APPENDIX

Additional Examples

INTRODUCTION

In this appendix additional examples will be presented showing the interested reader how to use the source code listings found in the supporting materials for this book. The listings are often slightly modified so that the reader will understand examples of specific changes that can be made in order to explore issues beyond the scope of the text.

SOFTWARE DETAILS

To facilitate learning, source code that is formatted for both IBM and Macintosh computers containing all of the text's MATLAB listings are included on the AIAA website. The equivalent FORTRAN source code listings can also be found on this website.

The MATLAB and FORTRAN source code should run, as is, with either MATLAB R2010b software or Version 11.05 of the Absoft Macintosh Pro Fortran compiler. The software has been tested on a MacPro Intel Mac Computer. Use of different computers or compilers in either the Macintosh- or IBM-compatible world may require some slight modification of the source code.

The naming conventions of the source code files for both languages are slightly different. The MATLAB naming convention is CxLy.M where x corresponds to Chapter number and y corresponds to listing number. In other words C4L2.M corresponds to MATLAB Listing 4.2 of the text (that is, Chapter 4, Listing 2). The FORTRAN naming convention is C4L2.F.

SENSITIVITY OF OPTIMAL GUIDANCE TO TIME TO GO ERRORS

In evaluating the performance of the optimal guidance law, we have seen tremendous performance benefits over proportional navigation in the presence of guidance system dynamics. It has been assumed that the time to go information, required by optimal guidance, was perfect. Adjoint Listing 8.2 was modified to

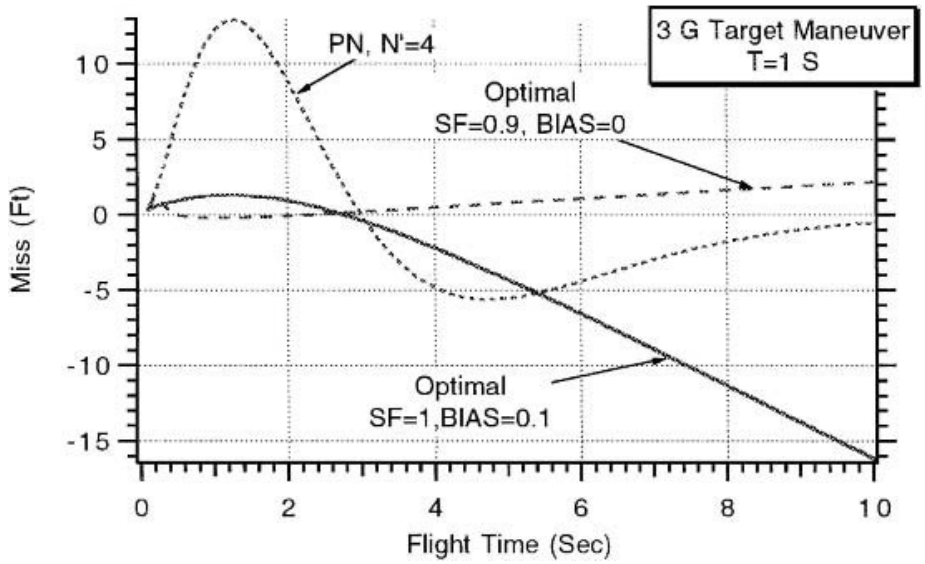

Fig. A.1 Time to go must be known accurately for optimal guidance to yield performance benefits.

include scale factor SF and bias BIAS errors on the estimated time to go TGOH when the optimal guidance option was used (APN = 2). An error-free time to go case would require SF = 1 and BIAS = 0. Time to go errors are not introduced into the proportional navigation option (APN = 0) because in practice this guidance law does not require time to go but works directly on the line-of-sight rate. Statements that have been modified from the original Listing 8.2 are highlighted in boldface in Listing A.1.

