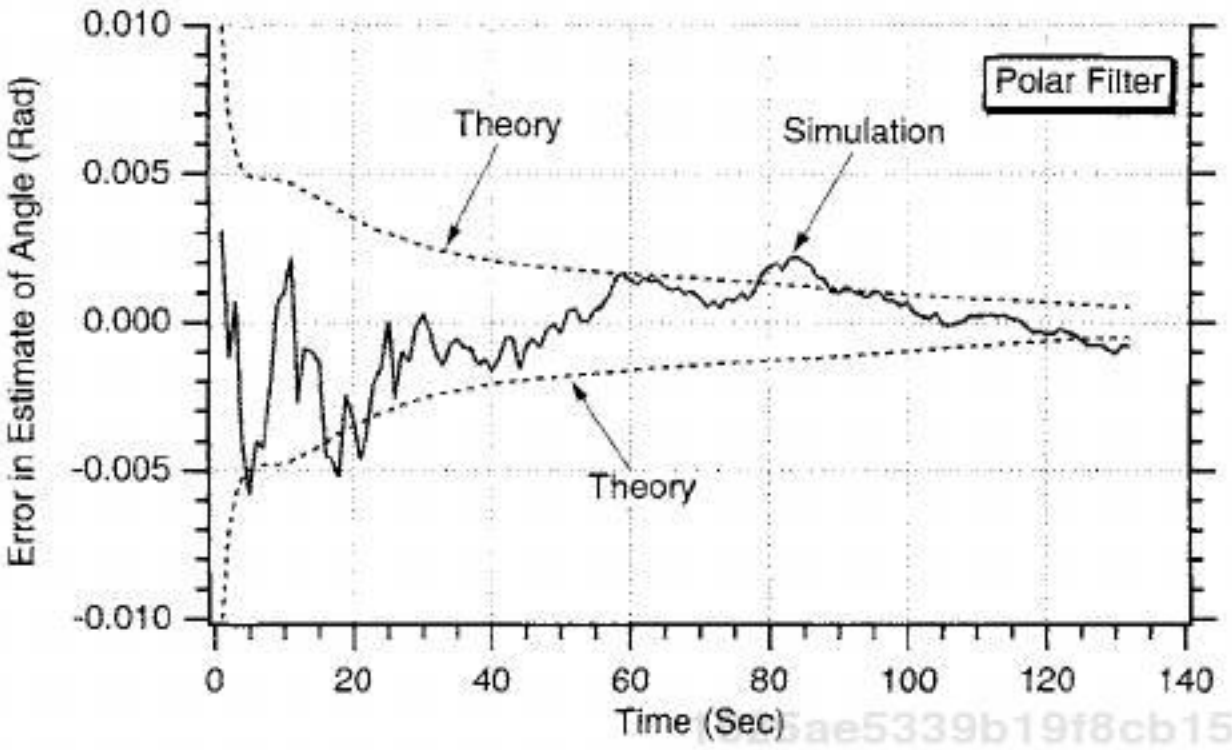

Fig. 9.19 Error in the estimate of angle indicates that extended polar Kalman filter appears to be working properly.

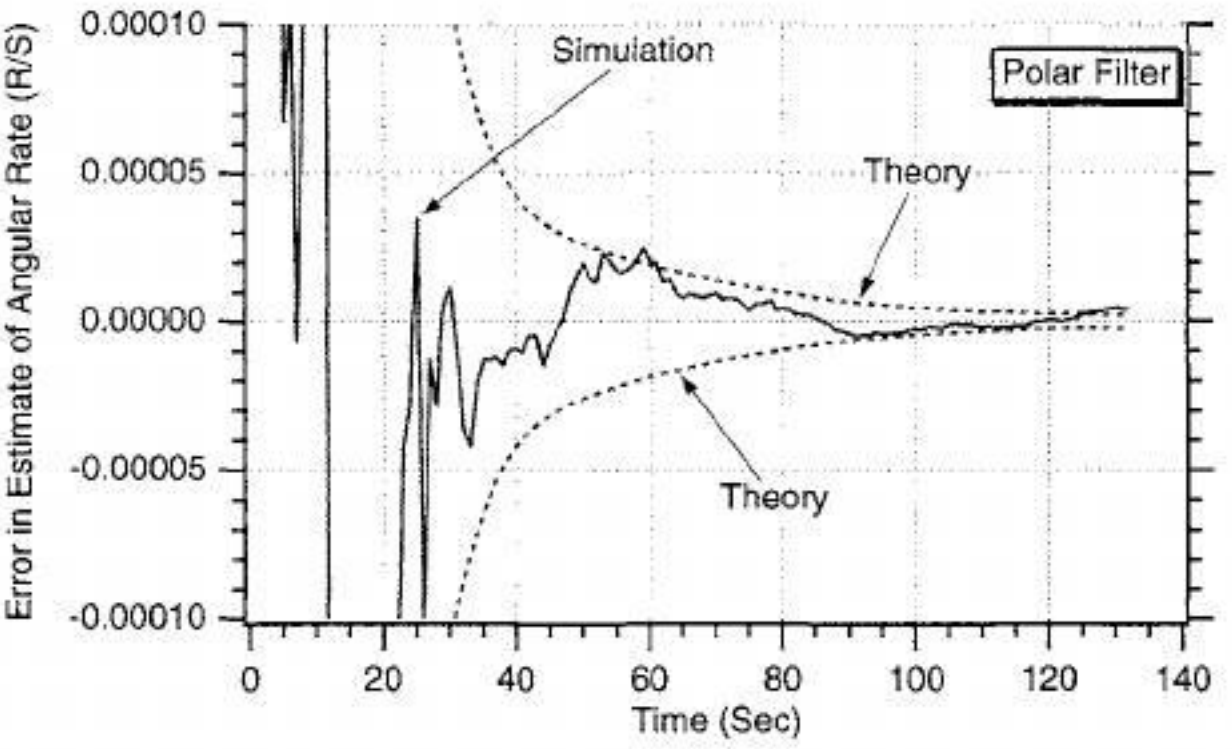

Fig. 9.20 Error in the estimate of angle rate indicates that extended polar Kalman filter appears to be working properly.

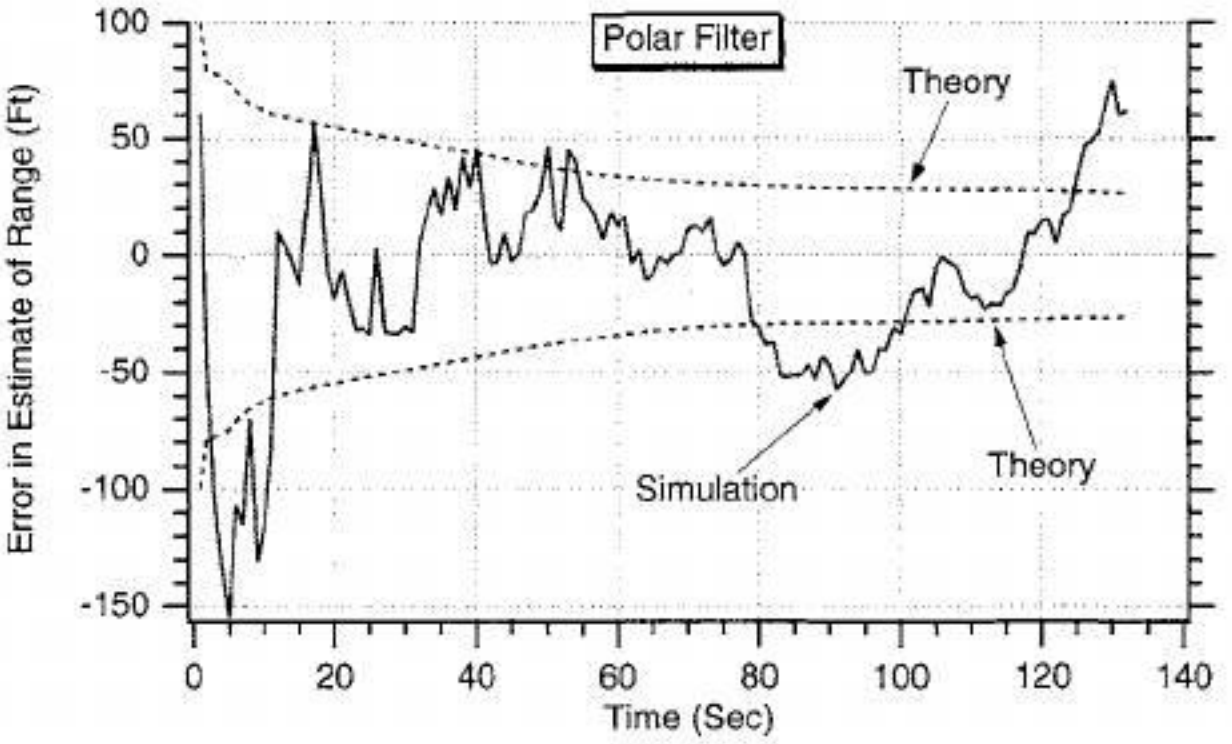

Fig. 9.21 Error in the estimate of range indicates that extended polar Kalman filter appears to be working properly.

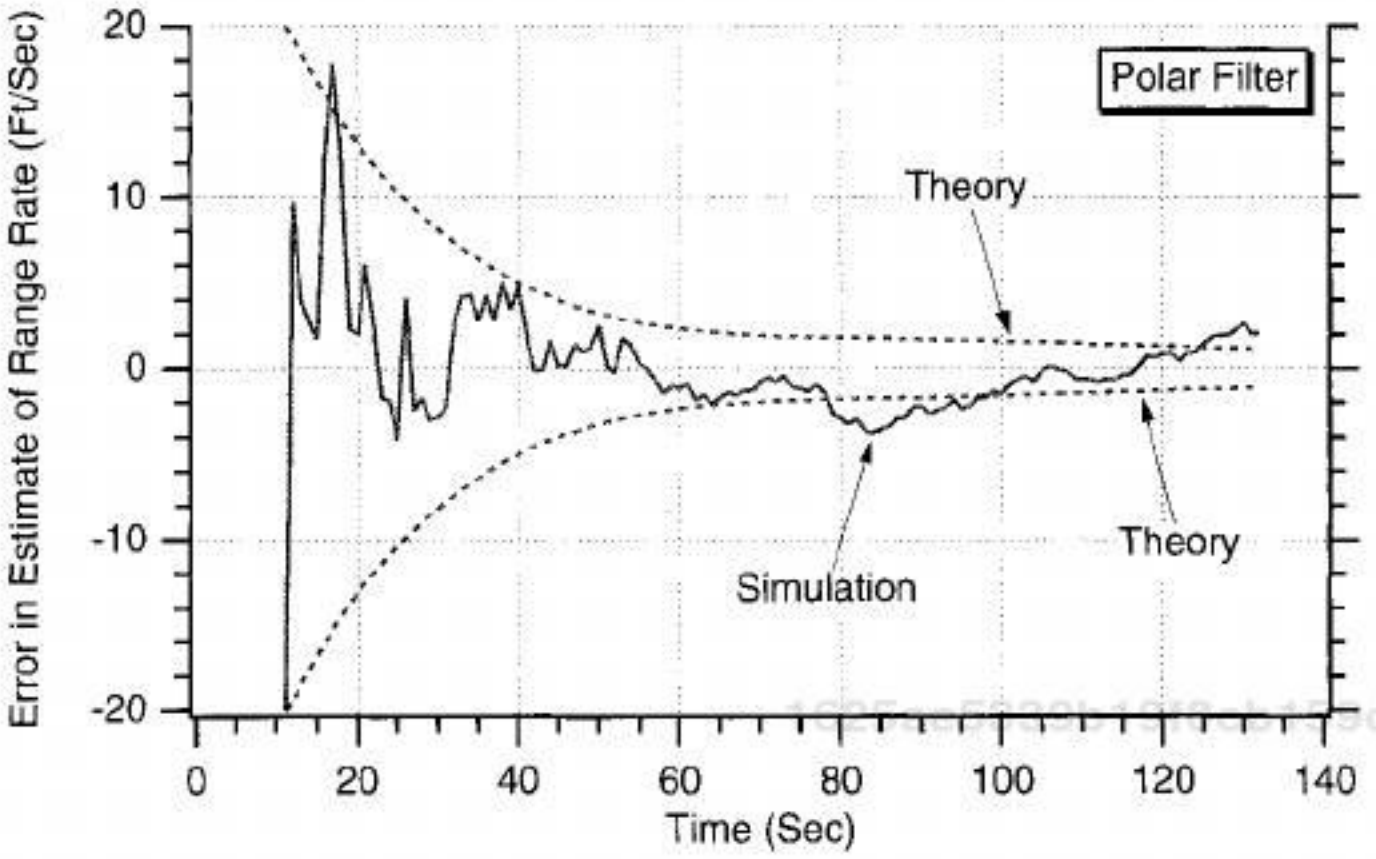

Fig. 9.22 Error in the estimate of range rate indicates that extended polar Kalman filter appears to be working properly.

range, and range rate, respectively. We can see from each of the plots that the single-run simulation error in the estimate is within the theoretical predictions of the covariance matrix approximately 68% of the time, indicating that the extended polar Kalman filter appears to be working properly.

In summary, we can say that there does not appear to be any advantage in using a polar extended Kalman filter for this example. The performance results of the filter are comparable to those of the Cartesian extended Kalman filter, although the computational burden is significantly greater. The increase in computational burden for the polar filter is primarily caused by the states having to be propagated forward at each sampling instant by numerical integration. With the Cartesian extended Kalman filter the fundamental matrix was exact, and numerical integration was not required for state propagation (i.e., multiplying the states by the fundamental matrix propagates the states).

Using Linear Decoupled Polynomial Kalman Filters

The cannon-launched projectile problem was ideal for an extended Kalman filter because either the measurements were nonlinear (if viewed in a Cartesian frame) or the system equations were nonlinear (if viewed in a polar frame). The problem also could have been manipulated to make it appropriate to apply two linear decoupled two-state polynomial Kalman filters—one in the downrange direction and the other in the altitude direction. This can be accomplished by pretending that the filter measures downrange and altitude (i.e., x_T^* and y_T^*), rather than angle and range (i.e., θ^* and r^*).

To find the equivalent noise on downrange and altitude, we have to go through the following procedure. We have already shown that downrange and altitude can be expressed in terms of range and angle according to

$$x_T = r\cos\theta + x_R$$

$$y_T = r\sin\theta + y_R$$

Using calculus, we can find the total differential of the preceding two equations as

$$\Delta x_T = \frac{\partial x_T}{\partial r}\Delta r + \frac{\partial x_T}{\partial \theta}\Delta\theta = \cos\theta\Delta r - r\sin\theta\Delta\theta$$

$$\Delta y_T = \frac{\partial y_T}{\partial r}\Delta r + \frac{\partial y_T}{\partial \theta}\Delta\theta = \sin\theta\Delta r + r\cos\theta\Delta\theta$$

We can square each of the preceding equations to obtain

$$\Delta x_T^2 = \cos^2\theta\Delta r^2 - 2r\sin\theta\cos\theta\Delta r\Delta\theta + r^2\sin^2\theta\Delta\theta^2$$
$$\Delta y_T^2 = \sin^2\theta\Delta r^2 + 2r\sin\theta\cos\theta\Delta r\Delta\theta + r^2\cos^2\theta\Delta\theta^2$$

To find the variance of the pseudonoise on downrange and altitude, we can take expectations of both sides of the equations, assuming that the actual range and angle noise are not correlated [i.e., $E(\Delta r\Delta\theta) = 0$). Therefore, we can say that

$$E(\Delta x_T^2) = \cos^2\theta E(\Delta r^2) + r^2\sin^2\theta E(\Delta\theta^2)$$
$$E(\Delta y_T^2) = \sin^2\theta E(\Delta r^2) + r^2\cos^2\theta E(\Delta\theta^2)$$

Because

$$\sigma_{x_T}^2 = E(\Delta x_T^2)$$
$$\sigma_{y_T}^2 = E(\Delta y_T^2)$$
$$\sigma_r^2 = E(\Delta r^2)$$
$$\sigma_\theta^2 = E(\Delta\theta^2)$$

we can also say that

$$\sigma_{x_T}^2 = \cos^2\theta\sigma_r^2 + r^2\sin^2\theta\sigma_\theta^2$$
$$\sigma_{y_T}^2 = \sin^2\theta\sigma_r^2 + r^2\cos^2\theta\sigma_\theta^2$$

In other words, we are pretending that we are measuring downrange and altitude with measurement accuracies described by the two preceding equations. We now have enough information for the design of the filters in each channel. In downrange there is no acceleration acting on the projectile. The model of the real world in this channel can be represented in state-space form as

$$\begin{bmatrix}\dot{x}_T\\ \ddot{x}_T\end{bmatrix} = \begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}\begin{bmatrix}x_T\\ \dot{x}_T\end{bmatrix} + \begin{bmatrix}0\\ u_s\end{bmatrix}$$

where u_s is white noise, which has been added to acceleration for possible protection as a result of uncertainties in modeling. Therefore, the continuous process-noise matrix is given by

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

where Φ_s is the spectral density of the white process noise. The systems dynamics matrix is obtained from the state-space equation by inspection and is given by

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

We have already shown in Chapter 4 that the fundamental matrix can be obtained exactly from this systems dynamics matrix and is given by

$$\Phi(\boldsymbol{t}) = \begin{bmatrix} 0 & t \\ 0 & 1 \end{bmatrix}$$

which means that the discrete fundamental matrix is

$$\Phi_k = \begin{bmatrix} 0 & T_s \\ 0 & 1 \end{bmatrix}$$

The discrete process-noise matrix is related to the continuous process-noise matrix according to

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

We have already derived the discrete process-noise matrix for this type of example in Chapter 4, and the results are given by

$$\boldsymbol{Q}_k = \Phi_s \begin{bmatrix} \dfrac{T_s^3}{3} & \dfrac{T_s^2}{2} \\ \dfrac{T_s^2}{2} & T_s \end{bmatrix}$$

The measurement equation is linear because we have assumed a pseudomeasurement and it is given by

$$x_T^* = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_T \\ \dot{x}_T \end{bmatrix} + v_x$$

Therefore, the measurement matrix is given by

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

In this case the measurement noise matrix is simply a scalar and is given by

$$\boldsymbol{R}_k = \sigma_{x_T}^2$$

We now have enough information to solve the Riccati equations for the Kalman gains in the downrange channel. Because the fundamental matrix is exact in this example, we can also use the preceding matrices in the linear Kalman-filtering equation

$$\hat{\boldsymbol{x}}_k = \Phi_k \hat{\boldsymbol{x}}_{k-1} + \boldsymbol{K}_k(z_k - \boldsymbol{H}\Phi_k \hat{\boldsymbol{x}}_{k-1})$$

to obtain

$$\begin{bmatrix} x_{T_k} \\ \hat{\dot{x}}_{T_k} \end{bmatrix} = \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{T_{k-1}} \\ \hat{\dot{x}}_{T_{k-1}} \end{bmatrix} + \begin{bmatrix} K_{1_k} \\ K_{2_k} \end{bmatrix} \left[x^*_{T_k} - \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{T_{k-1}} \\ \hat{\dot{x}}_{T_{k-1}} \end{bmatrix} \right]$$

Multiplying out the terms of the preceding matrix equation yields the two scalar equations, which represent the linear polynomial Kalman filter in the downrange channel or

$$\hat{x}_{T_k} = \hat{x}_{T_{k-1}} + T_s \hat{\dot{x}}_{T_{k-1}} + K_{1_k}(x^*_{T_k} - \hat{x}_{T_{k-1}} - T_s \hat{\dot{x}}_{T_{k-1}})$$

$$\hat{\dot{x}}_{T_k} = \hat{\dot{x}}_{T_{k-1}} + K_{2_k}(x^*_{T_k} - \hat{x}_{T_{k-1}} - T_s \hat{\dot{x}}_{T_{k-1}})$$

In the altitude channel there is gravity, and so the equations will be slightly different. The state-space equation describing the real world is now given by

$$\begin{bmatrix} \dot{y}_T \\ \ddot{y}_T \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y_T \\ \dot{y}_T \end{bmatrix} + \begin{bmatrix} 0 \\ -g \end{bmatrix} + \begin{bmatrix} 0 \\ u_s \end{bmatrix}$$

where g is gravity and is assumed to be known in advance. Therefore, the continuous known disturbance or control vector in this equation is given by

$$\boldsymbol{G} = \begin{bmatrix} 0 \\ -g \end{bmatrix}$$

Because the systems dynamics matrix in the altitude channel is identical to the one in the downrange channel, the fundamental matrix is also the same. There-

fore, the discrete control vector can be found from the continuous one from the relationship

$$\boldsymbol{G}_k = \int_0^{T_s} \Phi \boldsymbol{G}\, \mathrm{d}\tau = \int_0^{T_s} \begin{bmatrix} 0 & \tau \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ -g \end{bmatrix} \mathrm{d}\tau = \begin{bmatrix} -0.5T_s^2 g \\ -T_s g \end{bmatrix}$$

The measurement equation is given by

$$y_T^* = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} y_T \\ \dot{y}_T \end{bmatrix} + v_y$$

so that the measurement matrix in the altitude channel is the same as it was in the downrange channel. The measurement noise matrix is also simply a scalar in the altitude channel and is given by

$$\boldsymbol{R}_k = \sigma_{y_T}^2$$

We now have enough information to solve the Riccati equations for the Kalman gains in the altitude channel. Because the fundamental matrix is exact in this example, we can also use the preceding matrices in the Kalman-filtering equation

$$\hat{\boldsymbol{x}}_k = \Phi_k \hat{\boldsymbol{x}}_{k-1} + \boldsymbol{G}_k \boldsymbol{u}_{k-1} + \boldsymbol{K}_k (\boldsymbol{z}_k - \boldsymbol{H}\Phi_k \hat{\boldsymbol{x}}_{k-1} - \boldsymbol{H}\boldsymbol{G}_k \boldsymbol{u}_{k-1})$$

to obtain

$$\begin{bmatrix} y_{T_k} \\ \hat{\dot{y}}_{T_k} \end{bmatrix} = \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix} \begin{bmatrix} y_{T_{k-1}} \\ \hat{\dot{y}}_{T_{k-1}} \end{bmatrix} - \begin{bmatrix} 0.5gT_s^2 \\ gT_s \end{bmatrix} + \begin{bmatrix} K_{1_k} \\ K_{2_k} \end{bmatrix} \left[y_{T_k}^* - \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix} \begin{bmatrix} y_{T_{k-1}} \\ \hat{\dot{y}}_{T_{k-1}} \end{bmatrix} + \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 0.5gT_s^2 \\ gT_s \end{bmatrix} \right]$$

Multiplying out the terms of the preceding matrix equation yields the two scalar equations, which represent the linear polynomial Kalman filter in the downrange channel or

$$\hat{y}_{T_k} = \hat{y}_{T_{k-1}} + T_s \hat{\dot{y}}_{T_{k-1}} - 0.5gT_s^2 + K_{1_k}(x_{T_k}^* - \hat{x}_{T_{k-1}} - T_s \hat{\dot{x}}_{T_{k-1}} + 0.5gT_s^2)$$

$$\hat{\dot{y}}_{T_k} = \hat{\dot{y}}_{T_{k-1}} - gT_s + K_{2_k}(x_{T_k}^* - \hat{x}_{T_{k-1}} - T_s \hat{\dot{x}}_{T_{k-1}} + 0.5gT_s^2)$$

The preceding equations for the two linear decoupled polynomial Kalman filters and associated Riccati equations were programmed and are shown in Listing 9.4, along with a simulation of the real world. Again, we can see that the process-noise matrix is set to zero in this example (i.e., $\Phi_s = 0$). As before, we have initialized the states of the filter close to the true values. The position states are in error by 1000 ft, and the velocity states are in error by 100 ft/s, while the

Listing 9.4 Two decoupled polynomial linear Kalman filters for tracking projectile

```
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,O-Z)
      REAL*8 P(2,2),Q(2,2),M(2,2),PHI(2,2),HMAT(1,2),HT(2,1),PHIT(2,2)
      REAL*8 RMAT(1,1),IDN(2,2),PHIP(2,2),PHIPPHIT(2,2),HM(1,2)
      REAL*8 HMHT(1,1),HMHTR(1,1),HMHTRINV(1,1),MHT(2,1),K(2,1)
      REAL*8 KH(2,2),IKH(2,2)
      REAL*8 RMATY(1,1),PY(2,2),PHIPY(2,2),PHIPPHITY(2,2),MY(2,2)
      REAL*8 HMY(1,2),HMHTY(1,1),HMHTRY(1,1),HMHTRINVY(1,1)
      REAL*8 MHTY(2,1),KY(2,1),KHY(2,2),IKHY(2,2)
      INTEGER ORDER
      TS=1.
      ORDER=2
      PHIS=0.
      SIGTH=.01
      SIGR=100.
      VT=3000.
      GAMDEG=45.
      G=32.2
      XT=0.
      YT=0.
      XTD=VT*COS(GAMDEG/57.3)
      YTD=VT*SIN(GAMDEG/57.3)
      XR=100000.
      YR=0.
      OPEN(1,STATUS='UNKNOWN',FILE='DATFIL')
      OPEN(2,STATUS='UNKNOWN',FILE='COVFIL')
      T=0.
      S=0.
      H=.001
      DO 14 I=1,ORDER
      DO 14 J=1,ORDER
      PHI(I,J)=0.
      P(I,J)=0.
      Q(I,J)=0.
      IDN(I,J)=0.
14    CONTINUE
      PHI(1,1)=1.
      PHI(1,2)=TS
      PHI(2,2)=1.
      HMAT(1,1)=1.
      HMAT(1,2)=0.
      CALL MATTRN(PHI,ORDER,ORDER,PHIT)
      CALL MATTRN(HMAT,1,ORDER,HT)
      IDN(1,1)=1.
      IDN(2,2)=1.
      Q(1,1)=PHIS*TS*TS*TS/3.
```

(*continued*)

Listing 9.4 *(Continued)*

```
      Q(1,2)=PHIS*TS*TS/2.
      Q(2,1)=Q(1,2)
      Q(2,2)=PHIS*TS
      P(1,1)=1000.**2
      P(2,2)=100.**2
      PY(1,1)=1000.**2
      PY(2,2)=100.**2
      XTH=XT+1000.
      XTDH=XTD-100.
      YTH=YT-1000.
      YTDH=YTD-100.
      WHILE(YT>=0.)
            XTOLD=XT
            XTDOLD=XTD
            YTOLD=YT
            YTDOLD=YTD
            XTDD=0.
            YTDD=-G
            XT=XT+H*XTD
            XTD=XTD+H*XTDD
            YT=YT+H*YTD
            YTD=YTD+H*YTDD
            T=T+H
            XTDD=0.
            YTDD=-G
            XT=.5*(XTOLD+XT+H*XTD)
            XTD=.5*(XTDOLD+XTD+H*XTDD)
            YT=.5*(YTOLD+YT+H*YTD)
            YTD=.5*(YTDOLD+YTD+H*YTDD)
            S=S+H
            IF(S>=(TS-.00001))THEN
                  S=0.
                  THETH=ATAN2((YTH-YR),(XTH-XR))
                  RTH=SQRT((XTH-XR)**2+(YTH-YR)**2)
                  RMAT(1,1)=(COS(THETH)*SIGR)**2+(RTH*SIN(THETH)
     1               *SIGTH)**2
                  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)
                  CALL MATADD(HMHT,1,1,RMAT,HMHTR)
                    HMHTRINV(1,1)=1./HMHTR(1,1)
                  CALL MATMUL(M,ORDER,ORDER,HT,ORDER,1,MHT)
                  CALL MATMUL(MHT,ORDER,1,HMHTRINV,1,1,K)
                  CALL MATMUL(K,ORDER,1,HMAT,1,ORDER,KH)
```

(continued)

Listing 9.4 (*Continued*)

```
      CALL MATSUB(IDN,ORDER,ORDER,KH,IKH)
      CALL MATMUL(IKH,ORDER,ORDER,M,ORDER,
     ORDER,P)
      CALL GAUSS(THETNOISE,SIGTH)
      CALL GAUSS(RTNOISE,SIGR)
      THET=ATAN2((YT-YR),(XT-XR))
      RT=SQRT((XT-XR)**2+(YT-YR)**2)
      THETMEAS=THET+THETNOISE
      RTMEAS=RT+RTNOISE
      XTMEAS=RTMEAS*COS(THETMEAS)+XR
      RES1=XTMEAS-XTH-TS*XTDH
      XTH=XTH+TS*XTDH+K(1,1)*RES1
      XTDH=XTDH+K(2,1)*RES1
      RMATY(1,1)=(SIN(THETH)*SIGR)**2+
     1  (RTH*COS(THETH)*SIGTH)**2
      CALL MATMUL(PHI,ORDER,ORDER,PY,ORDER,
     1  ORDER,PHIPY)
      CALL MATMUL(PHIPY,ORDER,ORDER,PHIT,ORDER,
     1  ORDER,PHIPPHITY)
      CALL MATADD(PHIPPHITY,ORDER,ORDER,Q,MY)
      CALL MATMUL(HMAT,1,ORDER,MY,ORDER,ORDER,
     HMY)
      CALL MATMUL(HMY,1,ORDER,HT,ORDER,1,HMHTY)
      CALL MATADD(HMHTY,1,1,RMATY,HMHTRY)
     HMHTRINVY(1,1)=1./HMHTRY(1,1)
      CALL MATMUL(MY,ORDER,ORDER,HT,ORDER,1,
     MHTY)
      CALL MATMUL(MHTY,ORDER,1,HMHTRINVY,1,
     1,KY)
      CALL MATMUL(KY,ORDER,1,HMAT,1,ORDER,KHY)
      CALL MATSUB(IDN,ORDER,ORDER,KHY,IKHY)
      CALL MATMUL(IKHY,ORDER,ORDER,MY,ORDER,
     ORDER,PY)
      YTMEAS=RTMEAS*SIN(THETMEAS)+YR
      RES2=YTMEAS-YTH-TS*YTDH+.5*TS*TS*G
      YTH=YTH+TS*YTDH-.5*TS*TS*G+KY(1,1)*RES2
      YTDH=YTDH-TS*G+KY(2,1)*RES2
      ERRX=XT-XTH
      SP11=SQRT(P(1,1))
      ERRXD=XTD-XTDH
      SP22=SQRT(P(2,2))
      ERRY=YT-YTH
      SP11Y=SQRT(PY(1,1))
      ERRYD=YTD-YTDH
      SP22Y=SQRT(PY(2,2))
      WRITE(9,*)T,XT,XTH,XTD,XTDH,YT,YTH,YTD,YTDH
      WRITE(1,*)T,XT,XTH,XTD,XTDH,YT,YTH,YTD,YTDH
      WRITE(2,*)T,ERRX,SP11,-SP11,ERRXD,SP22,-SP22,
     1  ERRY,SP11Y,-SP11Y,ERRYD,SP22Y,-SP22Y
      ENDIF
```

(*continued*)

Listing 9.4 (*Continued*)

```
      END DO
      PAUSE
      CLOSE(1)
      CLOSE(2)
      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
```


initial covariance matrix reflects those errors. The simulation running time of Listing 9.4 is much faster than that of the extended Cartesian Kalman filter of Listing 9.1 because there are effectively fewer equations with two two-state filters than there are for those with one four-state filter.

The nominal case was run, and the resultant errors in the estimates for position and velocity in the downrange and altitude direction appear in Figs. 9.23–9.26. We can see from the four figures that the linear Kalman filters appear to be working correctly because the single-run simulation errors in the estimates appear to lie between the theoretical predictions of the associated covariance matrices approximately 68% of the time. However, if we compare these results to the errors in the estimates of the Cartesian extended Kalman filter shown in Figs. 9.8–9.12, we can see that these new results appear to be slightly worse than before. In other words, the errors in the estimates of the two two-state linear

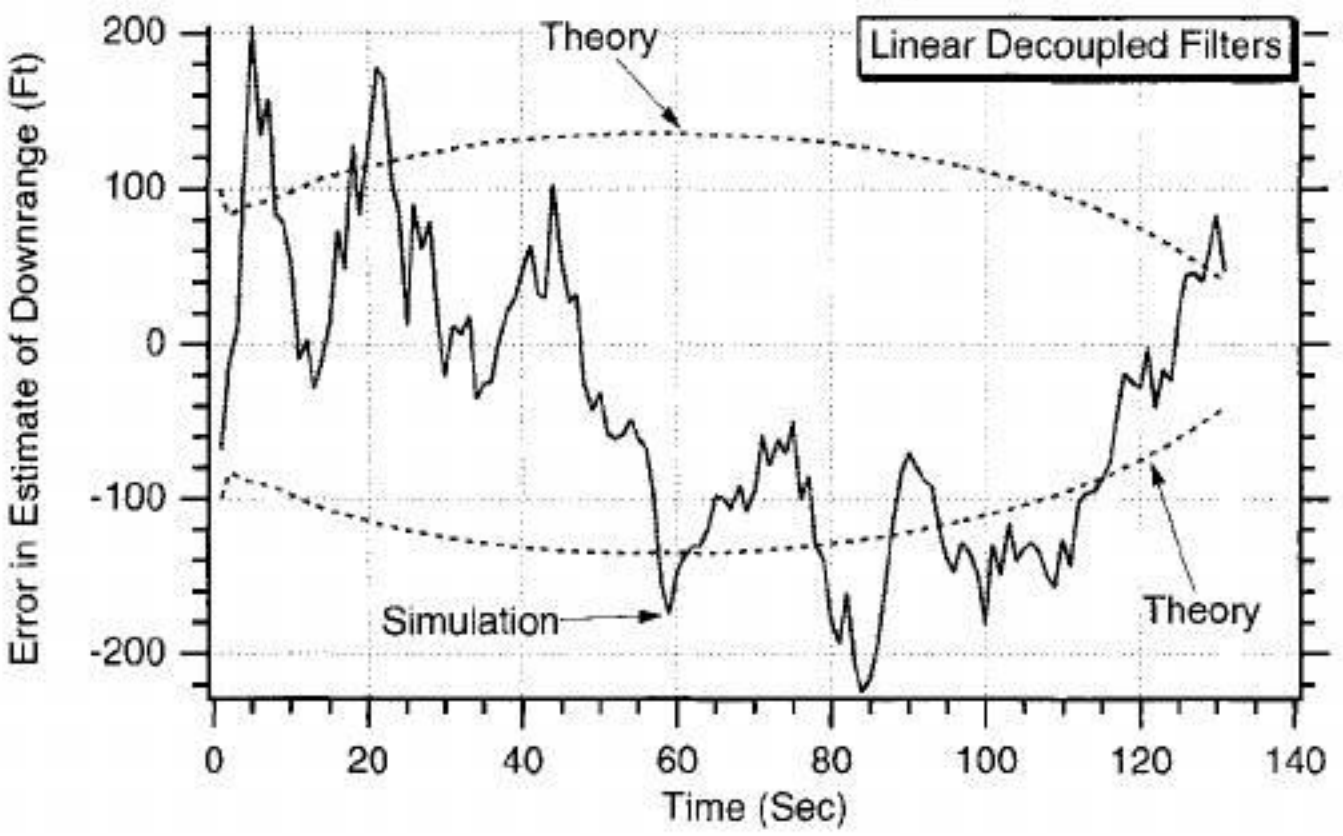

Fig. 9.23 Linear decoupled Kalman filter downrange error in the estimate of position is larger than that of extended Kalman filter.

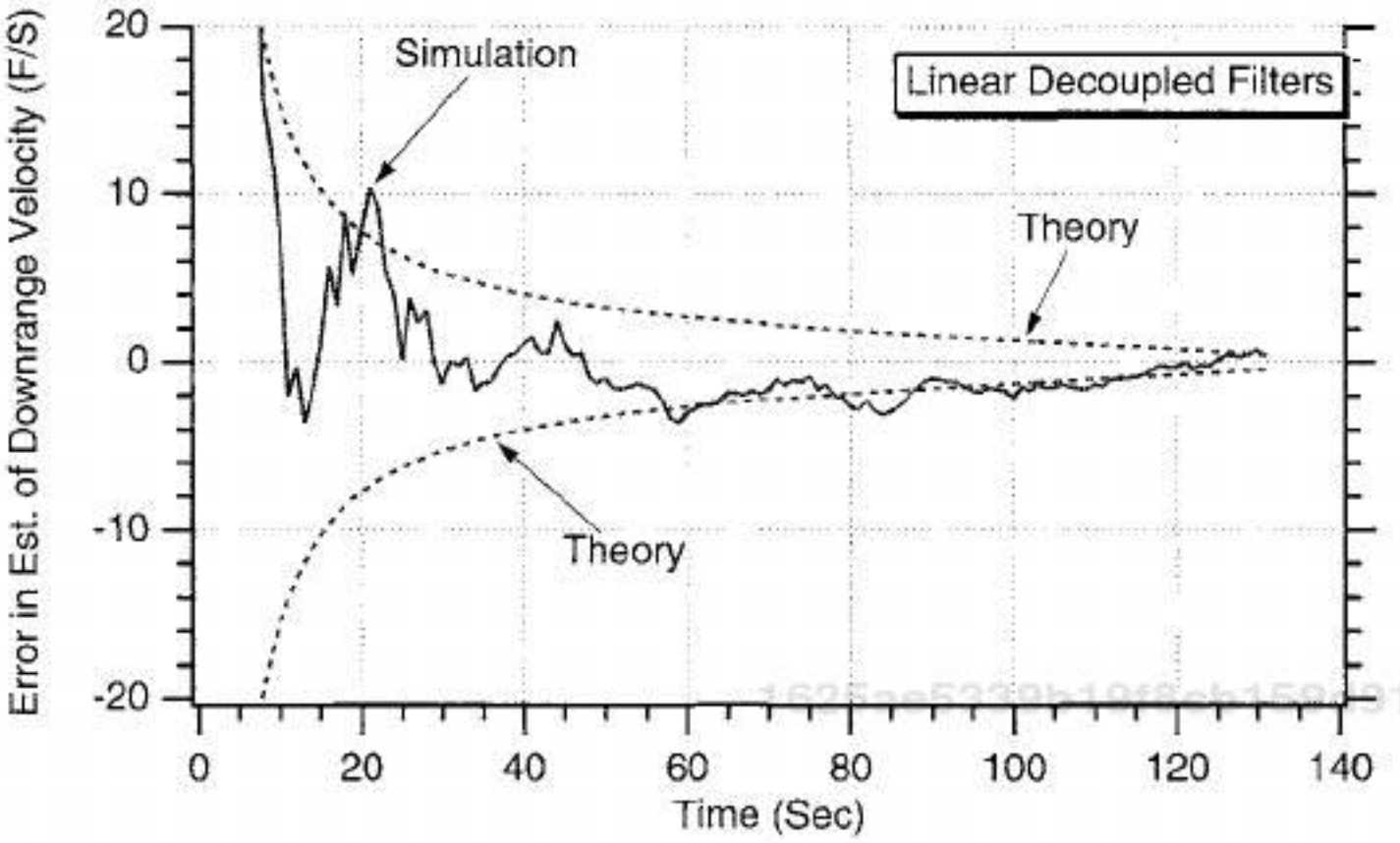

Fig. 9.24 Linear decoupled Kalman filter downrange error in the estimate of velocity is larger than that of extended Kalman filter.

decoupled polynomial Kalman filters appear to be larger than those of the four-state Cartesian extended Kalman filter.

Using Linear Coupled Polynomial Kalman Filters

In the preceding section we pretended that the noise in the altitude and downrange channels was not correlated, which allowed us to decouple the linear Kalman filters. Let us now remove that restriction in order to see what happens.

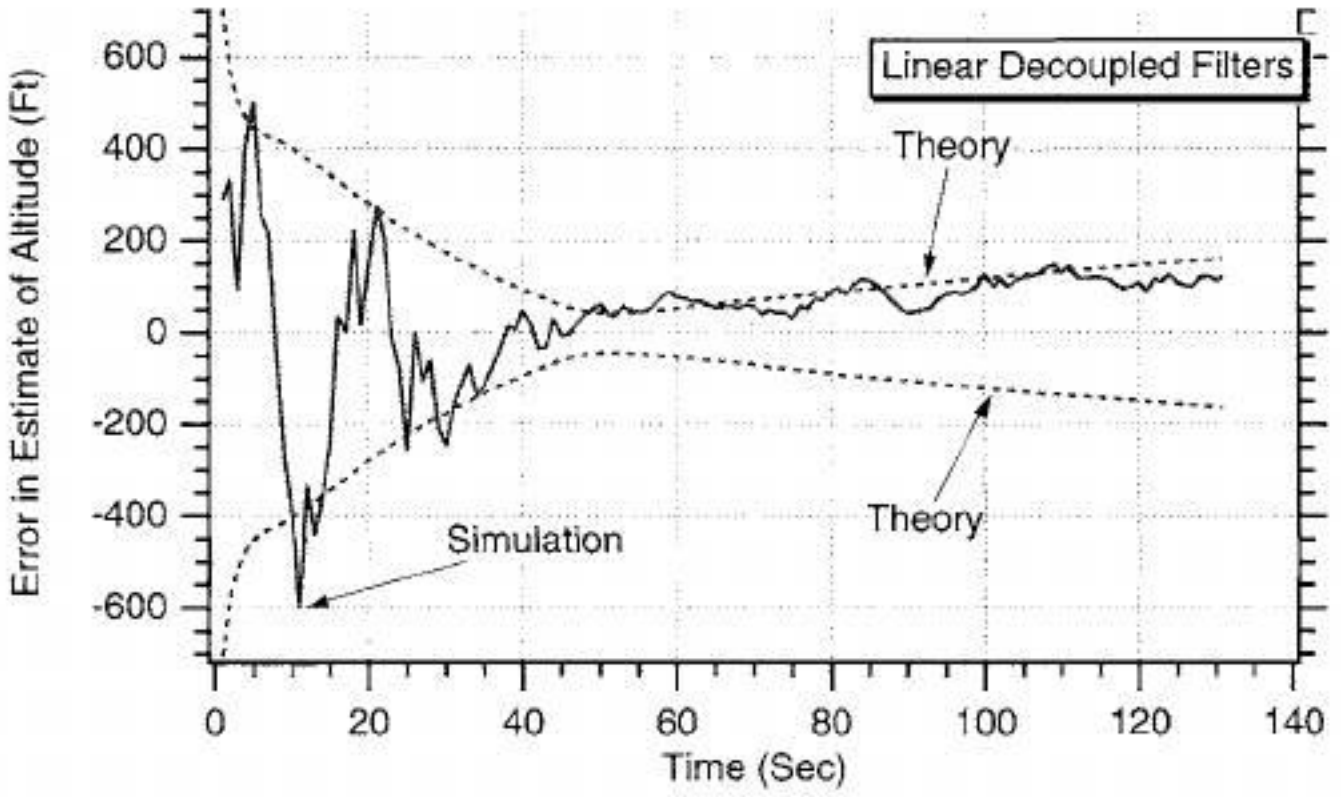

Fig. 9.25 Linear decoupled Kalman filter altitude error in the estimate of position is larger than that of extended Kalman filter.

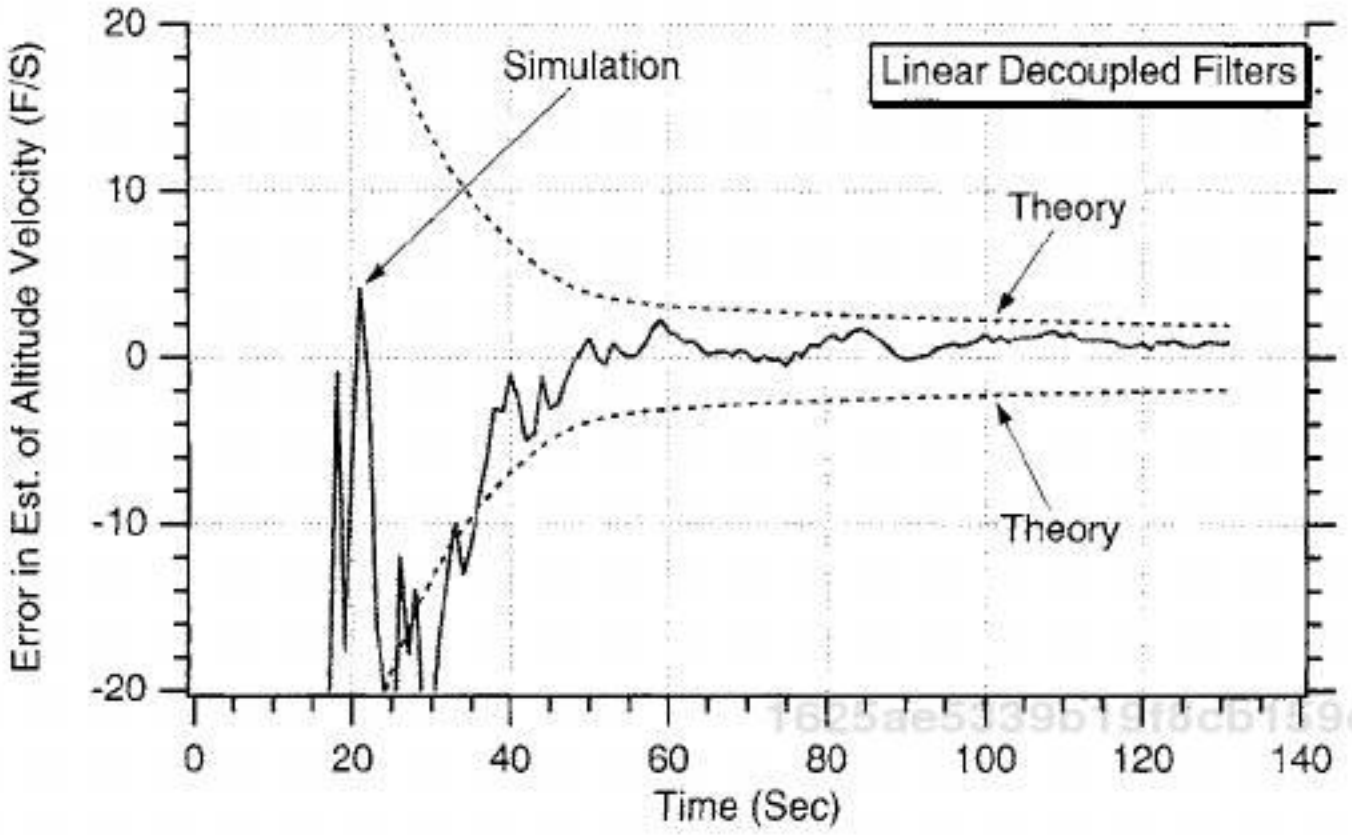

Fig. 9.26 Linear decoupled Kalman filter altitude error in the estimate of velocity is larger than that of extended Kalman filter.