Cases were rerun for proportional navigation (APN = 0), optimal guidance with a time to go scale factor error of 0.9 (APN = 2, SF = 0.9, BIAS = 0), and optimal guidance with a time to go bias error of 0.1 s (APN = 2, SF = 1, BIAS = 0.1). The miss distance sensitivity to a 3-*g* target maneuver when the guidance system time constant is 1 s is displayed in Fig. A.1. We can see that both scale factor and bias errors degrade the optimal guidance performance. Figure A.1 shows that the performance of optimal guidance with a 0.9 scale factor error is worse than that of proportional navigation for flight times greater than 8 s and with a 0.1 s bias error is worse than that of proportional navigation for flight times greater than 5.5 s. [1]. Therefore time to go must be known accurately in order for optimal guidance to perform better than proportional navigation.

LISTING A.1 ADJOINT SIMULATION OF OPTIMAL GUIDANCE SYSTEM (MODIFIED LISTING 8.2)

```
XNT=96.6;
XNP=4.;
TAU=1.;
TF=10.;
```

```
VM=3000.;
HEDEG=-20.;
APN=2;
BIAS=.1;
SF=1;
T=0.;
S=0.;
TP=T+.00001;
X1=0.;
X2=0.;
X3=1.;
X4=0.;
XNPP=0.;
H=.01;
HE=HEDEG/57.3;
n=0.;
while TP<=(TF-1e-5)
        X1OLD=X1;
        X2OLD=X2;
        X3OLD=X3;
        X4OLD=X4;
        STEP=1;
        FLAG=0;
        while STEP<=1
                if FLAG==1
                STEP=2;
                X1=X1+H*X1D;
                X2=X2+H*X2D;
                X3=X3+H*X3D;
                X4=X4+H*X4D;
                TP=TP+H;
                end
                TGO=TP+.00001;
                if APN==0
                C1=XNP/(TGO*TGO);
                C2=XNP/TGO;
                C3=0.;
                C4=0.;
                elseif APN==1
                C1=XNP/(TGO*TGO);
                C2=XNP/TGO;
                C3=.5*XNP;
                C4=0.;
                else
                                TGOH=SF*TGO+BIAS;
                                if TGOH<0
```

```
                                    TGOH=.0001;
                                end
                    X=TGOH/TAU;
                    TOP=6.*X*X*(exp(-X)-1.+X);
                    BOT1=2*X*X*X+3.+6.*X-6.*X*X;
                    BOT2=-12.*X*exp(-X)-3.*exp(-2.*X);
                    XNPP=TOP/(.0001+BOT1+BOT2);
                    C1=XNPP/(TGOH*TGOH);
                    C2=XNPP/TGOH;
                    C3=.5*XNPP;
                    C4=-XNPP*(exp(-X)+X-1.)/(X*X);
                    end
                    X1D=X2+C3*X4/TAU;
                    X2D=X3+C2*X4/TAU;
                    X3D=C1*X4/TAU;
                    X4D=-X4/TAU-X2+C4*X4/TAU;
                    FLAG=1;
            end
            FLAG=0;
            X1=(X1OLD+X1)/2+.5*H*X1D;
            X2=(X2OLD+X2)/2+.5*H*X2D;
            X3=(X3OLD+X3)/2+.5*H*X3D;
            X4=(X4OLD+X4)/2+.5*H*X4D;
            S=S+H;
            if S>=.0999
                    S=0.;
                    n=n+1;
                    ArrayTP(n)=TP;
                    ArrayXMNT(n)=XNT*X1;
                    ArrayXMHE(n)=-VM*HE*X2;
            end
end
figure
plot(ArrayTP,ArrayXMNT),grid
xlabel('Flight Time (Sec)')
ylabel('Target Maneuver Miss (Ft)')
clc
output=[ArrayTP',ArrayXMNT',ArrayXMHE'];
save datfil.txt output /ascii
disp 'simulation finished'
```

SIMULATING AN IMPULSE

So far, when the adjoint simulation technique has been used the impulse required for the implementation of the adjoint method has been simulated by finding the