Recall that we have already shown that downrange and altitude can be expressed in terms of range and angle according to

$$x_T = r\cos\theta + x_R$$
$$y_T = r\sin\theta + y_R$$

Using calculus, we can find the total differential of the preceding two equations as

$$\Delta x_T = \frac{\partial x_T}{\partial r}\Delta r + \frac{\partial x_T}{\partial \theta}\Delta\theta = \cos\theta\Delta r - r\sin\theta\Delta\theta$$

$$\Delta y_T = \frac{\partial y_T}{\partial r}\Delta r + \frac{\partial y_T}{\partial \theta}\Delta\theta = \sin\theta\Delta r + r\cos\theta\Delta\theta$$

Before, we squared each of the preceding equations to obtain

$$\Delta x_T^2 = \cos^2\theta\Delta r^2 - 2r\sin\theta\cos\theta\Delta r\Delta\theta + r^2\sin^2\theta\Delta\theta^2$$
$$\Delta y_T^2 = \sin^2\theta\Delta r^2 + 2r\sin\theta\cos\theta\Delta r\Delta\theta + r^2\cos^2\theta\Delta\theta^2$$

but now we also can find

$$\Delta x_T\Delta y_T = \sin\theta\cos\theta\Delta r^2 + r\sin^2\theta\Delta r\Delta\theta + r\cos^2\theta\Delta r\Delta\theta - r^2\sin\theta\cos\theta\Delta\theta^2$$

To find the variance of the pseudonoise, we can take expectations of both sides of the equations, assuming that the actual range and angle noise are not correlated [i.e., $E(\Delta r \Delta\theta) = 0$). Therefore, we can say that

$$E(\Delta x_T^2) = \cos^2\theta E(\Delta r^2) + r^2 \sin^2\theta E(\Delta\theta^2)$$

$$E(\Delta y_T^2) = \sin^2\theta E(\Delta r^2) + r^2 \cos^2\theta E(\Delta\theta^2)$$

$$E(\Delta x_T \Delta y_T) = \sin\theta\cos\theta E(\Delta r^2) - r^2 \sin\theta\cos\theta E(\Delta\theta^2)$$

Because

$$\sigma_{x_T}^2 = E(\Delta x_T^2)$$

$$\sigma_{y_T}^2 = E(\Delta y_T^2)$$

$$\sigma_{x_T y_T}^2 = E(\Delta x_T \Delta y_T)$$

$$\sigma_r^2 = E(r^2)$$

$$\sigma_\theta^2 = E(\Delta\theta^2)$$

we can say that

$$\sigma_{x_T}^2 = \cos^2\theta\theta\sigma_r^2 + r^2 \sin^2\theta\sigma_\theta^2$$

$$\sigma_{y_T}^2 = \sin^2\theta\theta\sigma_r^2 + r^2 \cos^2\theta\sigma_\theta^2$$

$$\sigma_{x_T y_T}^2 = \sin\theta\cos\theta\sigma_r^2 - r^2 \sin\theta\cos\theta\sigma_\theta^2$$

Therefore, unlike the preceding section, where we had a downrange and altitude scalar noise matrix, we now have a fully coupled 2×2 noise matrix given by

$$\boldsymbol{R}_k = \begin{bmatrix} \cos^2\theta\sigma_r^2 + r^2 \sin^2\theta\sigma_\theta^2 & \sin\theta\cos\theta\sigma_r^2 - r^2 \sin\theta\cos\theta\sigma_\theta^2 \\ \sin\theta\cos\theta\sigma_r^2 - r^2 \sin\theta\cos\theta\sigma_\theta^2 & \sin^2\theta\sigma_r^2 + r^2 \cos^2\theta\sigma_\theta^2 \end{bmatrix}$$

Because we are no longer decoupling the channels, the states upon which the new filter will be based are given by

$$\boldsymbol{x} = \begin{bmatrix} x_T \\ \dot{x}_T \\ y_T \\ \dot{y}_T \end{bmatrix}$$

and the pseudomeasurements are now linearly related to the states according to

$$\begin{bmatrix} x_T^* \\ y_T^* \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_T \\ \dot{x}_T \\ y_T \\ \dot{y}_T \end{bmatrix} + \begin{bmatrix} v_x \\ v_y \end{bmatrix}$$

From the preceding equation we can see that the measurement matrix is given by

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

Therefore, when the preceding Cartesian states are chosen, we have already shown that the state-space differential equation describing projectile motion is also linear and given by

$$\begin{bmatrix} \dot{x}_T \\ \ddot{x}_T \\ \dot{y}_T \\ \ddot{y}_T \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_T \\ \dot{x}_T \\ y_T \\ \dot{y}_T \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ -g \end{bmatrix} + \begin{bmatrix} 0 \\ u_s \\ 0 \\ u_s \end{bmatrix}$$

Again, notice that in the preceding equation gravity g is not a state that has to be estimated but is assumed to be known in advance. We also have added white process noise u_s to the acceleration portion of the equations as protection for effects that may not considered by the Kalman filter. The preceding equation is equivalent to the model used in deriving the extended Cartesian Kalman filter. The only difference in this formulation is that the actual measurements have been simplified to pseudomeasurements for purposes of making the filter linear.

From the preceding state-space equation we can see that systems dynamics matrix is given by

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

and we have already shown in this chapter that the fundamental matrix for this system of equations is

$$\Phi(t) = \begin{bmatrix} 0 & t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & t \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

$$\Phi_k = \begin{bmatrix} 0 & T_s & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T_s \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Because the continuous process-noise matrix is given by

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

we can say that, for this example, the continuous process-noise matrix is

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

where Φ_s is the spectral density of the white noise sources assumed to be on the downrange and altitude accelerations acting on the projectile. The discrete process-noise matrix can be derived from the continuous process-noise matrix according to

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

and we have already shown that the discrete process-noise matrix turns out to be

$$\boldsymbol{Q}_k = \begin{bmatrix} \frac{T_s^3\Phi_s}{3} & \frac{T_s^2\Phi_s}{2} & 0 & 0 \\ \frac{T_s^2\Phi_s}{2} & T_s\Phi_s & 0 & 0 \\ 0 & 0 & \frac{T_s^3\Phi_s}{3} & \frac{T_s^2\Phi_s}{2} \\ 0 & 0 & \frac{T_s^2\Phi_s}{2} & T_s\Phi_s \end{bmatrix}$$

From the state-space equation we can see that the continuous control vector in this equation is given by

$$\boldsymbol{G} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -g \end{bmatrix}$$

Therefore, the discrete control vector can be found from the continuous one from the relationship

$$\boldsymbol{G}_k = \int_0^{T_s} \Phi \boldsymbol{G}\, \mathrm{d}\tau = \int_0^{T_s} \begin{bmatrix} 0 & \tau & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \tau \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ -g \end{bmatrix} \mathrm{d}\tau = \begin{bmatrix} 0 \\ 0 \\ -0.5gT_s^2 \\ -gT_s \end{bmatrix}$$

We now have defined all of the matrices required to solve the Riccati equations. Because the fundamental matrix is exact in this example, we also can use the preceding matrices in the Kalman-filtering equation

$$\hat{\boldsymbol{x}}_k = \Phi_k \hat{\boldsymbol{x}}_{k-1} + \boldsymbol{G}_k \boldsymbol{u}_{k-1} + \boldsymbol{K}_k (\boldsymbol{z}_k - \boldsymbol{H}\Phi_k \hat{\boldsymbol{x}}_{k-1} - \boldsymbol{H}\boldsymbol{G}_k \boldsymbol{u}_{k-1})$$

to obtain

$$\begin{bmatrix} \hat{x}_{T_k} \\ \hat{\dot{x}}_{T_k} \\ \hat{y}_{T_k} \\ \hat{\dot{y}}_{T_k} \end{bmatrix} = \begin{bmatrix} 0 & T_s & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T_s \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{x}_{T_{k-1}} \\ \hat{\dot{x}}_{T_{k-1}} \\ \hat{y}_{T_{k-1}} \\ \hat{\dot{y}}_{T_{k-1}} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -0.5gT_s^2 \\ -gT_s \end{bmatrix}$$

$$+ \begin{bmatrix} K_{11_k} & K_{12_k} \\ K_{21_k} & K_{22_k} \\ K_{31_k} & K_{32_k} \\ K_{41_k} & K_{42_k} \end{bmatrix} \left[\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & T_s & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T_s \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{x}_{T-1} \\ \hat{\dot{x}}_{T_{k-1}} \\ \hat{y}_{T_{k-1}} \\ \hat{\dot{y}}_{T_{k-1}} \end{bmatrix} \right.$$

$$\left. - \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ -0.5gT_s^2 \\ -gT_s \end{bmatrix} \right]$$

Multiplying out the terms yields the following scalar equations:

$$\mathrm{Res}_1 = x^*_{T_k} - \hat{x}_{T_{k-1}} - T_s \hat{\dot{x}}_{T_{k-1}}$$

$$\mathrm{Res}_2 = y^*_{T_k} - \hat{y}_{T_{k-1}} - T_s \hat{\dot{y}}_{T_{k-1}} + 0.5gT_s^2$$

$$\hat{x}_{T_k} = \hat{x}_{T_{k-1}} + T_s \hat{\dot{x}}_{T_{k-1}} + K_{11_k}\mathrm{Res}_1 + K_{12_k}\mathrm{Res}_2$$

$$\hat{\dot{x}}_{T_k} = \hat{\dot{x}}_{T_{k-1}} + K_{21_k}\mathrm{Res}_1 + K_{22_k}\mathrm{Res}_2$$

$$\hat{y}_{T_k} = \hat{y}_{T_{k-1}} + T_s \hat{\dot{y}}_{T_{k-1}} - 0.5gT_s^2 + K_{31_k}\mathrm{Res}_1 + K_{32_k}\mathrm{Res}_2$$

$$\hat{\dot{y}}_{T_k} = \hat{\dot{y}}_{T_{k-1}} - gT_s + K_{41_k}\mathrm{Res}_1 + K_{42_k}\mathrm{Res}_2$$

The preceding equations for the four-state linear polynomial Kalman filters and associated Riccati equations were programmed and are shown in Listing 9.5, along with a simulation of the real world. Again, we can see that the process-noise matrix is set to zero in this example (i.e., $\Phi_s = 0$). As before, we have initialized the states of the filter close to the true values. The position states are in

Listing 9.5 Linear coupled four-state polynomial Kalman filter

```
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,O-Z)
        REAL*8 P(4,4),Q(4,4),M(4,4),PHI(4,4),HMAT(2,4),HT(2,4),PHIT(4,4)
        REAL*8 RMAT(2,2),IDN(4,4),PHIP(4,4),PHIPPHIT(4,4),HM(2,4)
        REAL*8 HMHT(2,2),HMHTR(2,2),HMHTRINV(2,2),MHT(4,2),K(4,2)
        REAL*8 KH(4,4),IKH(4,4)
        INTEGER ORDER
        TS=1.
        ORDER=4
        PHIS=0.
        SIGTH=.01
        SIGR=100.
        VT=3000.
        GAMDEG=45.
        G=32.2
        XT=0.
        YT=0.
        XTD=VT*COS(GAMDEG/57.3)
        YTD=VT*SIN(GAMDEG/57.3)
        XR=100000.
        YR=0.
        OPEN(1,STATUS='UNKNOWN',FILE='DATFIL')
        OPEN(2,STATUS='UNKNOWN',FILE='COVFIL')
        T=0.
        S=0.
        H=.001
        DO 14 I=1,ORDER
        DO 14 J=1,ORDER
        PHI(I,J)=0.
        P(I,J)=0.
        Q(I,J)=0.
        IDN(I,J)=0.
14      CONTINUE
        PHI(1,1)=1.
        PHI(1,2)=TS
        PHI(2,2)=1.
        PHI(3,3)=1.
```

(continued)

Listing 9.5 (*Continued*)

```
PHI(3,4)=TS
PHI(4,4)=1.
HMAT(1,1)=1.
HMAT(1,2)=0.
HMAT(1,3)=0.
HMAT(1,4)=0.
HMAT(2,1)=0.
HMAT(2,2)=0.
HMAT(2,3)=1.
HMAT(2,4)=0.
CALL  MATTRN(PHI,ORDER,ORDER,PHIT)
CALL  MATTRN(HMAT,2,ORDER,HT)
IDN(1,1)=1.
IDN(2,2)=1.
IDN(3,3)=1.
IDN(4,4)=1.
Q(1,1)=PHIS*TS*TS*TS/3.
Q(1,2)=PHIS*TS*TS/2.
Q(2,1)=Q(1,2)
Q(2,2)=PHIS*TS
Q(3,3)=PHIS*TS*TS*TS/3.
Q(3,4)=PHIS*TS*TS/2.
Q(4,3)=Q(3,4)
Q(4,4)=PHIS*TS
P(1,1)=1000.**2
P(2,2)=100.**2
P(3,3)=1000.**2
P(4,4)=100.**2
XTH=XT+1000.
XTDH=XTD-100.
YTH=YT-1000.
YTDH=YTD+100.
WHILE(YT>=0.)
        XTOLD=XT
        XTDOLD=XTD
        YTOLD=YT
        YTDOLD=YTD
        XTDD=0.
        YTDD=-G
        XT=XT+H*XTD
        XTD=XTD+H*XTDD
        YT=YT+H*YTD
        YTD=YTD+H*YTDD
        T=T+H
        XTDD=0.
        YTDD=-G
        XT=.5*(XTOLD+XT+H*XTD)
        XTD=.5*(XTDOLD+XTD+H*XTDD)
        YT=.5*(YTOLD+YT+H*YTD)
```

(*continued*)

Listing 9.5 (*Continued*)

```
                YTD=.5*(YTDOLD+YTD+H*YTDD)
                S=S+H
                IF(S>=(TS-.00001))THEN
                      S=0.
                      THETH=ATAN2((YTH-YR),(XTH-XR))
                      RTH=SQRT((XTH-XR)**2+(YTH-YR)**2)
                      RMAT(1,1)=(COS(THETH)*SIGR)**2+(RTH*SIN
     1                  (THETH)*SIGTH)**2
                      RMAT(2,2)=(SIN(THETH)*SIGR)**2+(RTH*COS
     1                  (THETH)*SIGTH)**2
                      RMAT(1,2)=SIN(THETH)*COS(THETH)*(SIGR**2-
     1                  (RTH*SIGTH)**2)
                      RMAT(2,1)=RMAT(1,2)
                      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,2,ORDER,M,ORDER,
                        ORDER,HM)
                      CALL MATMUL(HM,2,ORDER,HT,ORDER,2,HMHT)
                      CALL MATADD(HMHT,2,2,RMAT,HMHTR)
                        DET=HMHTR(1,1)*HMHTR(2,2)-
                        HMHTR(1,2)*HMHTR(2,1)
                      HMHTRINV(1,1)=HMHTR(2,2)/DET
                      HMHTRINV(1,2)=-HMHTR(1,2)/DET
                      HMHTRINV(2,1)=-HMHTR(2,1)/DET
                      HMHTRINV(2,2)=HMHTR(1,1)/DET
                      CALL MATMUL(M,ORDER,ORDER,HT,ORDER,
                        2,MHT)
                      CALL MATMUL(MHT,ORDER,2,HMHTRINV,2,2,K)
                      CALL MATMUL(K,ORDER,2,HMAT,2,ORDER,KH)
                      CALL MATSUB(IDN,ORDER,ORDER,KH,IKH)
                      CALL MATMUL(IKH,ORDER,ORDER,M,ORDER,
                        ORDER,P)
                      CALL GAUSS(THETNOISE,SIGTH)
                      CALL GAUSS(RTNOISE,SIGR)
                      THET=ATAN2((YT-YR),(XT-XR))
                      RT=SQRT((XT-XR)**2+(YT-YR)**2)
                      THETMEAS=THET+THETNOISE
                      RTMEAS=RT+RTNOISE
                      XTMEAS=RTMEAS*COS(THETMEAS)+XR
                      YTMEAS=RTMEAS*SIN(THETMEAS)+YR
                      RES1=XTMEAS-XTH-TS*XTDH
                      RES2=YTMEAS-YTH-TS*YTDH+.5*TS*TS*G
                      XTH=XTH+TS*XTDH+K(1,1)*RES1+K(1,2)*RES2
                      XTDH=XTDH+K(2,1)*RES1+K(2,2)*RES2
                      YTH=YTH+TS*YTDH-.5*TS*TS*G+K(3,1)*RES1+
     1                    K(3,2)*RES2
                      YTDH=YTDH-TS*G+K(4,1)*RES1+K(4,2)*RES2
```

(*continued*)

Listing 9.5 (*Continued*)

```
                ERRX=XT-XTH
                SP11=SQRT(P(1,1))
                ERRXD=XTD-XTDH
                SP22=SQRT(P(2,2))
                ERRY=YT-YTH
                SP33=SQRT(P(3,3))
                ERRYD=YTD-YTDH
                SP44=SQRT(P(4,4))
                WRITE(9,*)T,XT,XTH,XTD,XTDH,YT,YTH,YTD,YTDH
                WRITE(1,*)T,XT,XTH,XTD,XTDH,YT,YTH,YTD,YTDH
                WRITE(2,*)T,ERRX,SP11,-SP11,ERRXD,SP22,-SP22,
1                 ERRY,SP33,-SP33,ERRYD,SP44,-SP44
            ENDIF
            END DO
            PAUSE
            CLOSE(1)
            CLOSE(2)
            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
```

error by 1000 ft, and the velocity states are in error by 100 ft/s, while the initial covariance matrix reflects those errors.

The nominal case of Listing 9.5 was run in which there was no process noise. Figures 9.27–9.30 display the errors in the estimates of the downrange and altitude states. By comparing these figures with those of the Cartesian extended Kalman filter (see Figs. 9.8–9.11), we can see that the results are now virtually identical. In other words, in the cannon-launched projectile example there is no degradation in performance by using a linear coupled filter with pseudomeasurements as opposed to the extended Cartesian Kalman filter with the actual measurements. However, there was some degradation in performance when we used decoupled linear filters, as was done in the preceding section.

Robustness Comparison of Extended and Linear Coupled Kalman Filters

So far we have seen that both the extended and linear coupled Kalman filters appear to have approximately the same performance. However, these results were based on work in which the initialization errors in the filter states were relatively modest. Let us first see how the extended Kalman filter reacts to larger initialization errors.

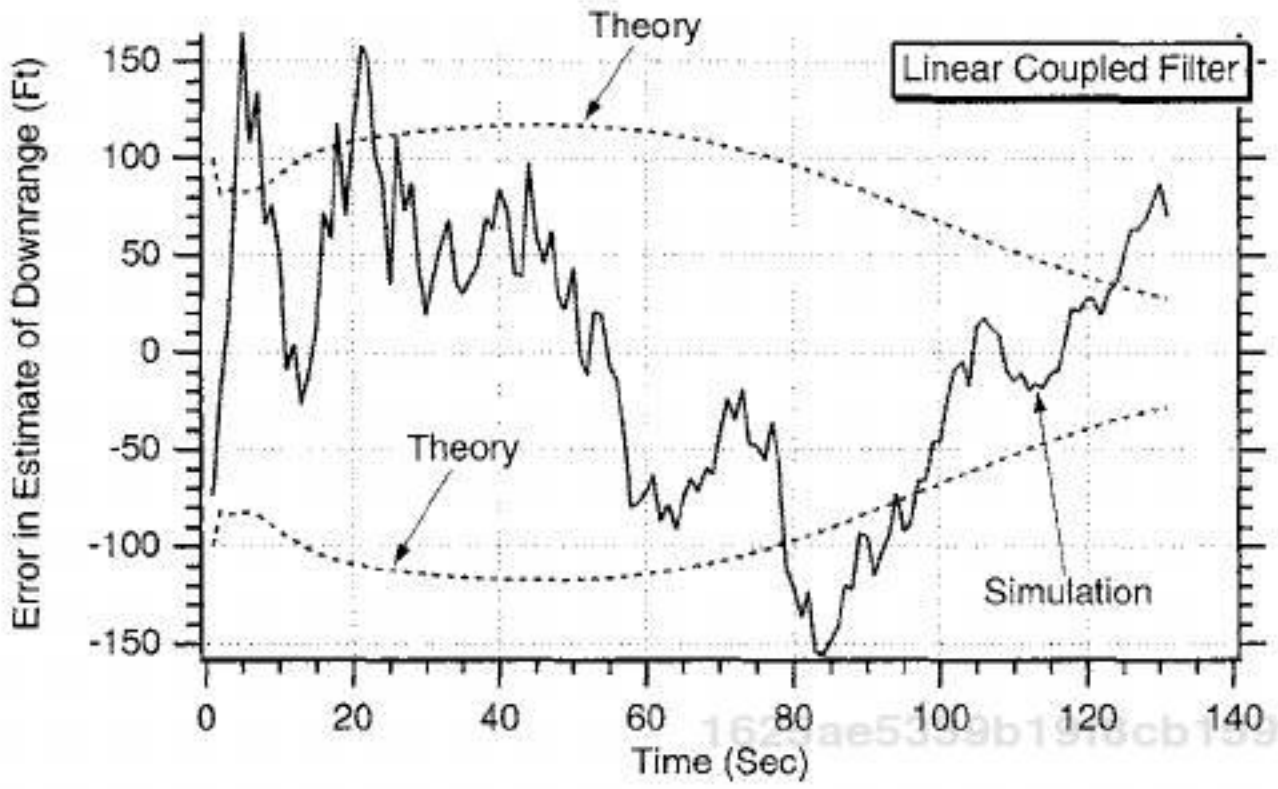

Fig. 9.27 Error in estimate of downrange is the same for the linear coupled and extended Kalman filters.

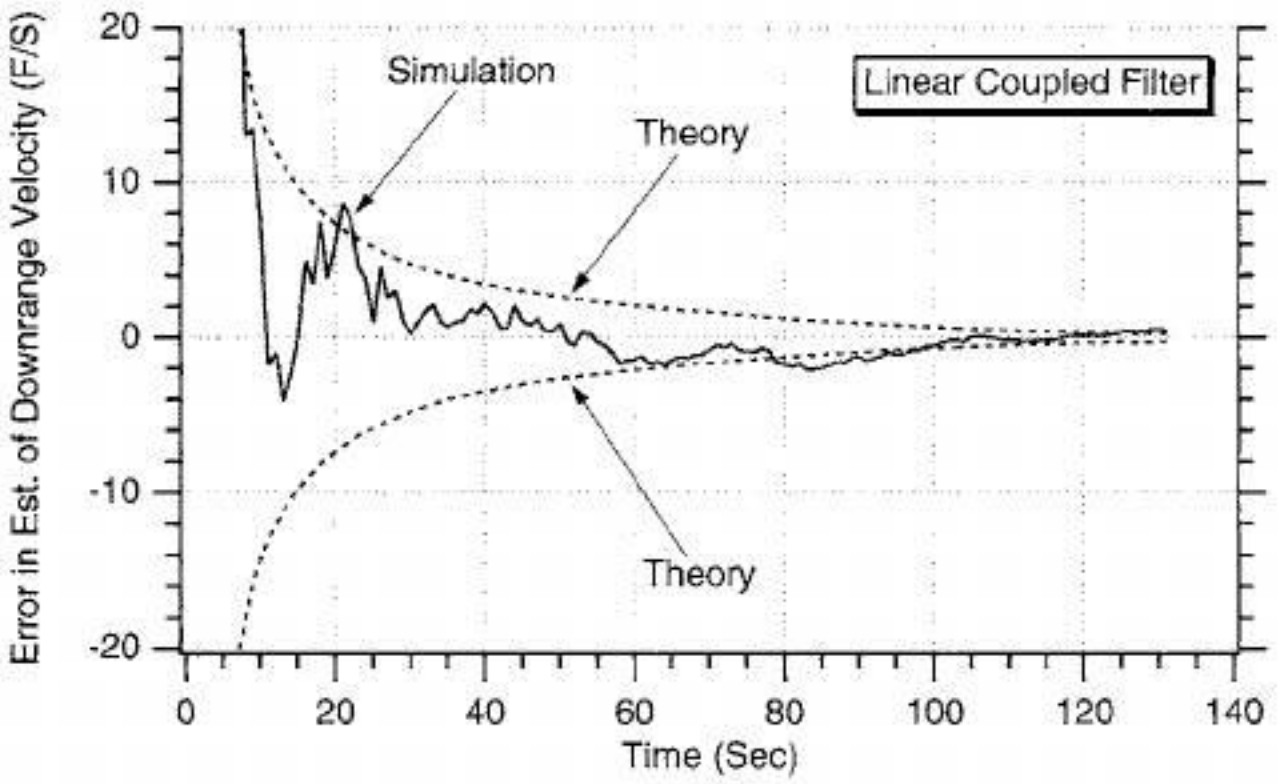

Fig. 9.28 Error in estimate of downrange velocity is the same for the linear coupled and extended Kalman filters.

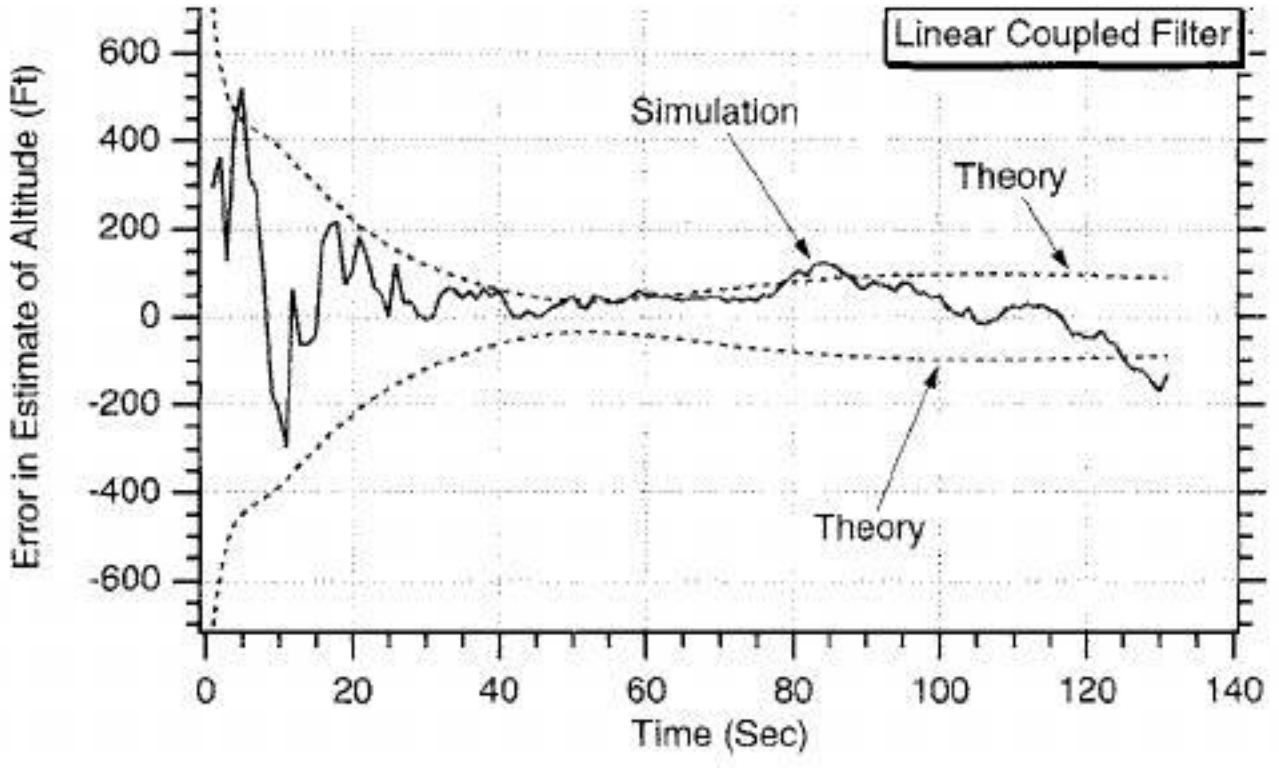

Fig. 9.29 Error in estimate of altitude is the same for the linear coupled and extended Kalman filters.

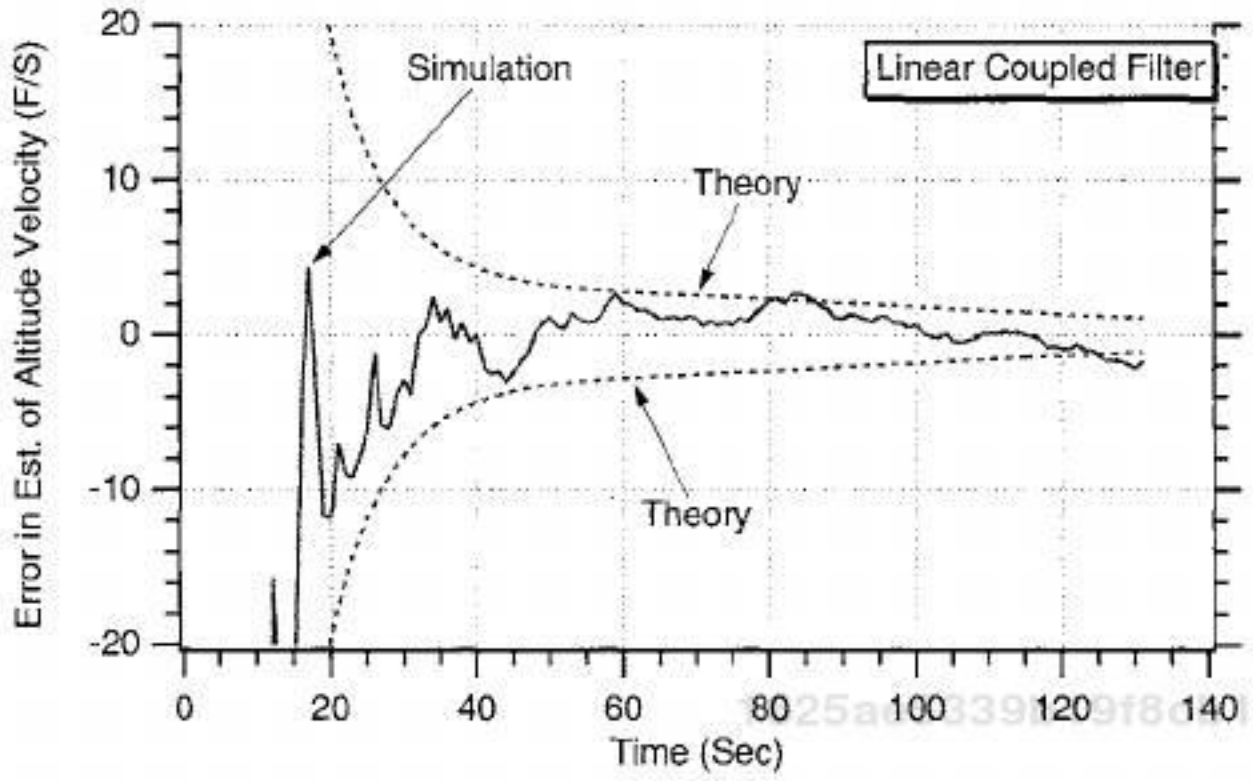

Fig. 9.30 Error in estimate of altitude velocity is the same for the linear coupled and extended Kalman filters.

In all of the experiments of this chapter, the initial state estimates of the filter were the true states plus an error of 1000 ft on position and 100 ft/s on velocity, as indicated here:

$$\begin{bmatrix} \hat{x}_T(0) \\ \hat{\dot{x}}_T(0) \\ \hat{y}_T(0) \\ \hat{\dot{y}}_T(0) \end{bmatrix} = \begin{bmatrix} x_T(0) \\ \dot{x}_T(0) \\ y_T(0) \\ \dot{y}_T(0) \end{bmatrix} + \begin{bmatrix} 1000 \\ -100 \\ -1000 \\ 100 \end{bmatrix}$$

The initial covariance matrix of both the linear and extended Kalman filters reflected the initialization errors by having each diagonal element be the square of the appropriate initialization error or

$$\boldsymbol{P}_0 = \begin{bmatrix} 1000^2 & 0 & 0 & 0 \\ 0 & 100^2 & 0 & 0 \\ 0 & 0 & 1000^2 & 0 \\ 0 & 0 & 0 & 100^2 \end{bmatrix}$$

To see how sensitive the extended Kalman filter is to initialization errors, let us first double those errors so that the initial estimates of the extended Kalman filter and its covariance matrix are now given by

$$\begin{bmatrix} \hat{x}_T(0) \\ \hat{\dot{x}}_T(0) \\ \hat{y}_T(0) \\ \hat{\dot{y}}_T(0) \end{bmatrix} = \begin{bmatrix} x_T(0) \\ \dot{x}_T(0) \\ y_T(0) \\ \dot{y}_T(0) \end{bmatrix} + \begin{bmatrix} 2000 \\ -200 \\ -2000 \\ 200 \end{bmatrix}$$

$$\boldsymbol{P}_0 = \begin{bmatrix} 2000^2 & 0 & 0 & 0 \\ 0 & 200^2 & 0 & 0 \\ 0 & 0 & 2000^2 & 0 \\ 0 & 0 & 0 & 200^2 \end{bmatrix}$$

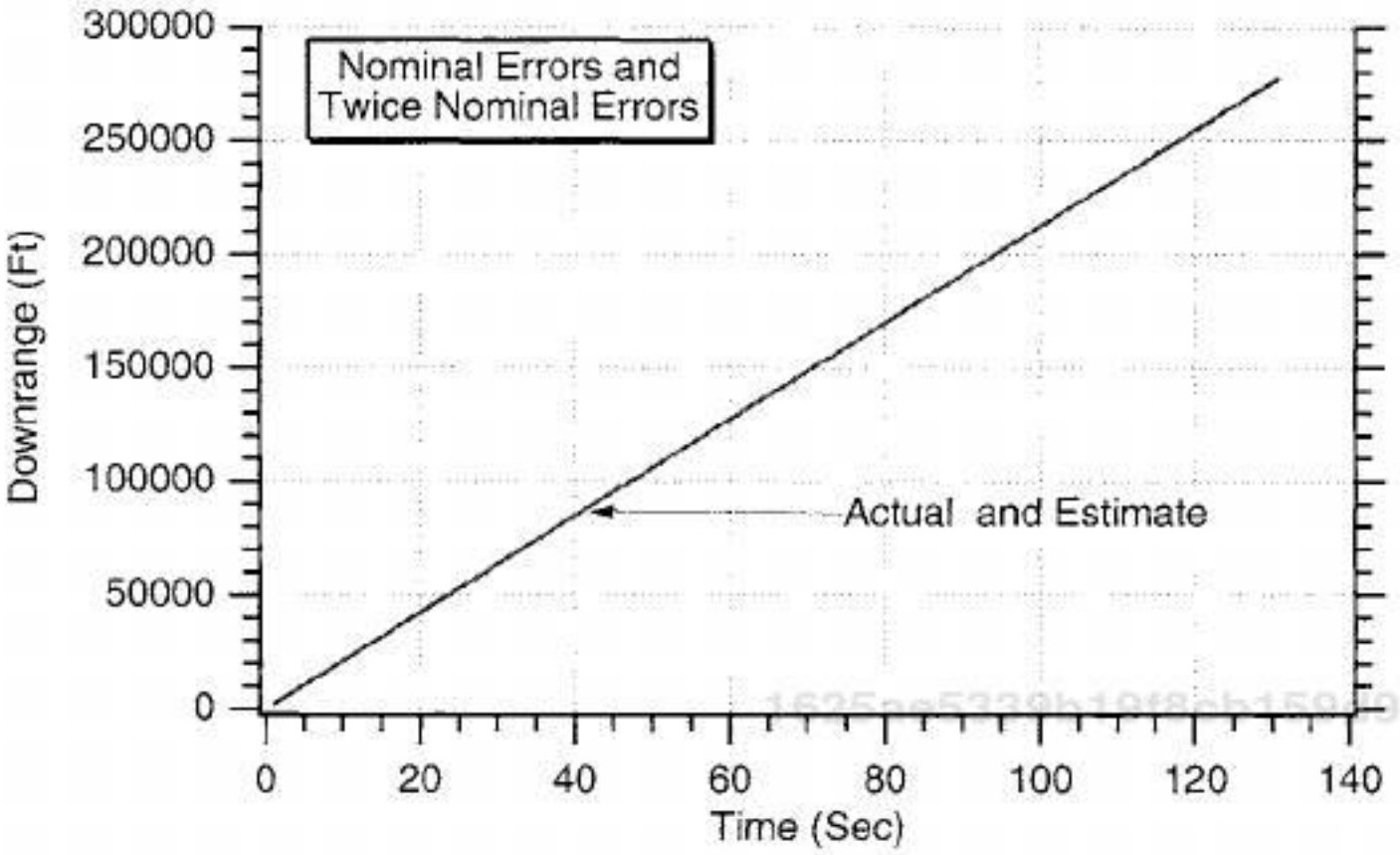

Fig. 9.31 Extended Kalman filter appears to yield good estimates even when initialization errors are twice as large as nominal.

Listing 9.1, which represents the extended Cartesian Kalman filter, was rerun with the preceding initialization errors. We can see from Fig. 9.31 that the estimate of the downrange location of the projectile appears to be independent of initialization error because the estimates do not change when the initialization errors are doubled. In addition, the estimates of the projectile location appear to be very close to the actual projectile location.

Next, the initialization errors were made five times larger than the nominal values so that the initial state estimates and covariance matrix are now given by

$$\begin{bmatrix} \hat{x}_T(0) \\ \hat{\dot{x}}_T(0) \\ \hat{y}_T(0) \\ \hat{\dot{y}}_T(0) \end{bmatrix} = \begin{bmatrix} x_T(0) \\ \dot{x}_T(0) \\ y_T(0) \\ \dot{y}_T(0) \end{bmatrix} + \begin{bmatrix} 5000 \\ -500 \\ -5000 \\ 500 \end{bmatrix}$$

$$\boldsymbol{P}_0 = \begin{bmatrix} 5000^2 & 0 & 0 & 0 \\ 0 & 500^2 & 0 & 0 \\ 0 & 0 & 5000^2 & 0 \\ 0 & 0 & 0 & 500^2 \end{bmatrix}$$

Listing 9.1 was rerun with the new initialization errors. We can see from Fig. 9.32 that with the larger initialization errors the extended Kalman filter's estimate of projectile downrange degrades severely. In addition, we can see that for the first 10 s there is a filter transient caused by the initialization errors. However, after the transient settles out the filter estimate of the projectile downrange position severely lags the actual downrange location.

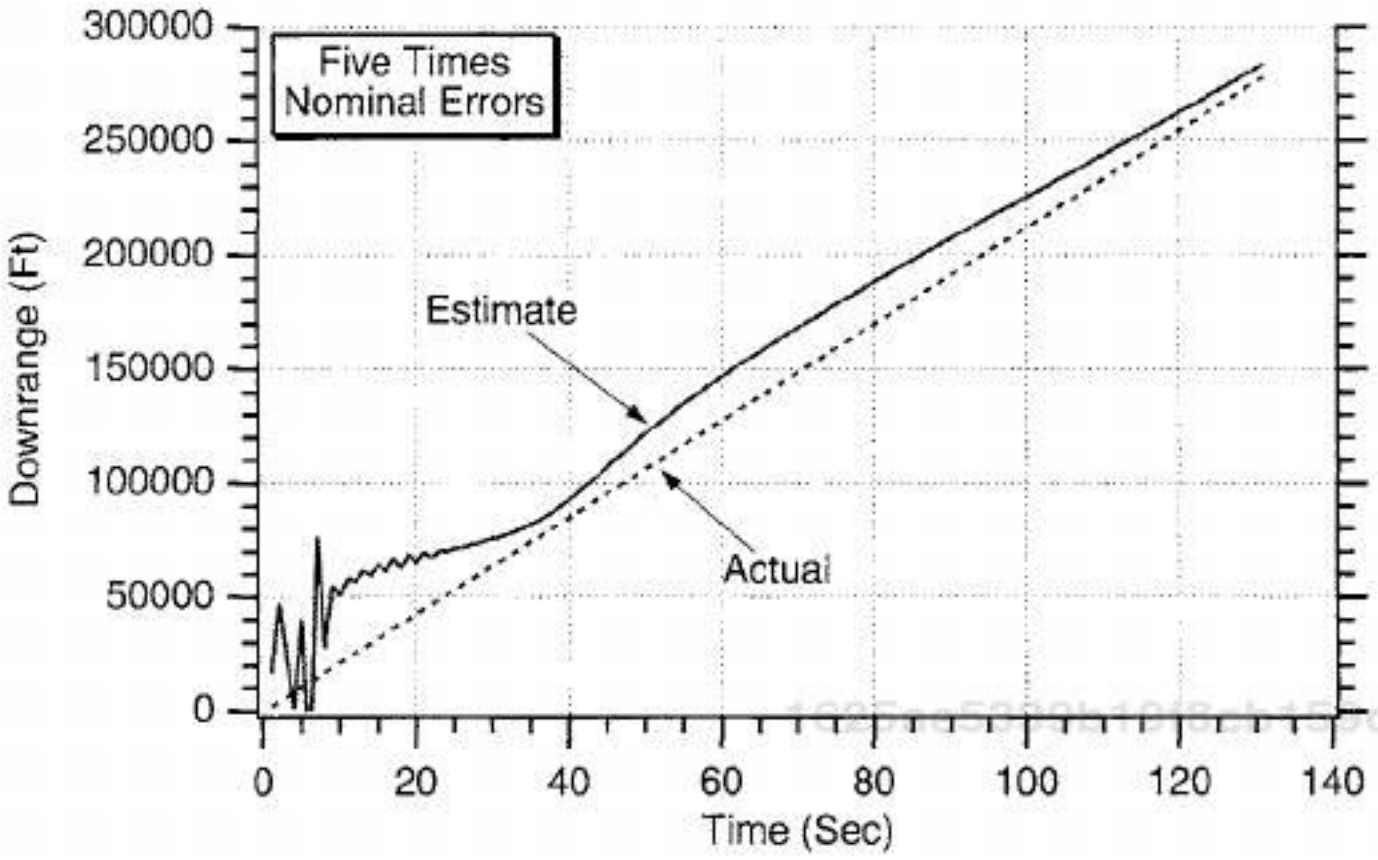

Fig. 9.32 Estimates from extended Kalman filter degrade severely when initialization errors are five time larger than nominal.

Next, the extended Kalman filter's initialization errors were made 10 times larger than the nominal values. The new initial state estimates and initial covariance matrix are now given by

$$\begin{bmatrix} \hat{x}_T(0) \\ \hat{\dot{x}}_T(0) \\ \hat{y}_T(0) \\ \hat{\dot{y}}_T(0) \end{bmatrix} = \begin{bmatrix} x_T(0) \\ \dot{x}_T(0) \\ y_T(0) \\ \dot{y}_T(0) \end{bmatrix} + \begin{bmatrix} 10000 \\ -1000 \\ -10000 \\ 1000 \end{bmatrix}$$

$$\boldsymbol{P}_0 = \begin{bmatrix} 10{,}000^2 & 0 & 0 & 0 \\ 0 & 1{,}000^2 & 0 & 0 \\ 0 & 0 & 10{,}000^2 & 0 \\ 0 & 0 & 0 & 1{,}000^2 \end{bmatrix}$$

Listing 9.1 was rerun with the preceding initialization conditions. We can see from Fig. 9.33 that when the initialization errors of the filter estimate and the initial covariance matrix are 10 times the nominal value the estimate of projectile, downrange is worthless. We can see that the filter estimate of the downrange location of the projectile is monotonically decreasing, whereas the actual location is monotonically increasing.