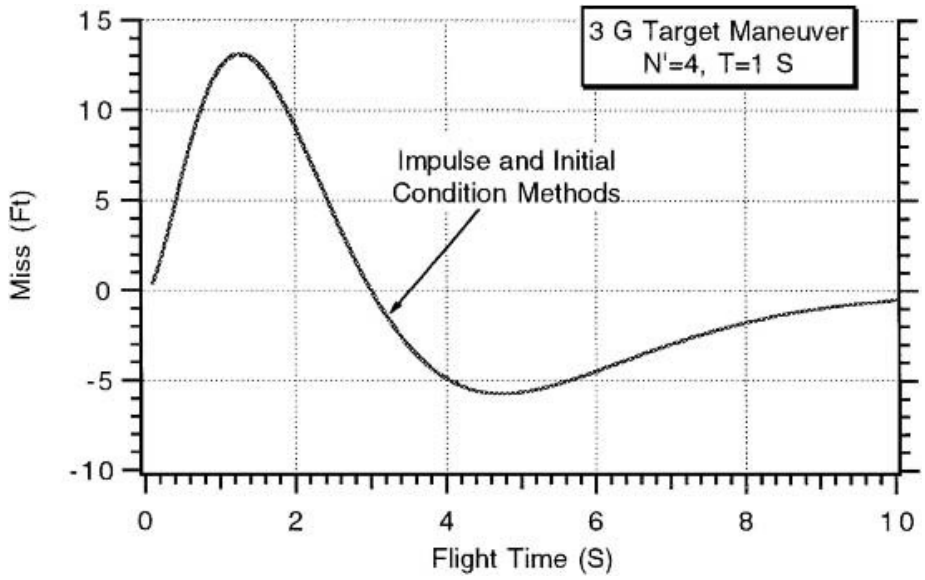

Fig. A.2 Both impulse and initial condition methods are equivalent for target maneuver disturbance.

appropriate initial conditions on the various integrators in the system. In some applications, in which the adjoint method is being automated, it may be advantageous to actually simulate the impulse rather than trying to develop the logic in finding the appropriate initial conditions [2]. Listing 3.1 has been modified to show how the impulse can be simulated. Changes to the original source code have been highlighted in boldface. We can see that before the STEP = 1 statement the height of the impulse is chosen so that the simulated impulse has unit area. The width of the impulse is half an integration interval since the differential equations are called twice with the second-order Runge–Kutta integration technique. Before the STEP = 1 statement we can see that the impulse is added only once to the derivative of *x*3. When we come back to this section of code at other times the value of the simulated impulse will be zero.

We can see from Figs. A.2 and A.3 that the miss due to target maneuver and heading error is identical whether we are using Listing 3.1 where initial conditions are used for the impulse or Listing A.2 where the impulse is actually simulated. However it is important to note that when the impulse is simulated the answers are more sensitive to the integration step size than when the initial condition method is used. This means that the answers in Listing A.2 will start to diverge from the true answers sooner than the answers from Listing 3.1 if the integration interval is made larger.