All of the initialization experiments conducted so far in this section had zero process noise (i.e., $\Phi_s = 0$). We know from Chapter 6 that the addition of process noise will widen the bandwidth of the Kalman filter, and hopefully this also will improve the extended Kalman filter's transient response. The case in which the initialization errors for the extended Kalman were 10 times larger than nominal and the filter's estimates were worthless was reexamined when process-noise was present. Figure 9.34 shows that the addition of process noise (i.e., $\Phi_s = 1$) now

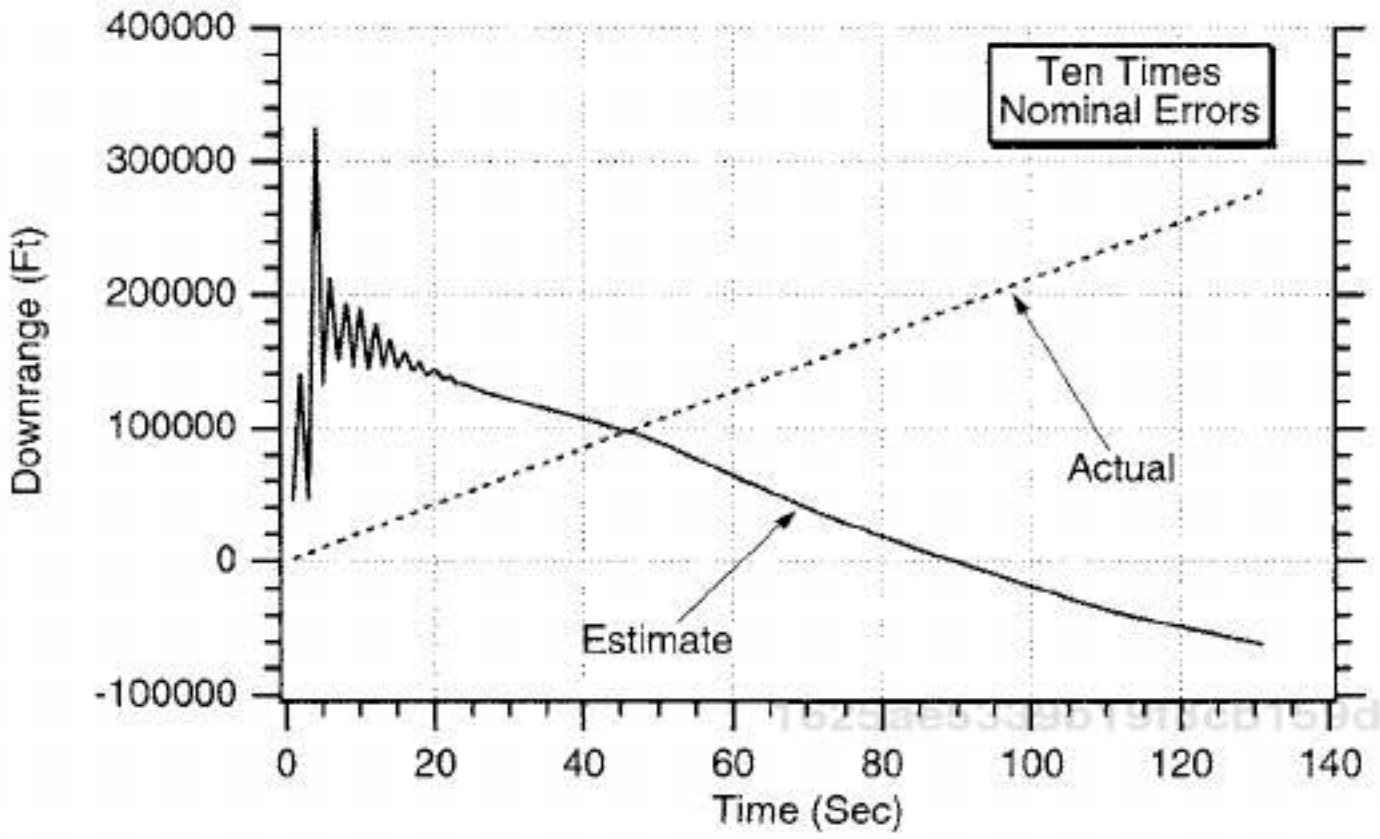

Fig. 9.33 Estimates from extended Kalman filter are worthless when initialization errors are 10 times larger than nominal.

enables the extended Kalman filter to estimate the downrange location of the projectile. Figures 9.34 and 9.35 show that increasing the process noise enables the filter to eliminate the estimation transient error associated with the large initialization errors more rapidly. This is not surprising because we already know from Chapter 6 that increasing the process noise will increase the bandwidth of the Kalman filter and thus improve its transient response.

For more precise work we must also see how the actual errors in the estimates compare to the covariance matrix predictions for the errors in the estimates. Figure 9.36 examines the error in the estimate of the projectile's downrange location. We can see that the extended Kalman filter with process noise (i.e., $\Phi_s = 1000$) now appears to be working correctly because, after an initial transient

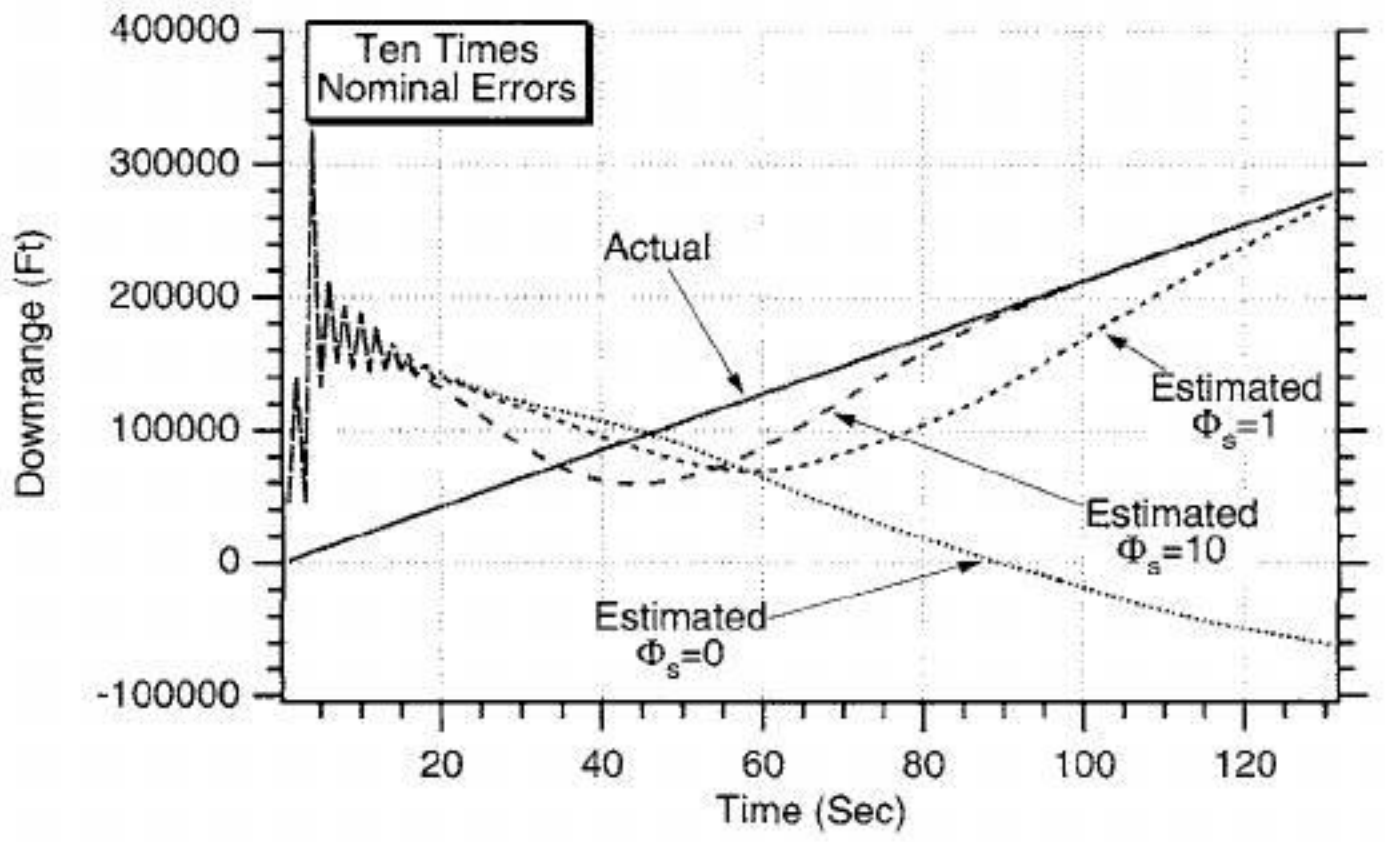

Fig. 9.34 Addition of process noise enables extended Kalman filter to better estimate downrange in presence of large initialization errors.

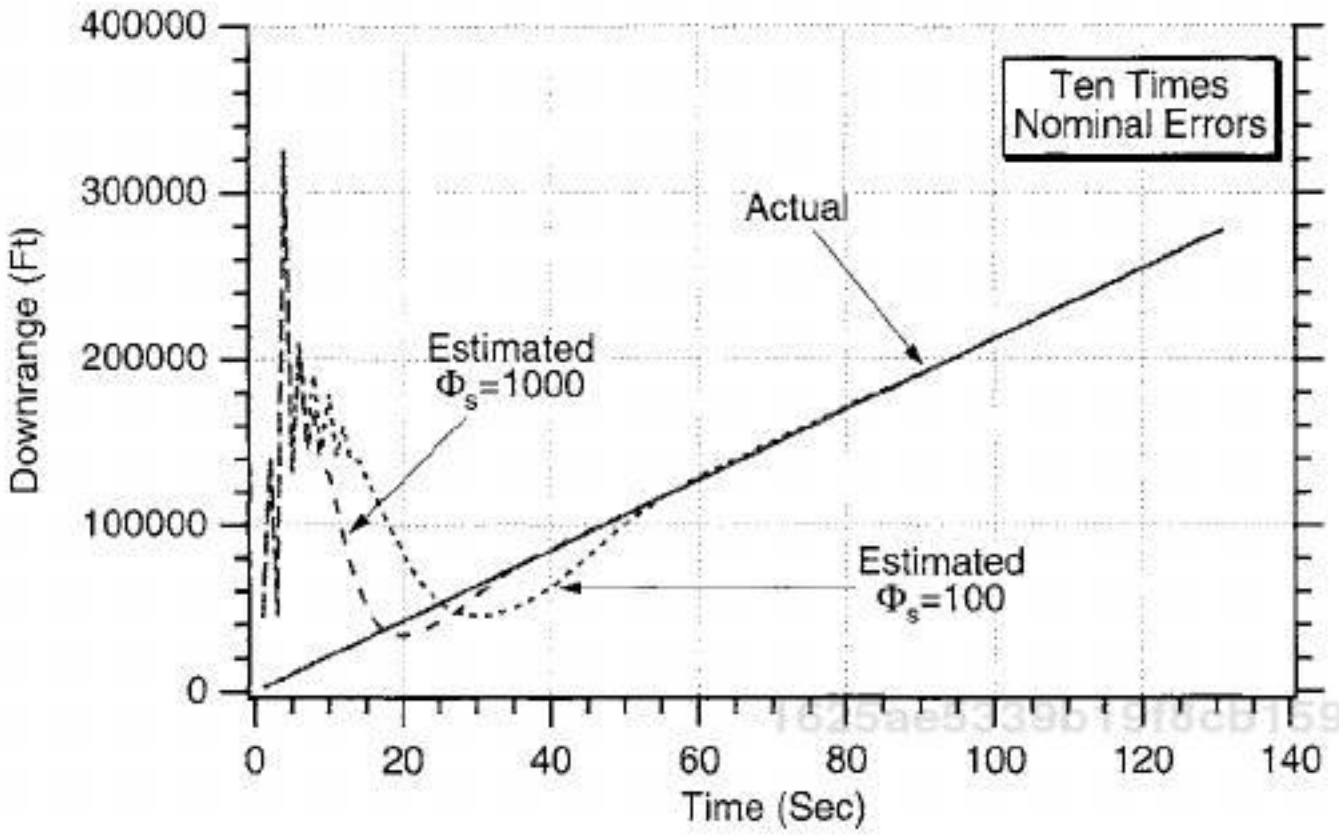

Fig. 9.35 Making process noise larger further improves extended Kalman filter's ability to estimate downrange in presence of large initialization errors.

period, the error in the estimate of projectile downrange position agrees with the results of the covariance matrix. In other words, the simulated error in the estimate appears to lie within the theoretical bounds (i.e., square root of P_{11}) approximately 68% of the time.

We now ask ourselves how the linear coupled polynomial Kalman filter would have performed under similar circumstances. Listing 9.5, which we have already shown has the linear coupled polynomial Kalman filter, was rerun for the case in which the initialization errors were 10 times their nominal value. The linear coupled polynomial Kalman filter was run *without* process-noise. We can see from Fig. 9.37 that under these adverse conditions the linear filter does not appear to have a problem in estimating the downrange location of the projectile. Again,

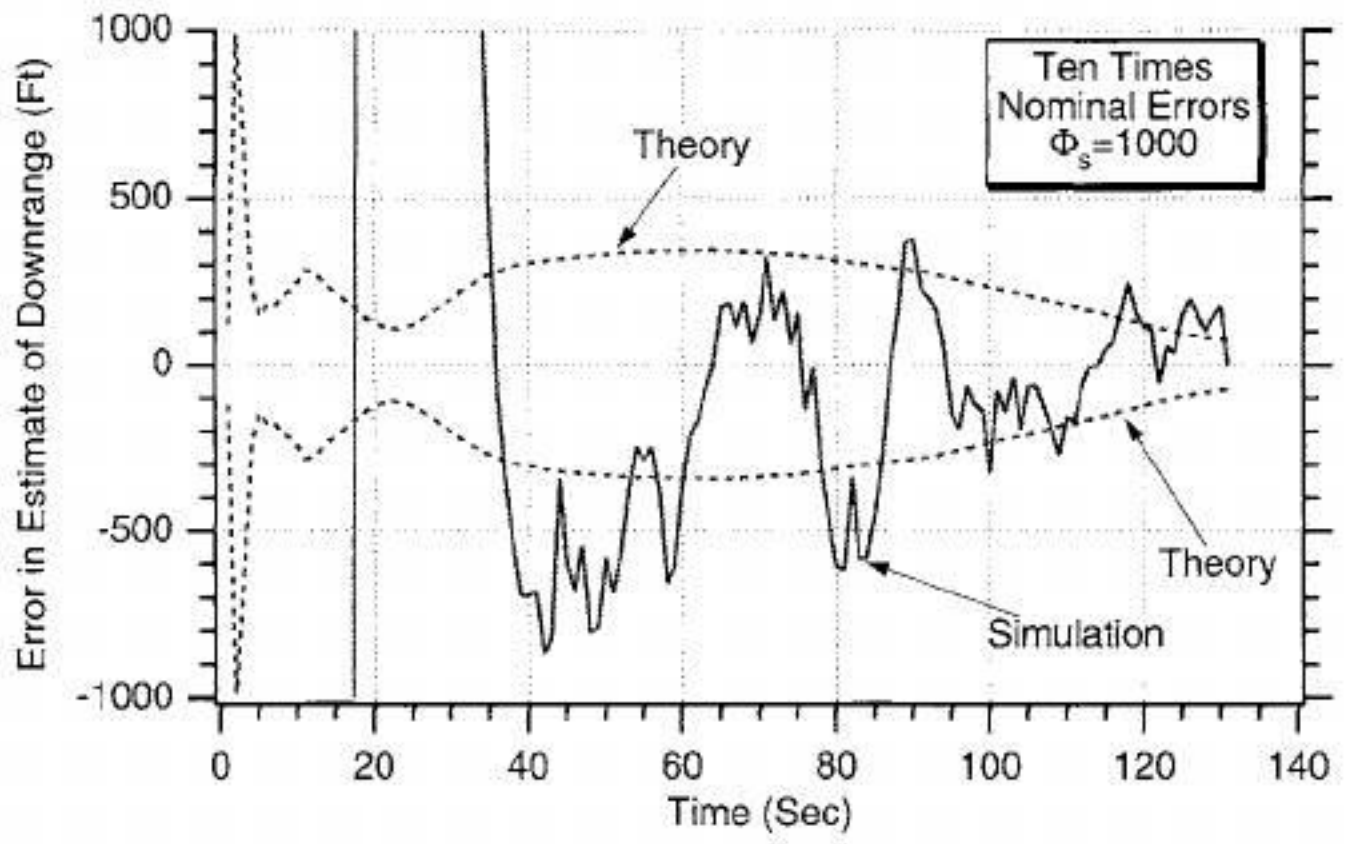

Fig. 9.36 After an initial transient period simulation results agree with covariance matrix predictions.

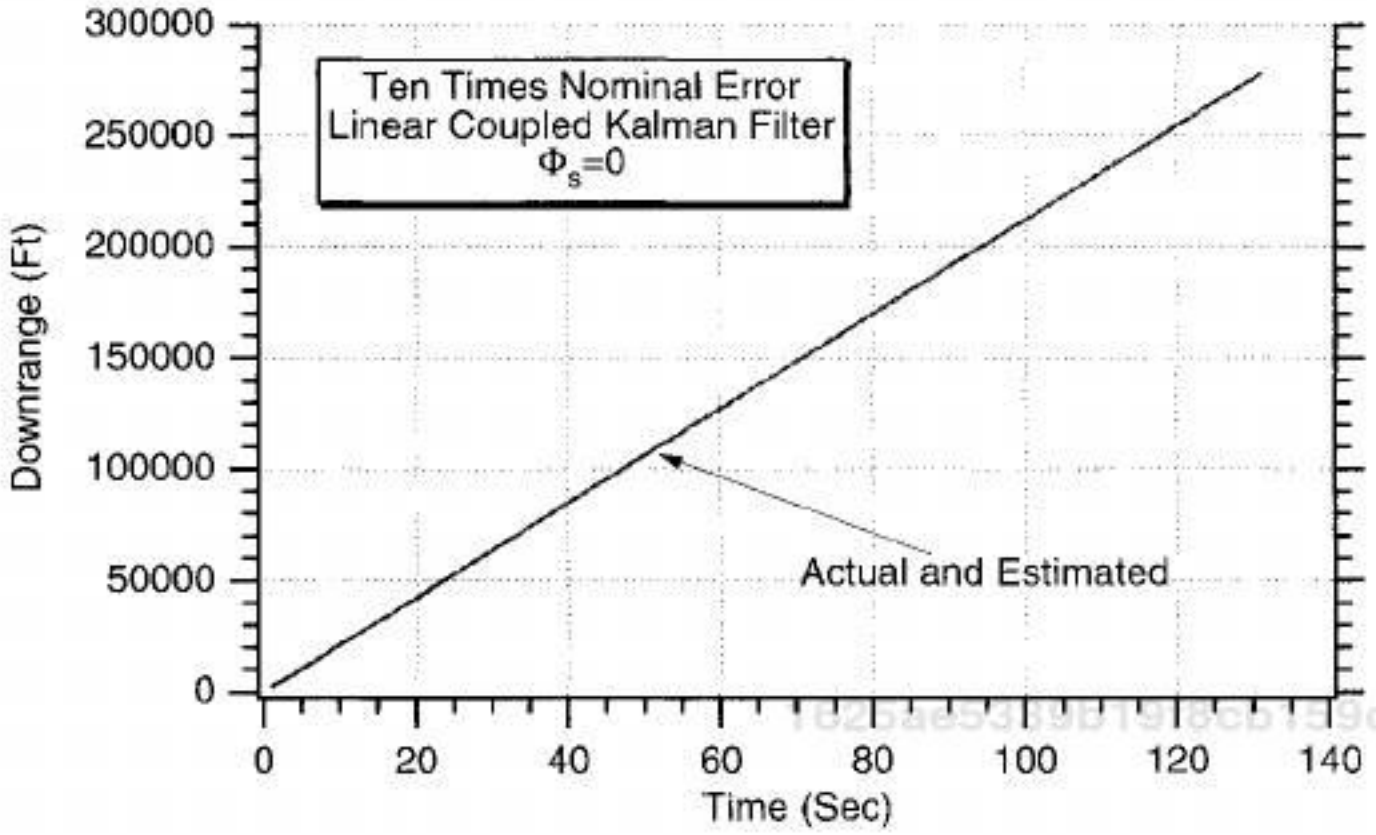

Fig. 9.37 Linear coupled polynomial Kalman filter does not appear to be sensitive to large initialization errors.

we can examine filter performance more critically by examining the error in the estimate from simulation and comparing it to the covariance matrix prediction. We can see from Fig. 9.38 that the single-run simulation results of the error in the estimate of the projectile downrange location appear to lie within the theoretical bounds approximately 68% of the time. Therefore, the linear coupled polynomial Kalman filter *without* process-noise appears to be working in the presence of large initialization errors. If we compare these results to the case in which the initialization errors were their nominal values (see Fig. 9.8), we can see that the performance of the linear filter does not appear to depend on the size of the initialization error.

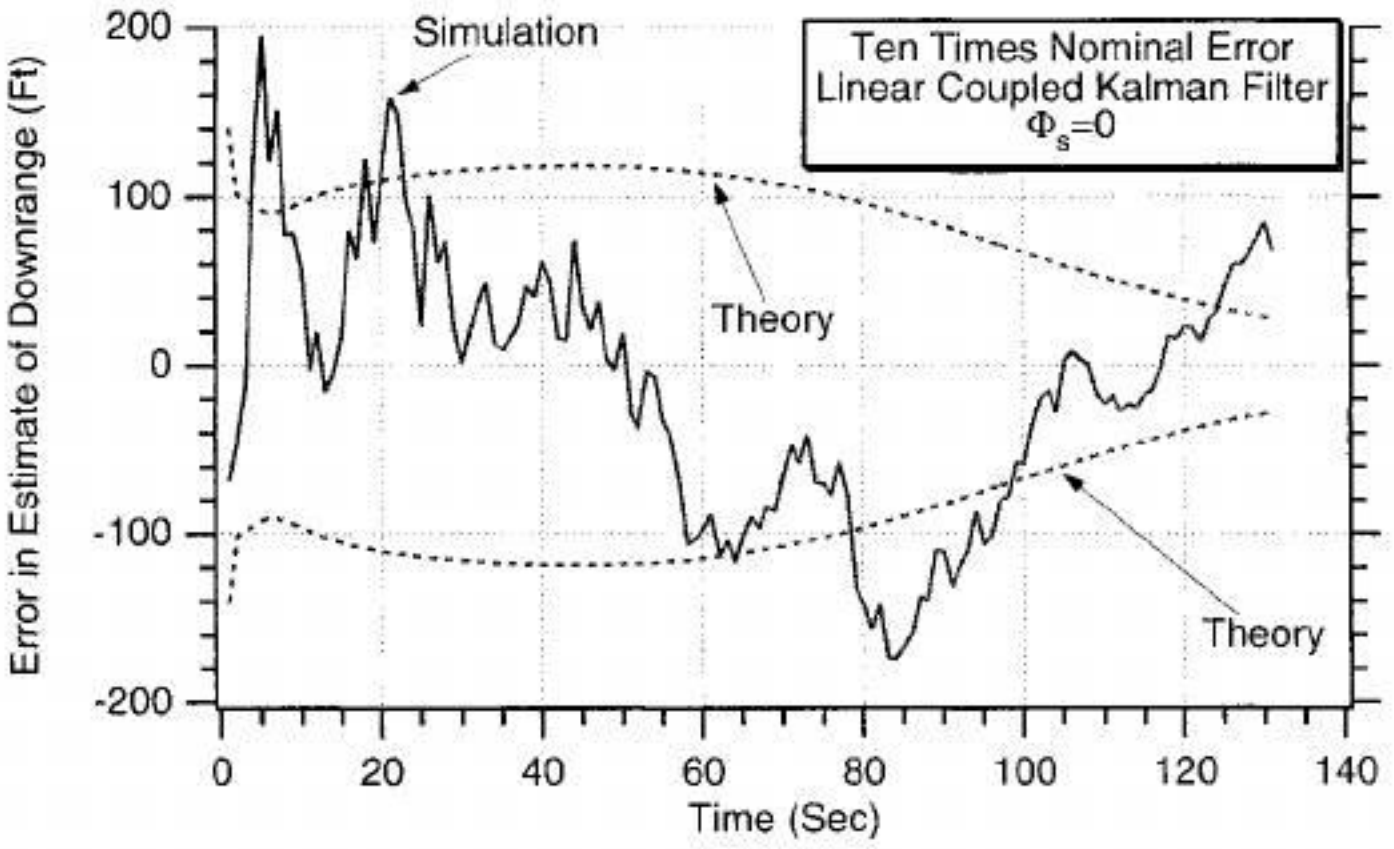

Fig. 9.38 Linear coupled polynomial Kalman filter appears to be working correctly in presence of large initialization errors.

If we compare the preceding results to those of the extended Kalman filter (see Fig. 9.36), we can see that the performance of the linear Kalman filter is far superior to that of the extended Kalman filter. When the initialization errors were 10 times the nominal value, the extended filter yielded estimates for projectile downrange location with errors on the order of 300 ft, while the linear filter errors were on the order of 100 ft.

To further demonstrate the robustness of the linear coupled polynomial Kalman filter to large initialization errors, another experiment was conducted. In this case the initial state estimates were set to zero or

$$\begin{bmatrix} \hat{x}_T(0) \\ \hat{\dot{x}}_T(0) \\ \hat{y}_T(0) \\ \hat{\dot{y}}_T(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The diagonal elements of the initial covariance matrix were then set to infinity to account for the large uncertainties in the initial errors in the estimates or

$$\boldsymbol{P}_0 = \begin{bmatrix} \infty & 0 & 0 & 0 \\ 0 & \infty & 0 & 0 \\ 0 & 0 & \infty & 0 \\ 0 & 0 & 0 & \infty \end{bmatrix}$$

Figure 9.39 shows sample results for the linear coupled polynomial Kalman filter. We can see from Fig. 9.39 that the errors in the estimate of downrange are well behaved, even when the filter is extremely poorly initialized. Recall that when the diagonal elements of the initial covariance matrix are infinite and the process-noise matrix is zero we have a recursive least-squares filter. By comparing Fig 9.39 with Fig. 9.8, we can see that the filter performance is virtually identical, indicating once again that initialization is not an important issue for a linear Kalman filter. However, good initialization is critical for an extended Kalman filter.

Therefore, if we have a problem in which both a linear and extended Kalman filter work and yield comparable performance, it is safer to use the linear Kalman filter because it is more robust when large initialization errors are present.

Summary

In this chapter we have shown how an extended Kalman filter could be built in either the Cartesian or polar coordinate systems. For the cannon-launched projectile problem, in which the filter was in the Cartesian system, the state-space equation was linear, but the measurement equation was nonlinear. If the filter were in the polar system, the state-space equation was nonlinear, but the measurement equation was linear. We saw that for the example considered the performance of both extended Kalman filters was approximately the same. In other words, if computational requirements were not an issue the preferred

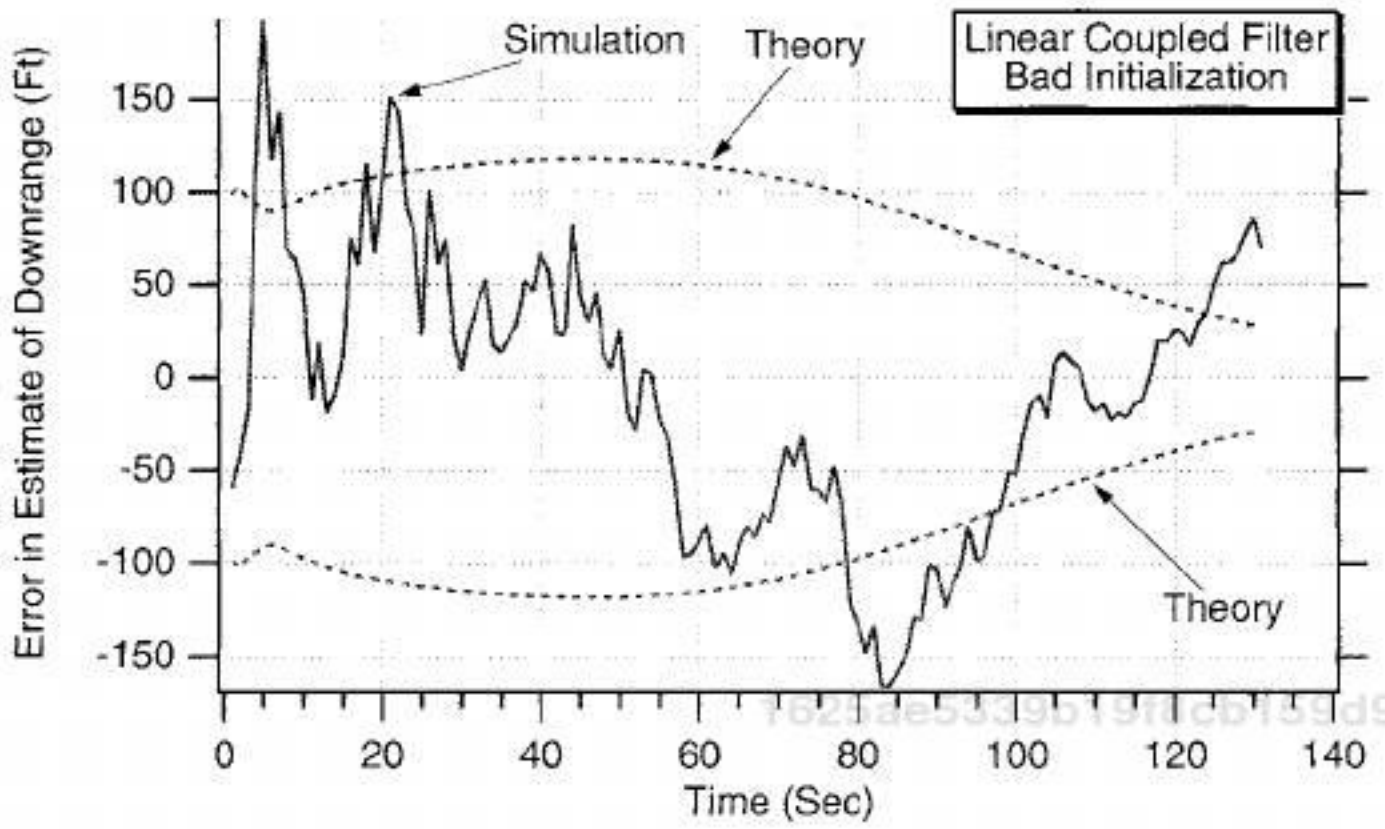

Fig. 9.39 Linear coupled Kalman filter performance appears to be independent of initialization errors.

coordinate system for the extended Kalman filter was a matter of personal preference and not related to performance issues. The polar filter was more computationally expensive than the Cartesian filter because numerical integration was required for state propagation. We also saw that a linear coupled Kalman filter that had near identical performance to that of the extended Kalman filters could also be built. The advantage of the linear filter was that it was more robust, as illustrated by an experiment in which the filter was improperly initialized. The linear coupled filter worked fine under such adverse conditions, whereas the extended Kalman filter failed. Process noise could be used to fix the extended Kalman filter, but the price paid was much larger estimation errors when compared to the linear Kalman filter. We also saw that if computation was an issue the linear filter also could be decoupled. The computer throughput requirements were better in the resultant filter pair because a pair of two-state linear filters require significantly less computation than one linear four-state filter. However, the price for reduced computation was a slight degradation in filter performance as measured by the estimation errors.

Reference

[1] Richards, J. A., Sears, F. W., Wehr, M. R., and Zemansky, M. W., *Modern University Physics, Part 1: Mechanics and Thermodynamics*, Addison Wesley Longman, Reading, MA, 1960, pp. 54–74.

Chapter 10

Tracking a Sine Wave

Introduction

IN THIS chapter we will attempt to apply extended Kalman filtering to a problem we briefly investigated with a linear Kalman filter. We will revisit the problem of tracking a sinusoidal signal measurement corrupted by noise. We have already shown that if the frequency of the sinusoid was known in advance the signal could be tracked quite well with a linear Kalman filter. However, if the frequency of the sinusoidal signal is unknown either our model of the real world or our measurement model becomes nonlinear, and we must resort to an extended Kalman filter. Several possible extended Kalman filters for this application, each of which has different states, will be explored in this chapter.

Extended Kalman Filter

In Chapter 5 we attempted to estimate the states (i.e., derivatives) of a sinusoidal signal based on noisy measurement of the sinusoidal signal. We showed that a linear polynomial Kalman filter was adequate for estimation purposes, but a much better linear Kalman filter, which made use of the fact that we knew that the true signal was sinusoidal, could be constructed. However, it was also demonstrated that if our a priori information was in error (i.e., knowledge of the frequency of the sinusoid is in error) the performance of the better linear Kalman filter deteriorated to the point where the estimates were no better, and possibly worse, than that of the linear polynomial Kalman filter (i.e., linear Kalman filter required no a priori information at all). In this section we will attempt to build an extended Kalman filter that operates on the sinusoidal measurement but also takes into account that the frequency of the sinusoidal signal is unknown and must also be estimated.[1,2]

In some of the one-dimensional work that follows, we will make use of the concept of negative and positive frequencies. Some of these frequency concepts may be difficult to understand from a physical point of view. These concepts are much easier to visualize in two dimensions. For example, in the two-dimensional x-y plane, circular motion can be described by the equations

$$x = A \sin \omega t$$

$$y = A \cos \omega t$$

In this case a positive frequency means that circular motion will be clockwise, and a negative frequency will describe circular motion in the counterclockwise direction.

In this chapter we consider the one-dimensional sinusoidal signal given by

$$x = A \sin \omega t$$

We can define a new variable ϕ to be

$$\phi = \omega t$$

If the frequency of the sinusoid is constant, we can take the derivative of the preceding equation to obtain

$$\dot{\phi} = \omega$$

Similarly, by assuming that the frequency and amplitude of the sinusoid are constant we are also saying that

$$\dot{\omega} = 0$$
$$\dot{A} = 0$$

We can express the preceding set of scalar equations that model our version of the real world in state-space form as

$$\begin{bmatrix} \dot{\phi} \\ \dot{\omega} \\ \dot{A} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \phi \\ \omega \\ A \end{bmatrix} + \begin{bmatrix} 0 \\ u_{s1} \\ u_{s2} \end{bmatrix}$$

where u_{s1} and u_{s2} are white process noise sources that have been added to the derivatives of frequency and amplitude. These white noise sources may be required later to get the filter to work if we encounter problems. The process noise has been added to the derivatives of the states that are the least certain because we really do not know if the frequency and amplitude of the sinusoid will be constant. The preceding state-space equation is linear for this particular formulation. From the preceding state-space equation the continuous process-noise matrix can be written by inspection as

$$\boldsymbol{Q} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \Phi_{s1} & 0 \\ 0 & 0 & \Phi_{s2} \end{bmatrix}$$

where Φ_{s1} and Φ_{s2} are the spectral densities of the white noise sources u_{s1} and u_{s2}. The systems dynamics matrix also can be written by inspection of the state-space equation as

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

Because $\boldsymbol{F}^2$ is zero as a result of

$$\boldsymbol{F}^2 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

the fundamental matrix can be expressed exactly as the two-term Taylor-series expansion

$$\Phi = \boldsymbol{I} + \boldsymbol{F}t = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} t = \begin{bmatrix} 1 & t & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Therefore, the exact discrete fundamental matrix can be obtained by replacing t with T_s, yielding

$$\Phi_k = \begin{bmatrix} 1 & T_s & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Although the state-space equation is linear, the measurement equation is nonlinear for this particular formulation of the problem. In fact, information concerning the sinusoidal nature of the signal is buried in the measurement equation. We have constructed the problem so that the linearized measurement equation (i.e., we are actually measuring x, which is not a state, thus making the measurement nonlinear) is given by

$$\Delta x^* = \begin{bmatrix} \dfrac{\partial x}{\partial \phi} & \dfrac{\partial x}{\partial \omega} & \dfrac{\partial x}{\partial A} \end{bmatrix} \begin{bmatrix} \Delta\phi \\ \Delta\omega \\ \Delta A \end{bmatrix} + v$$

where v is white measurement noise. To calculate the partial derivatives of the preceding equation, we first recall that

$$x = A \sin \omega t = A \sin \phi$$

Therefore, the partial derivatives in the measurement matrix are easily evaluated as

$$\frac{\partial x}{\partial \phi} = A\cos\phi$$

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

$$\frac{\partial x}{\partial A} = \sin\phi$$

making the linerized measurement matrix (i.e., matrix of partial differentials)

$$\boldsymbol{H} = [A\cos\phi \quad 0 \quad \sin\phi]$$

The measurement matrix is evaluated at the projected estimates of A and ϕ. In this formulation the measurement noise is a scalar. Therefore, the discrete measurement noise matrix will also be a scalar given by

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

where σ_x^2 is the variance of the measurement noise.

Recall that the discrete process-noise matrix can be found from the continuous process-noise matrix according to

$$\boldsymbol{Q}_k = \int_0^{T_2} \Phi(\tau)\boldsymbol{Q}\Phi^T(\tau)\,\mathrm{d}t$$

Substitution of the appropriate matrices into the preceding expression yields

$$\boldsymbol{Q}_k = \int_0^{T_s} \begin{bmatrix} 1 & \tau & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & \Phi_{s1} & 0 \\ 0 & 0 & \Phi_{s2} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ \tau & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathrm{d}\tau$$

After multiplying out the three matrices we obtain

$$\boldsymbol{Q}_k = \int_0^{T_s} \begin{bmatrix} \tau^2\Phi_{s1} & \tau\Phi_{s1} & 0 \\ \tau\Phi_{s1} & \Phi_{s1} & 0 \\ 0 & 0 & \Phi_{s2} \end{bmatrix} \mathrm{d}\tau$$

Finally, integrating the preceding expression shows the discrete process-noise matrix to be

$$\boldsymbol{Q}_k = \begin{bmatrix} \dfrac{\Phi_{s1} T_s^3}{3} & \dfrac{\Phi_{s1} T_s^2}{2} & 0 \\ \dfrac{\Phi_{s1} T_s^2}{2} & \Phi_{s1} T_s & 0 \\ 0 & 0 & \Phi_{s2} T_s \end{bmatrix}$$

We now have enough information to solve the matrix Riccati equations for the Kalman gains.

Because the fundamental matrix is exact in this application, we can also use it to exactly propagate the state estimates in the extended Kalman filter over the sampling interval. In other words, by using the fundamental matrix we can propagate the states from time $k-1$ to time k or in matrix form

$$\begin{bmatrix} \bar{\phi}_k \\ \bar{\omega}_k \\ \bar{A}_k \end{bmatrix} = \begin{bmatrix} 1 & T_s & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{\phi}_{k-1} \\ \hat{\omega}_{k-1} \\ \hat{A}_{k-1} \end{bmatrix}$$

We can multiply out the preceding matrix equation to get the three scalar equations

$$\begin{aligned} \bar{\phi}_k &= \hat{\phi}_{k-1} + \hat{\omega}_{k-1} T_s \\ \bar{\omega}_k &= \hat{\omega}_{k-1} \\ \bar{A}_k &= \hat{A}_{k-1} \end{aligned}$$

The linearized measurement matrix is used in the Riccati equations, but we do not have to use that matrix in the computation of the residual for the actual extended Kalman-filtering equations. We can do better by using the actual nonlinear equation for the residual or

$$\text{Res}_k = x_k^* - A_k \sin \phi_k$$

where the residual has been computed using the projected values of the states (i.e., barred quantities in the preceding equation). Now the extended Kalman-filtering equations can be written as

$$\begin{aligned} \hat{\phi}_k &= \bar{\phi}_k + K_{1_k} \text{Res}_k \\ \hat{\omega}_k &= \bar{\omega}_k + K_{2_k} \text{Res}_k \\ \hat{A}_k &= \bar{A}_k + K_{3_k} \text{Res}_k \end{aligned}$$

Listing 10.1 presents the extended Kalman filter for estimating the states of a noisy sinusoidal signal whose frequency is unknown. We can see that the

Listing 10.1 Extended Kalman filter for sinusoidal signal with unknown frequency

```
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,O-Z)
        REAL*8 P(3,3),Q(3,3),M(3,3),PHI(3,3),HMAT(1,3),HT(3,1),PHIT(3,3)
        REAL*8 RMAT(1,1),IDN(3,3),PHIP(3,3),PHIPPHIT(3,3),HM(1,3)
        REAL*8 HMHT(1,1),HMHTR(1,1),HMHTRINV(1,1),MHT(3,1),K(3,1)
        REAL*8 KH(3,3),IKH(3,3)
        INTEGER ORDER
        A=1.
        W=1.
        TS=.1
        ORDER=3
        PHIS1=0.
        PHIS2=0.
        SIGX=1.
        OPEN(1,STATUS='UNKNOWN',FILE='DATFIL')
        OPEN(2,STATUS='UNKNOWN',FILE='COVFIL')
        T=0.
        S=0.
        H=.001
        DO 14 I=1,ORDER
        DO 14 J=1,ORDER
        PHI(I,J)=0.
        P(I,J)=0.
        Q(I,J)=0.
        IDN(I,J)=0.
14      CONTINUE
        RMAT(1,1)=SIGX**2
        IDN(1,1)=1.
        IDN(2,2)=1.
        IDN(3,3)=1.
        PHIH=0.
        WH=2.
        AH=3.
        P(1,1)=0.
        P(2,2)=(W-WH)**2
        P(3,3)=(A-AH)**2
        XT=0.
        XTD=A*W
        WHILE(T<=20.)
              XTOLD=XT
              XTDOLD=XTD
              XTDD=-W*W*XT
              XT=XT+H*XTD
              XTD=XTD+H*XTDD
              T=T+H
              XTDD=-W*W*XT
```

(*continued*)

Listing 10.1 (*Continued*)

```
      XT=.5*(XTOLD+XT+H*XTD)
      XTD=.5*(XTDOLD+XTD+H*XTDD)
      S=S+H
      IF(S>=(TS-.00001))THEN
                  S=0.
                  PHI(1,1)=1.
                  PHI(1,2)=TS
                  PHI(2,2)=1.
                  PHI(3,3)=1.
                  Q(1,1)=TS*TS*TS*PHIS1/3.
                  Q(1,2)=.5*TS*TS*PHIS1
                  Q(2,1)=Q(1,2)
                  Q(2,2)=PHIS1*TS
                  Q(3,3)=PHIS2*TS
                  PHIB=PHIH+WH*TS
                  HMAT(1,1)=AH*COS(PHIB)
                  HMAT(1,2)=0.
                  HMAT(1,3)=SIN(PHIB)
                  CALL  MATTRN(PHI,ORDER,ORDER,PHIT)
                  CALL  MATTRN(HMAT,1,ORDER,HT)
                  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)
                  CALL  MATADD(HMHT,ORDER,ORDER,RMAT,
                    HMHTR)HMHTRINV(1,1)=1./HMHTR(1,1)
                  CALL  MATMUL(M,ORDER,ORDER,HT,ORDER,1,
                    MHT)
                  CALL  MATMUL(MHT,ORDER,1,HMHTRINV,1,1,K)
                  CALL  MATMUL(K,ORDER,1,HMAT,1,ORDER,KH)
                  CALL  MATSUB(IDN,ORDER,ORDER,KH,IKH)
                  CALL  MATMUL(IKH,ORDER,ORDER,M,ORDER,
                    ORDER,P)
      CALL  GAUSS(XTNOISE,SIGX)
      XTMEAS=XT+XTNOISE
      RES=XTMEAS-AH*SIN(PHIB)
      PHIH=PHIB+K(1,1)*RES
      WH=WH+K(2,1)*RES
      AH=AH+K(3,1)*RES
      ERRPHI=W*T-PHIH
      SP11=SQRT(P(1,1))
      ERRW=W-WH
      SP22=SQRT(P(2,2))
      ERRA=A-AH
      SP33=SQRT(P(3,3))
```

(*continued*)

Listing 10.1 (*Continued*)

```
                    XTH=AH*SIN(PHIH)
                    XTDH=AH*WH*COS(PHIH)
                    WRITE(9,*)T,XT,XTH,XTD,XTDH,W,WH,A,AH,W*T,
                      PHIH
                    WRITE(1,*)T,XT,XTH,XTD,XTDH,W,WH,A,AH,W*T,
                      PHIH
                    WRITE(2,*)T,ERRPHI,SP11,-SP11,ERRW,SP22,-SP22,
     1                ERRA,SP33,-SP33
                ENDIF
            END DO
            PAUSE
            CLOSE(1)
            CLOSE(2)
            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
```

simulation is initially set up to run without process noise (i.e., PHIS1 = PHIS2 = 0). The actual sinusoidal signal has unity amplitude, and the standard deviation of the measurement noise is also unity. The frequency of the actual sinusoidal signal is unity. The filter's initial state estimates of ϕ is set to zero, whereas the estimate of A has been set to three (rather than unity). The initial estimate of the frequency is set to two (rather than unity). Values are used for the initial covariance matrix to reflect the uncertainties in our initial state estimates. Because the fundamental matrix is exact in this filter formulation, a special subroutine is *not* required in this simulation to integrate the state equations over the sampling interval to obtain the state projections.

The nominal case of Listing 10.1 (i.e., in which there is no process noise) was run. We can see from Figs. 10.1 and 10.2 that for this run the extended Kalman filter is able to estimate the frequency and amplitude of the sinusoidal signal quite well when the actual frequency of the sinusoid is 1 rad/s and the filter's initial frequency estimate is 2 rad/s. After approximately 5 s the frequency and amplitude estimates are near perfect.

To test the filter's robustness, another case was run in which the actual frequency of the sinusoid is negative, but the filter is initialized with a positive frequency. We can see from Fig. 10.3 that now the filter is unable to estimate the negative frequency. In fact the final frequency estimate of the extended Kalman filter is approximately zero. Figure 10.4 indicates that under these circumstances the filter is also unable to estimate the signal amplitude. The figure indicates that the extended Kalman filter estimates the amplitude to be three, rather than the true value of unity.

Even though at this point we know the filter is not satisfactory in meeting our original goals, we would like to examine it further to see if we can learn

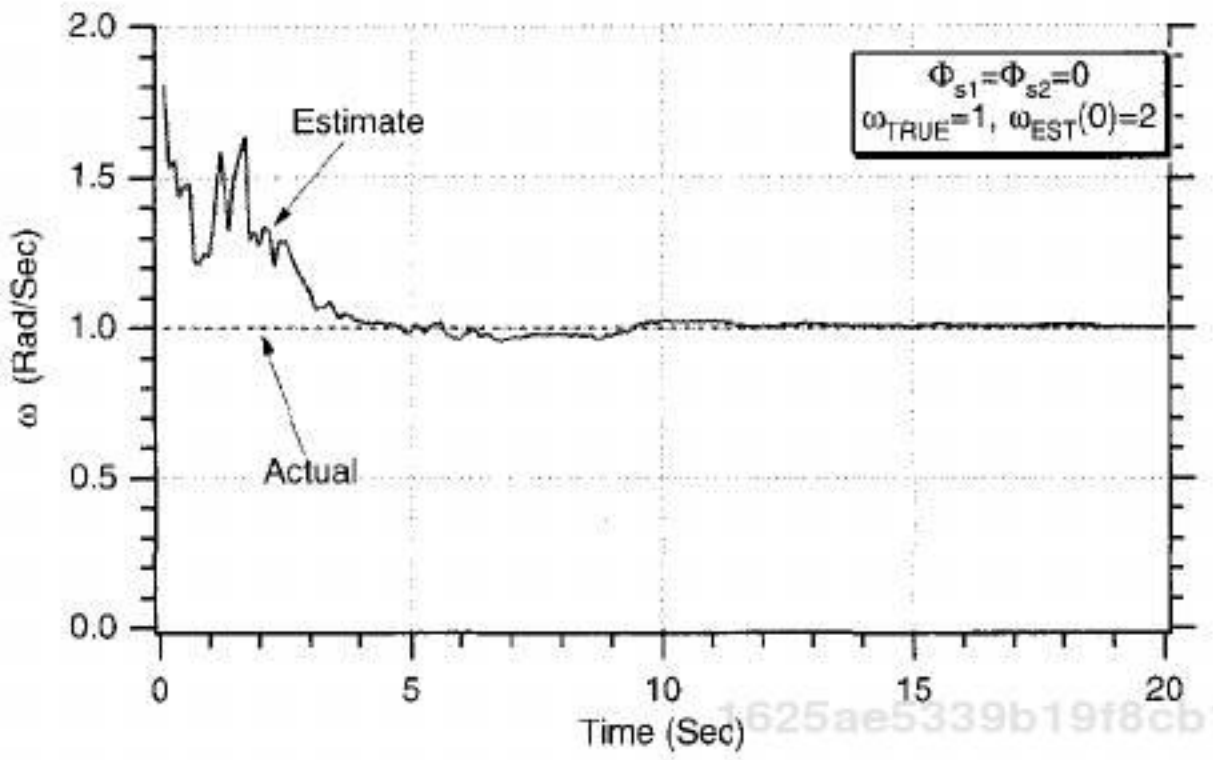

Fig. 10.1 Extended Kalman filter is able to estimate positive frequency when initial frequency estimate is also positive.

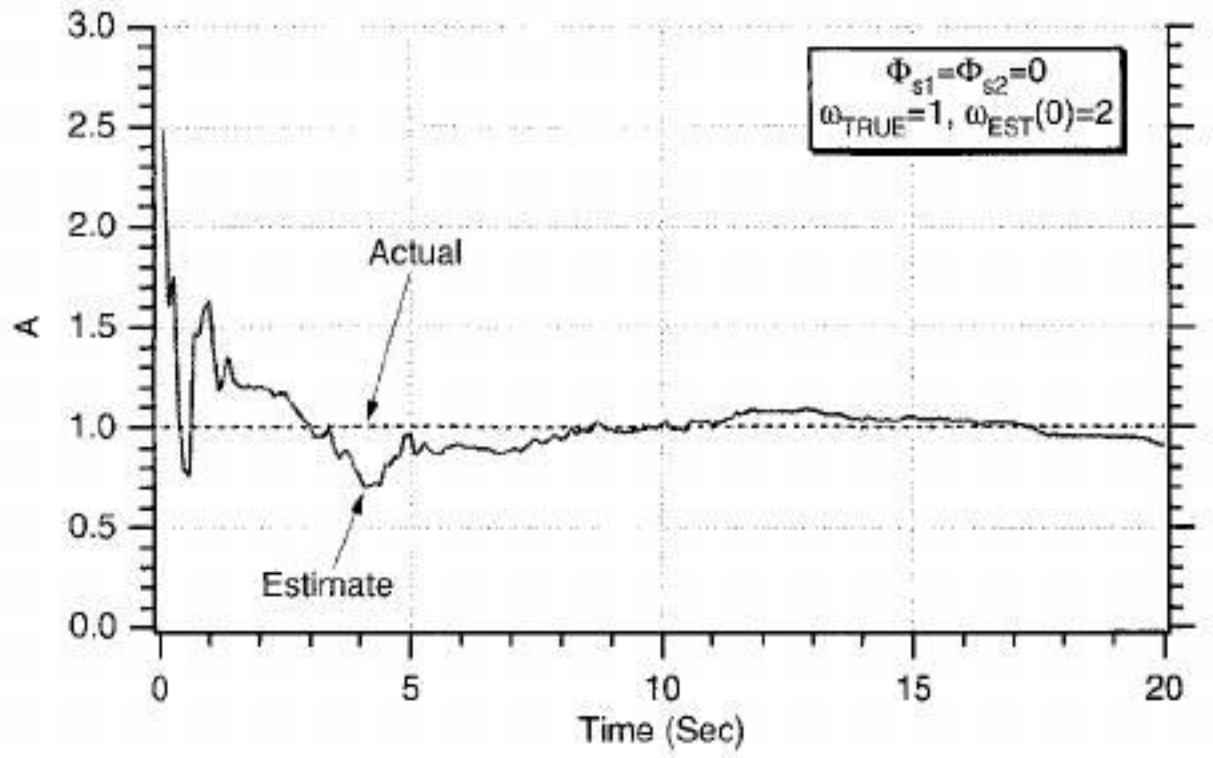

Fig. 10.2 Extended Kalman filter is able to estimate amplitude when actual frequency is positive and initial frequency estimate is also positive.

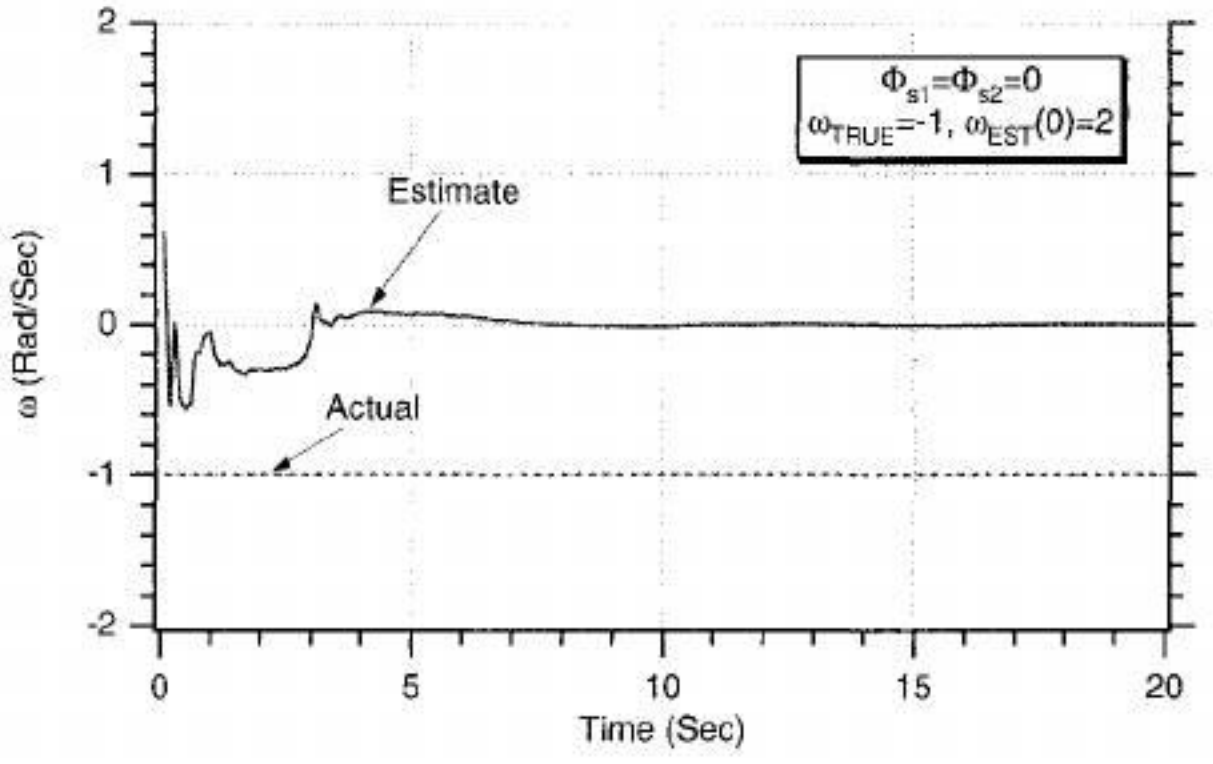

Fig. 10.3 Extended Kalman filter is unable to estimate negative frequency when initial frequency estimate is positive.

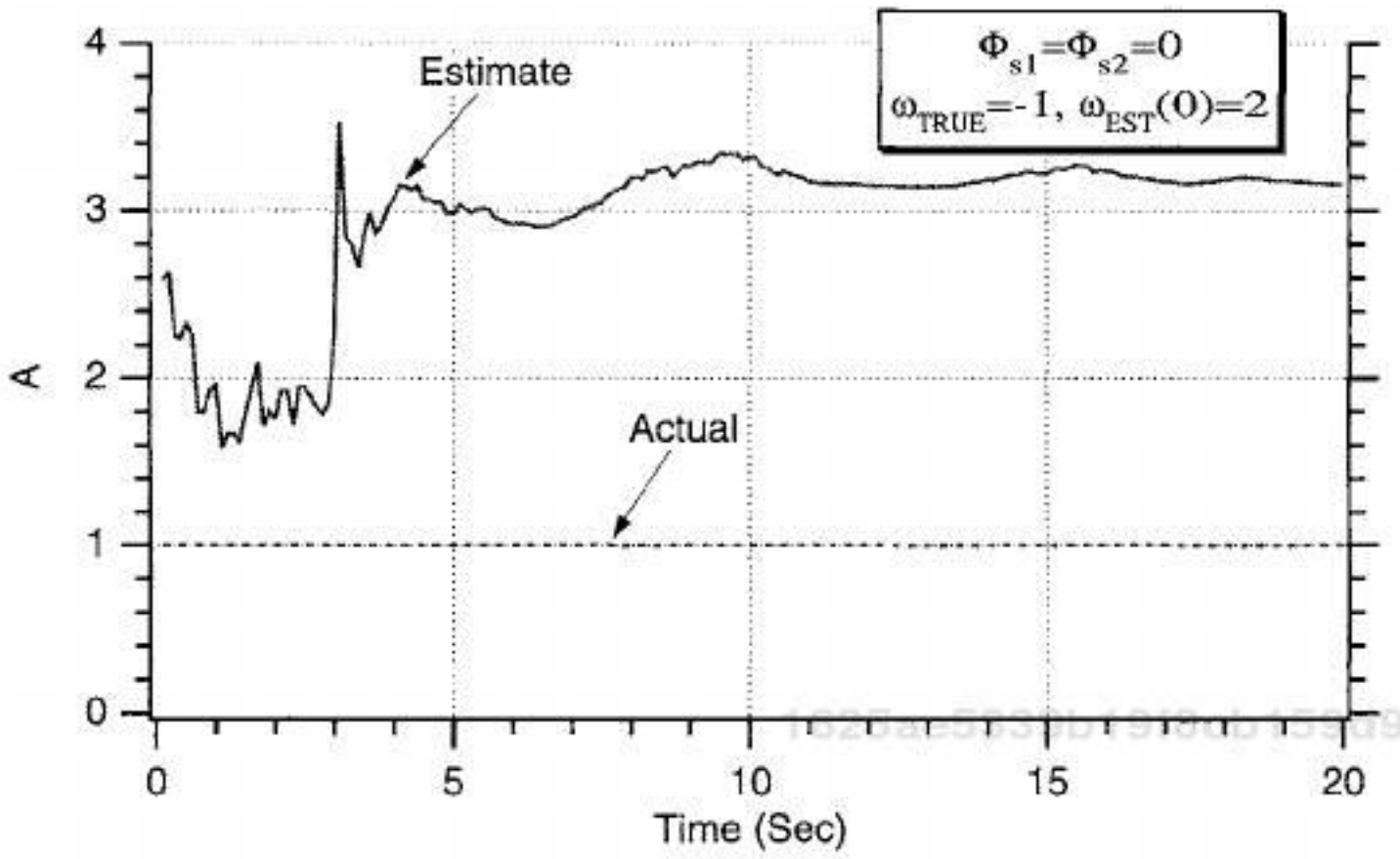

Fig. 10.4 Extended Kalman filter is unable to estimate amplitude when actual frequency is negative and initial frequency estimate is positive.

something that will be of future use. To further test the filter's properties, another case was run in which the actual frequency of the sinusoid is negative, but this time the filter is initialized with a negative frequency. We can see from Fig. 10.5 that the filter is now able to estimate the negative frequency fairly rapidly. Figure 10.6 also indicates that under these circumstances the filter is also able to estimate the signal amplitude accurately.

As a final test of the filter's properties, another case was run in which the actual frequency of the sinusoid is positive while the filter is initialized with a negative frequency. We can see from Fig. 10.7 that after approximately 10 s the filter is now able to estimate the positive frequency. However, Fig. 10.8 indicates that

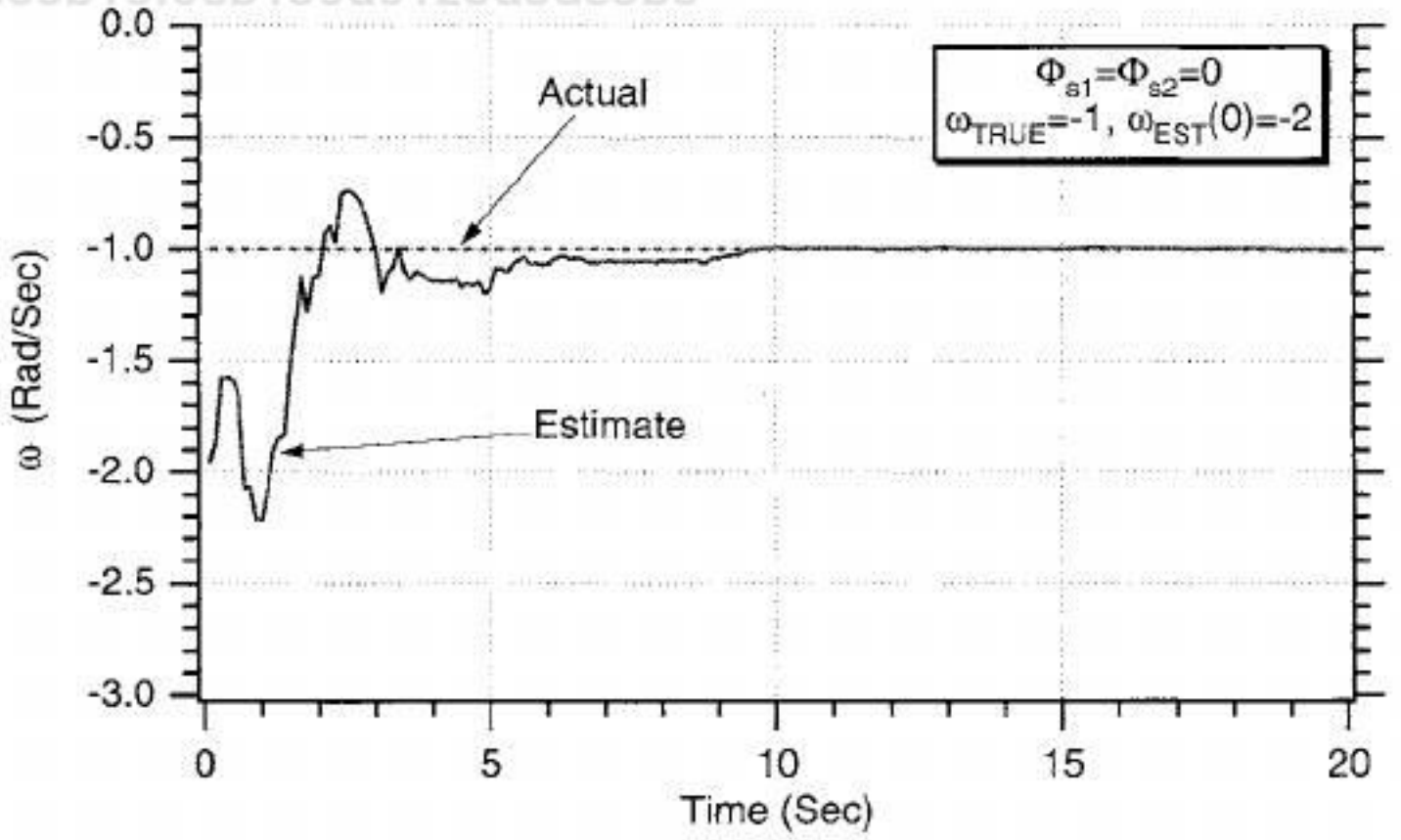

Fig. 10.5 Extended Kalman filter is now able to estimate negative frequency when initial frequency estimate is also negative.

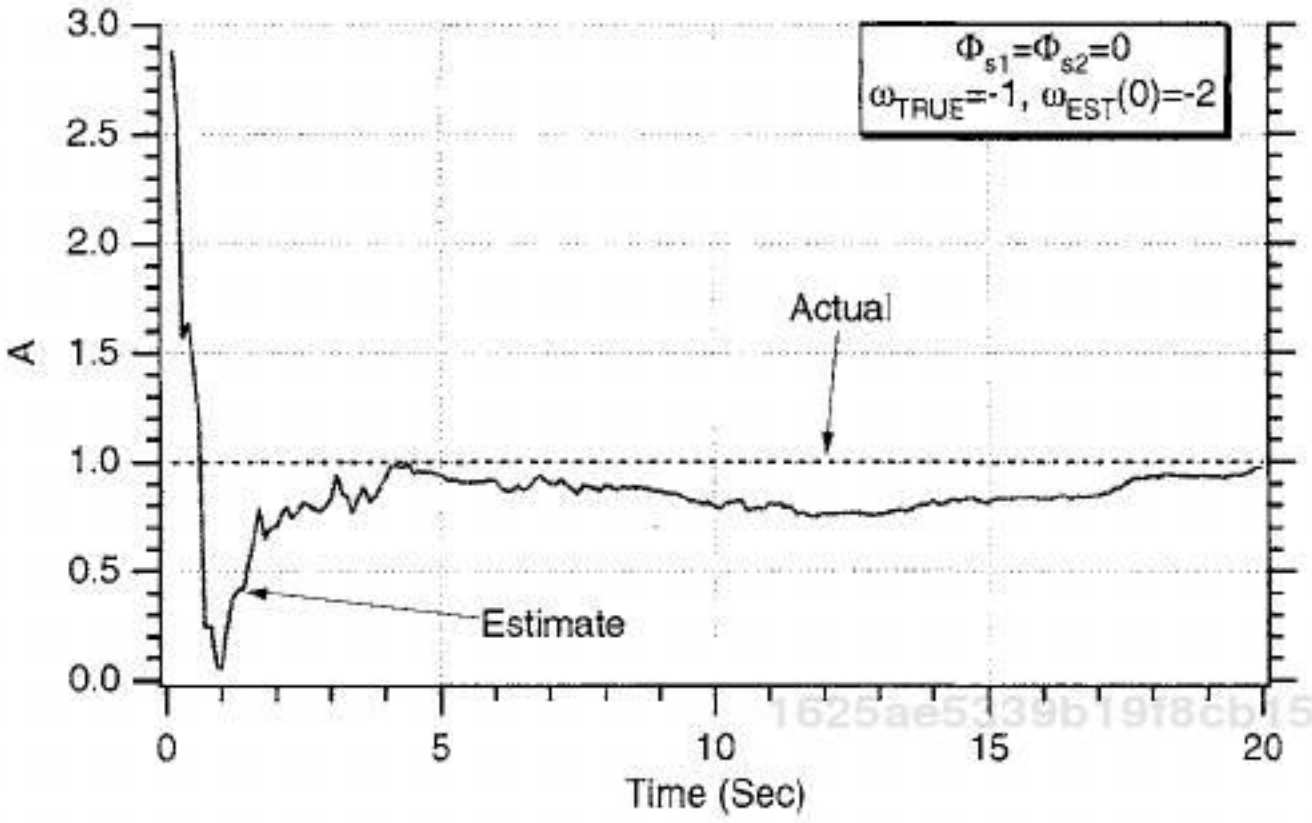

Fig. 10.6 Extended Kalman filter is now able to estimate amplitude when actual frequency is negative and initial frequency estimate is also negative.

under these circumstances the filter is unable to estimate the signal amplitude. The actual amplitude is 1, and the estimated amplitude is approximately 0.6.

From the preceding set of experiments, it appears that the extended Kalman filter only works if the initial estimate of the signal frequency is of the same sign as the actual frequency. We have already seen that when the signs of the actual frequency and initial frequency estimate are mismatched either the signal frequency or amplitude or both could not be estimated.

Sometimes adding process noise is the engineering fix for getting a Kalman filter to perform properly and robustly. Figures 10.9 and 10.10 reexamine the case where the actual frequency of the sinusoid is positive but the initial frequency estimate of the filter is negative. If we now add process noise (i.e.,

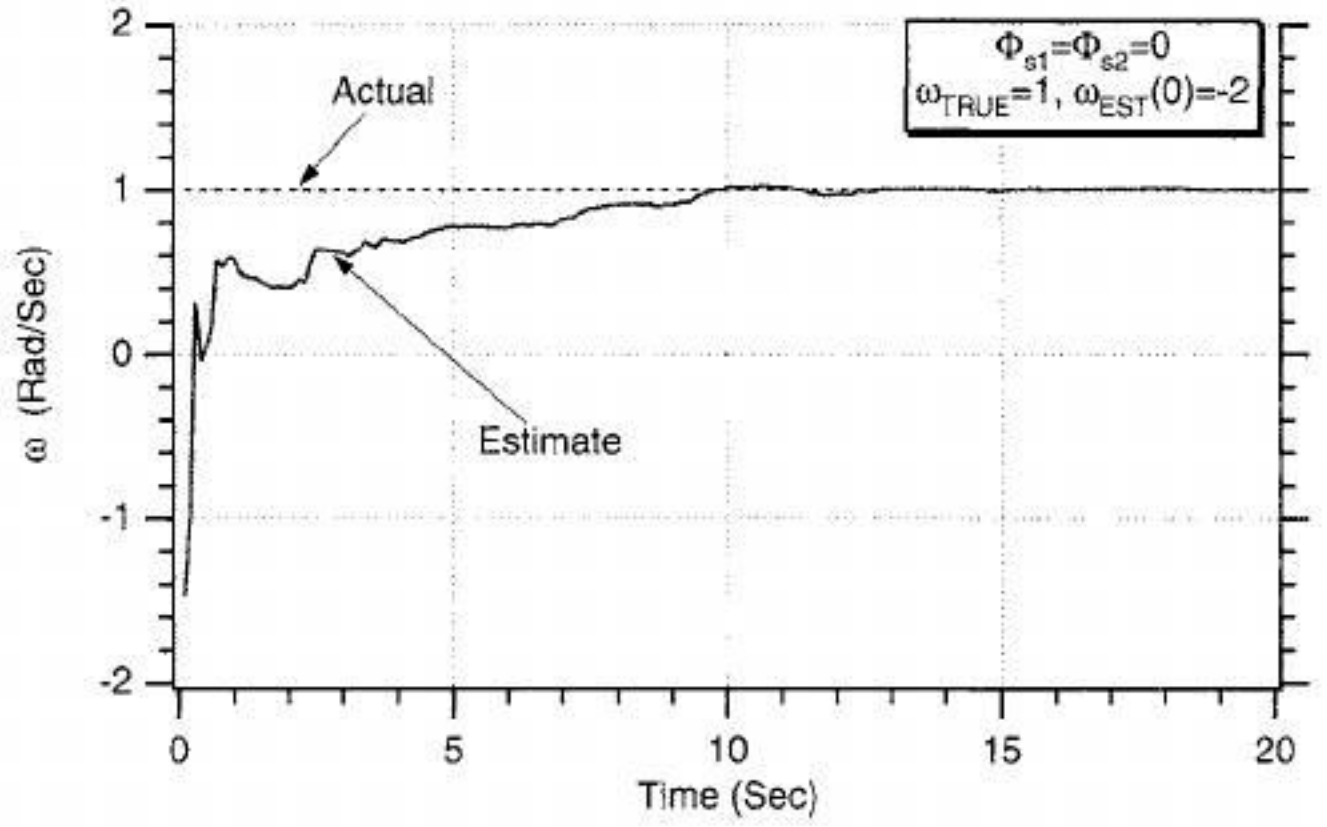

Fig. 10.7 Extended Kalman filter is able to estimate positive frequency when initial frequency estimate is negative.

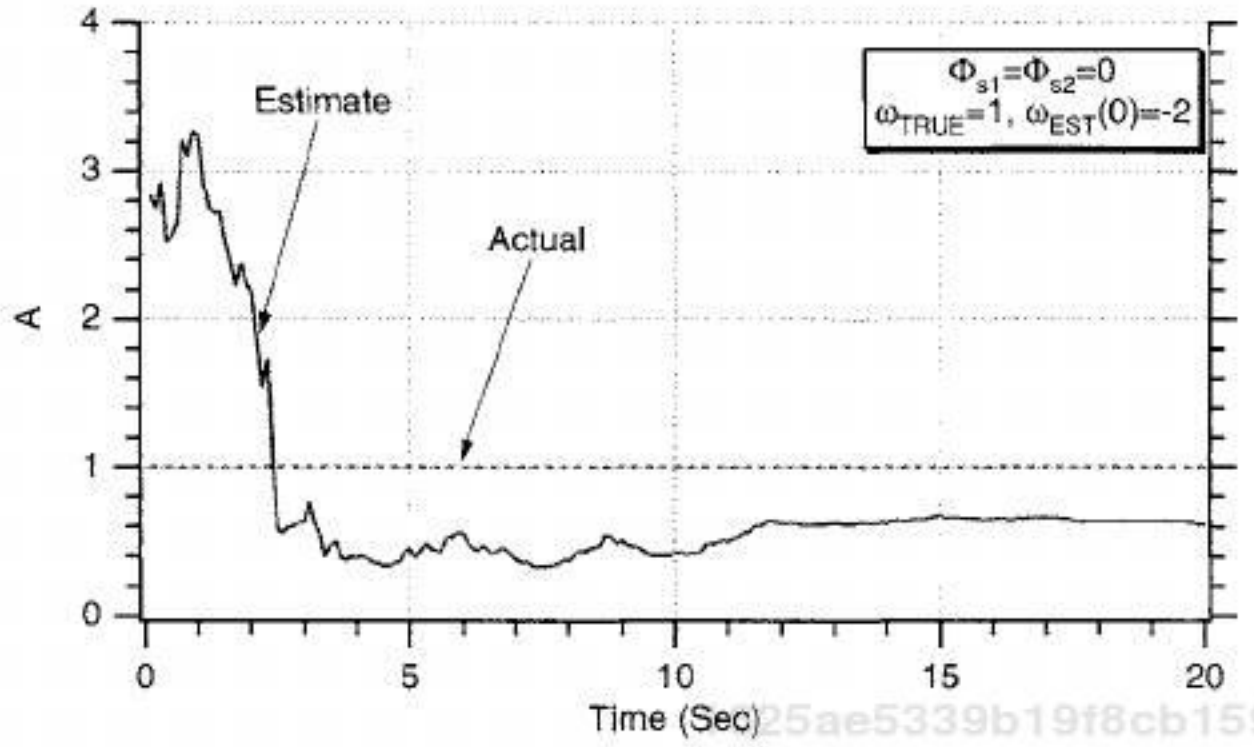

Fig. 10.8 Extended Kalman filter is not able to estimate amplitude when actual frequency is positive and initial frequency estimate is negative.

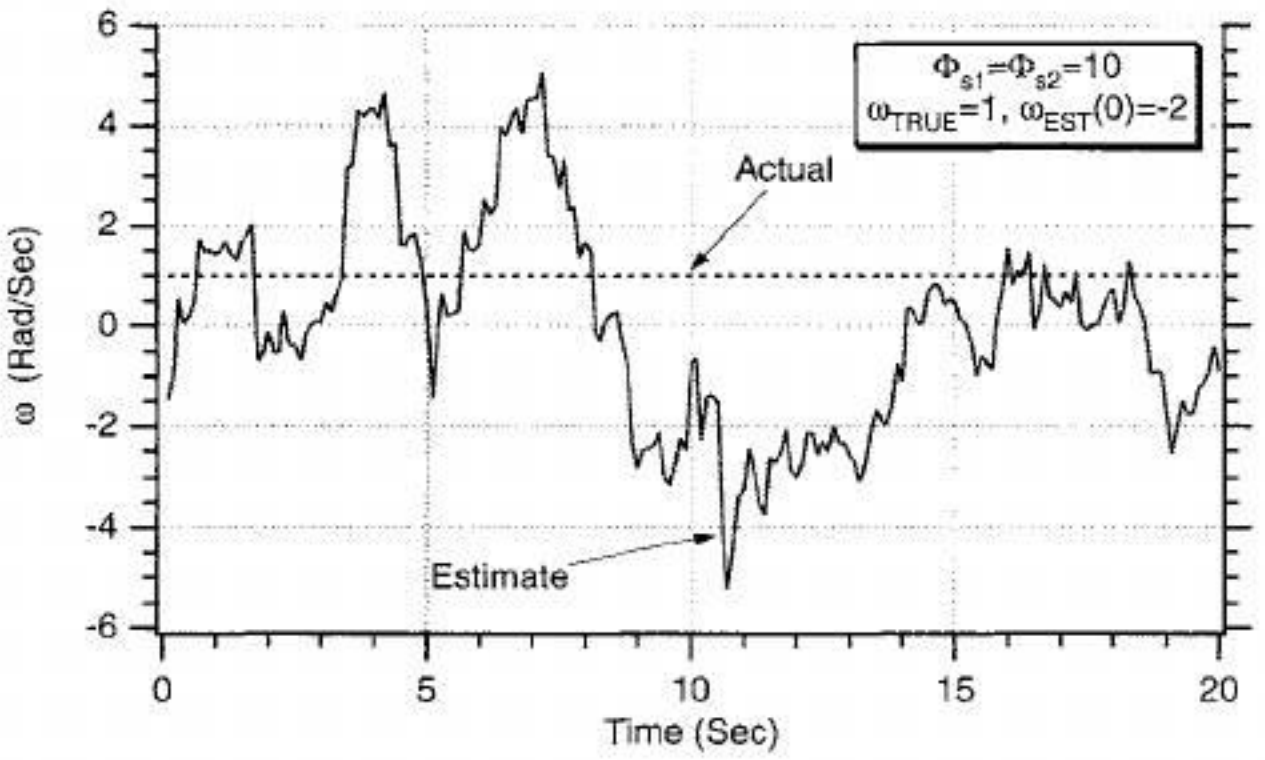

Fig. 10.9 Addition of process noise is not the engineering fix to enable filter to estimate frequency.

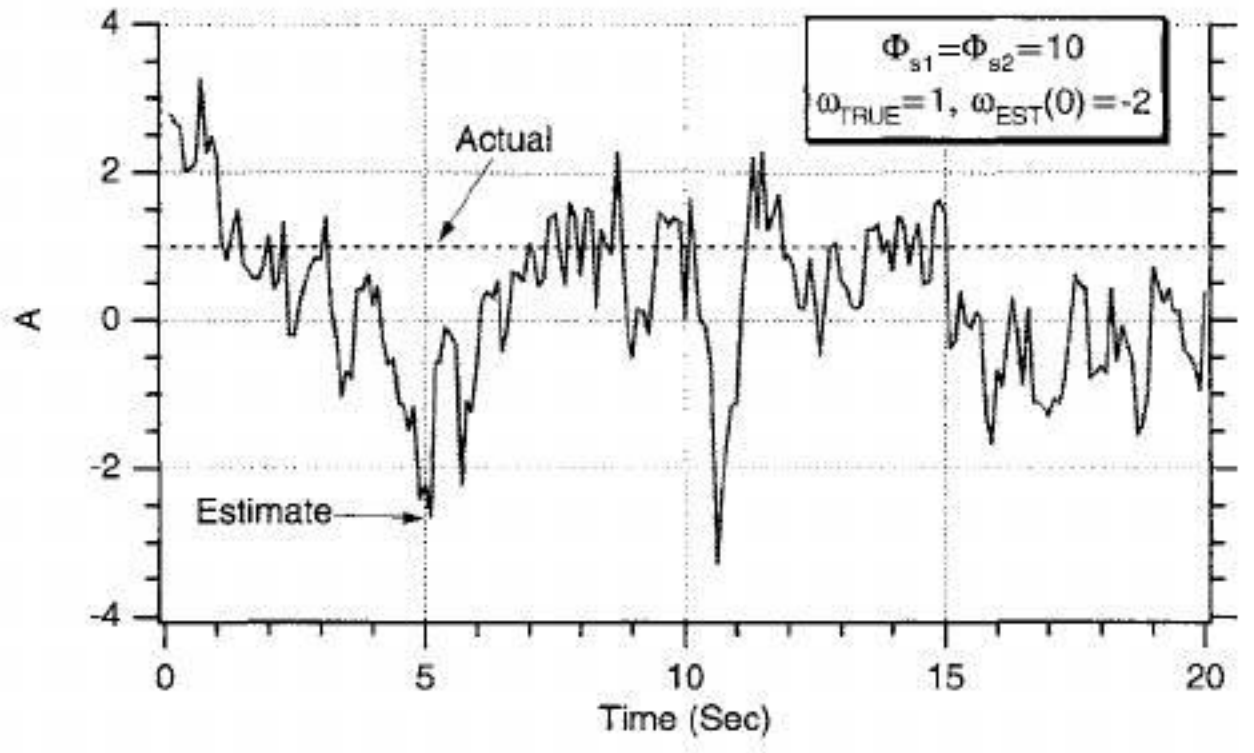

Fig. 10.10 Addition of process noise is not the engineering fix to enable filter to estimate amplitude.

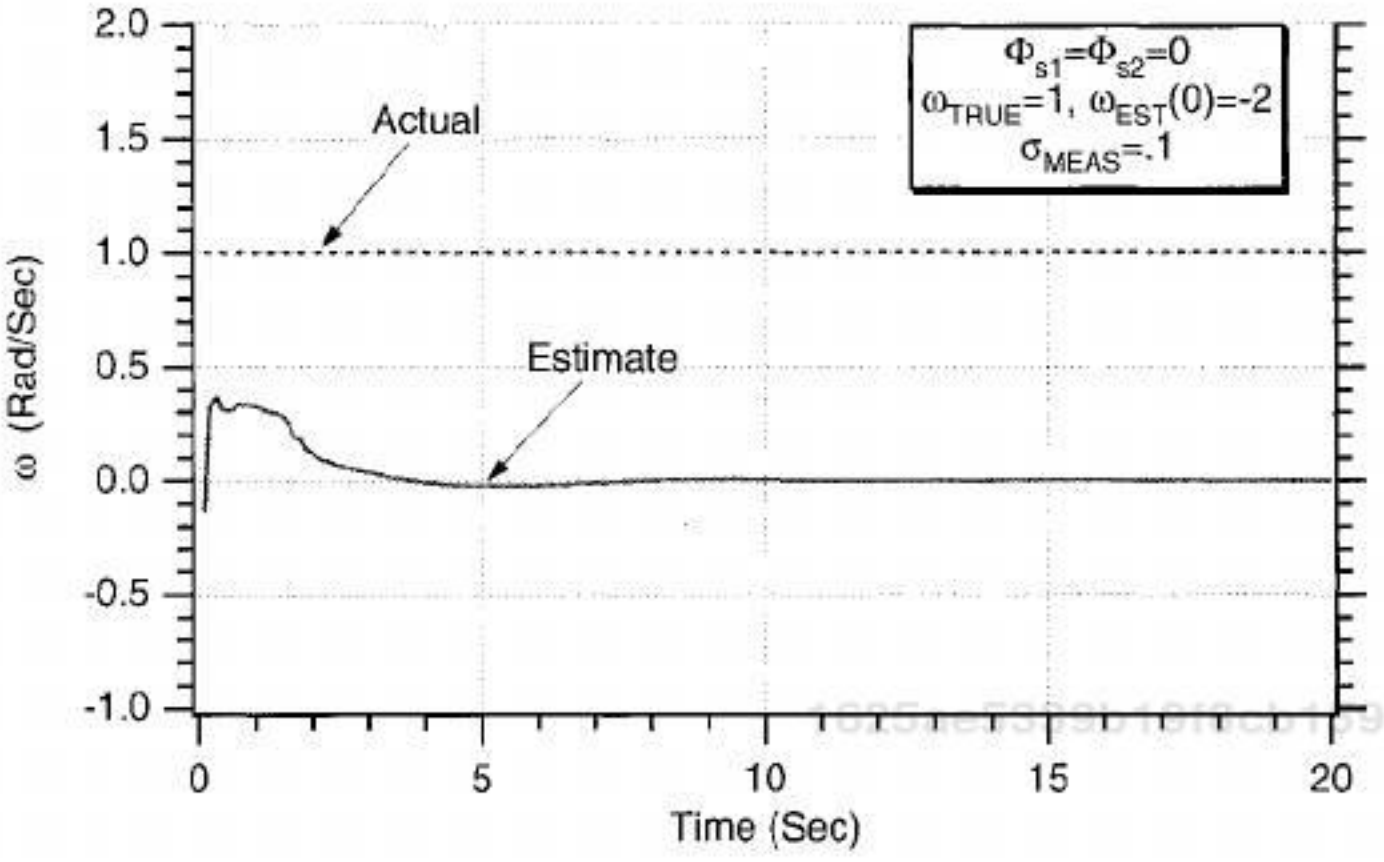

Fig. 10.11 Reducing measurement noise by an order of magnitude does not enable Kalman filter to estimate positive frequency when initial frequency estimate is negative.

$\Phi_{s1} = 10, \quad \Phi_{s2} = 10$), the extended Kalman filter's frequency and amplitude estimates are considerably worse than they were when there was no process noise at all (see Figs. 10.7–10.8). Therefore, it appears that the extended Kalman filter is not able to estimate frequency and amplitude under all circumstances, even in the presence of process noise.

It also was hypothesized that perhaps there was too much measurement noise for the extended Kalman filter to work properly. Figure 10.11 indicates that even if we reduce the measurement noise by an order of magnitude the extended Kalman filter is still unable to estimate the positive frequency of the sinusoid when the initial filter frequency estimate is negative. In addition, from Fig. 10.12

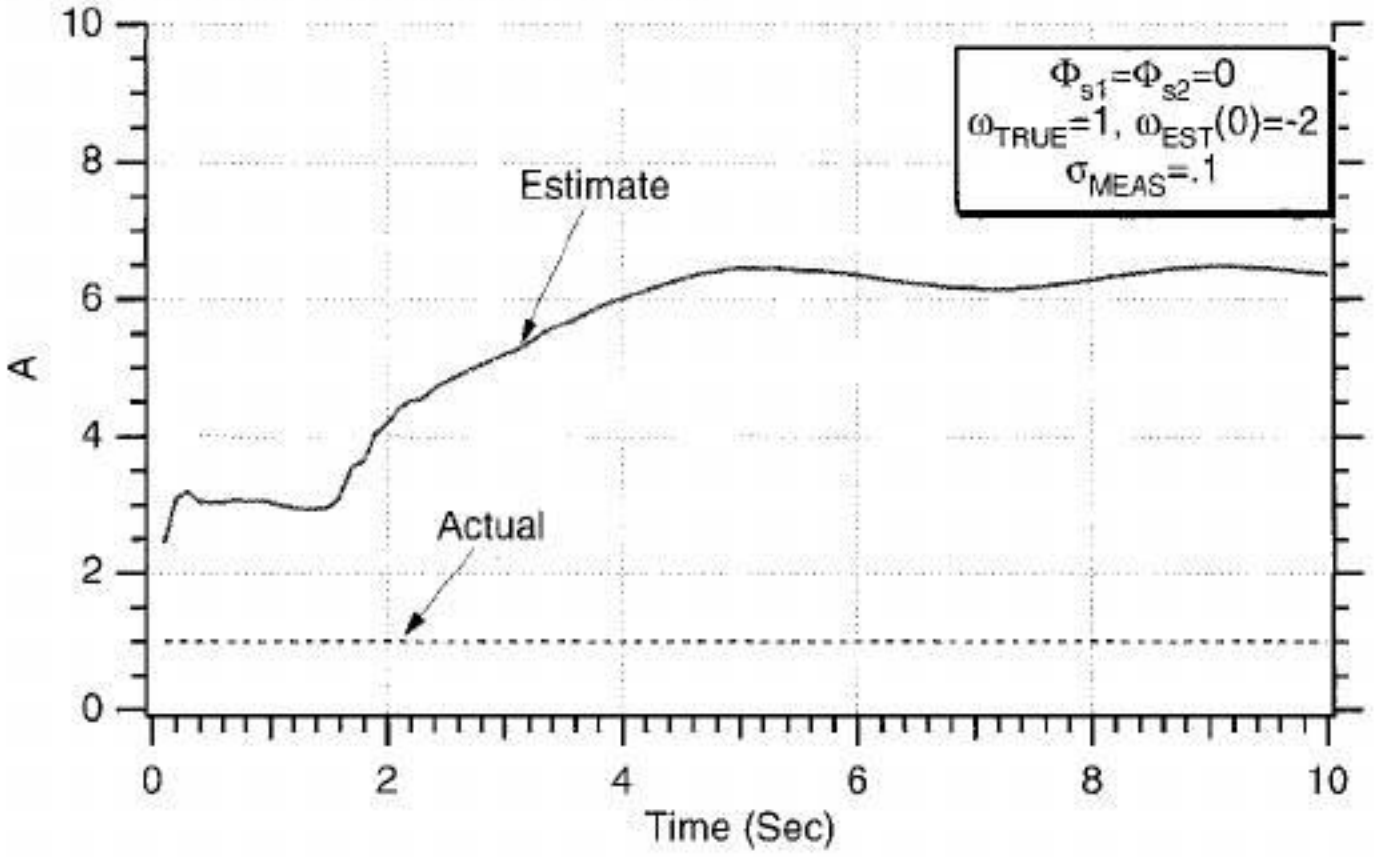

Fig. 10.12 Reducing measurement noise by an order of magnitude does not enable Kalman filter to estimate amplitude when actual frequency is positive and initial frequency estimate is negative.

we can see that the filter is also unable to estimate the amplitude under the same conditions. In other words, the filter's inability to estimate frequency and amplitude properly under a variety of initial conditions is not caused by the fact that the measurement noise is too large.