LISTING A.2 SIMULATING AN IMPULSE RATHER THAN USING INITIAL CONDITIONS FOR USE IN THE ADJOINT METHOD (EQUIVALENT TO LISTING 3.1)

```
n=0.;
XNP=4;
```

```
XNT=96.6;
TAU=1;
TF=10;
VM=3000;
HEDEG=-20;
T=0.;
S=0.;
TP=T+.00001;
X1=0;
X2=0;
X3=0;
X4=0;
H=.01;
HE=HEDEG/57.3;
while TP<=(TF-1e-5)
        S=S+H;
        X1OLD=X1;
        X2OLD=X2;
        X3OLD=X3;
        X4OLD=X4;
        if TP<H/2
                IMPULSE=2./H;
        else
                IMPULSE=0;
        end
        STEP=1;
        FLAG=0;
        while STEP<=1
                if FLAG==1
                        STEP=2;
                        X1=X1+H*X1D;
                        X2=X2+H*X2D;
                        X3=X3+H*X3D;
                        X4=X4+H*X4D;
                        TP=TP+H;
                end
                X1D=X2;
                X2D=X3;
                Y1=(X4-X2)/TAU;
                TGO=TP+.00001;
                if STEP==2
                        IMPULSE=0;
                end
                X3D=XNP*Y1/TGO+IMPULSE;
                X4D=-Y1;
                FLAG=1;
```

```
        end
        FLAG=0;
        X1=(X1OLD+X1)/2+.5*H*X1D;
        X2=(X2OLD+X2)/2+.5*H*X2D;
        X3=(X3OLD+X3)/2+.5*H*X3D;
        X4=(X4OLD+X4)/2+.5*H*X4D;
        if S>=.0999
                S=0.;
n=n+1;
ArrayTP(n)=TP;
                ArrayXMNT(n)=XNT*X1;
                ArrayXMHE(n)=-VM*HE*X2;
        end
end
figure
plot(ArrayTP,ArrayXMNT),grid
xlabel('Flight Time (Sec)')
ylabel('Target Maneuver Miss (Ft)')
figure
plot(ArrayTP,ArrayXMHE),grid
xlabel('Flight Time (Sec)')
ylabel('Heading Error Miss (Ft)')
clc
output=[ArrayTP',ArrayXMNT',ArrayXMHE'];
save datfil.txt output /ascii
disp 'simulation finished'
```

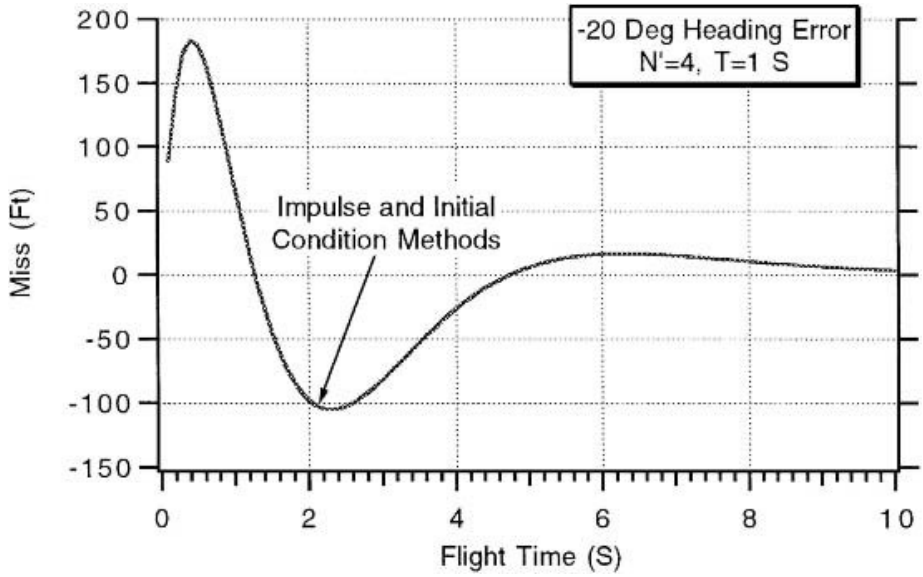

Fig. A.3 Both impulse and initial condition methods are equivalent for heading error disturbance.

DIFFERENT GUIDANCE SYSTEM DISTRIBUTIONS

So far in the text we have chosen the binomial distribution as our canonic guidance system form (that is, all equal time constants). The canonic guidance system form was chosen for simplicity since only the total guidance system time constant had to be specified. It was never demonstrated how the binomial guidance system compared to other canonic guidance system forms in terms of the resultant performance projections.