The conclusion reached is that the extended Kalman filter we have formulated in this section does not appear to be working satisfactorily if the filter is not properly initialized. Is it possible that, for this problem, the frequency of the sinusoid is unobservable?

Two-State Extended Kalman Filter with a Priori Information

So far we have shown that the extended Kalman filter of the preceding section was unable to estimate the frequency and amplitude of the sinusoidal signal unless the filter was properly initialized. With perfect hindsight this now seems reasonable because if we have a signal given by

$$x = A \sin \omega t$$

it is possible to determine if a negative value of x is a result of either a negative frequency ω or a negative amplitude A. Therefore, to gain a deeper understanding of the problem it is hypothesized that if the amplitude of the sinusoid is known in advance we should be able to estimate the frequency based on measurements of x. For the academic purpose of testing our hypothesis, we will derive another extended Kalman filter, which assumes that the amplitude of the sinusoidal signal is known precisely. Under these circumstances the model of the real world can be simplified from that of the preceding section to

$$\begin{bmatrix} \dot{\phi} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \phi \\ \omega \end{bmatrix} + \begin{bmatrix} 0 \\ u_s \end{bmatrix}$$

where u_s is white process noise that has been added to the derivative of frequency to account for the fact that we may not be modeling the real world perfectly. Therefore, the continuous process-noise matrix can be written by inspection of the preceding state-space equation as

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

where Φ_s is the spectral density of the white process noise. The systems dynamics matrix also can be obtained by inspection of the state-space equation representing the real world as

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

Because $\boldsymbol{F}^2$ is zero, the fundamental matrix can be represented exactly by the two-term Taylor-series expansion

$$\Phi = \boldsymbol{I} + \boldsymbol{F}t = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} t = \begin{bmatrix} 1 & t \\ 0 & 1 \end{bmatrix}$$

which means that the discrete fundamental matrix is given by

$$\Phi_k = \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix}$$

Although the state-space equation representing our model of the real world is linear, the measurement equation (i.e., we are actually measuring x, which is not a state) is nonlinear. Therefore, the linearized measurement equation is given by

$$\Delta x^* = \begin{bmatrix} \dfrac{\partial x}{\partial \phi} & \dfrac{\partial x}{\partial \omega} \end{bmatrix} \begin{bmatrix} \Delta\phi \\ \Delta\omega \end{bmatrix} + v$$

where v is white measurement noise. We have already shown in the preceding section that

$$x = A \sin \omega t = A \sin \phi$$

the partial derivatives of the measurement matrix can be easily evaluated as

$$\frac{\partial x}{\partial \phi} = A \cos \phi$$

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

making the linearized measurement matrix

$$\boldsymbol{H} = [A \cos \phi \quad 0]$$

The measurement matrix will be evaluated at the projected estimate of ϕ. In this example, because the measurement noise is a scalar, the discrete measurement noise matrix will also be a scalar given by

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

where σ_x^2 is the variance of the measurement noise. Recall that the discrete process-noise matrix can be found from the continuous process-noise matrix according to

$$\boldsymbol{Q}_k = \int_0^{T_2} \Phi(\tau) \boldsymbol{Q} \Phi^T(\tau)\, \mathrm{d}t$$

Substitution of the appropriate matrices into the preceding expression yields

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

After multiplying out the three matrices, we get

$$\boldsymbol{Q}_k = \int_0^{T_s} \begin{bmatrix} \tau^2 \Phi_s & \tau \Phi_s \\ \tau \Phi_s & \Phi_s \end{bmatrix} d\tau$$

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

$$\boldsymbol{Q}_k = \begin{bmatrix} \dfrac{\Phi_s T_s^3}{3} & \dfrac{\Phi_s T_s^2}{2} \\ \dfrac{\Phi_s T_s^2}{2} & \Phi_s T_s \end{bmatrix}$$

We now have enough information to solve the matrix Riccati equations for the Kalman gains.

Because the fundamental matrix is also exact in this application, we can also use it to propagate the state estimates in the extended Kalman filter over the sampling interval. Using the fundamental matrix, we can propagate the states from time $k-1$ to time k or in matrix form

$$\begin{bmatrix} \bar{\phi}_k \\ \bar{\omega}_k \end{bmatrix} = \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{\phi}_{k-1} \\ \hat{\omega}_{k-1} \end{bmatrix}$$

We can multiply out the preceding matrix equation to get the two scalar equations

$$\bar{\phi}_k = \hat{\phi}_{k-1} + \hat{\omega}_{k-1} T_s$$
$$\bar{\omega}_k = \hat{\omega}_{k-1}$$

Again, as was the case in the preceding section, because the measurement matrix is linearized we do not have to use it in the calculation of the residual for the actual extended Kalman-filtering equations. Instead, we can do better by using the actual nonlinear equation for the residual or

$$\text{Res}_k = x_k^* - A \sin \bar{\phi}_k$$

In the preceding equation the amplitude A is not estimated but is assumed to be known a prior. The extended Kalman-filtering equations can now be written as

$$\hat{\phi}_k = \bar{\phi}_k + K_{1_k} \text{Res}_k$$
$$\hat{\omega}_k = \bar{\omega}_k + K_{2_k} \text{Res}_k$$

where the barred quantities represent projected states that have been already defined.

Listing 10.2 presents the two-state extended Kalman filter for estimating the states of a noisy sinusoidal signal whose frequency is unknown but whose amplitude is known. As was the case with Listing 10.1, we can see that the simulation is initially set up to run without process noise (i.e., PHIS = 0). The actual sinusoidal signal still has unity amplitude, and the standard deviation of the measurement noise is also unity. We can see from Listing 10.2 that the frequency of the actual sinusoidal signal is also unity. The filter's initial state estimate of ϕ is set to zero because time is initially zero (i.e., $\phi = \omega t$), while the initial estimate of the frequency is set to two (rather than unity). Values are used for the initial covariance matrix to reflect the uncertainties in our initial state estimates. Again, because the fundamental matrix is exact, a special subroutine is not required in this simulation to integrate the state equations over the sampling interval in order to obtain the state projections.

Listing 10.2 Two-state extended Kalman filter with a priori information for estimating frequency of sinusoidal signal

```
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,O-Z)
        REAL*8 P(2,2),Q(2,2),M(2,2),PHI(2,2),HMAT(1,2),HT(2,1),PHIT(2,2)
        REAL*8 RMAT(1,1),IDN(2,2),PHIP(2,2),PHIPPHIT(2,2),HM(1,2)
        REAL*8 HMHT(1,1),HMHTR(1,1),HMHTRINV(1,1),MHT(2,1),K(2,1)
        REAL*8 KH(2,2),IKH(2,2)
        INTEGER ORDER
        TS=.1
        A=1.
        W=1.
        PHIS=0.
        SIGX=1.
        ORDER=2
        OPEN(1,STATUS='UNKNOWN',FILE='DATFIL')
        OPEN(2,STATUS='UNKNOWN',FILE='COVFIL')
        T=0.
        S=0.
        H=.001
        DO 14 I=1,ORDER
        DO 14 J=1,ORDER
        PHI(I,J)=0.
        P(I,J)=0.
        Q(I,J)=0.
        IDN(I,J)=0.
14      CONTINUE
        RMAT(1,1)=SIGX**2
```

(continued)

Listing 10.2 (*Continued*)

```
IDN(1,1)=1.
IDN(2,2)=1.
PHIH=0.
WH=2.
P(1,1)=0.**2
P(2,2)=(W-WH)**2
XT=0.
XTD=A*W
WHILE(T<=20.)
        XTOLD=XT
        XTDOLD=XTD
        XTDD=-W*W*XT
        XT=XT+H*XTD
        XTD=XTD+H*XTDD
        T=T+H
        XTDD=-W*W*XT
        XT=.5*(XTOLD+XT+H*XTD)
        XTD=.5*(XTDOLD+XTD+H*XTDD)
        S=S+H
        IF(S>=(TS-.00001))THEN
                S=0.
                PHI(1,1)=1.
                PHI(1,2)=TS
                PHI(2,2)=1.
                Q(1,1)=TS*TS*TS*PHIS/3.
                Q(1,2)=.5*TS*TS*PHIS
                Q(2,1)=Q(1,2)
                Q(2,2)=PHIS*TS
                PHIB=PHIH+WH*TS
                HMAT(1,1)=COS(PHIB)
                HMAT(1,2)=0.
                CALL MATTRN(PHI,ORDER,ORDER,PHIT)
                CALL MATTRN(HMAT,1,ORDER,HT)
                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)
                CALL MATADD(HMHT,ORDER,ORDER,RMAT,
                  HMHTR)
                HMHTRINV(1,1)=1./HMHTR(1,1)
                CALL MATMUL(M,ORDER,ORDER,HT,ORDER,1,
                  MHT)
                CALL MATMUL(MHT,ORDER,1,HMHTRINV,1,1,K)
                CALL MATMUL(K,ORDER,1,HMAT,1,ORDER,KH)
                CALL MATSUB(IDN,ORDER,ORDER,KH,IKH)
                CALL MATMUL(IKH,ORDER,ORDER,M,ORDER,
```

(*continued*)

Listing 10.2 *(Continued)*

```
      ORDER,P)
      CALL GAUSS(XTNOISE,SIGX)
      XTMEAS=XT+XTNOISE
      RES=XTMEAS-A*SIN(PHIB)
      PHIH=PHIB+K(1,1)*RES
      WH=WH+K(2,1)*RES
      PHIREAL=W*T
      ERRPHI=PHIREAL-PHIH
      SP11=SQRT(P(1,1))
      ERRW=W-WH
      SP22=SQRT(P(2,2))
      XTH=A*SIN(PHIH)
      XTDH=A*WH*COS(PHIH)
      WRITE(9,*)T,XT,XTH,XTD,XTDH,W,WH,PHI,PHIH
      WRITE(1,*)T,XT,XTH,XTD,XTDH,W,WH,PHI,PHIH
      WRITE(2,*)T,ERRPHI,SP11,-SP11,ERRW,SP22,-SP22
    ENDIF
  END DO
  PAUSE
  CLOSE(1)
  CLOSE(2)
  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 10.2 was run, and we can see from Fig. 10.13 that the new two-state extended Kalman filter appears to be working because it is able to estimate the positive frequency when the filter frequency estimate is initialized positive. However, under these idealized initialization conditions the preceding extended Kalman filter also had similar performance (see Fig 10.1).

Another case was run in which the actual frequency was negative and the initial frequency estimate was positive. We can see from Fig. 10.14 that now the new two-state extended Kalman filter is able to estimate the negative frequency quite well under these circumstances, whereas the preceding three-state extended Kalman filter was unable to estimate the negative frequency under similar circumstances (see Fig. 10.3). Thus, it appears that when the amplitude of the sinusoid is known in advance it is possible to estimate the frequency of the sinusoid.

Figures 10.15 and 10.16 complete further experiments in which we are attempting to estimate the frequency for various filter frequency initialization conditions. As expected, Fig. 10.15 shows that a negative frequency can be easily estimated when the initial filter frequency estimate is negative. This is not

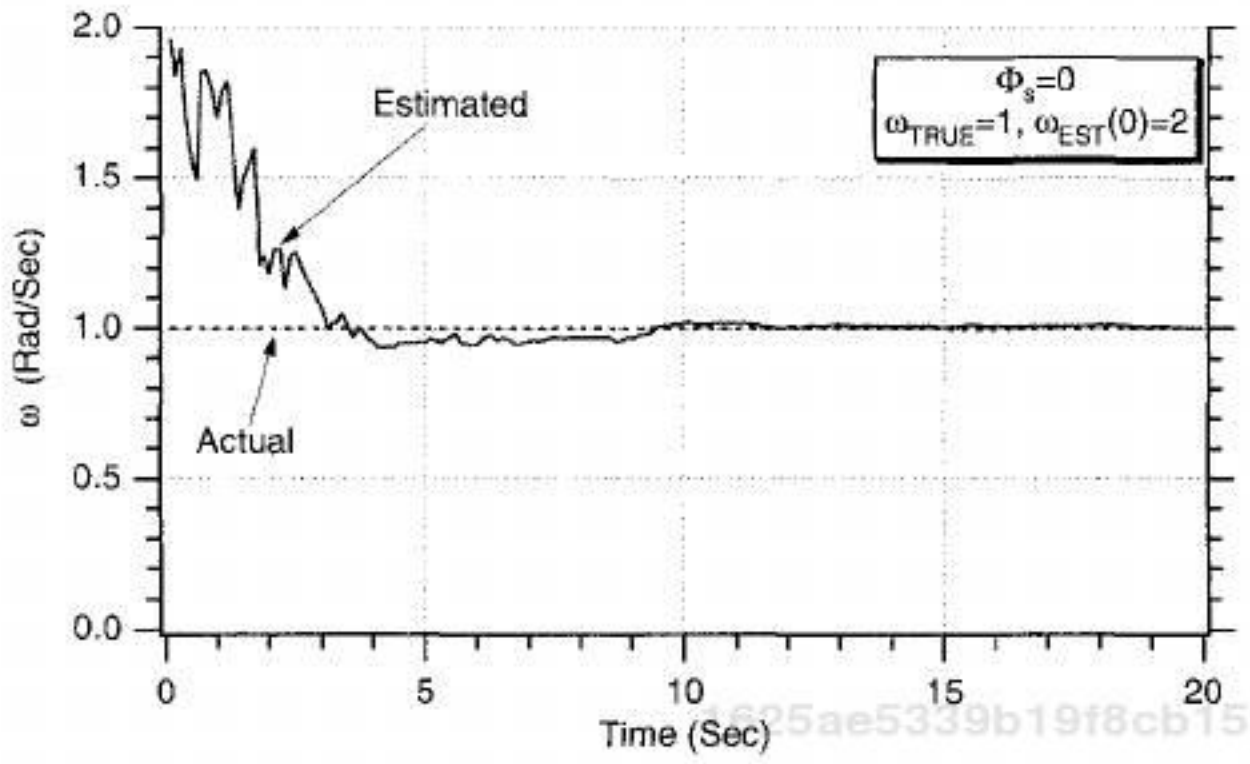

Fig. 10.13 Two-state extended Kalman filter estimates positive frequency when initial frequency estimate is also positive.

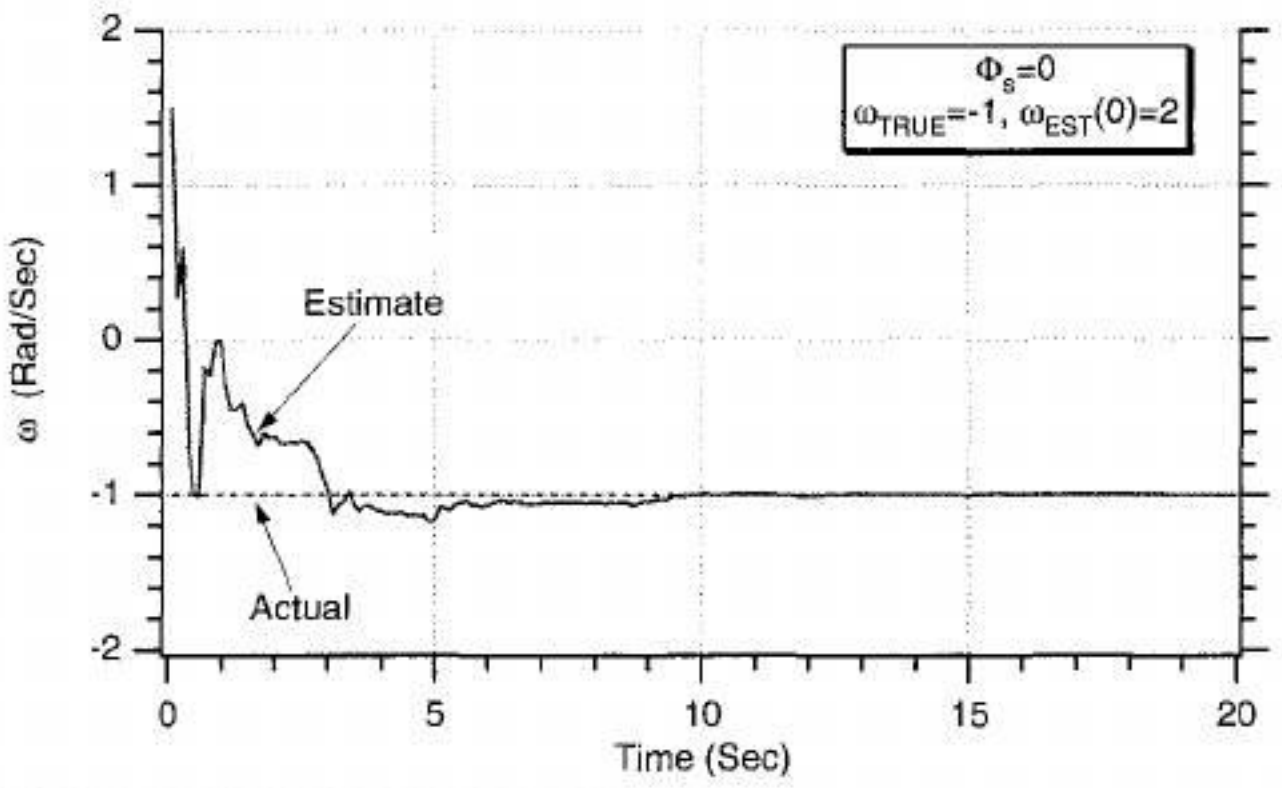

Fig. 10.14 New two-state extended Kalman filter estimates negative frequency when initialized positive.

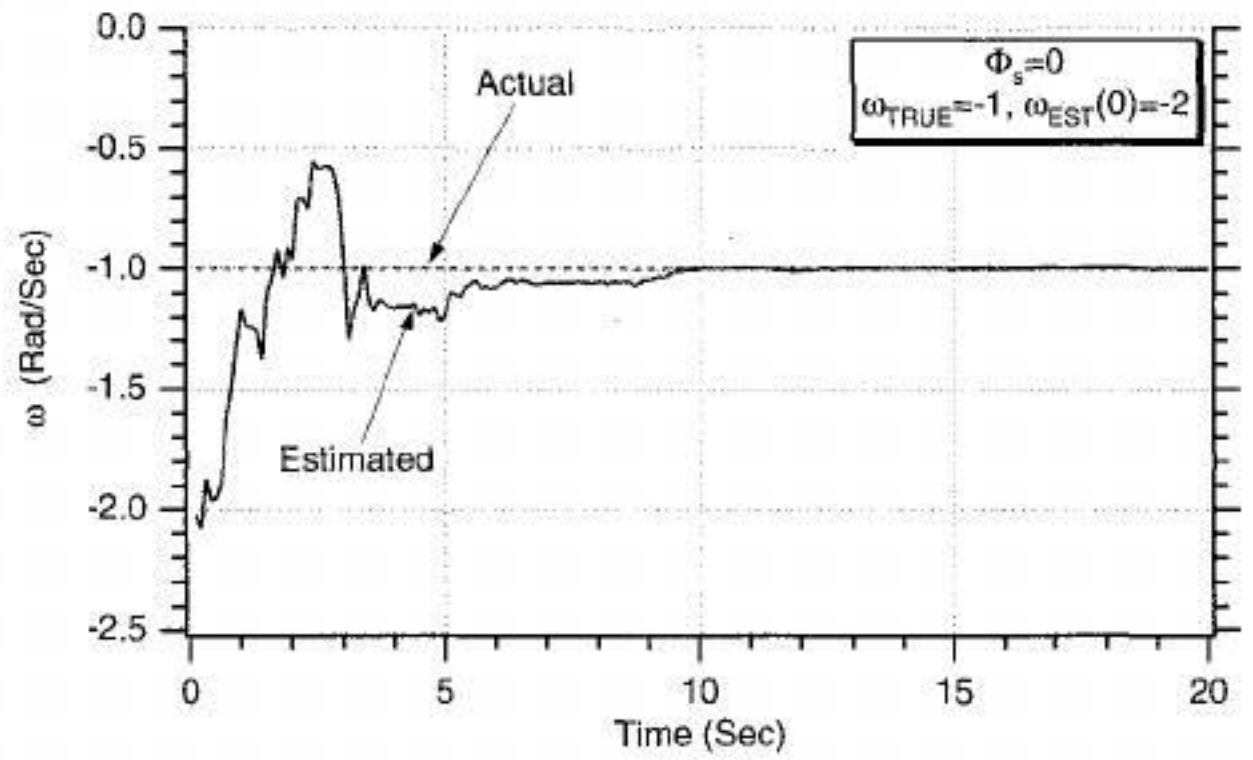

Fig. 10.15 New two-state extended Kalman filter estimates negative frequency when initialized negative.

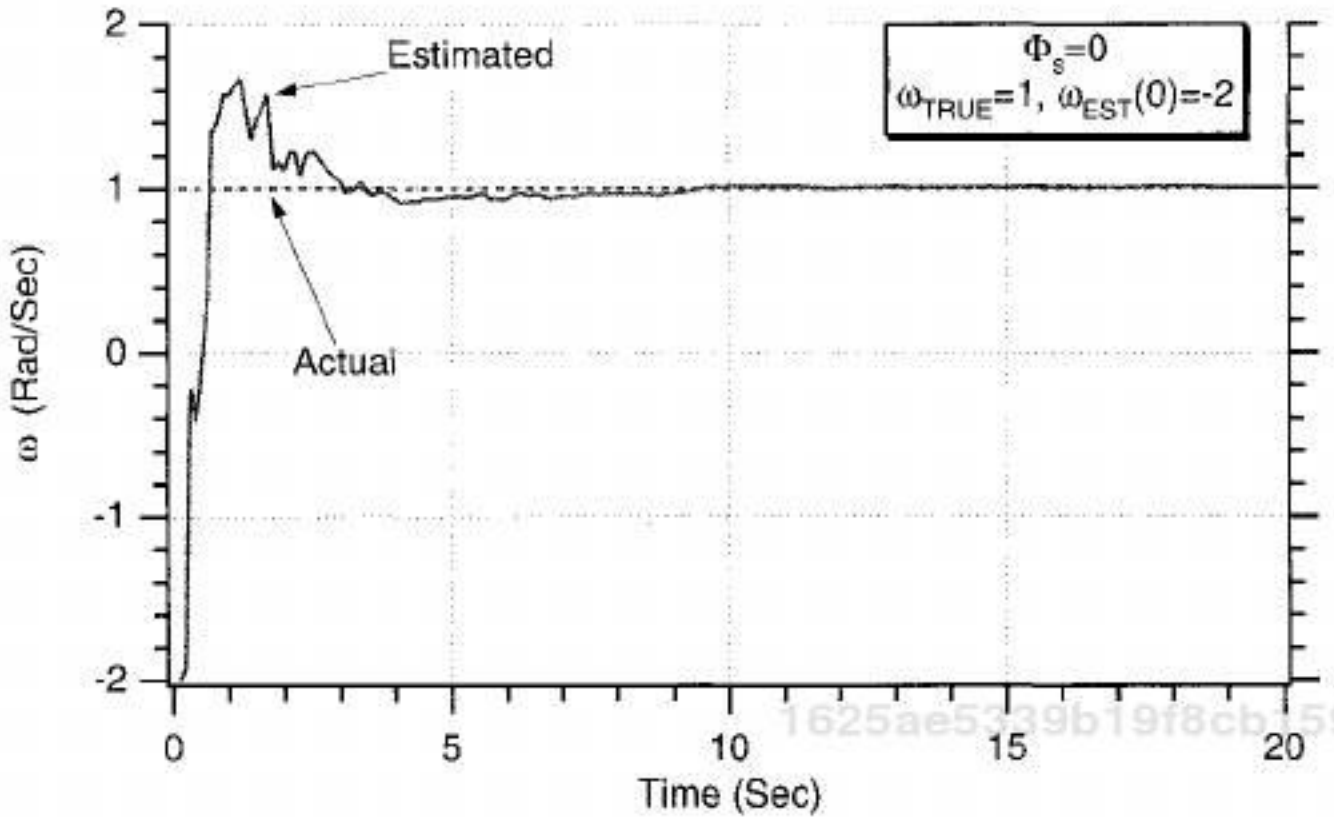

Fig. 10.16 New two-state extended Kalman filter estimates positive frequency when initialized negative.

surprising because the preceding three-state extended Kalman filter also could have accomplished this task. However, Fig. 10.16 shows that a positive frequency also can be estimated when the initial filter frequency estimate is negative. The preceding three-state extended Kalman filter would have failed at this attempt.

To see if the new two-state extended Kalman filter was really working all of the time, even more runs were made. Listing 10.2 was modified slightly so that it could be run in the Monte Carlo mode. Code was changed so that 10 runs could be made, one after another, each with a different noise stream. The first case that was examined in detail was the one in which the actual frequency of the sinusoid was negative and the filter's initial estimate of the frequency was positive. The single-run simulation results of Fig. 10.14 indicated that the filter was working. However, we can see from Fig. 10.17 that one case out of 10 failed to estimate the

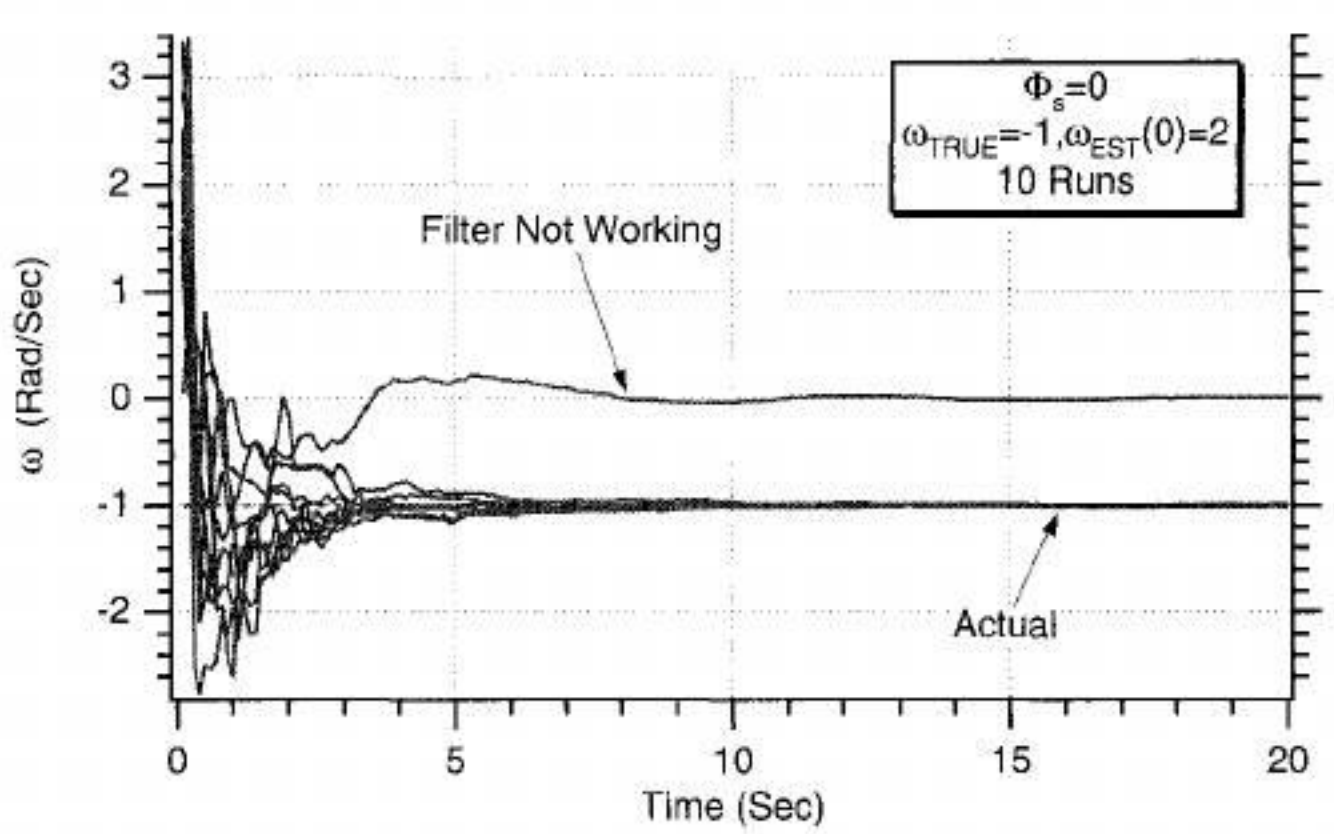

Fig. 10.17 New two-state extended Kalman filter does not estimate negative frequency all of the time when filter is initialized positive.

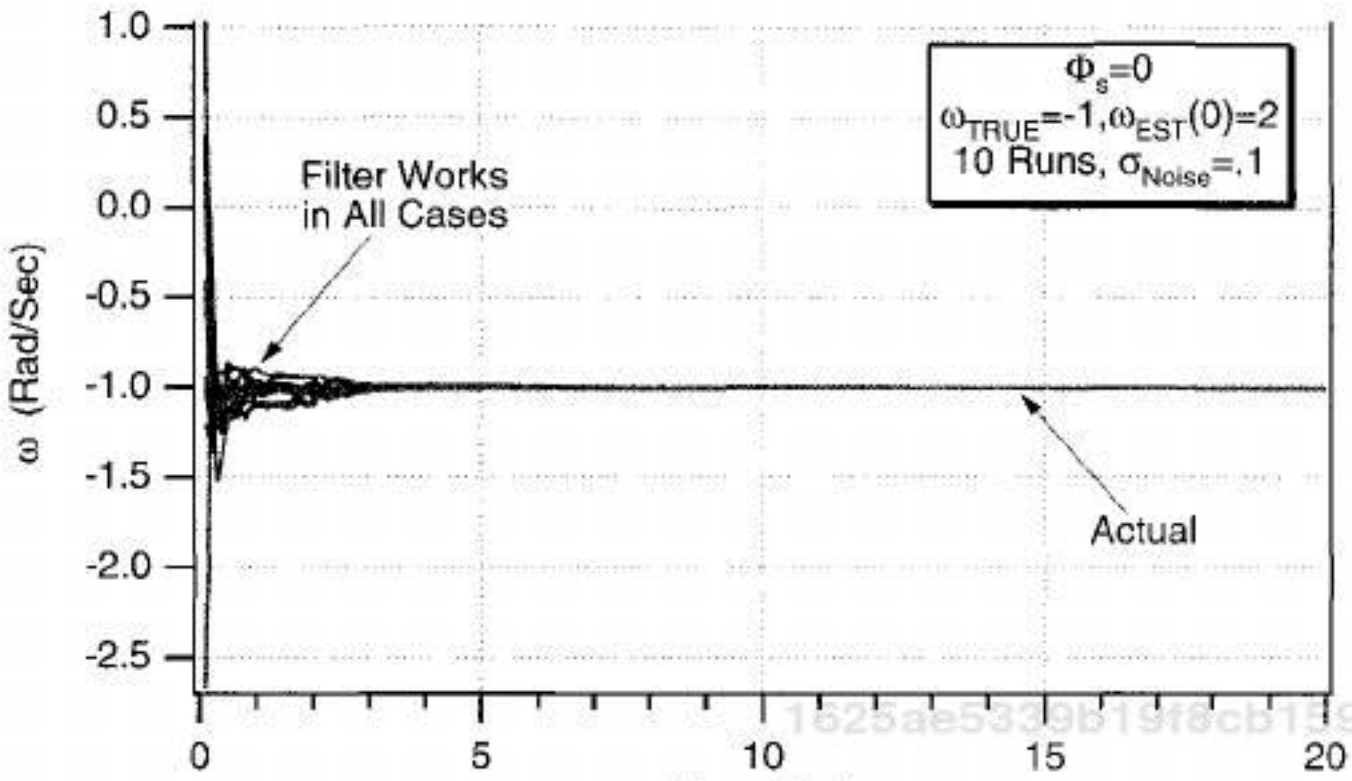

Fig. 10.18 New two-state extended Kalman filter works all of the time when the measurement noise is reduced by an order of magnitude.

negative frequency. In that particular case the filter thought the frequency was zero.

Because the filter should work all of the time, it was hypothesized that perhaps there was too much measurement noise. The standard deviation of the measurement noise was unity, and the amplitude of the sinusoidal signal was also unity. Figure 10.18 shows that when the measurement noise is reduced by an order of magnitude to 0.1 the filter is now successful in estimating the negative frequency in all 10 cases.

In the preceding figure the frequency estimates looked excellent. However, the real performance of the two-state extended Kalman filter is determined by the error in the estimate of frequency. Figure 10.19 displays the error in the estimate

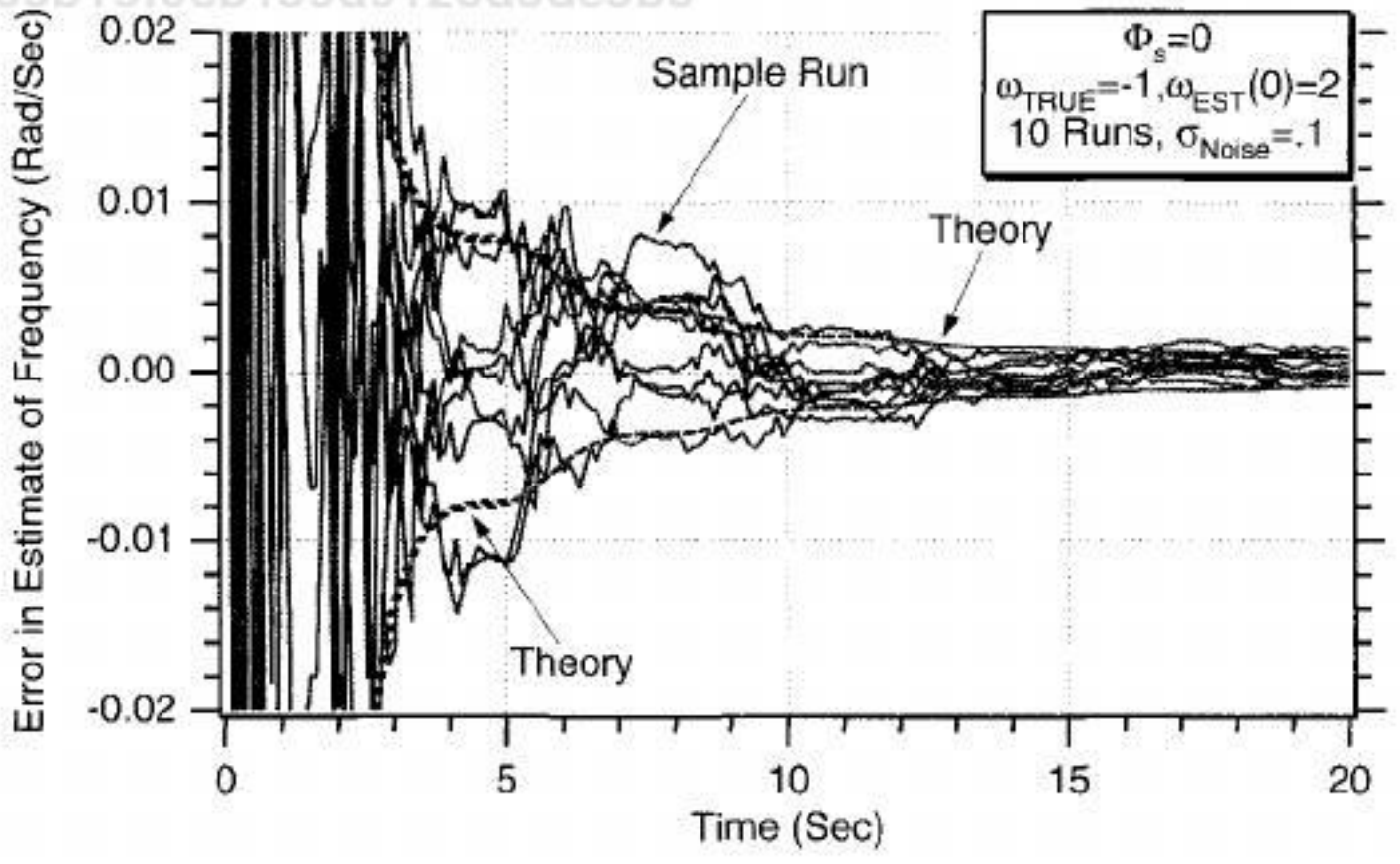

Fig. 10.19 Error in the estimate results indicate that the new two-state extended Kalman filter is able to estimate frequency.

of frequency for the case in which the actual frequency is negative, but the filter's initial estimate of frequency is positive when the measurement noise is 0.1. The theoretical error bounds for the error in the estimate of frequency, obtained by taking the square root of the second diagonal element of the covariance matrix, are also displayed in Fig. 10.19. We can see that for all 10 runs the simulated error in the estimate of frequency lies within the theoretical bounds most of the time, indicating that the new two-state extended Kalman filter is working properly.

The two-state extended Kalman filter derived in this section was an academic exercise in explaining the failure of the three-state extended Kalman filter of the preceding section. We still want to find out if an extended Kalman filter can be designed to estimate the signal states and frequency under poor initialization conditions.

Alternate Extended Kalman Filter for Sinusoidal Signal

In this section we will try again to build an extended Kalman filter that makes use of the fact that the signal is sinusoidal but also takes into account that both the frequency and amplitude of the sinusoid are also unknown.

Recall that the actual signal is given by

$$x = A \sin \omega t$$

As was just mentioned, in this model we will assume that both the amplitude A and the frequency ω of the sinusoid are unknown. If we take the derivative of the preceding equation, we also get a sinusoid or

$$\dot{x} = A\omega \cos \omega t$$

Taking the derivative again yields

$$\ddot{x} = -A\omega^2 \sin \omega t$$

Thus, we can see that the preceding second-order differential equation can be expressed in terms of the first equation or

$$\ddot{x} = -\omega^2 x$$

In other words, integrating the preceding equation twice, with the appropriate initial conditions, will yield the original sinusoidal signal. The preceding equation is especially useful because the sinusoidal term and signal amplitude have been eliminated and the second derivative of the signal or third state has been expressed in terms of the signal or first state. This is precisely what we desire for expressing the problem in state-space notation.

If we assume that the sinusoidal frequency is an unknown constant, its derivative must be zero. To account for the fact that the sinusoidal frequency may not be a constant, it is prudent to add process noise to the derivative of

frequency. Therefore, the two differential equations representing the real world in this example are

$$\ddot{x} = -\omega^2 x$$
$$\dot{\omega} = u_s$$

where u_s is white process noise with spectral density Φ_s. The preceding differential equations can be expressed as three first-order differential equations (i.e., one of the equations is redundant) in state-space form as

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -\omega^2 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \omega \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ u_s \end{bmatrix}$$

The preceding equation is nonlinear because a state also shows up in the 3×3 matrix multiplying the state vector. From the preceding equation we can see that the systems dynamics matrix turns out to be

$$\boldsymbol{F} = \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} = \begin{bmatrix} \dfrac{\partial \dot{x}}{\partial x} & \dfrac{\partial \dot{x}}{\partial \dot{x}} & \dfrac{\partial \dot{x}}{\partial \omega} \\ \dfrac{\partial \ddot{x}}{\partial x} & \dfrac{\partial \ddot{x}}{\partial \dot{x}} & \dfrac{\partial \ddot{x}}{\partial \omega} \\ \dfrac{\partial \dot{\omega}}{\partial x} & \dfrac{\partial \dot{\omega}}{\partial \dot{x}} & \dfrac{\partial \dot{\omega}}{\partial \omega} \end{bmatrix}$$

where the partial derivatives are evaluated at the current estimates. Taking the partial derivatives in this example can be done by inspection, and the resultant systems dynamics matrix is given by

$$\boldsymbol{F} = \begin{bmatrix} 0 & 1 & 0 \\ -\hat{\omega}^2 & 0 & -2\hat{\omega}\hat{x} \\ 0 & 0 & 0 \end{bmatrix}$$

where the terms in the systems dynamics matrix have been evaluated at the current state estimates. In this example the exact fundamental matrix will be difficult, if not impossible, to find. If we assume that the elements of the systems dynamics matrix are approximately constant between sampling instants, then we use a two-term Taylor-series approximation for the fundamental matrix, yielding

$$\Phi(t) \approx I + \boldsymbol{F}t = \begin{bmatrix} 1 & t & 0 \\ -\hat{\omega}^2 t & 1 & -2\hat{\omega}\hat{x}t \\ 0 & 0 & 1 \end{bmatrix}$$

Therefore, the discrete fundamental matrix can be obtained by substituting T_s for t or

$$\Phi_k \approx \begin{bmatrix} 1 & T_s & 0 \\ -\hat{\omega}_{k-1}^2 T_s & 1 & -2\hat{\omega}_{k-1}\hat{x}_{k-1} T_s \\ 0 & 0 & 1 \end{bmatrix}$$

The continuous process-noise matrix can by found from the original state-space equation to be

$$\boldsymbol{Q} = E(ww^T) = E\left[\begin{bmatrix} 0 \\ 0 \\ u_s \end{bmatrix} [0 \quad 0 \quad u_s]\right] = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \Phi_s \end{bmatrix}$$

Recall that the discrete process-noise matrix can be found from the continuous process-noise matrix according to

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

By substitution of the appropriate matrices into the preceding equation, we obtain

$$\boldsymbol{Q}_k = \int_0^{T_s} \begin{bmatrix} 1 & \tau & 0 \\ -\hat{\omega}^2\tau & 1 & -2\hat{\omega}\hat{x}\tau \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \Phi_s \end{bmatrix} \begin{bmatrix} 1 & -\hat{\omega}^2\tau & 0 \\ \tau & 1 & 0 \\ 0 & -2\hat{\omega}\hat{x}\tau & 1 \end{bmatrix} \mathrm{d}\tau$$

Multiplying out the three matrices yields

$$\boldsymbol{Q}_k = \int_0^{T_s} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 4\hat{\omega}^2\hat{x}^2\tau^2\Phi_s & -2\hat{\omega}\hat{x}\tau\Phi_s \\ 0 & -2\hat{\omega}\hat{x}\tau\Phi_s & \Phi_s \end{bmatrix} \mathrm{d}\tau$$

If we again assume that the states are approximately constant between the sampling intervals, then the preceding integral can easily be evaluated as

$$\boldsymbol{Q}_k = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1.333\hat{\omega}^2\hat{x}^2 T_s^3\Phi_s & -\hat{\omega}\hat{x}T_s^2\Phi_s \\ 0 & -\hat{\omega}\hat{x}T_s^2\Phi_s & T_s\Phi_s \end{bmatrix}$$

In this problem we are assuming that the measurement is of the first state plus noise or

$$x_k^* = x_k + v_k$$

Therefore, the measurement is linearly related to the states according to

$$x_k^* = [1 \quad 0 \quad 0]\begin{bmatrix} x \\ \dot{x} \\ \omega \end{bmatrix} + v_k$$

The measurement matrix can be obtained from the preceding equation by inspection as

$$\boldsymbol{H} = [1 \quad 0 \quad 0]$$

In this example the discrete measurement noise matrix is a scalar and is given by

$$\boldsymbol{R}_k = E(\boldsymbol{v}_k \boldsymbol{v}_k^T) = \sigma_x^2$$

We now have enough information to solve the matrix Riccati equations for the Kalman gains.

For this example the projected states in the actual extended Kalman-filtering equations do not have to use the approximation for the fundamental matrix. Instead the state projections, indicated by an overbar, can be obtained by numerically integrating the nonlinear differential equations over the sampling interval. Therefore, the extended Kalman-filtering equations can then be written as

$$\hat{x}_k = \bar{x}_k + K_{1_k}(x_k^* - \bar{x}_k)$$
$$\hat{\dot{x}}_k = \bar{\dot{x}}_k + K_{2_k}(x_k^* - \bar{x}_k)$$
$$\hat{\omega}_k = \hat{\omega}_{k-1} + K_{3_k}(x_k^* - \bar{x}_k)$$

Listing 10.3 presents the alternate three-state extended Kalman filter for estimating the states of a noisy sinusoidal signal whose frequency is unknown. We can see once again that the simulation is initially set up to run without process noise (i.e., PHIS=0). The actual sinusoidal signal has unity amplitude, and the standard deviation of the measurement noise is also unity. We can see from Listing 10.3 that the frequency of the actual sinusoidal signal is also unity. The filter's initial state estimates of x and $\dot{x}$ are set to zero, while the initial estimate of the frequency is set to two (rather than unity). In other words, the second and third states are mismatched from the real world. Values are used for the initial covariance matrix to reflect the uncertainties in our initial state estimates. As was the case in Chapter 9, subroutine PROJECT is used to integrate the nonlinear equations using Euler integration over the sampling interval to obtain the state projections for x and $\dot{x}$.

A case was run using the nominal values of Listing 10.3 (i.e., no process noise), and we can see from Figs. 10.20–10.22 that the actual quantities and their estimates are quite close. An additional comparison of the results of our alternate three-state extended Kalman filter with the results of the first- and second-order polynomial Kalman filters of Chapter 5 indicates that we are estimating the states more accurately than those imperfect linear filters. However, our state estimates

Listing 10.3 Three-state alternate extended Kalman filter for sinusoidal signal with unknown frequency

```
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,O-Z)
        REAL*8 P(3,3),Q(3,3),M(3,3),PHI(3,3),HMAT(1,3),HT(3,1),PHIT(3,3)
        REAL*8 RMAT(1,1),IDN(3,3),PHIP(3,3),PHIPPHIT(3,3),HM(1,3)
        REAL*8 HMHT(1,1),HMHTR(1,1),HMHTRINV(1,1),MHT(3,1),K(3,1),
          F(3,3)
        REAL*8 KH(3,3),IKH(3,3)
        INTEGER ORDER
        HP=.001
        W=1.
        A=1.
        TS=.1
        ORDER=3
        PHIS=0.
        SIGX=1.
        OPEN(1,STATUS='UNKNOWN',FILE='DATFIL')
        OPEN(2,STATUS='UNKNOWN',FILE='COVFIL')
        T=0.
        S=0.
        H=.001
        DO 14 I=1,ORDER
        DO 14 J=1,ORDER
        F(I,J)=0.
        PHI(I,J)=0.
        P(I,J)=0.
        Q(I,J)=0.
        IDN(I,J)=0.
14      CONTINUE
        RMAT(1,1)=SIGX**2
        IDN(1,1)=1.
        IDN(2,2)=1.
        IDN(3,3)=1.
        P(1,1)=SIGX**2
        P(2,2)=2.**2
        P(3,3)=2.**2
        XTH=0.
        XTDH=0.
        WH=2.
        XT=0.
        XTD=A*W
        WHILE(T<=20.)
              XTOLD=XT
              XTDOLD=XTD
```

(continued)

Listing 10.3 *(Continued)*

```
XTDD=-W*W*XT
XT=XT+H*XTD
XTD=XTD+H*XTDD
T=T+H
XTDD=-W*W*XT
XT=.5*(XTOLD+XT+H*XTD)
XTD=.5*(XTDOLD+XTD+H*XTDD)
S=S+H
IF(S>=(TS-.00001))THEN
        S=0.
        F(1,2)=1.
        F(2,1)=-WH**2
        F(2,3)=-2.*WH*XTH
        PHI(1,1)=1.
        PHI(1,2)=TS
        PHI(2,1)=-WH*WH*TS
        PHI(2,2)=1.
        PHI(2,3)=-2.*WH*XTH*TS
        PHI(3,3)=1.
        Q(2,2)=4.*WH*WH*XTH*XTH*TS*TS*TS*PHIS/3.
        Q(2,3)=-2.*WH*XTH*TS*TS*PHIS/2.
        Q(3,2)=Q(2,3)
        Q(3,3)=PHIS*TS
        HMAT(1,1)=1.
        HMAT(1,2)=0.
        HMAT(1,3)=0.
        CALL MATTRN(PHI,ORDER,ORDER,PHIT)
        CALL MATTRN(HMAT,1,ORDER,HT)
        CALL MATMUL(PHI,ORDER,ORDER,P,ORDER,
          ORDER,PHIP)
        CALL MATMUL(PHIP,ORDER,ORDER,PHIT,ORDER,
          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)
        CALL MATADD(HMHT,ORDER,ORDER,RMAT,
          HMHTR)
        HMHTRINV(1,1)=1./HMHTR(1,1)
        CALL MATMUL(M,ORDER,ORDER,HT,ORDER,1,
          MHT)
        CALL MATMUL(MHT,ORDER,1,HMHTRINV,1,1,K)
        CALL MATMUL(K,ORDER,1,HMAT,1,ORDER,KH)
        CALL MATSUB(IDN,ORDER,ORDER,KH,IKH)
        CALL MATMUL(IKH,ORDER,ORDER,M,ORDER,
          ORDER,P)
        CALL GAUSS(XTNOISE,SIGX)
        XTMEAS=XT+XTNOISE
        CALL PROJECT(T,TS,XTH,XTDH,XTB,XTDB,HP,WH)
        RES=XTMEAS-XTB
```

(continued)

Listing 10.3 *(Continued)*

```
                XTH=XTB+K(1,1)*RES
                XTDH=XTDB+K(2,1)*RES
                WH=WH+K(3,1)*RES
                ERRX=XT-XTH
                SP11=SQRT(P(1,1))
                ERRXD=XTD-XTDH
                SP22=SQRT(P(2,2))
                ERRW=W-WH
                SP33=SQRT(P(3,3))
                WRITE(9,*)T,XT,XTH,XTD,XTDH,W,WH
                WRITE(1,*)T,XT,XTH,XTD,XTDH,W,WH
                WRITE(2,*)T,ERRX,SP11,-SP11,ERRXD,SP22,-SP22,
                  ERRW,SP33,-SP33
1               ENDIF
            END DO
            PAUSE
            CLOSE(1)
            CLOSE(2)
            END

            SUBROUTINE PROJECT(TP,TS,XTP,XTDP,XTH,XTDH,HP,W)
            IMPLICIT REAL*8 (A-H)
            IMPLICIT REAL*8 (O-Z)
            T=0.
            XT=XTP
            XTD=XTDP
            H=HP
            WHILE(T<=(TS-.0001))
                  XTDD=-W*W*XT
                  XTD=XTD+H*XTDD
                  XT=XT+H*XTD
                  T=T+H
            END DO
            XTH=XT
            XTDH=XTD
            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
```

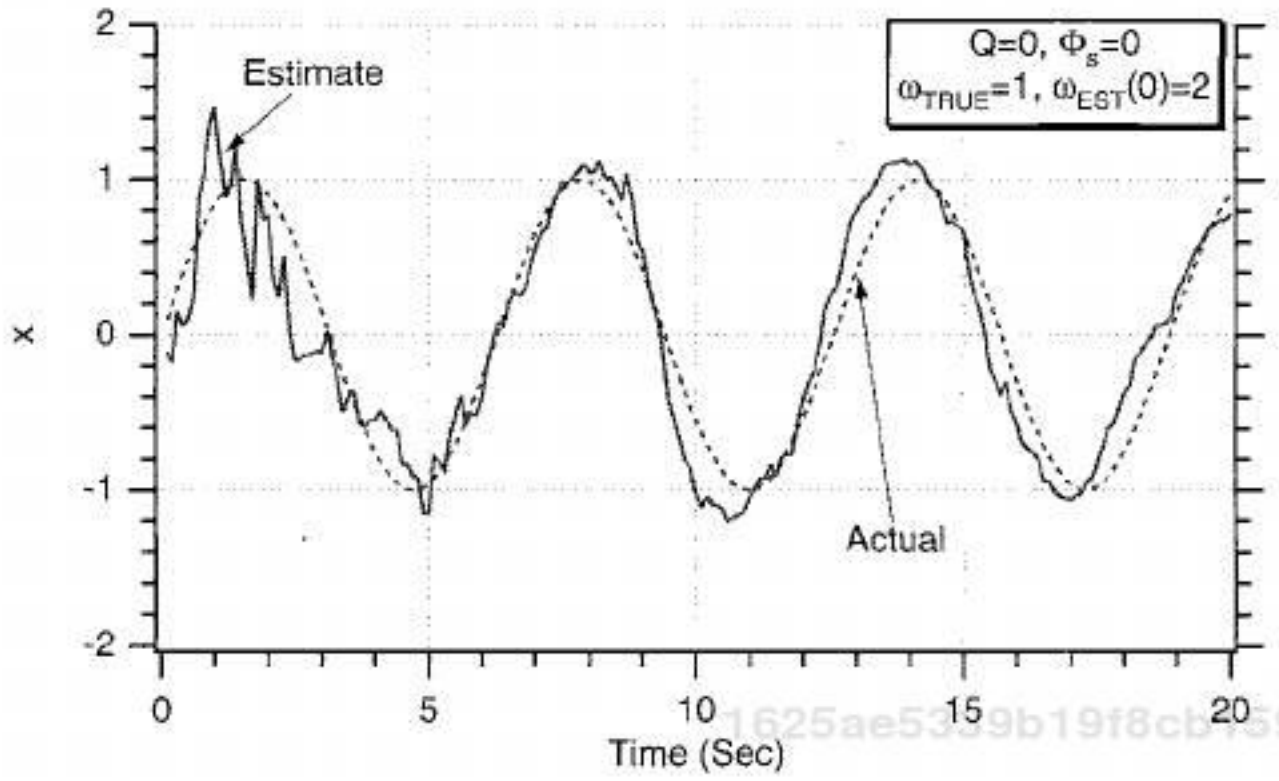

Fig. 10.20 Alternate three-state extended Kalman filter estimates first state well.

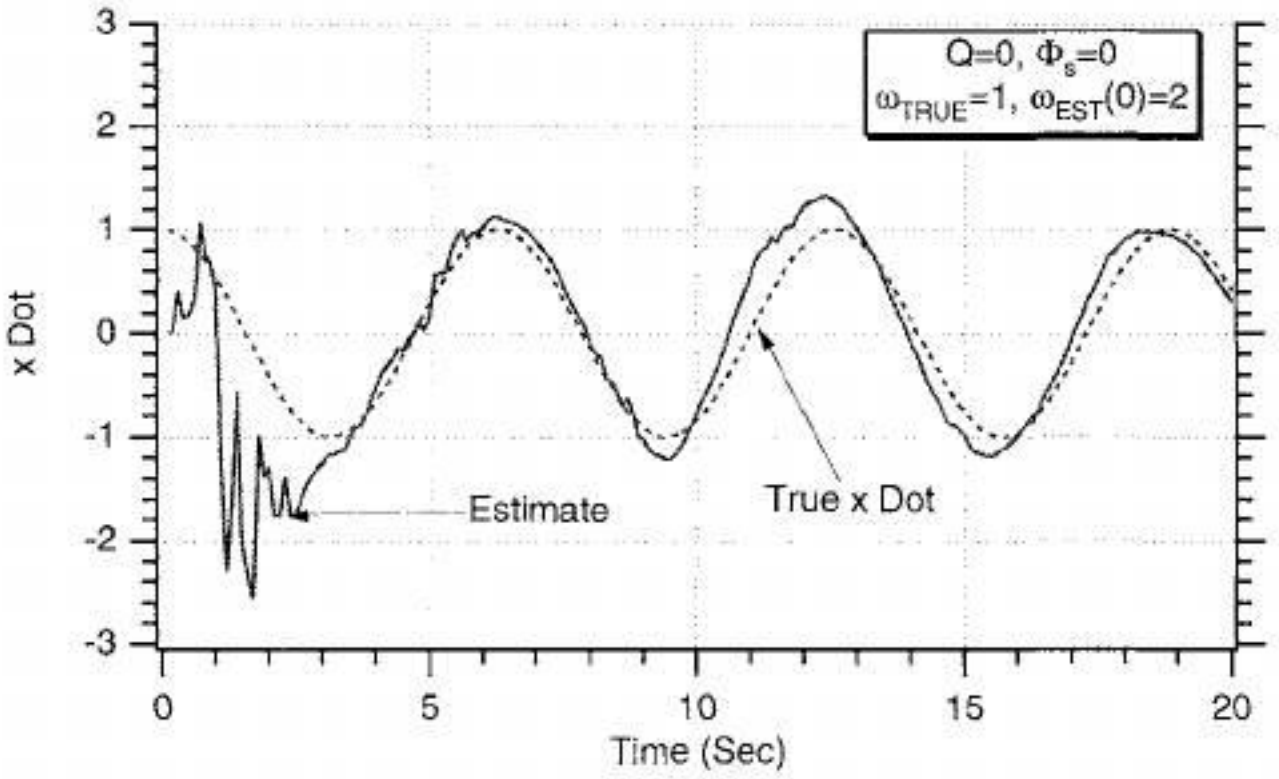

Fig. 10.21 Alternate three-state extended Kalman filter estimates second state well, even though filter is not initialized correctly.

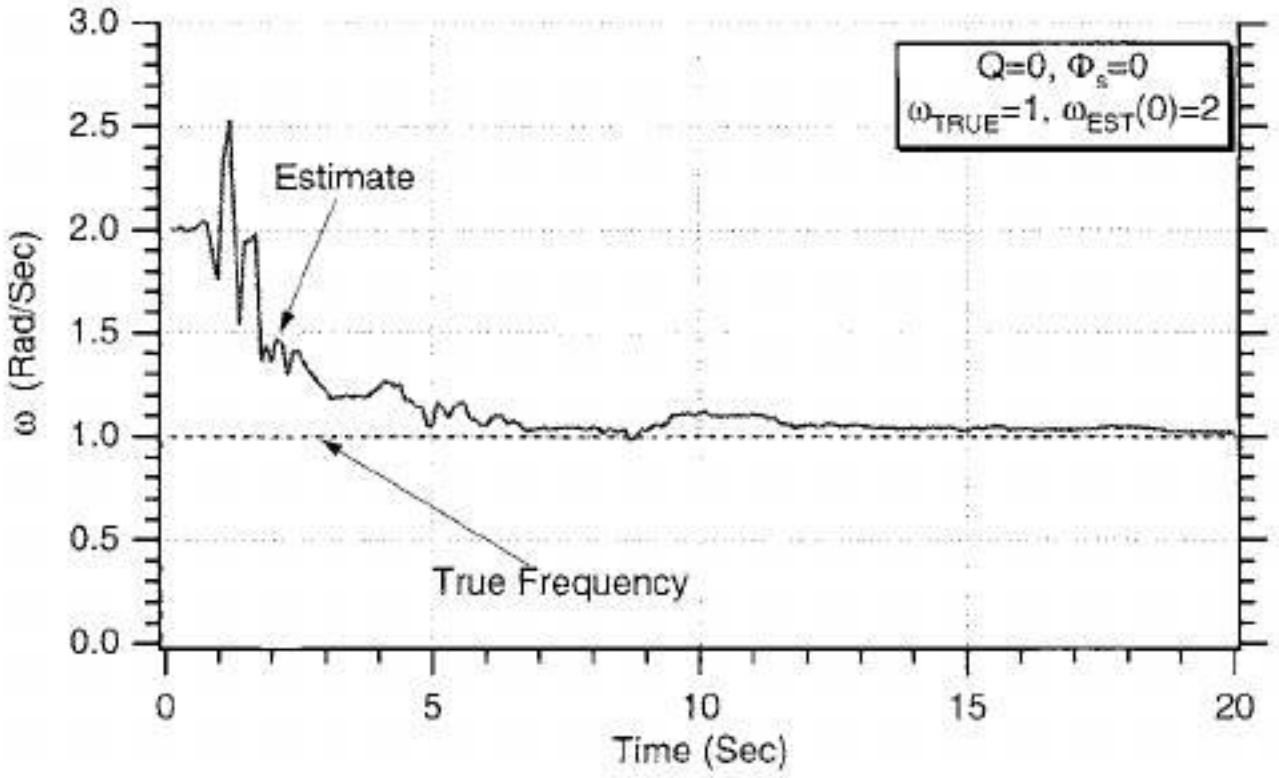

Fig. 10.22 Alternate three-state extended Kalman filter appears able to estimate the frequency of the sinusoid even though filter is not initialized correctly.

are not quite as good as those obtained from the linear sinusoidal Kalman filter of Chapter 5 (i.e., the one in which the fundamental matrix is exact) in which the frequency is known a priori. However, we are estimating the states far more accurately than the linear sinusoidal Kalman filter of Chapter 5, when the frequency is in error. From Fig. 10.22 we can see that it takes approximately 5 s to obtain an accurate estimate of the frequency for the parameters considered.

The fact that the new alternate three-state extended Kalman filter appears to be working under the benign initialization condition means that the filter is now ready for further testing. We will try to make the filter fail, as was done in the preceding two sections of this chapter. We will now consider the more difficult condition where the actual frequency is negative but the initial filter frequency estimate is positive. At first glance Fig. 10.23 appears to be telling us that the new filter also has failed in estimating the frequency. However, a closer examination of Fig. 10.23 indicates that the magnitude of the frequency has been estimated correctly, but the sign has not been estimated correctly.

To see if it is of any practical utility to only be able to estimate the magnitude of the frequency, more information is displayed in Figs. 10.24 and 10.25. Figure 10.24 compares the actual and estimated signal x, whereas Fig. 10.25 compares the actual and estimated derivative of the signal or $\dot{x}$. We can see that in both figures the estimates and actual quantities are very close even though the sign of the estimated frequency magnitude is not correct, indicating that the filter is working properly.

Another experiment was conducted in which the actual frequency of the signal was minus one, but the initial estimate of the frequency was minus two. Under these conditions we do not expect any problems, and Fig. 10.26 confirms that the alternate three-state extended Kalman filter estimates the frequency quite well.

Another experiment was conducted in which the actual frequency of the signal was plus one, but the initial estimate of the frequency was minus two. Under these conditions we expect problems, and Fig. 10.27 indicates that the alternate three-

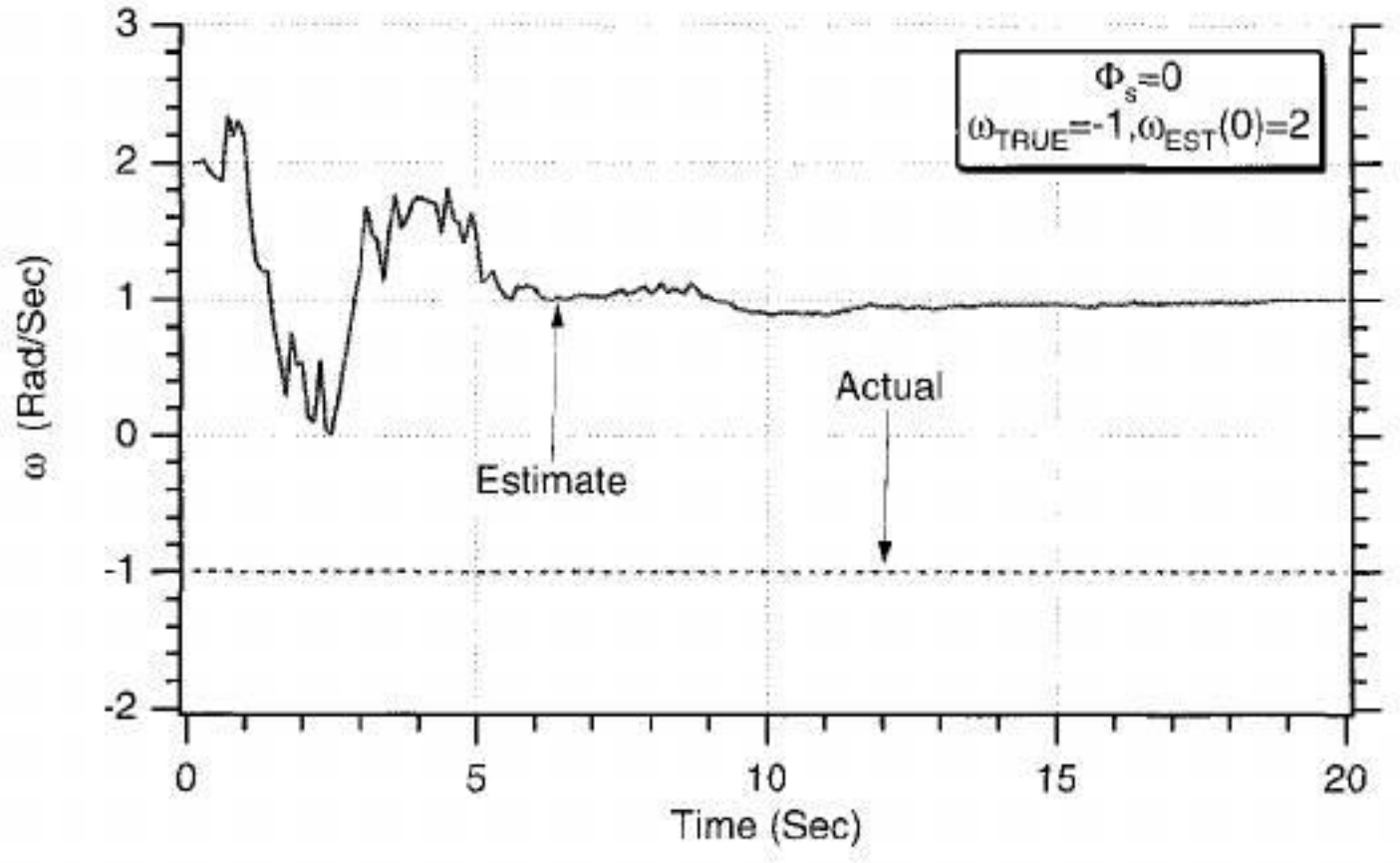

Fig. 10.23 Alternate three-state extended Kalman filter appears able to estimate the magnitude of the frequency but not its sign when the filter is not initialized correctly.

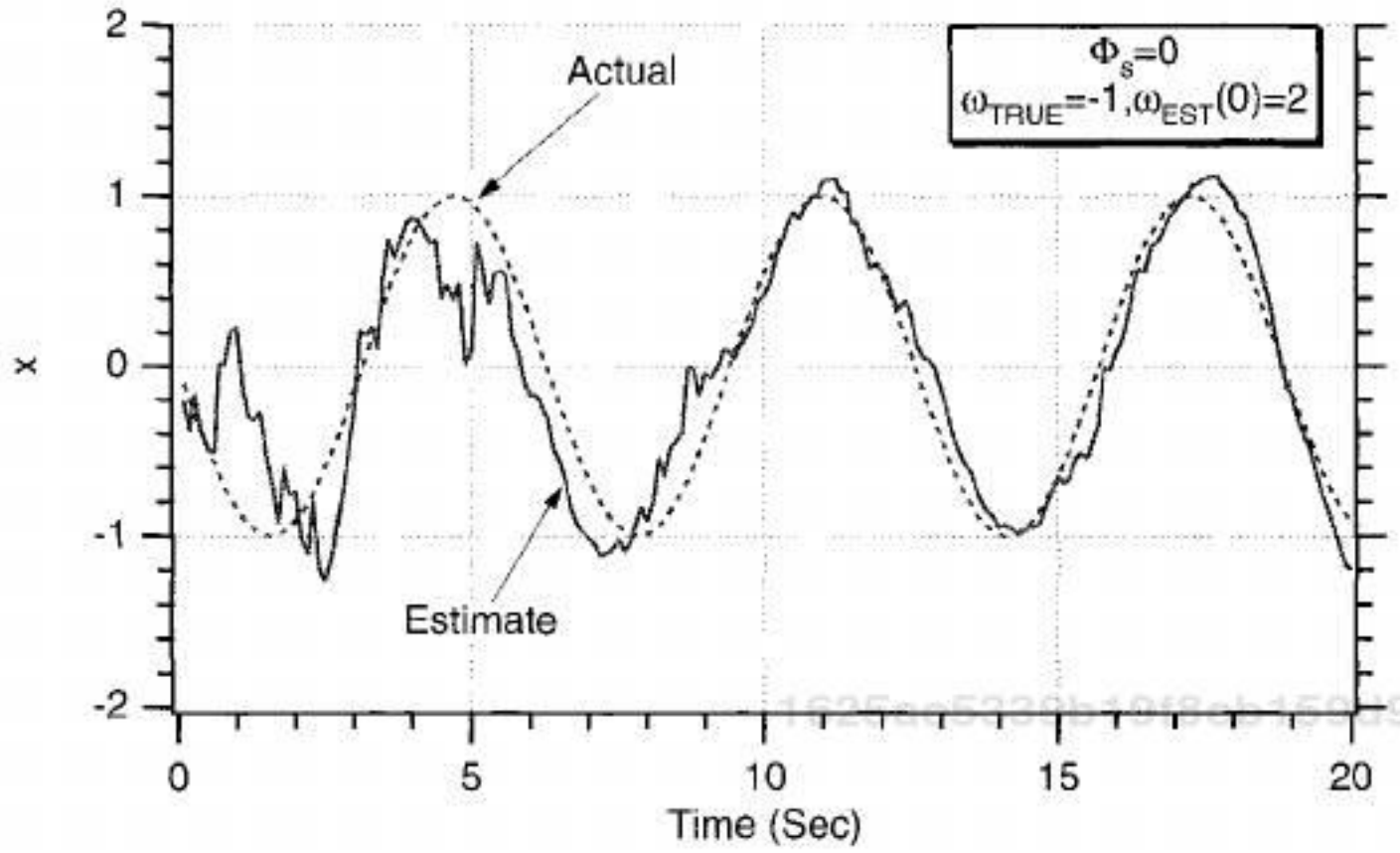

Fig. 10.24 Alternate three-state extended Kalman filter appears able to estimate the signal when the filter is not initialized correctly.

state extended Kalman filter estimates the magnitude of the frequency quite well but estimates the wrong sign.

To again see if it is of any practical utility to be able to estimate the magnitude of the frequency, more information is displayed in Figs. 10.28 and 10.29. Figure 10.28 compares the actual and estimated signal x, whereas Fig. 10.29 compares the actual and estimated derivative of the signal or $\dot{x}$. We can see that in both figures the estimates and actual quantities are very close, indicating that the filter is working properly even though we are not able to estimate the sign of the frequency.

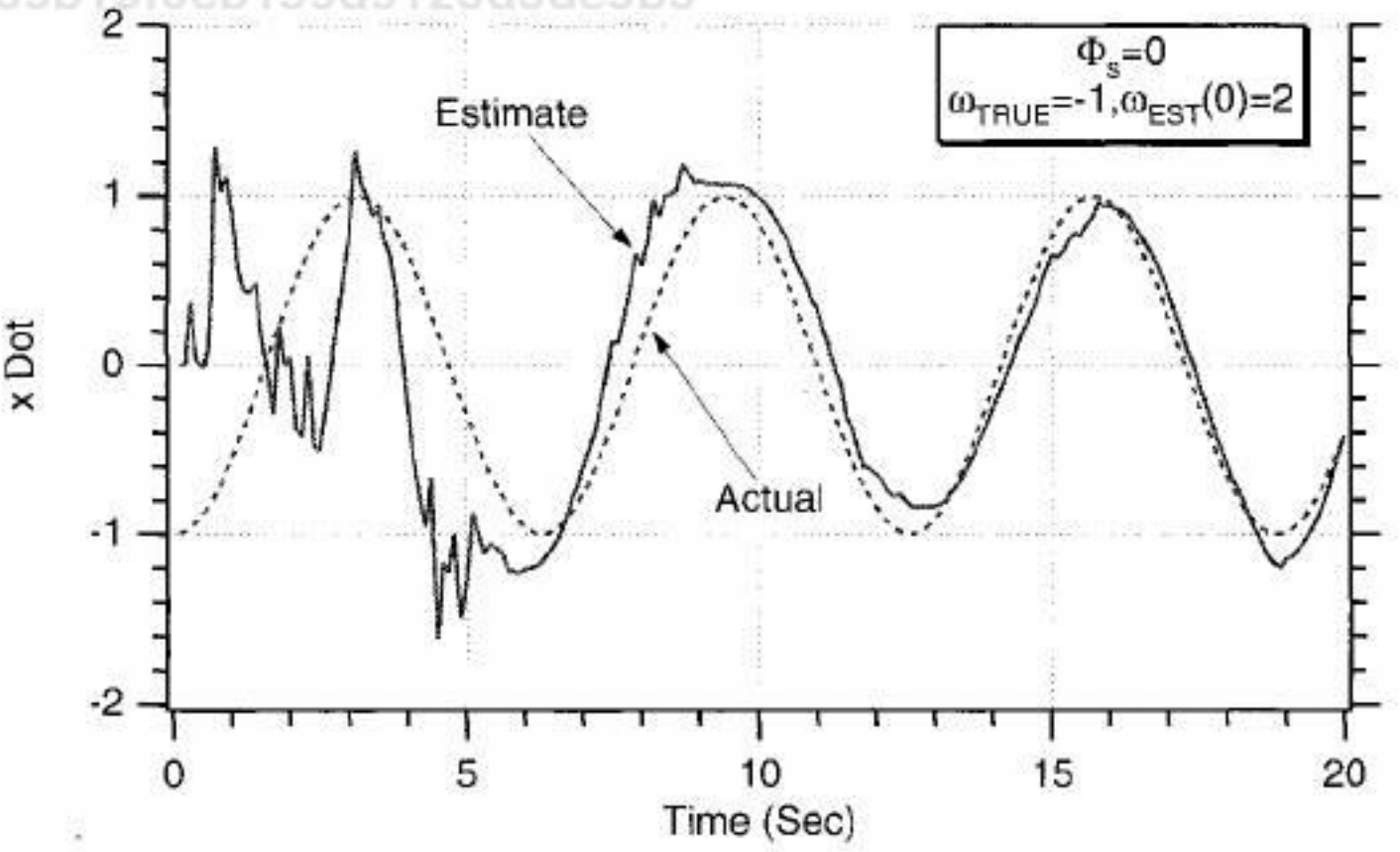

Fig. 10.25 Alternate three-state extended Kalman filter appears able to estimate the derivative of the signal when filter is not initialized correctly.

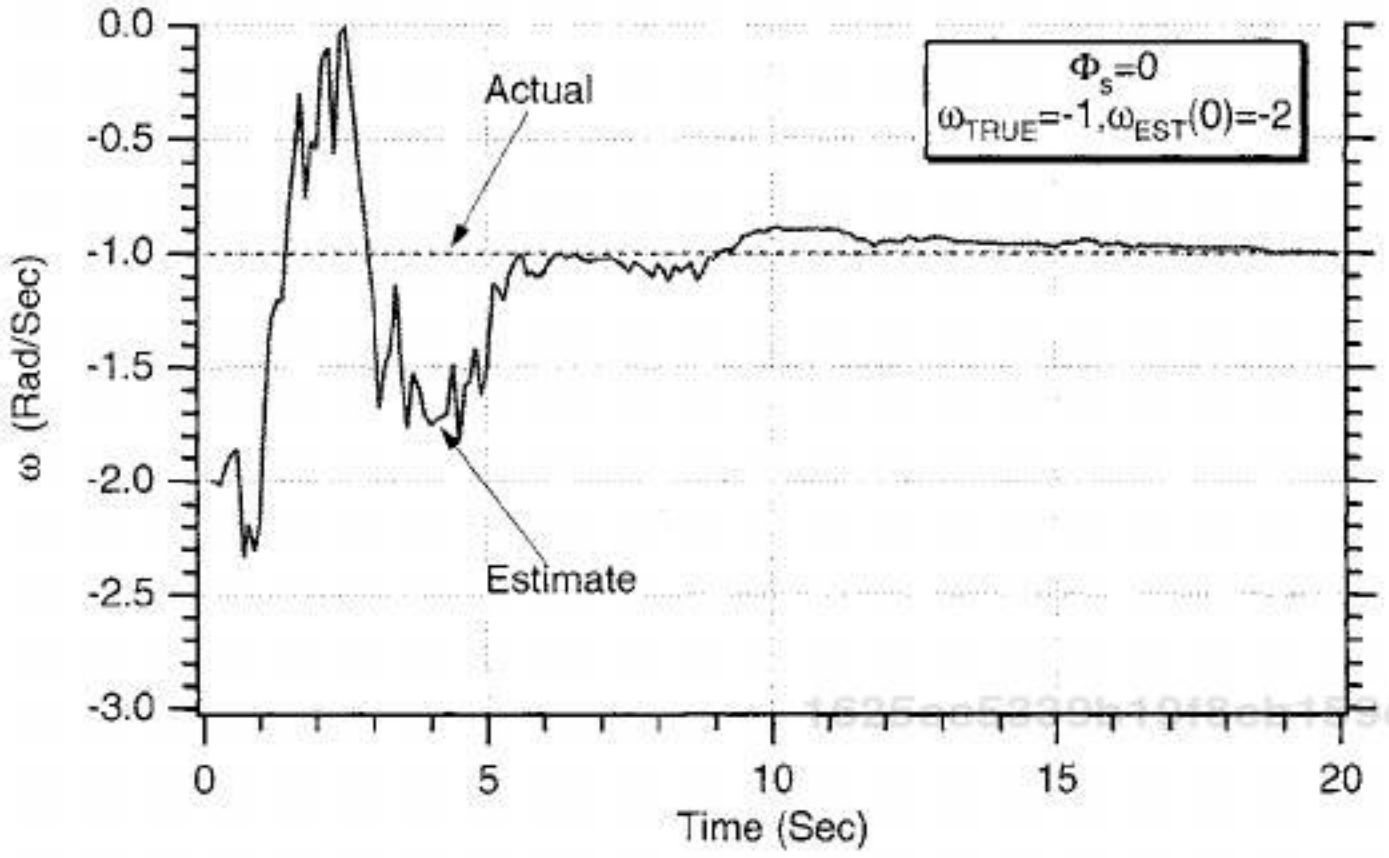

Fig. 10.26 Alternate three-state extended Kalman filter appears able to estimate the frequency correctly when frequency and initial estimate are both negative.

Figures 10.30–10.32 display the theoretical errors in the state estimates (i.e., obtained by taking the square root of the appropriate diagonal element of the covariance matrix) and the single-run simulation errors for each of the three states. The single-run errors in the estimates appear to be within the theoretical bounds approximately 68% of the time, indicating that the extended Kalman filter seems to be behaving properly. Because this example has zero process noise, the errors in the estimates diminish as more measurements are taken.

To verify that the filter was actually working all of the time under a variety of initialization conditions, Listing 10.3 was slightly modified to give it Monte Carlo

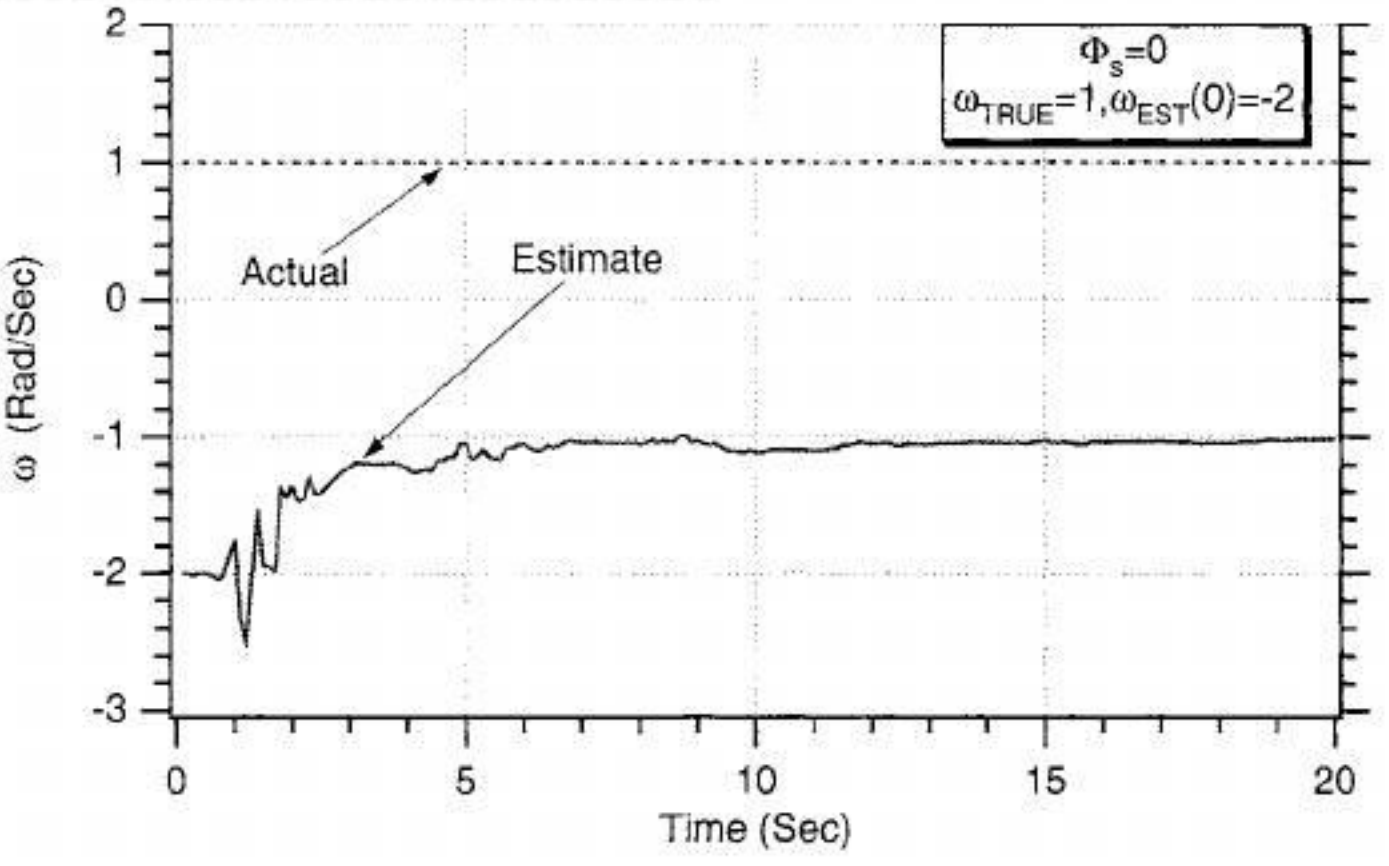

Fig. 10.27 Alternate three-state extended Kalman filter appears able to estimate the magnitude of the frequency but not its sign when the filter is not initialized correctly.

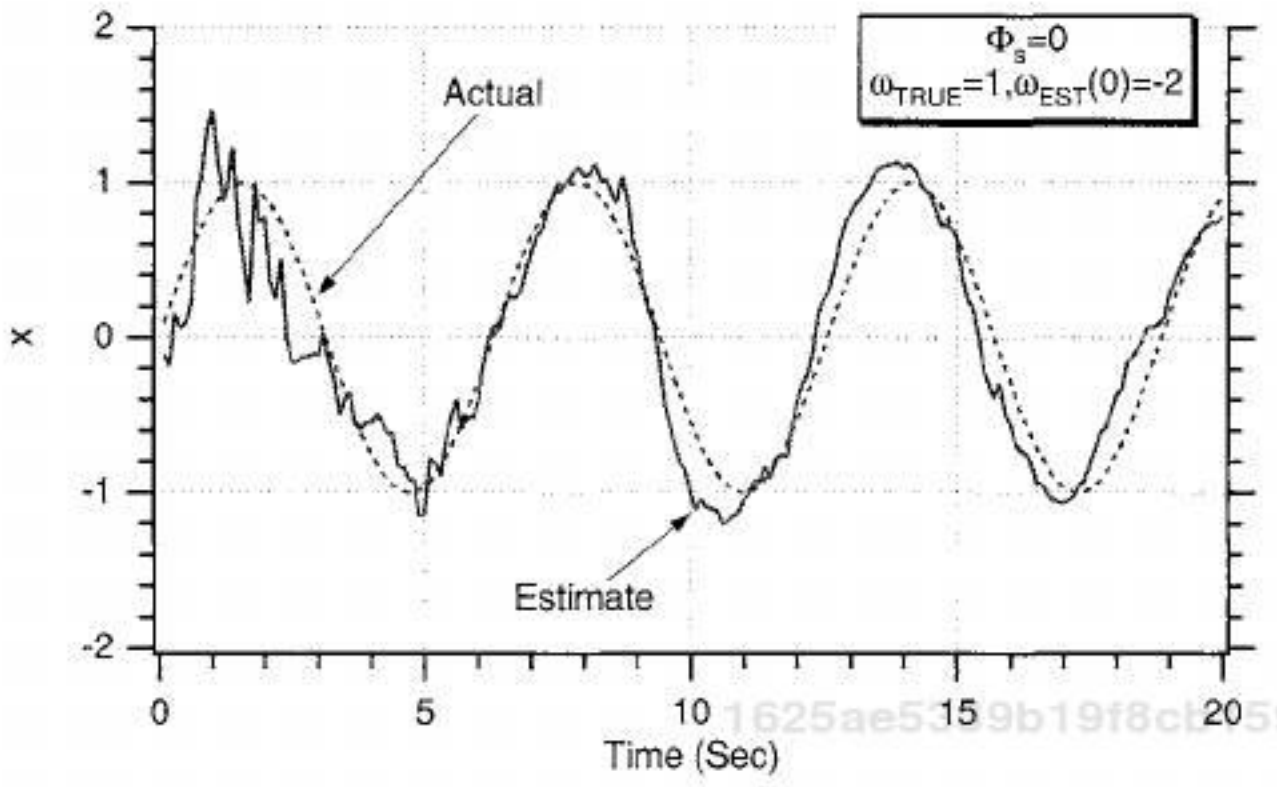

Fig. 10.28 Alternate three-state extended Kalman filter appears able to estimate the signal when the filter is not initialized correctly.

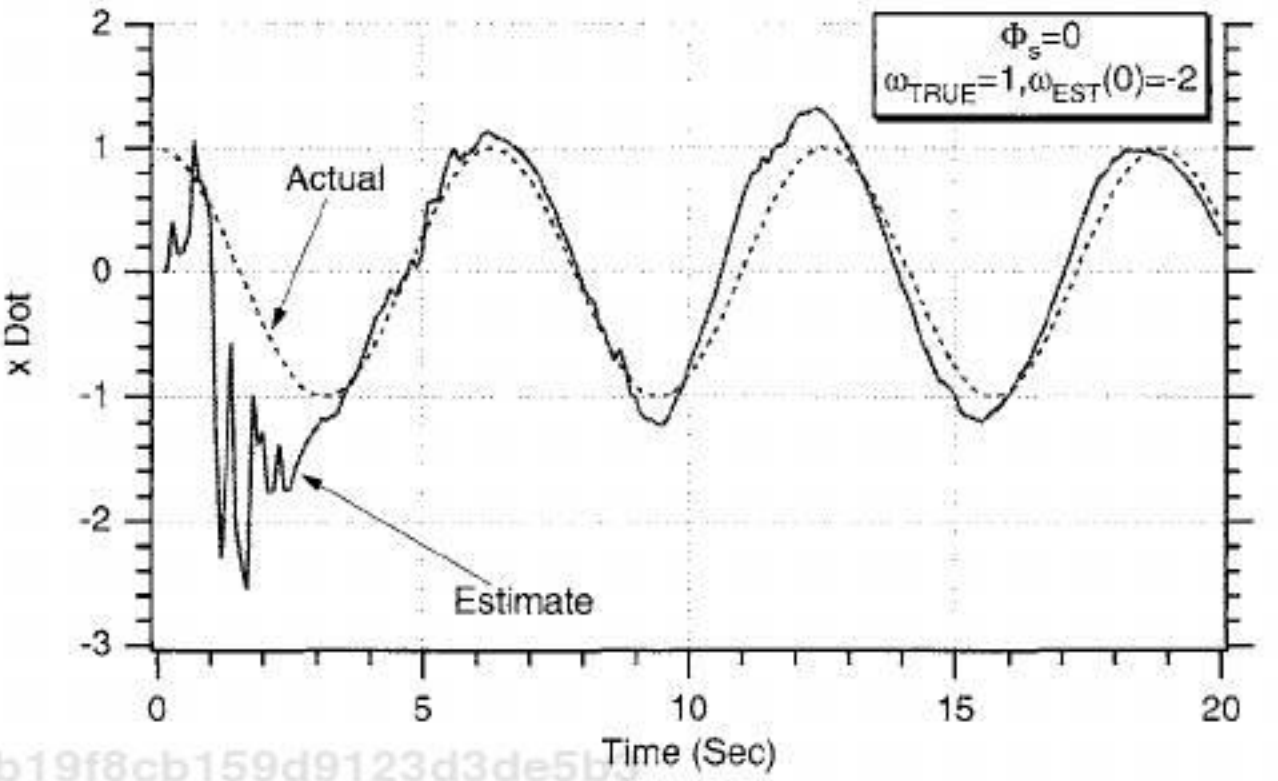

Fig. 10.29 Alternate three-state extended Kalman filter appears able to estimate the derivative of the signal when the filter is not initialized correctly.

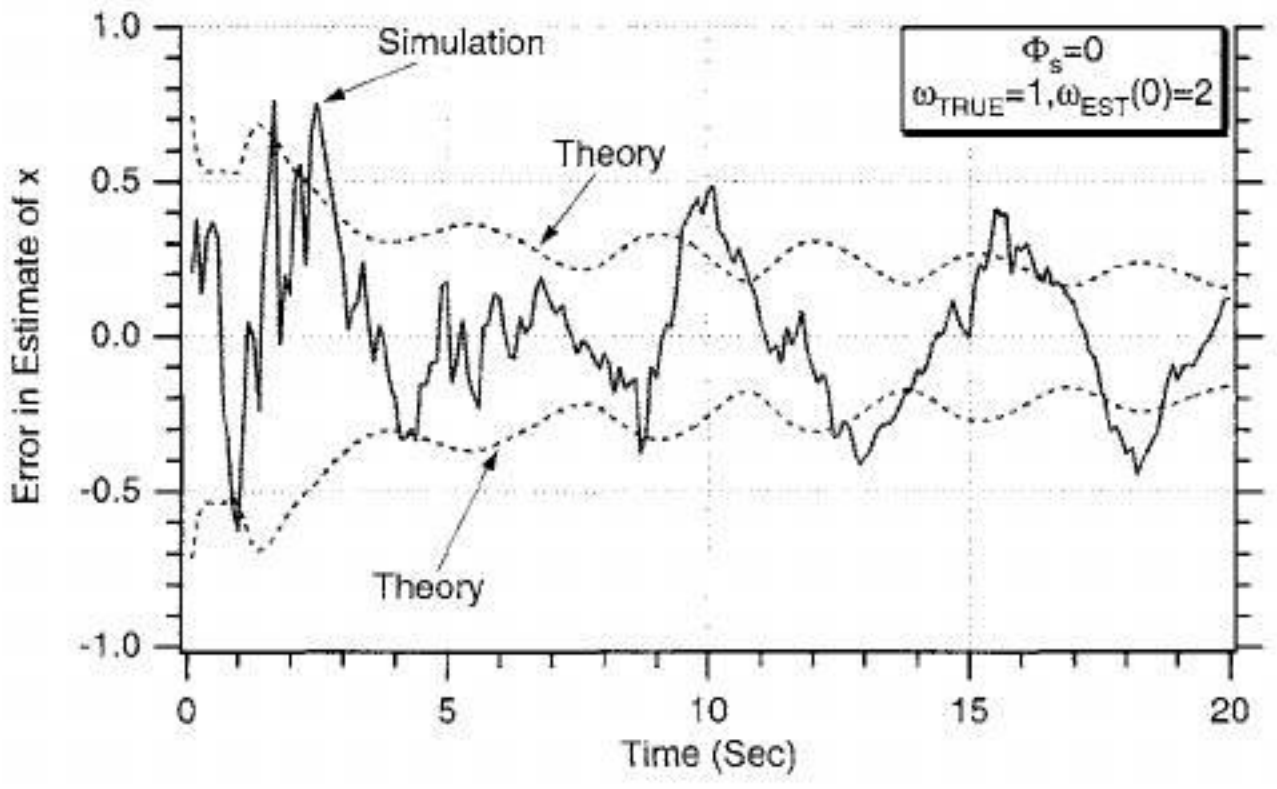

Fig. 10.30 Error in the estimate of first state agrees with theory.

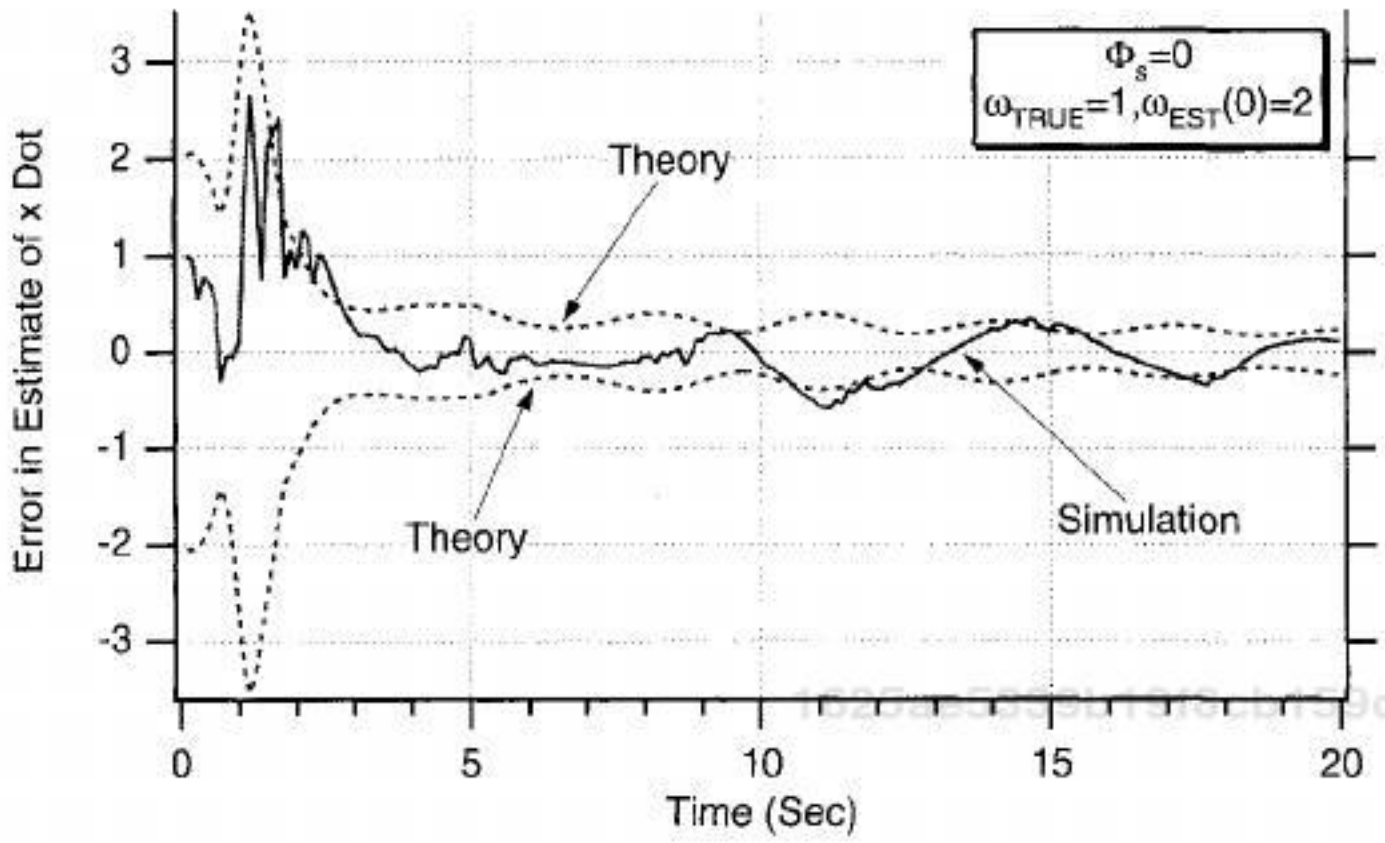

Fig. 10.31 Error in the estimate of second state agrees with theory.

capabilities. Ten-run Monte Carlo sets were run with the alternate three-state extended Kalman filter for the four different frequency and estimated frequency initialization conditions. The standard deviation of the measurement noise for these experiments was unity. The results, which are presented in Figs. 10.33–10.36, show that the correct magnitude of the frequency is always estimated accurately, regardless of initialization. We have already seen that when the correct frequency magnitude is estimated the other states also will be accurate. Thus, we can conclude that the alternate three-state extended Kalman filter is effective in estimating the states of a sinusoid.

We have seen that the alternate three-state extended Kalman filter is only able to estimate both the magnitude and sign of the frequency exactly when the initial

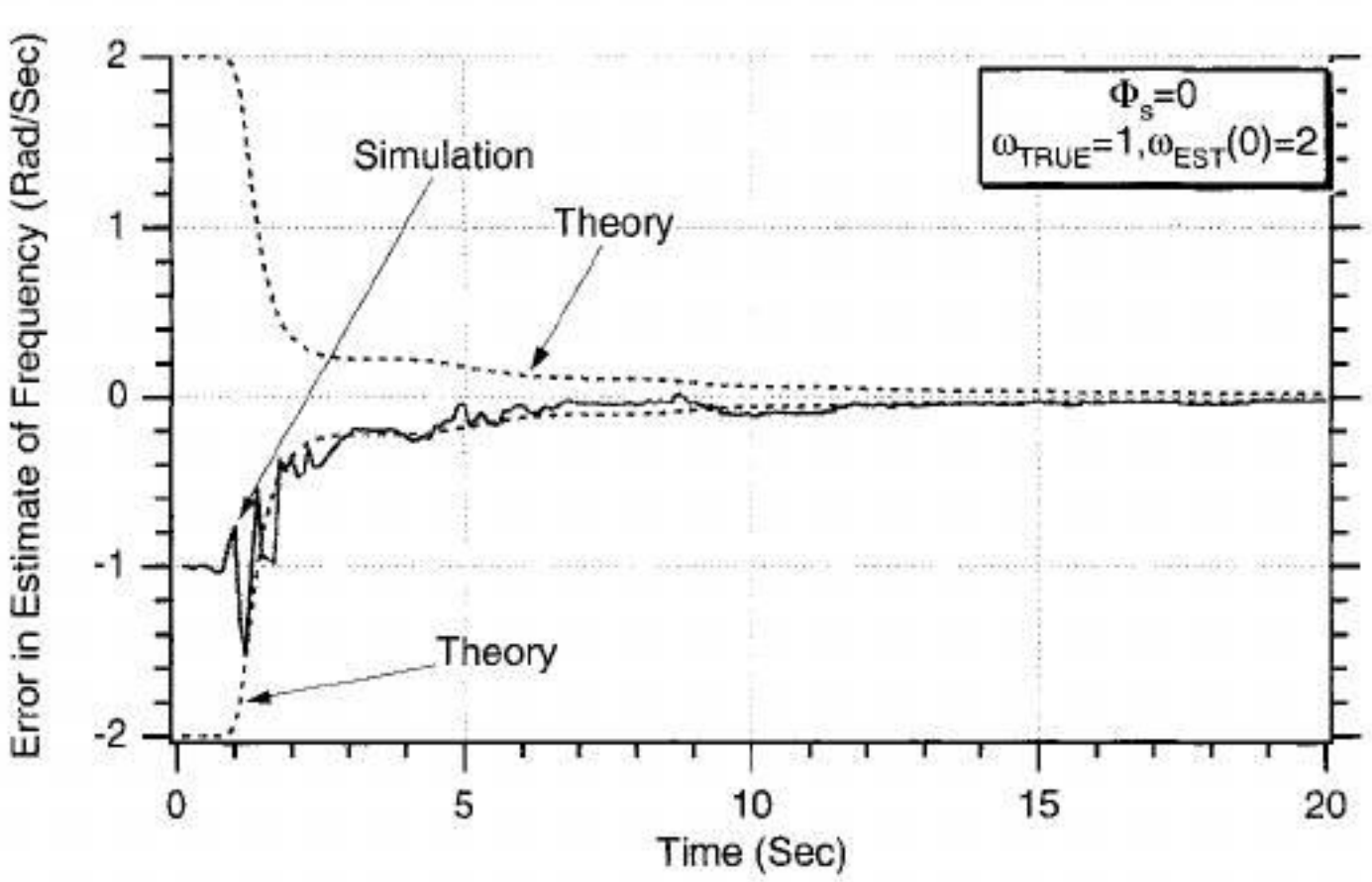

Fig. 10.32 Error in the estimate of third state agrees with theory.

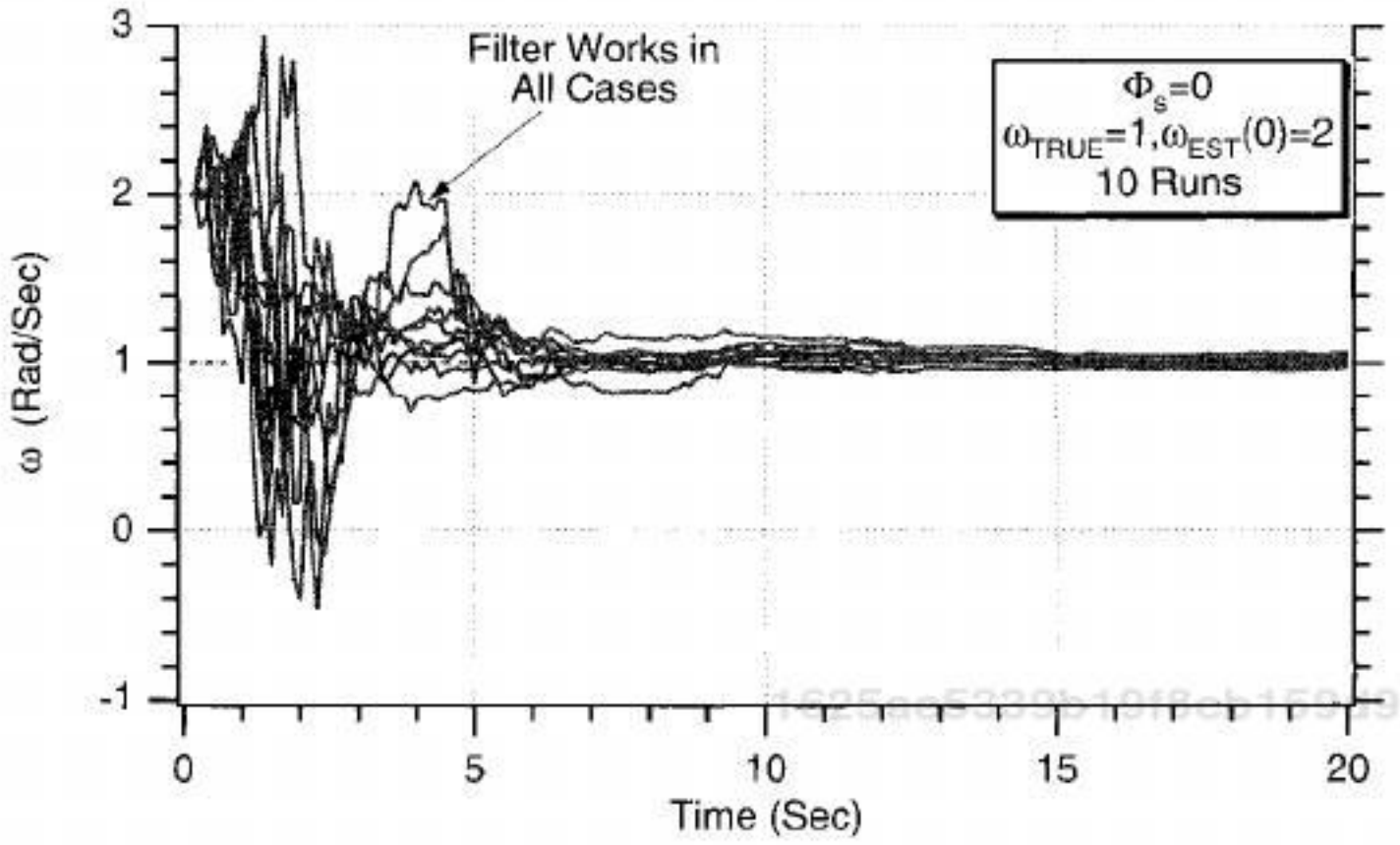

Fig. 10.33 Alternate three-state extended Kalman filter estimates correct frequency when initial frequency estimate is of correct sign.

frequency estimate of the filter is of the same sign as the actual frequency. When this condition is not met, the filter is only able to estimate the correct magnitude of the frequency but not the sign. It is hypothesized that the reason for the filter's lack of ability in always distinguishing between positive and negative frequencies is caused by the formulation of the state-space equations where an ω^2 term appears. Apparently, we cannot distinguish between positive and negative frequencies with only the ω^2 term (i.e., there is no ω term). However, if we are only interested in obtaining estimates of x and $\dot{x}$, knowing the magnitude of the frequency is sufficient (i.e., the sign of the frequency is not important).

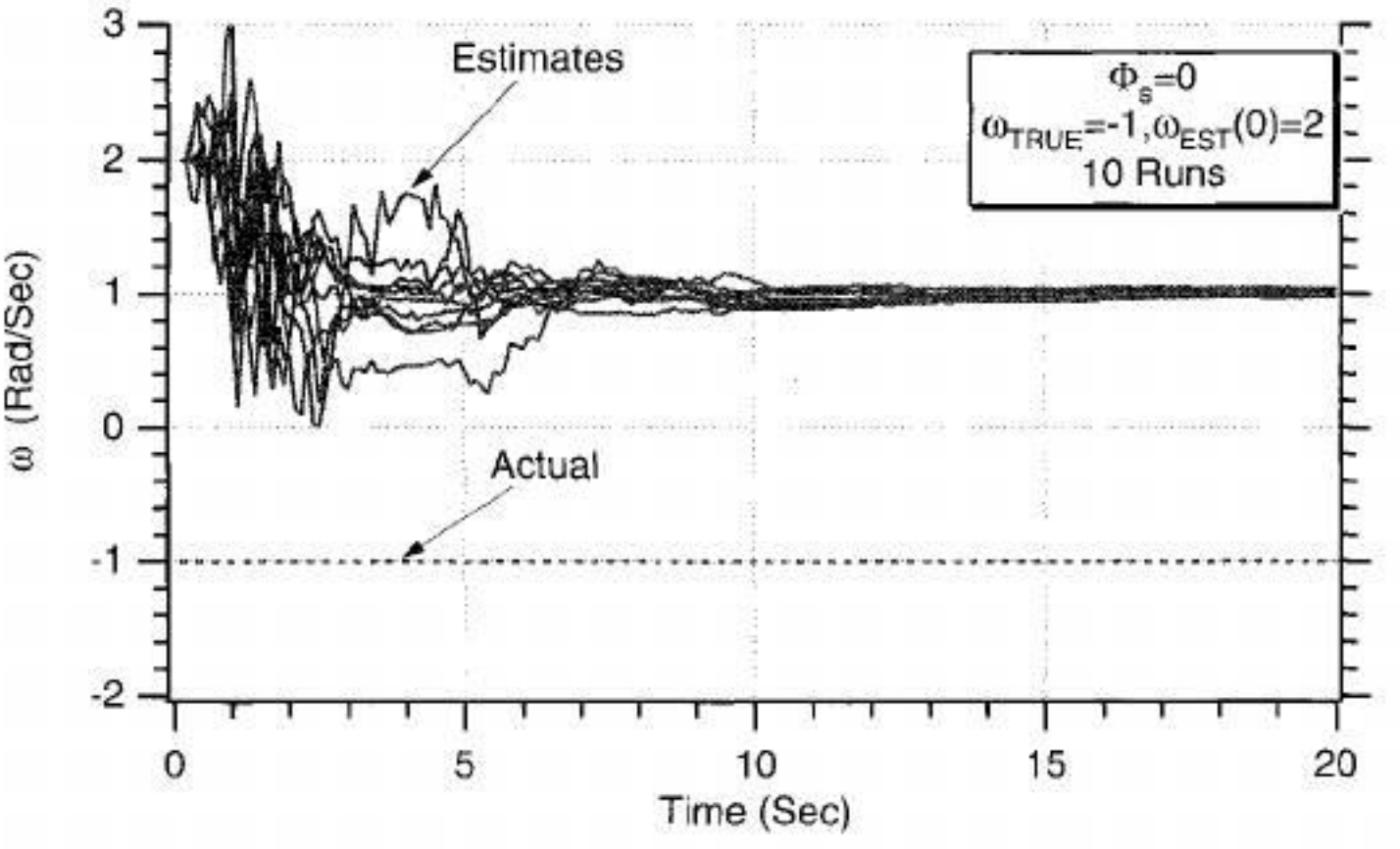

Fig. 10.34 Alternate three-state extended Kalman filter estimates correct frequency magnitude when initial frequency estimate is of the wrong sign.

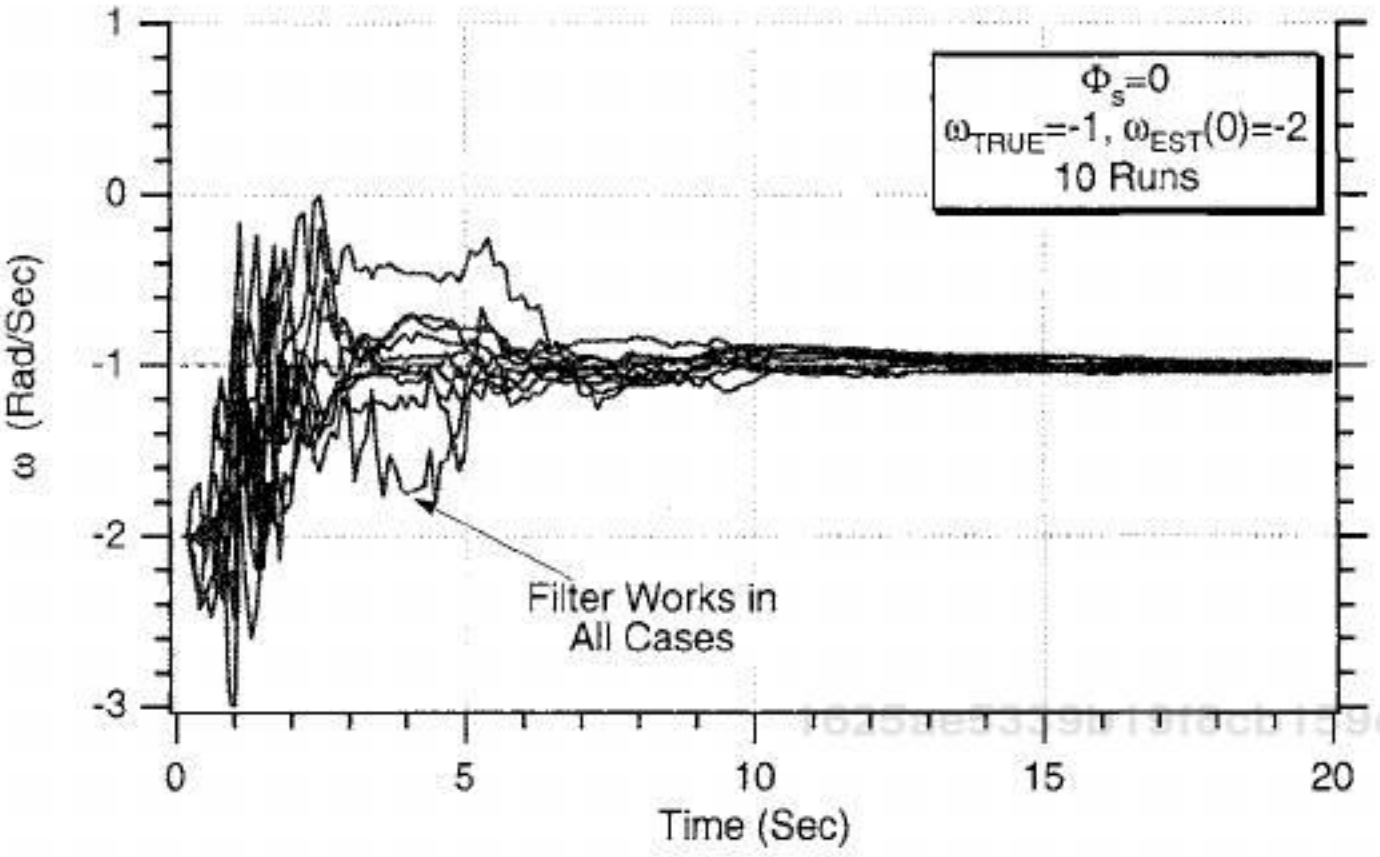

Fig. 10.35 Alternate three-state extended Kalman filter estimates correct frequency when initial frequency estimate is of correct sign.

Another Extended Kalman Filter for Sinusoidal Model

We have just derived an extended Kalman filter in the preceding section in which the sinusoidal frequency ω was a state. We demonstrated that in order to obtain accurate estimates of the states of a sinusoidal signal it was not important to estimate the sign of the sinusoidal frequency but only its magnitude. With that information we will see if we can do even better in the estimation process by reforming the problem to match our new experience base. Let us define a new state z, which is simply the square of the frequency or

$$z = \omega^2$$

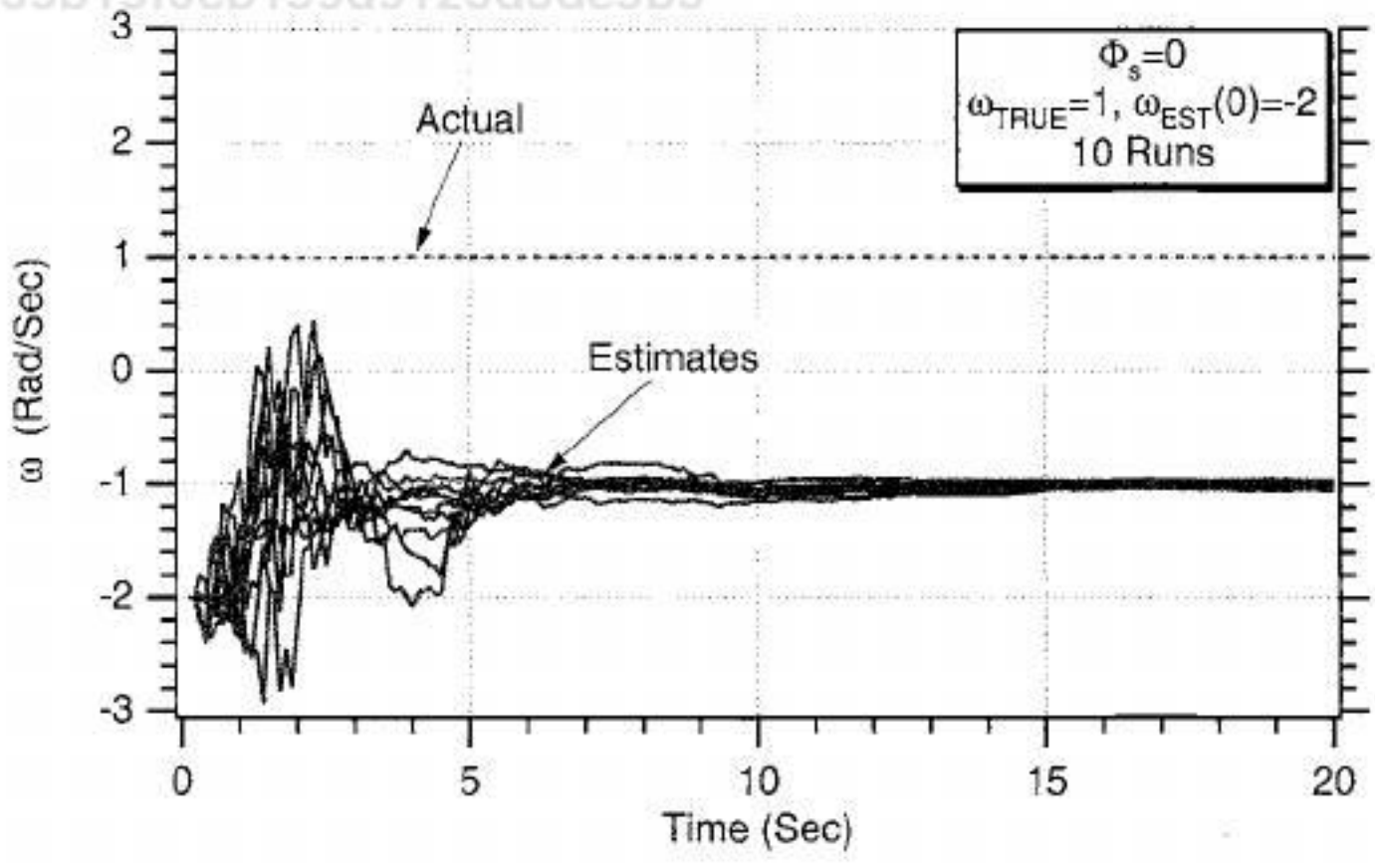

Fig. 10.36 Alternate three-state extended Kalman filter estimates correct frequency magnitude when initial frequency estimate is of the wrong sign.

Under these new circumstances the differential equations representing the real world in this new model are now given by

$$\ddot{x} = -zx$$

$$\dot{z} = u_s$$

where u_s is white process noise with spectral density Φ_s. The preceding differential equations can be expressed as three first-order differential equations (i.e., one of the equations is redundant) in state-space form as

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -z & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ z \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ u_s \end{bmatrix}$$

The preceding equation is also nonlinear because a state also shows up in the 3×3 matrix multiplying the state vector. From the preceding equation we know that the systems dynamics matrix is given by

$$\boldsymbol{F} = \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} = \begin{bmatrix} \dfrac{\partial \dot{x}}{\partial x} & \dfrac{\partial \dot{x}}{\partial \dot{x}} & \dfrac{\partial \dot{x}}{\partial z} \\ \dfrac{\partial \ddot{x}}{\partial x} & \dfrac{\partial \ddot{x}}{\partial \dot{x}} & \dfrac{\partial \ddot{x}}{\partial z} \\ \dfrac{\partial \dot{z}}{\partial x} & \dfrac{\partial \dot{z}}{\partial \dot{x}} & \dfrac{\partial \dot{z}}{\partial z} \end{bmatrix}$$

where the partial derivatives are evaluated at the current estimates. Taking the partial derivatives in this example can be done by inspection, and the resultant systems dynamics matrix is given by

$$\boldsymbol{F} = \begin{bmatrix} 0 & 1 & 0 \\ -\hat{z} & 0 & -\hat{x} \\ 0 & 0 & 0 \end{bmatrix}$$

where the terms in the systems dynamics matrix have been evaluated at the current state estimates. As was the case in the last section, the exact fundamental matrix will be difficult, if not impossible, to find. If we assume that the elements of the systems dynamics matrix are approximately constant between sampling instants, then we use a two-term Taylor-series approximation for the fundamental matrix, yielding

$$\Phi(t) \approx I + \boldsymbol{F}t = \begin{bmatrix} 1 & t & 0 \\ -\hat{z}t & 1 & -\hat{x}t \\ 0 & 0 & 1 \end{bmatrix}$$

Therefore, the discrete fundamental matrix can be obtained by substituting T_s for t or

$$\Phi_k \approx \begin{bmatrix} 1 & T_s & 0 \\ -\hat{z}_{k-1}T_s & 1 & -\hat{x}_{k-1}T_s \\ 0 & 0 & 1 \end{bmatrix}$$

As was the case in the preceding section, the continuous process-noise matrix can by found from the original state-space equation to be

$$\boldsymbol{Q} = E(ww^T) = E\left[\begin{bmatrix} 0 \\ 0 \\ u_s \end{bmatrix}[0 \quad 0 \quad u_s]\right] = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \Phi_s \end{bmatrix}$$

Recall that the discrete process-noise matrix can be found from the continuous process-noise matrix according to

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

By substitution of the appropriate matrices into the preceding equation, we obtain

$$\boldsymbol{Q}_k = \int_0^{T_s} \begin{bmatrix} 1 & \tau & 0 \\ -\hat{z}\tau & 1 & -\hat{x}\tau \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \Phi_s \end{bmatrix} \begin{bmatrix} 1 & -\hat{z}\tau & 0 \\ \tau & 1 & 0 \\ 0 & -\hat{x}\tau & 1 \end{bmatrix} \mathrm{d}\tau$$

Multiplying out the three matrices yields

$$\boldsymbol{Q}_k = \int_0^{T_s} \begin{bmatrix} 0 & 0 & 0 \\ 0 & \hat{x}^2\tau^2\Phi_s & -\hat{x}\tau\Phi_s \\ 0 & -\hat{x}\tau\Phi_s & \Phi_s \end{bmatrix} \mathrm{d}\tau$$

If we again assume that the states are approximately constant between the sampling intervals, then the preceding integral can easily be evaluated as

$$\boldsymbol{Q}_k = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.333\hat{x}^2T_s^3\Phi_s & -0.5\hat{x}T_s^2\Phi_s \\ 0 & -0.5\hat{x}T_s^2\Phi_s & T_s\Phi_s \end{bmatrix}$$

In this problem we are also assuming that the measurement is of the first state plus noise or

$$x_k^* = x_k + v_k$$

Therefore, the measurement is linearly related to the states according to

$$x_k^* = [1 \quad 0 \quad 0]\begin{bmatrix} x \\ \dot{x} \\ z \end{bmatrix} + v_k$$

The measurement matrix can be obtained from the preceding equation by inspection as

$$\boldsymbol{H} = [1 \quad 0 \quad 0]$$

In this example the discrete measurement noise matrix is a scalar and is given by

$$\boldsymbol{R}_k = E(v_k v_k^T) = \sigma_x^2$$

We now have enough information to solve the matrix Riccati equations for the Kalman gains.

As was the case in the preceding section, the projected states in the actual extended Kalman-filtering equations do not have to use the approximation for the fundamental matrix. Instead, the state projections, indicated by an overbar, can be obtained by numerically integrating the nonlinear differential equations over the sampling interval. Therefore, the extended Kalman-filtering equations can then be written as

$$\hat{x}_k = \bar{x}_k + K_{1_k}(x_k^* - \bar{x}_k)$$
$$\hat{\dot{x}}_k = \bar{\dot{x}}_k + K_{2_k}(x_k^* - \bar{x}_k)$$
$$\hat{z}_k = \hat{z}_{k-1} + K_{3_k}(x_k^* - \bar{x}_k)$$

Listing 10.4 presents another three-state extended Kalman filter for estimating the states of a noisy sinusoidal signal whose frequency is unknown. As was the case in the last section, the simulation is initially set up to run without process noise (i.e., PHIS = 0). The actual sinusoidal signal has unity amplitude, and the standard deviation of the measurement noise is also unity. We can see from Listing 10.4 that the frequency of the actual sinusoidal signal is unity. The filter's initial state estimates of x and $\dot{x}$ are set to zero. Because the initial estimate of the frequency in the previous section was two (rather than unity), the initial value of the estimate of z is 4 (i.e., $z = \omega^2 = 2^2 = 4$). In other words, as was the case in the preceding section, the second and third states are mismatched from the real world. Values are used for the initial covariance matrix to reflect the uncertainties in our initial state estimates. Subroutine PROJECT is still used to integrate the nonlinear equations, using Euler integration over the sampling interval to obtain the state projections for x and $\dot{x}$.

A case was run using the nominal values of Listing 10.4 (i.e., no process noise), and we can see from a comparison that the actual quantities and their estimates in Figs. 10.37–10.39 are quite close. If we compare the estimates of x and $\dot{x}$ (i.e., Figs. 10.37 and 10.38) with the preceding results from our alternate

Listing 10.4 Another three-state extended Kalman filter for sinusoidal signal with unknown frequency