In this section we will consider two additional guidance system forms. The only feature that the various guidance systems will have in common with the binomial guidance system is that the total guidance system time constant will be the same. The fifth-order binomial guidance system adjoint block diagram of Fig. 6.5 was modified so that other guidance system configurations could be studied by input changes and the resultant adjoint block diagram appears in Fig. A.4. We can see that if $QD = 0$ and all the time constants are the same (i.e., $T_1 = T_2 = T_3 = T_4 = T_5$) we have the fifth-order binomial guidance system that we have already studied. However, now we have the flexibility to make the individual time constants different from one another. If $QD = 1$ in Fig. A.4, we have three real poles and a quadratic distribution for the guidance system. Again the real poles can all have different values. Note that the only disturbance in the guidance system of Fig. A.4 is a step target maneuver.

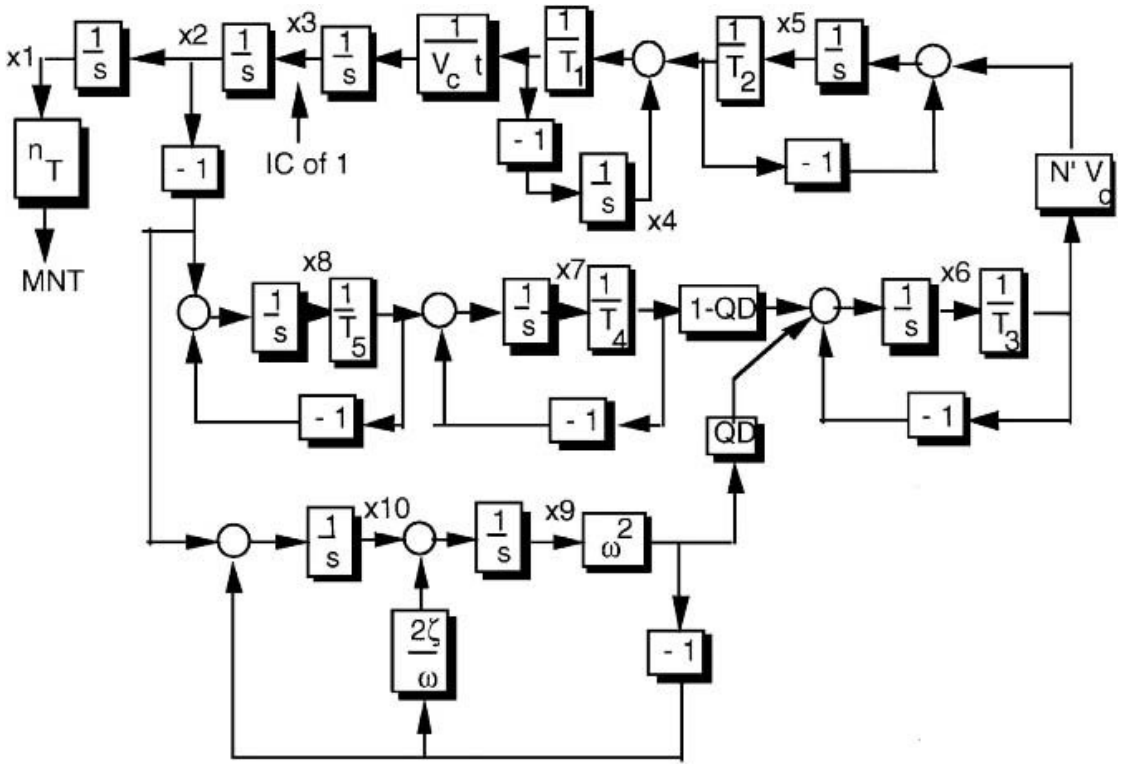

Fig. A.4 Canonic guidance system adjoint block diagram.

The adjoint simulation of Listing 6.1 was modified to correspond to Fig. A.4, and the resultant simulation appears in Listing A.3. The statements that have changed from Listing 6.1 have been highlighted in boldface. We can see from the listing that the only disturbance to the guidance system is a 1-*g* target maneuver.