```
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,O-Z)
        REAL*8 P(3,3),Q(3,3),M(3,3),PHI(3,3),HMAT(1,3),HT(3,1),PHIT(3,3)
        REAL*8 RMAT(1,1),IDN(3,3),PHIP(3,3),PHIPPHIT(3,3),HM(1,3)
        REAL*8 HMHT(1,1),HMHTR(1,1),HMHTRINV(1,1),MHT(3,1),K(3,1),
          F(3,3)
        REAL*8 KH(3,3),IKH(3,3)
        INTEGER ORDER
        HP=.001
        W=1.
        WH=2.
        A=1.
        TS=.1
        ORDER=3
        PHIS=0.
        SIGX=1.
        OPEN(1,STATUS='UNKNOWN',FILE='DATFIL')
        OPEN(2,STATUS='UNKNOWN',FILE='COVFIL')
        T=0.
        S=0.
        H=.001
        DO 14 I=1,ORDER
        DO 14 J=1,ORDER
        F(I,J)=0.
        PHI(I,J)=0.
        P(I,J)=0.
        Q(I,J)=0.
        IDN(I,J)=0.
14      CONTINUE
        RMAT(1,1)=SIGX**2
        IDN(1,1)=1.
        IDN(2,2)=1.
        IDN(3,3)=1.
        P(1,1)=SIGX**2
        P(2,2)=2.**2
        P(3,3)=4.**2
        XTH=0.
        XTDH=0.
        ZH=WH**2
        XT=0.
        XTD=A*W
        WHILE(T<=20.)
              XTOLD=XT
              XTDOLD=XTD
```

(continued)

Listing 10.4 (*Continued*)

```
XTDD=-W*W*XT
XT=XT+H*XTD
XTD=XTD+H*XTDD
T=T+H
XTDD=-W*W*XT
XT=.5*(XTOLD+XT+H*XTD)
XTD=.5*(XTDOLD+XTD+H*XTDD)
S=S+H
IF(S>=(TS-.00001))THEN
        S=0.
        F(1,2)=1.
        F(2,1)=-WH**2
        F(2,3)=-2.*WH*XTH
        PHI(1,1)=1.
        PHI(1,2)=TS
        PHI(2,1)=-ZH*TS
        PHI(2,2)=1.
        PHI(2,3)=-XTH*TS
        PHI(3,3)=1.
        Q(2,2)=XTH*XTH*TS*TS*TS*PHIS/3.
        Q(2,3)=-XTH*TS*TS*PHIS/2.
        Q(3,2)=Q(2,3)
        Q(3,3)=PHIS*TS
        HMAT(1,1)=1.
        HMAT(1,2)=0.
        HMAT(1,3)=0.
        CALL MATTRN(PHI,ORDER,ORDER,PHIT)
        CALL MATTRN(HMAT,1,ORDER,HT)
        CALL MATMUL(PHI,ORDER,ORDER,P,ORDER,
          ORDER,PHIP)
        CALL MATMUL(PHIP,ORDER,ORDER,PHIT,ORDER,
          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)
        CALL MATADD(HMHT,ORDER,ORDER,RMAT,
          HMHTR)HMHTRINV(1,1)=1./HMHTR(1,1)
        CALL MATMUL(M,ORDER,ORDER,HT,ORDER,1,
          MHT)
        CALL MATMUL(MHT,ORDER,1,HMHTRINV,1,1,K)
        CALL MATMUL(K,ORDER,1,HMAT,1,ORDER,KH)
        CALL MATSUB(IDN,ORDER,ORDER,KH,IKH)
        CALL MATMUL(IKH,ORDER,ORDER,M,ORDER,
          ORDER,P)
        CALL GAUSS(XTNOISE,SIGX)
        XTMEAS=XT+XTNOISE
        CALL PROJECT(T,TS,XTH,XTDH,XTB,XTDB,HP,ZH)
        RES=XTMEAS-XTB
```

(*continued*)

Listing 10.4 *(Continued)*

```
      XTH=XTB+K(1,1)*RES
      XTDH=XTDB+K(2,1)*RES
      ZH=ZH+K(3,1)*RES
      ERRX=XT-XTH
      SP11=SQRT(P(1,1))
      ERRXD=XTD-XTDH
      SP22=SQRT(P(2,2))
      Z=W**2
      ERRZ=Z-ZH
      SP33=SQRT(P(3,3))
      WH=SQRT(ZH)

      WRITE(9,*)T,XT,XTH,XTD,XTDH  ,Z,ZH
                WRITE(1,*)T,XT,XTH,XTD,XTDH  ,Z,ZH
                WRITE(2,*)T,ERRX,SP11,-SP11,ERRXD,SP22,-SP22,
     1            ERRZ,SP33,-SP33
           ENDIF
      END DO
      PAUSE
      CLOSE(1)
      CLOSE(2)
      END