LISTING A.3 CANONIC GUIDANCE SYSTEM ADJOINT SIMULATION

```
n=0;
QD=0.;
XNT=32.2;
XNP=4.;
T1=.0667;
T2=.133;
T3=.2;
T4=.267;
T5=.333;
W=10;
Z=.7;
TF=10;
VC=4000;
T=0.;
S=0;
TP=T+.00001;
X1=0;
X2=0;
X3=1;
X4=0;
X5=0;
X6=0;
X7=0;
X8=0;
X9=0;
X10=0;
H=.01;
while TP<=(TF-1e-5)
        S=S+H;
        X1OLD=X1;
        X2OLD=X2;
        X3OLD=X3;
        X4OLD=X4;
        X5OLD=X5;
        X6OLD=X6;
        X7OLD=X7;
        X8OLD=X8;
```

```
X9OLD=X9;
X10OLD=X10;
STEP=1;
FLAG=0;
while STEP<=1
        if FLAG==1
        STEP=2;
                        X1=X1+H*X1D;
                        X2=X2+H*X2D;
                        X3=X3+H*X3D;
                        X4=X4+H*X4D;
                        X5=X5+H*X5D;
                        X6=X6+H*X6D;
                        X7=X7+H*X7D;
                        X8=X8+H*X8D;
                        X9=X9+H*X9D;
                        X10=X10+H*X10D;
                        TP=TP+H;
                end
                X1D=X2;
                X2D=X3;
                TGO=TP+.00001;
                X3D=(X4+X5/T2)/(VC*TGO*T1);
                X4D=-(X4+X5/T2)/T1;
                X5D=-X5/T2+XNP*VC*X6/T3;
                X6D=-X6/T3+QD*W*W*X9+(1.-QD)*X7/T4;
                X7D=-X7/T4+X8/T5;
                X8D=-X8/T5-X2;
                X9D=X10-2.*Z*W*X9;
                X10D=-W*W*X9-X2;
                FLAG=1;
end
FLAG=0;
X1=(X1OLD+X1)/2+.5*H*X1D;
X2=(X2OLD+X2)/2+.5*H*X2D;
X3=(X3OLD+X3)/2+.5*H*X3D;
X4=(X4OLD+X4)/2+.5*H*X4D;
X5=(X5OLD+X5)/2+.5*H*X5D;
X6=(X6OLD+X6)/2+.5*H*X6D;
X7=(X7OLD+X7)/2+.5*H*X7D;
X8=(X8OLD+X8)/2+.5*H*X8D;
X9=(X9OLD+X9)/2+.5*H*X9D;
X10=(X10OLD+X10)/2+.5*H*X10D;
if S>=.0999
        S=0;
        XMNT=XNT*X1;
```

```
            n=n+1;
        ArrayTP(n)=TP;
        ArrayXMNT(n)=XNT*X1;
        end
end
figure
plot(ArrayTP,ArrayXMNT),grid
xlabel('Flight Time (Sec)')
ylabel('Target Maneuver Miss (Ft)')
clc
output=[ArrayTP',ArrayXMNT'];
save datfil.txt output /ascii

disp 'simulation finished'
```

For reference the first guidance system studied is the fifth-order binomial guidance system in which all the individual time constants are 0.2 s or

$$\frac{n_L}{\dot{\lambda}} = \frac{N'V_c}{(1+0.2\,\mathrm{s})(1+0.2\,\mathrm{s})(1+0.2\,\mathrm{s})(1+0.2\,\mathrm{s})(1+0.2\,\mathrm{s})}$$

We can see that the total time constant of the preceding expression is 1 s since

$$T = 0.2 + 0.2 + 0.2 + 0.2 + 0.2 = 1\,\mathrm{s}$$

The next guidance system under consideration has five unequal time constants where the second time constant is twice as big as the first, the third time constant is three times as big as the first, the fourth time constant is four times bigger than the first, and the fifth time constant is five times bigger than the first or

$$\frac{n_L}{\dot{\lambda}} = \frac{N'V_c}{(1+0.0667\,\mathrm{s})(1+0.133\,\mathrm{s})(1+0.2\,\mathrm{s})(1+0.267\,\mathrm{s})(1+0.333\,\mathrm{s})}$$

Again we can see that the total time constant of the preceding expression is also 1 s since

$$T = 0.0667 + 0.133 + 0.2 + 0.267 + 0.333 = 1\,\mathrm{s}$$

The last fifth-order guidance system configuration studied is the one with three real poles and a quadratic distribution or

$$\frac{n_L}{\dot{\lambda}} = \frac{N'V_c}{(1+0.1\,\mathrm{s})(1+0.2\,\mathrm{s})(1+0.56\,\mathrm{s})[1+(2*0.7/10)\mathrm{s}+(\mathrm{s}^2/10^2)]}$$

Again we can see that the total time constant of the preceding expression is 1 s since

$$T = 0.1 + 0.2 + 0.56 + 2*0.7/10 = 1\,\mathrm{s}$$

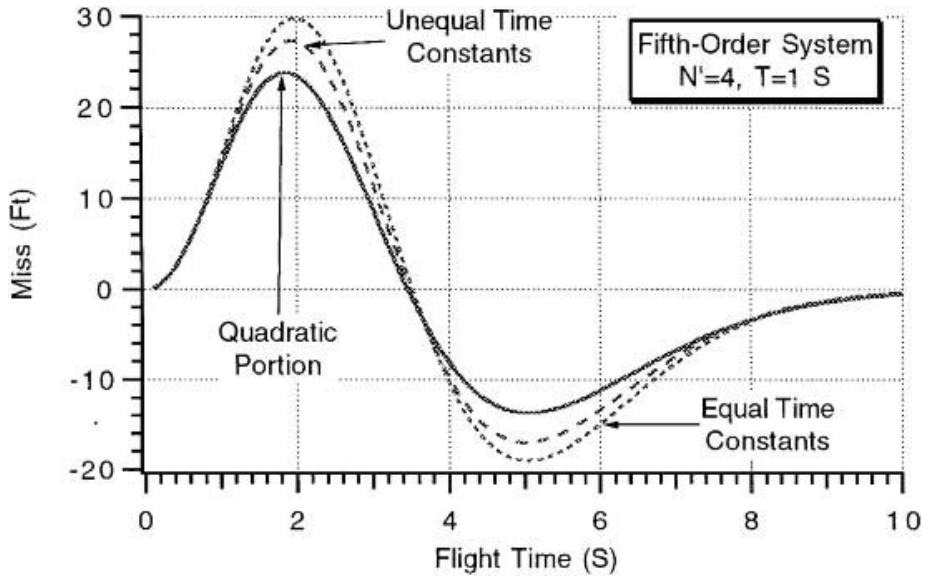

Fig. A.5 All fifth-order guidance system configurations yield approximately same answers if total guidance system time constant is same.

The adjoint simulation results appear in Fig A.5 where the miss distance due to a 1-*g* step target maneuver disturbance is presented as a function of flight time We can see that all of the miss distance curves are close to one another, indicating that the performance of all of the guidance system configurations considered are approximately the same since the total guidance system time constant of each guidance systems is 1 s. This means that the value of the total guidance system time constant is far more important than the guidance system pole distribution.