      SUBROUTINE PROJECT(TP,TS,XTP,XTDP,XTH,XTDH,HP,Z)
      IMPLICIT REAL*8 (A-H)
      IMPLICIT REAL*8 (O-Z)
      T=0.
      XT=XTP
      XTD=XTDP
      H=HP
      WHILE(T<=(TS-.0001))
           XTDD=-Z*XT
           XTD=XTD+H*XTDD
           XT=XT+H*XTD
           T=T+H
      END DO
      XTH=XT
      XTDH=XTD
      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
```

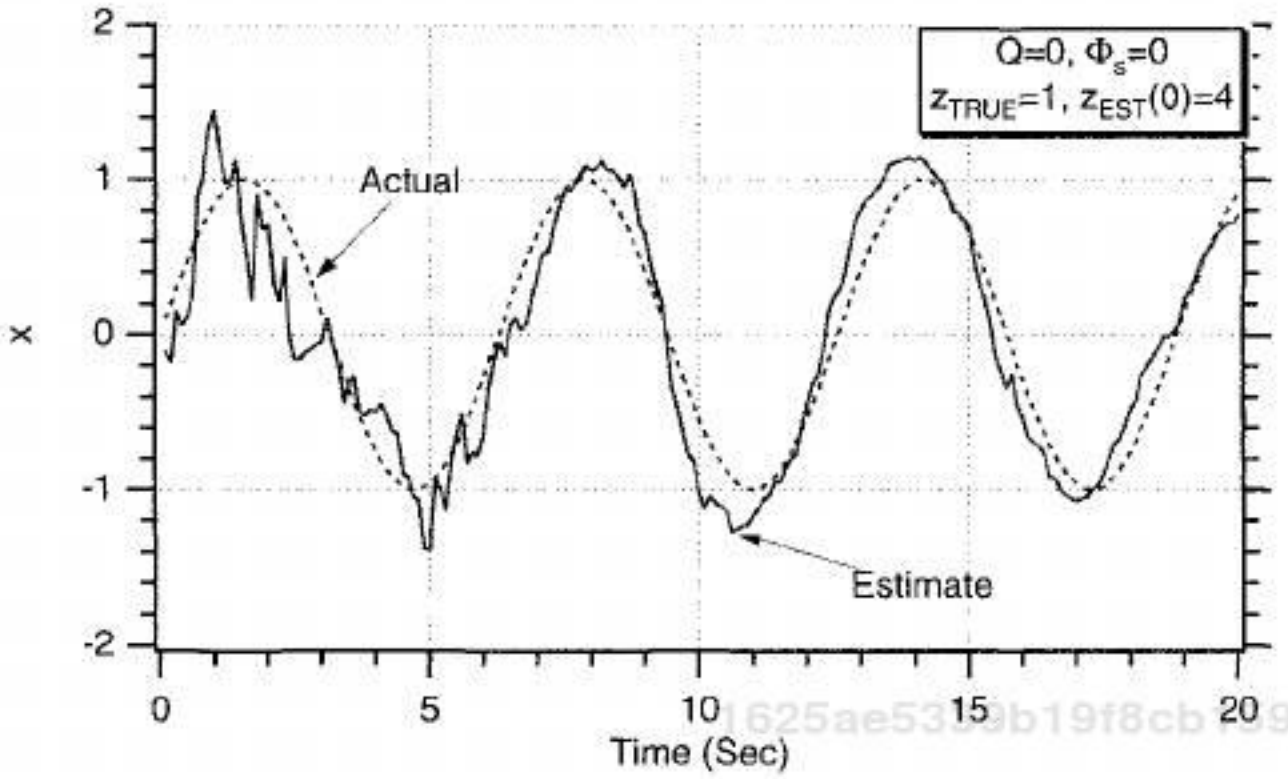

Fig. 10.37 New extended Kalman filter estimates first state quite well.

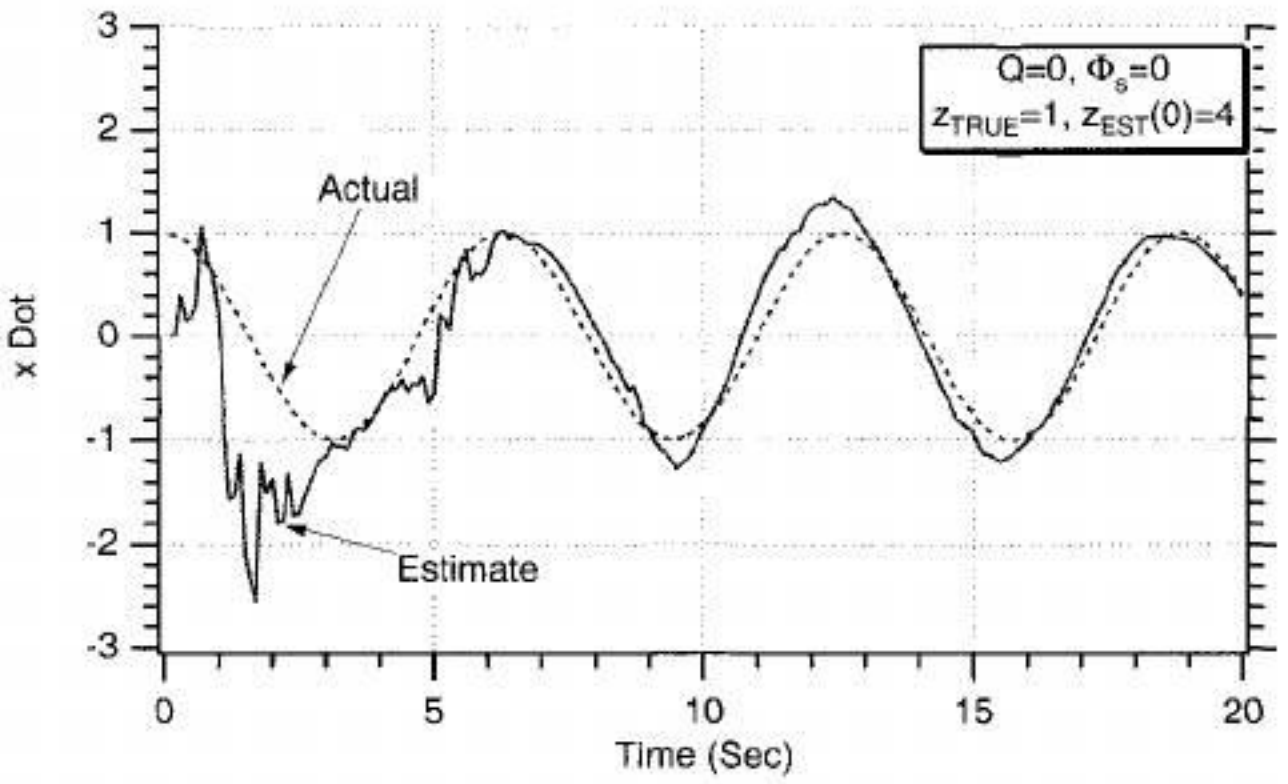

Fig. 10.38 New extended Kalman filter estimates second state well, even though that state is not correctly initialized.

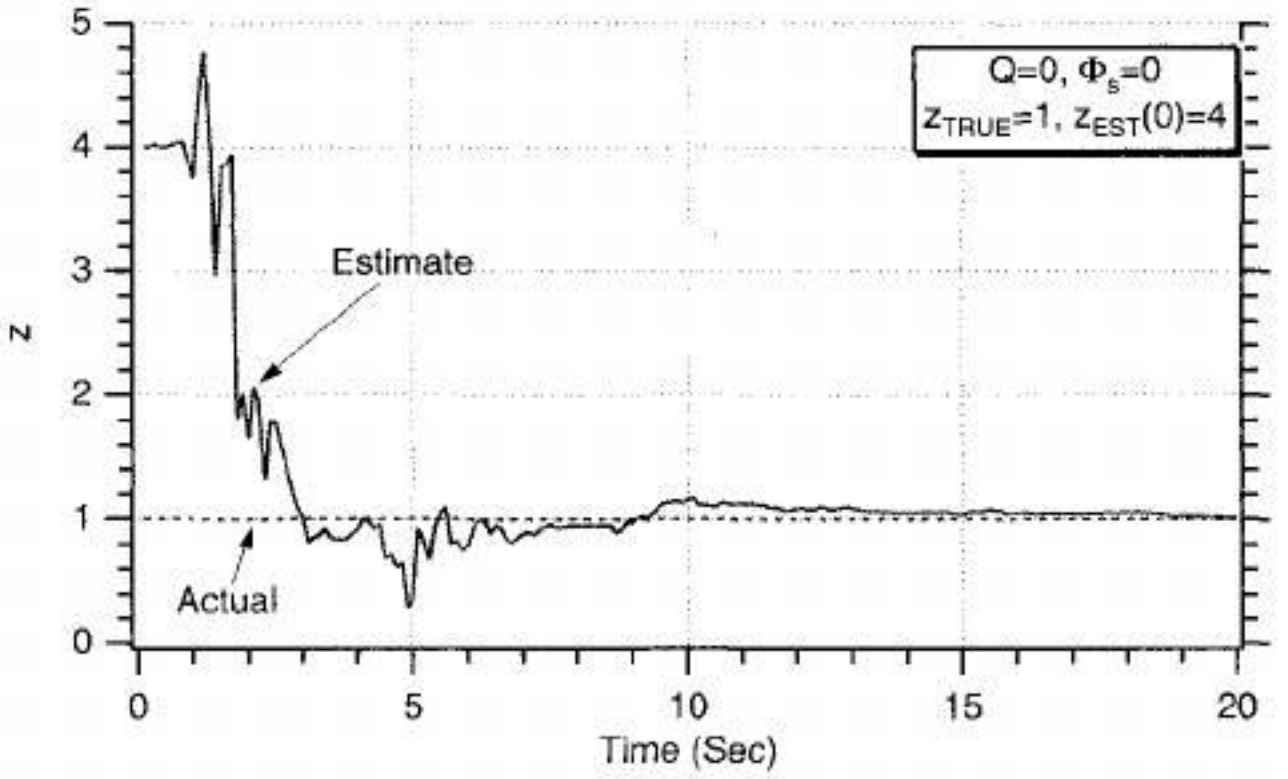

Fig. 10.39 After 5 s the new extended Kalman filter is able to estimate the square of frequency.

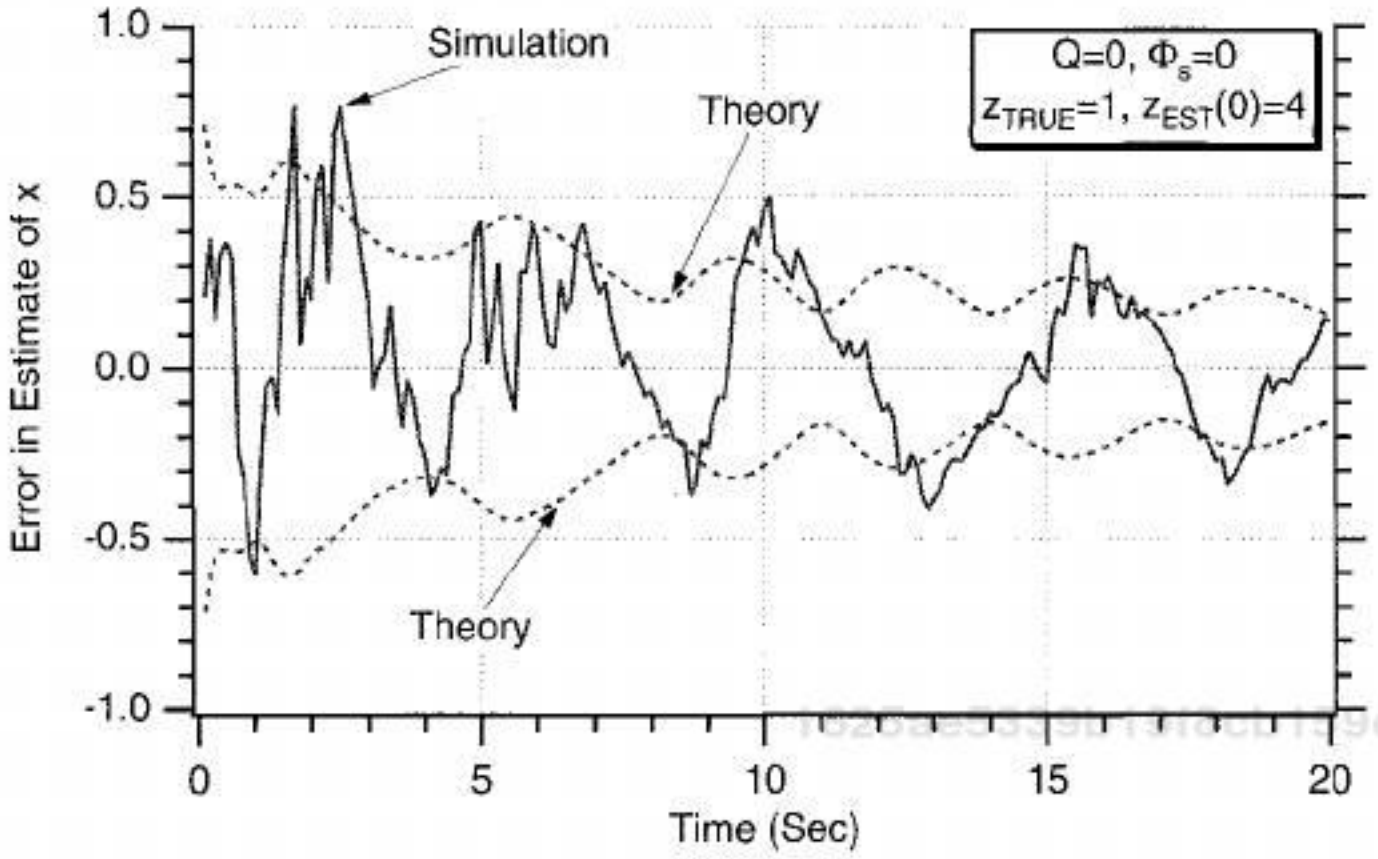

Fig. 10.40 Error in the estimate of first state agrees with theory.

extended Kalman filter (ie., Figs. 10.20 and 10.21), we can see that the new results are virtually identical to the preceding results. Our new filter does not appear to be any better or worse than the preceding filter. From Fig. 10.39 we can see that it takes approximately 5 s to obtain an accurate estimate of z (i.e., the square of the frequency) for the parameters considered.

Figures 10.40–10.42 display the theoretical errors in the state estimates (i.e., obtained by taking the square root of the appropriate diagonal element of the covariance matrix) and the single-run errors for each of the three states obtained from the simulation. The single-run errors in the estimates appear to be within the theoretical bounds approximately 68% of the time, indicating that the extended Kalman filter seems to be behaving properly. Because this example has zero process noise, the errors in the estimates diminish as more measurements are taken. The results of Figs. 10.40 and 10.41 are virtually identical to the results of

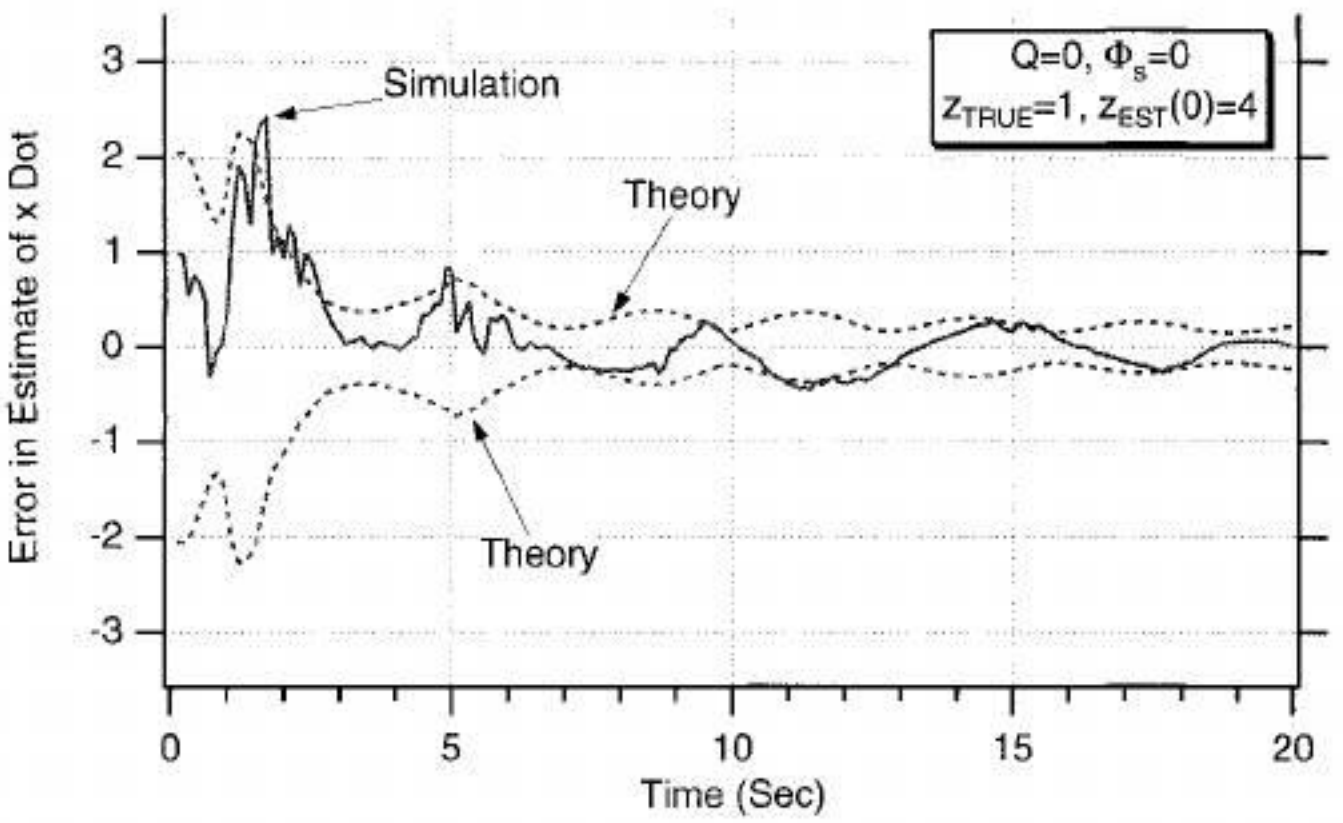

Fig. 10.41 Error in the estimate of second state agrees with theory.

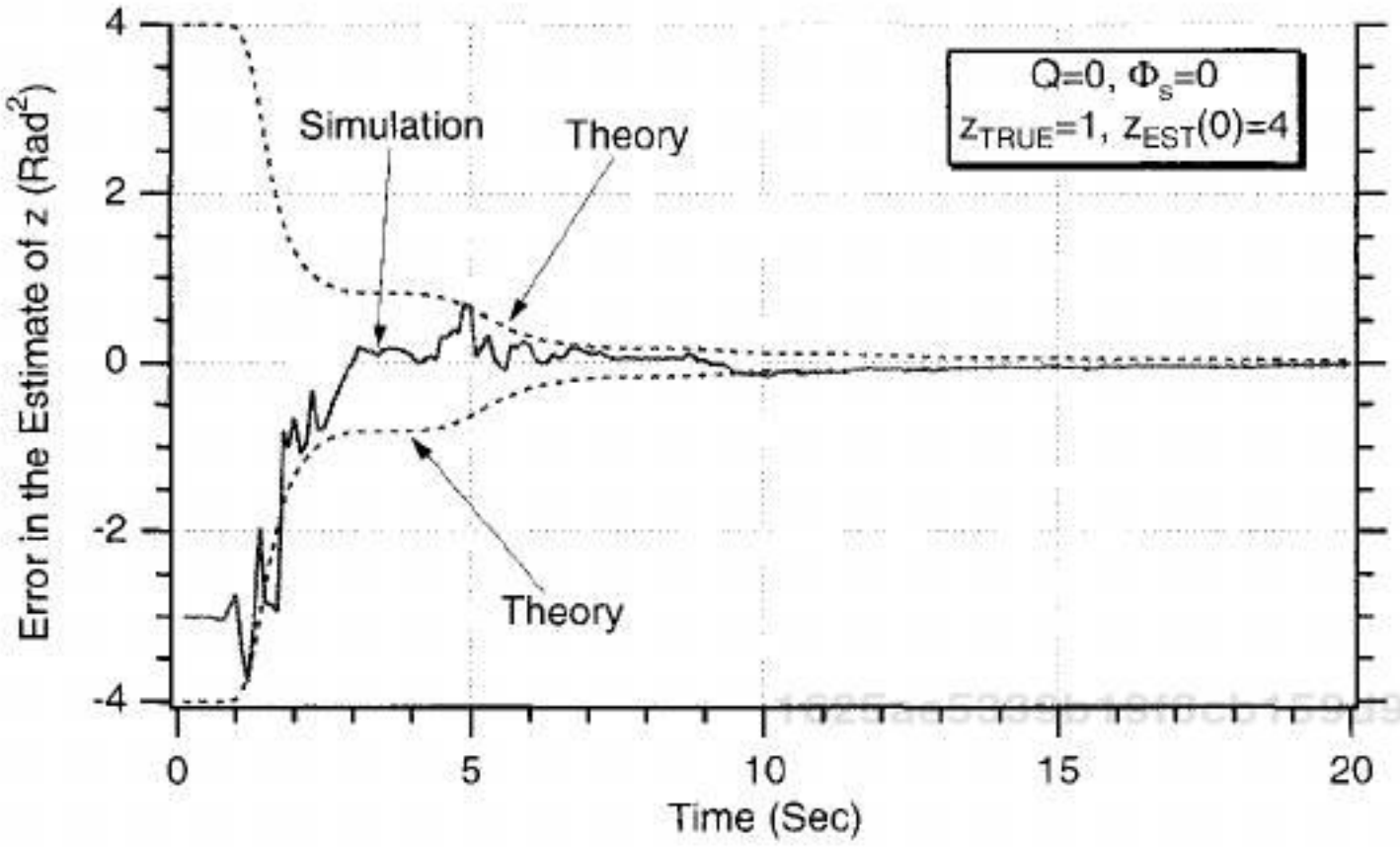

Fig. 10.42 Error in the estimate of third state agrees with theory.

Figs. 10.30 and 10.31. Thus, it appears that there is virtually no difference between the extended Kalman filter of this section and the one of the preceding section.

To verify that the filter was actually working all of the time, Listing 10.4 was slightly modified to give it Monte Carlo capabilities. A ten-run Monte Carlo set was run with the new three-state extended Kalman filter. Figure 10.43 shows that we are able to accurately estimate z in each of the 10 runs. Thus, we can conclude that the new three-state extended Kalman filter is also effective in estimating the states of a sinusoid.

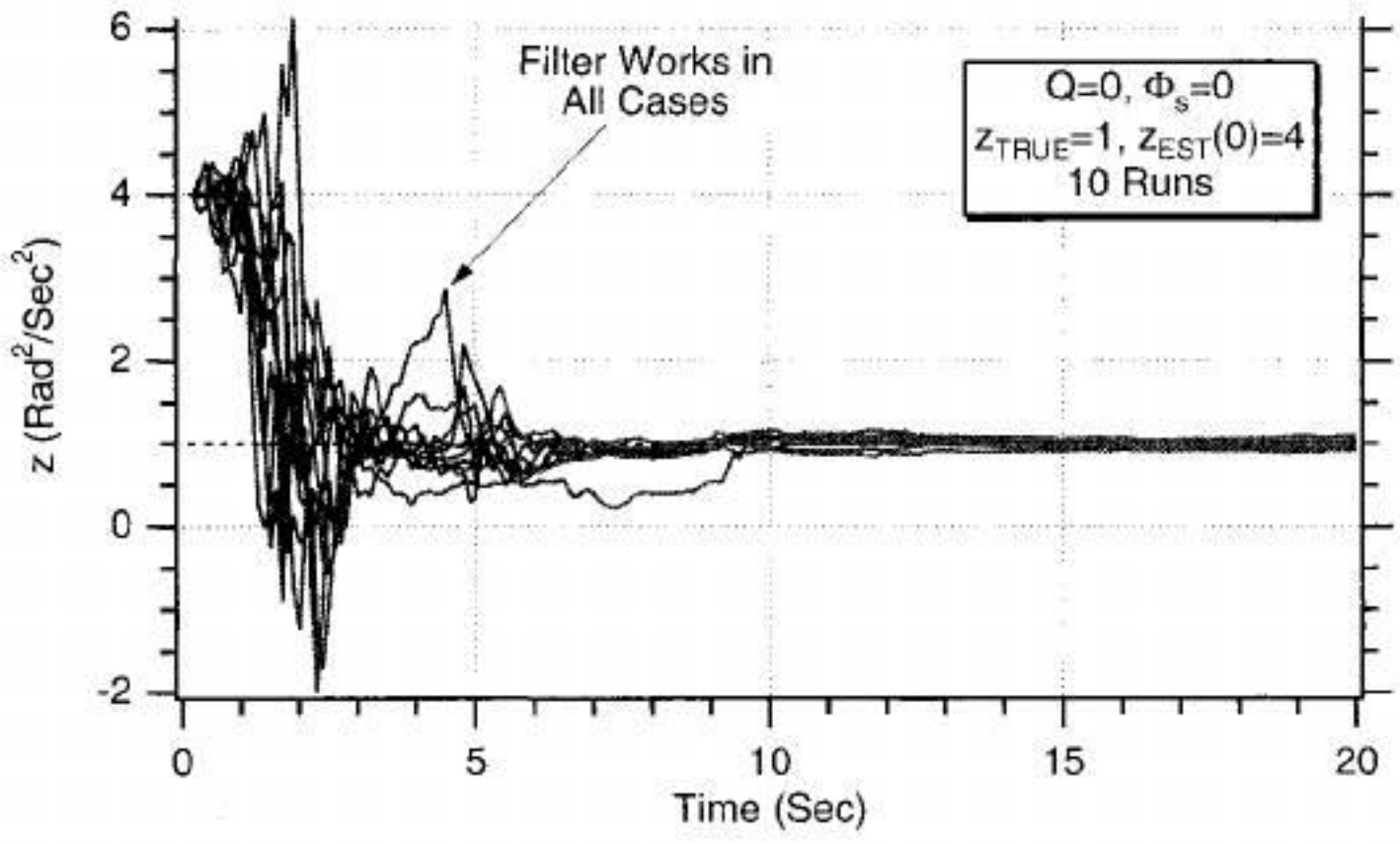

Fig. 10.43 New three-state extended Kalman filter is also effective in estimating the square of the frequency of a sinusoid.

Summary

We have developed several extended Kalman filters that attempt to estimate the states of a sinusoidal signal way based upon noisy measurements of the signal. We have demonstrated in this chapter that simply choosing states and building an extended Kalman filter does not guarantee that it will actually work as expected if programmed correctly. Sometimes, some states that are being estimated are not observable. Numerous experiments were conducted, and various extended Kalman filters were designed, each of which had different states, to highlight some of the issues. An extended Kalman filter was finally built that could estimate all of the states under a variety of initialization conditions, but it could not always distinguish between positive and negative frequencies (i.e., always a correct estimate of frequency magnitude but not always a correct estimate of the frequency sign). Monte Carlo experiments confirmed that this particular extended Kalman filter appeared to be very robust. An alternate form of the extended Kalman filter, in which the square of the frequency was being estimated, was shown to be equally as robust.

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.

This page intentionally left blank

Chapter 11

Satellite Navigation

Introduction

IN THIS chapter we will show how extended Kalman filtering can be used to locate a receiver based on range measurements from a number of satellites to the receiver. Although many simplifications are made, some aspects of problems of this type are similar to those encountered by a receiver processing measurements from the global positioning system (GPS).[1] We will first investigate the case in which filtering is not used at all, but we solve simply the noisy algebraic equations for the receiver location. Next, we will attempt to improve the accuracy in locating the receiver by filtering the range measurements. We will then show that even more accuracy can be achieved by using extended Kalman filtering. Once the filter is built, we will see what happens to accuracy and to the filter when the receiver is moving. All of the experiments will be conducted in two dimensions with either one or two satellites.

Problem with Perfect Range Measurements

Consider two satellites traveling in orbit in a two-dimensional world. If we are considering a flat Earth, then the satellites will appear as if they are each traveling at constant altitude. Let us assume that at any time the two satellite locations [(x_{R1}, y_{R1}) and (x_{R2}, y_{R2})] are known perfectly. In addition, let us first consider the case where each satellite is able to take perfect range measurements r_1 and r_2 to a receiver at location (x, y), as shown in Fig. 11.1. It seems reasonable to believe that two range measurements are sufficient for determining the location of the receiver.

To calculate the location of the receiver numerically, let us first recall that the range from each of the satellites to the receiver can be written by inspection of Fig. 11.1 as

$$r_1 = \sqrt{(x_{R1} - x)^2 + (y_{R1} - y)^2}$$

$$r_2 = \sqrt{(x_{R2} - x)^2 + (y_{R2} - y)^2}$$

The preceding two equations have two unknowns (i.e., x and y are unknowns); because we are assuming that we know the location of both satellites perfectly (i.e., x_{R1}, y_{R1}, x_{R2}, and y_{R2} are known), we have enough information to solve for

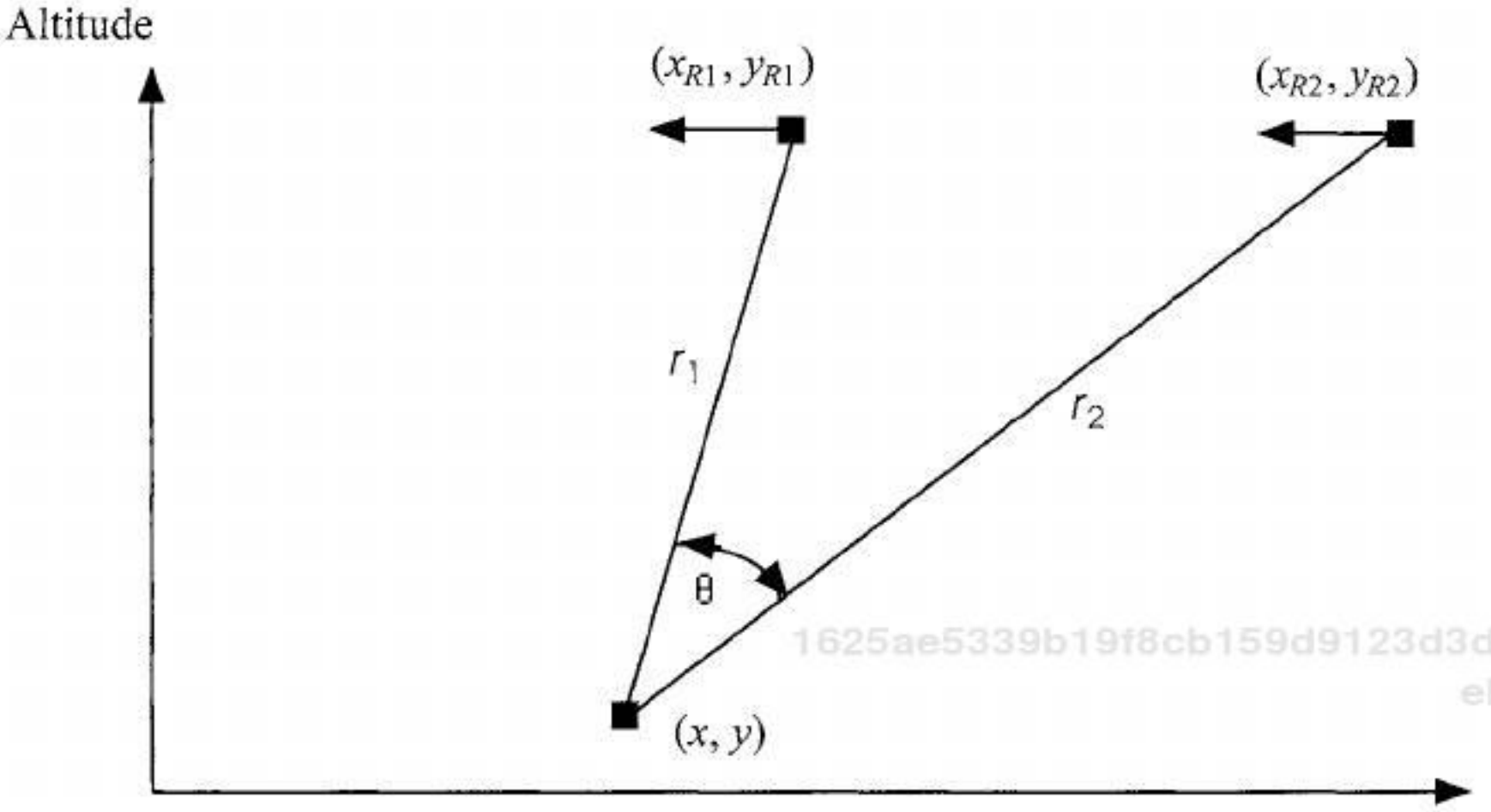

Fig. 11.1 Two satellites making range measurements to a receiver.

the location of the receiver. Squaring and multiplying out each of the terms of the two preceding equations yields

$$r_1^2 = x_{R1}^2 - 2x_{R1}x + x^2 + y_{R1}^2 - 2y_{R1}y + y^2$$

$$r_2^2 = x_{R2}^2 - 2x_{R2}x + x^2 + y_{R2}^2 - 2y_{R2}y + y^2$$

By subtracting the second equation from the first equation and combining like terms, we get

$$r_1^2 - r_2^2 = 2x(x_{R2} - x_{R1}) + 2y(y_{R2} - y_{R1}) + x_{R1}^2 + y_{R1}^2 - x_{R2}^2 - y_{R2}^2$$

Finally, solving for x in terms of y yields

$$x = -\frac{y(y_{R2} - y_{R1})}{(x_{R2} - x_{R1})} + \frac{r_1^2 - r_2^2 - x_{R1}^2 - y_{R1}^2 + x_{R2}^2 + y_{R2}^2}{2(x_{R2} - x_{R1})}$$

By defining

$$A = -\frac{(y_{R2} - y_{R1})}{(x_{R2} - x_{R1})}$$

$$B = \frac{r_1^2 - r_2^2 - x_{R1}^2 - y_{R1}^2 + x_{R2}^2 + y_{R2}^2}{2(x_{R2} - x_{R1})}$$

we achieve simplification and obtain

$$x = Ay + B$$

Substituting the preceding solution into the square of the first range equation yields

$$r_1^2 = x_{R1}^2 - 2x_{R1}(Ay + B) + (Ay + B)^2 + y_{R1}^2 - 2y_{R1}y + y^2$$

It is obvious that the preceding equation is a quadratic in terms of y, which can be rewritten as

$$0 = y^2(1 + A^2) + y(-2Ax_{R1} + 2AB - 2y_{R1}) + x_{R1}^2 - 2x_{R1}B + y_{R1}^2 - r_1^2$$

Therefore, we can considerably simplify the preceding quadratic equation by defining

$$a = 1 + A^2$$

$$b = -2Ax_{R1} + 2AB - 2y_{R1}$$

$$c = x_{R1}^2 - 2x_{R1}B + y_{R1}^2 - r_1^2$$

Using the preceding definitions yields the simplified quadratic equation in y as

$$0 = ay^2 + by + c$$

It is well known that the solution to the preceding equation has two roots, indicating that there are two possible receiver locations based only on range information. Using common sense or a priori information (i.e., receiver is not below ground or in outer space), we can throw away the extraneous root to obtain

$$y = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

Once we have solved for y we can then solve for x as

$$x = Ay + B$$

In practice the solution to the two preceding equations yields the coordinates of the estimated location of the receiver. Only in the case in which the satellite location is known precisely and the range measurements from the satellite to the receiver are perfect should the preceding equations yield exact solutions. We will see later that satellite geometry will also be an important factor in determining the solutions.

To help us better understand the satellite geometry, we will use the angle θ, shown in Fig. 11.1, between the two range measurements as a way of specifying

the geometry. We can see from Fig. 11.1 that the angle θ can be calculated from the vector dot product of the two ranges or

$$\theta = \cos^{-1} \frac{\overline{r_1} \cdot \overline{r_2}}{|\overline{r_1}||\overline{r_2}|}$$

Figure 11.1 shows that the range vectors can be expressed as

$$\overline{r_1} = (x_{R1} - x)\bar{i} + (y_{R1} - y)\bar{j}$$

$$\overline{r_2} = (x_{R2} - x)\bar{i} + (y_{R2} - y)\bar{j}$$

whereas the range magnitudes are simply

$$|\overline{r_1}| = r_1$$

$$|\overline{r_2}| = r_2$$

Therefore, substitution yields the solution for the angle θ as

$$\theta = \cos^{-1} \frac{(x_{R1} - x)(x_{R2} - x) + (y_{R1} - y)(y_{R2} - y)}{r_1 r_2}$$

To check the preceding equations, a case was set up in which the satellites were considered to be in an orbit similar to the GPS satellites. In the planar flat Earth model of Fig. 11.1, this means that the satellites travel in a straight line at a fixed altitude. We are assuming that for this example the receiver is located at the origin (i.e., 0 ft downrange and 0 ft in altitude) of Fig. 11.1. Both satellites are at a 20,000 km altitude and are traveling at a speed of 14,600 ft/s (to the left).[1] Initially, let us assume that the first satellite is 1,000,000 ft downrange of the receiver, while the second satellite is 500,000 ft downrange of the receiver. Estimates of the receiver location are calculated every second for 100 s. In the simulation we integrate satellite velocity to get the satellite location. Because in this example the satellite velocity is constant, we could have integrated the satellite velocity, obtaining the location of the satellites as

$$x_{R1} = \dot{x}_{R1t} + x_{R1}(0)$$

$$x_{R2} = \dot{x}_{R2t} + x_{R2}(0)$$

$$y_{R1} = y_{R1}(0)$$

$$y_{R2} = y_{R2}(0)$$

For this example the initial satellite locations are given by

$$x_{R1}(0) = 1{,}000{,}000\,\text{ft}$$

$$x_{R2}(0) = 500{,}000\,\text{ft}$$

$$y_{R1}(0) = 20{,}000 * 3280\,\text{ft}$$

$$y_{R2}(0) = 20{,}000 * 3280\,\text{ft}$$

and the satellite velocities are

$$\dot{x}_{R1} = -14{,}600\,\text{ft/s}$$

$$\dot{x}_{R2} = -14{,}600\,\text{ft/s}$$

Because the satellites are moving, the angle θ will change with time.

The preceding equations for the motion of the two satellites, the angle between the two range vectors, and, most importantly, the solution for the receiver location were programmed and appear in Listing 11.1. Although it was not necessary, we can see from the listing that the satellite velocities are numerically integrated to get the satellite position. Every second, the actual and estimated receiver location is printed out along with the angle between the two range vectors.

The nominal case of Listing 11.1 was run, and the estimated location of the receiver (i.e., XH and YH) matched the actual location of the receiver (i.e., X and Y), indicating the formulas that were just derived are correct. Note that the calculations are repeated every second because the satellites are moving. However, all calculations yield the same location for the receiver, which is at the origin of our coordinate system.

Estimation Without Filtering

To get a feeling for how well we can estimate the location of the stationary receiver in the presence of noisy range measurements from both satellites, the simulation of Listing 11.1 was slightly modified to account for the fact that the range measurements might not be perfect. The resultant simulation with the noisy range measurements appears in Listing 11.2. We can see from the listing that zero-mean, Guassian noise with a standard deviation 300 ft was added to each range measurement every second (i.e., $T_s = 1$). The formulas of the preceding section were used to estimate the receiver location from the noisy measurements, as shown in Listing 11.2. Changes to the original simulation of Listing 11.1 to account for the noisy range measurements are highlighted in bold in Listing 11.2.

The nominal case of Listing 11.2 was run. Figures 11.2 and 11.3 compare the actual and estimated downrange and altitude receiver locations based on the noisy range measurements. We can see that a 300-ft 1 σ noise error on range yields extremely large downrange location errors for this particular geometry. In this

Listing 11.1 Simulation to see if receiver location can be determined from two perfect range measurements

```
      IMPLICIT REAL*8(A-H)
      IMPLICIT REAL*8(O-Z)
      X=0.
      Y=0.
      XR1=1000000.
      YR1=20000.*3280.
      XR2=500000.
      YR2=20000.*3280.
      OPEN(1,STATUS='UNKNOWN',FILE='DATFIL')
      TS=1.
      TF=100.
      T=0.
      S=0.
      H=.01
      WHILE(T<=TF)
            XR1OLD=XR1
            XR2OLD=XR2
            XR1D=-14600.
            XR2D=-14600.
            XR1=XR1+H*XR1D
            XR2=XR2+H*XR2D
            T=T+H
            XR1D=-14600.
            XR2D=-14600.
            XR1=.5*(XR1OLD+XR1+H*XR1D)
            XR2=.5*(XR2OLD+XR2+H*XR2D)
            S=S+H
            IF(S>=(TS-.00001))THEN
                  S=0.
                  R1=SQRT((XR1-X)**2+(YR1-Y)**2)
                  Rs=SQRT((XR2-X)**2+(YR2-Y)**2)
                  A=(YR1-YR2)/(XR2-XR1)
                  B1=(R1**2-R2**2-XR1**2-YR1**2+XR2**2+YR2**2)/
     1               (2.*(XR2-XR1))
                  A=1.+A1**2
                  B=2.*A1*B1-2.*A1*XR1-2.*YR1
                  C=XR1**2-2.*XR1*B1+YR1**2-R1**2
                  YH=(-B-SQRT(B**2-4.*A*C))/(2.*A)
                  XH=A1*YH+B1
                  THET=ACOS(((XR1-X)*(XR2-X)+(YR1-Y)*(YR2-Y))/
     1               (R1*R2))
                  WRITE(9,*)T,X,XH,Y,YH,57.3*THET
                  WRITE(1,*)T,X,XH,Y,YH,57.3*THET
            ENDIF
      END DO
      PAUSE
      CLOSE(1)
      END
```

Listing 11.2 Simulation to see if receiver location can be determined from two noisy range measurements

```
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)
    SIGNOISE=300.
    X=0.
    Y=0.
    XR1=1000000.
    YR1=20000.*3280.
    YR2=500000.
    YR2=20000.*3280.
    OPEN(1,STATUS='UNKNOWN',FILE='DATFIL')
    TS=1.
    TF=100.
    T=0.
    S=0.
    H=.01
    WHILE(T <=TF)
            XR1OLD=XR1
            XR2OLD=XR2
            XR1D=-14600.
            XR2D=-14600.
            XR1=XR1+H*XR1D
            XR2=XR2+H*XR2D
            T=T+H
            XR1D=-14600.
            XR2D=-14600.
            XR1=.5*(XR1OLD+XR1+H*XR1D)
            XR2=.5*(XR2OLD+XR2+H*XR2D)
            S=S+H
            IF(S>=(TS-.00001))THEN
                    S=0.
            CALL GAUSS(R1NOISE,SIGNOISE)
            CALL GAUSS(R2NOISE,SIGNOISE)
            R1=SQRT((XR1-X)**2+(YR1-Y)**2)
            R2=SQRT((XR2-X)**2+(YR2-Y)**2)
            R1S=R1+R1NOISE
            R2S=R2+R2NOISE
            A1=(YR1-YR2)/(XR2-XR1)
            B1=(R1S**2-R2S**2-XR1**2-YR1**2+XR2**2+YR2**2)/
l               (2.*(XR2-XR1))
            A=1.+A1**2
            B=2.*A1*B1-2.*A1*XR1-2.*YR1
            C=XR1**2-2.*XR1*B1+YR1**2-R1S**2
            YH=(-B-SQRT(B**2-4.*A*C))/(2.*A)
            XH=A1*YH+B1
```

(*continued*)

Listing 11.2 *(Continued)*

```
                    THET=ACOS(((XR1-X)*(XR2-X)+(YR1-Y)*(YR2-Y))/
     1                (R1*R2))
                    WRITE(9,*)T,X,XH,Y,YH,57.3*THET,R1,R2
                    WRITE(1,*)T,X,XH,Y,YH,57.3*THET,R1,R2
                ENDIF
            END DO
            PAUSE
            CLOSE(1)
            END
C SUBROUTINE GAUSS IS SHOWN IN LISTING 1.8
```


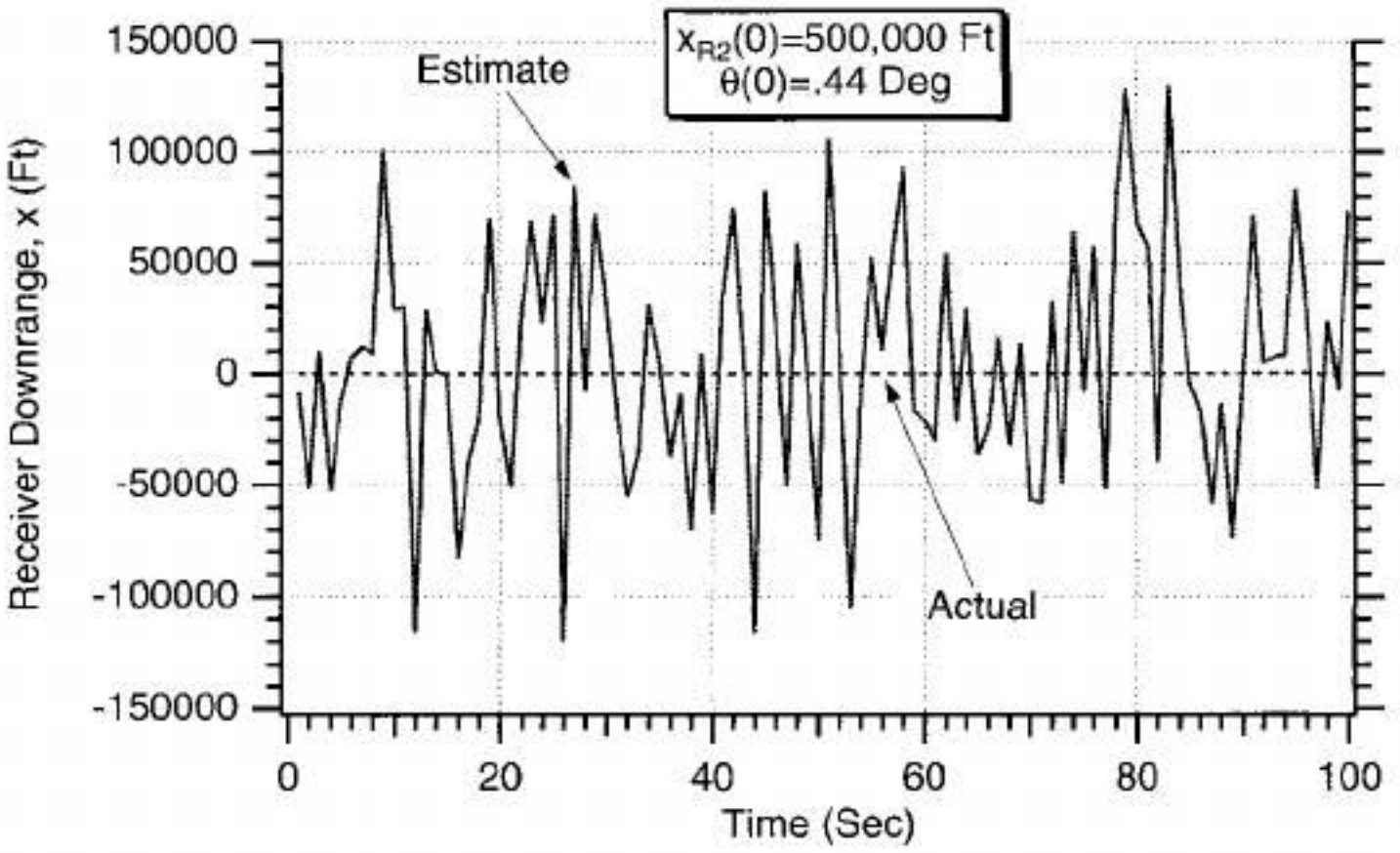

Fig. 11.2 Using raw measurements yields large downrange receiver location errors.

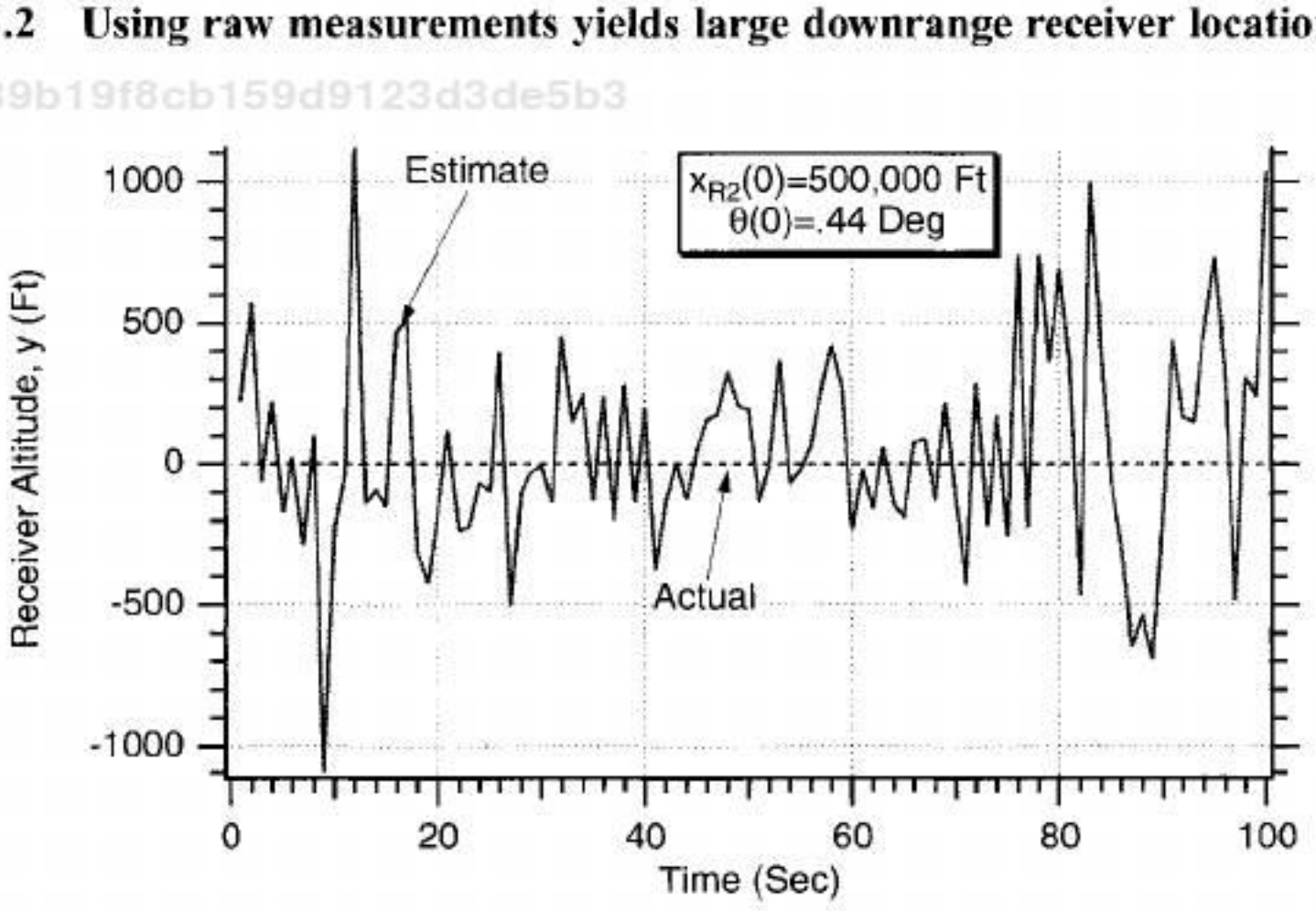

Fig. 11.3 Using raw range measurements yields large altitude receiver location errors.