SAMPLING EXPERIMENTS

In Chapters 7 and 9 we conducted simplified data rate studies with both the digital fading memory and Kalman noise filters. We concluded that the miss distance due to noise and target maneuver tended to decrease as the data rate increased (that is, sampling time decreased). For simplicity, in the data rate studies, the measurement noise standard deviation was held constant as the data rate changed. In many systems, when one gets into the details of the signal processing, it becomes readily apparent that the data rate and measurement noise standard deviation are *not* independent. In these systems the measurement noise spectral density Φ remains constant, which means that the standard deviation of the simulated digital measurement noise is proportional to the square root of the data rate (in other words, inversely proportional to the square root of the sampling time T_s) or

$$\sigma = \sqrt{\frac{\Phi}{T_s}}$$

Therefore if we double the data rate (that is, sampling time halved), we must also increase the standard deviation of the simulated digital measurement noise by 41.4% ($2^{.5} = 1.414$). In this section we shall repeat the experiments of Chapters 7 and 9 to see if the miss due to digital measurement noise still decreases with increasing data rate when the measurement noise spectral density is held constant.

The miss distance adjoint program of Listing 7.3 represents the adjoint of a digital two-state fading memory filter in the homing loop. The program was modified so that a change in the data rate would cause a change in the standard deviation of the measurement noise according to the preceding relationship under the constant spectral density assumption. It was assumed that a sampling time of 0.1 s (10 Hz data rate) corresponded to 1 mr of measurement noise. Adjoint runs were made in which the sampling time was considered a parameter. Figure A.6 shows that the standard deviation of the miss distance due to digital measurement noise still decreases with increasing data rate, although not as dramatically as was the case in Fig. 7.22.

The Monte Carlo simulation used to generate Fig. 9.12 was also modified so that the equivalent spectral density of the measurement noise would remain constant and the standard deviation of the digital measurement noise would vary with data rate. Fifty run Monte Carlo sets were run for 20 different values of flight time at data rates of 2 Hz ($T_s = 0.5$ s), 10 Hz ($T_s = 0.1$ s), and 20 Hz ($T_s = 0.05$ s). We can see from Fig. A.7 that, although the noise miss distance dependence on data rate is not as dramatic as in Fig. 9.13, the miss distance still decreases with increasing data rate (decreasing sampling time).

In summary, we can say that even in cases in which it is appropriate to hold the measurement noise spectral density constant when data rate studies are conducted, the noise induced miss still decreases with increasing data rate when either

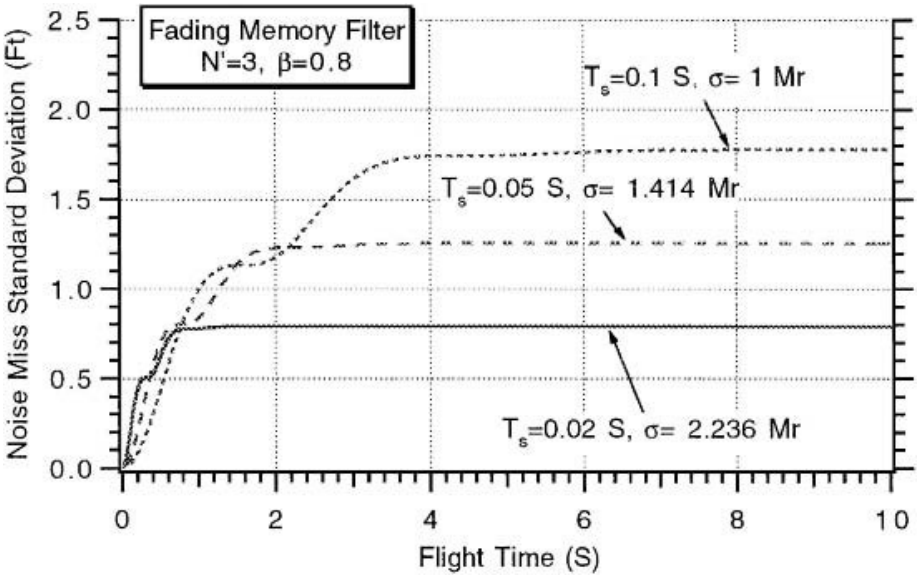

Fig. A.6 Increasing data rate still reduces miss due to digital measurement noise.

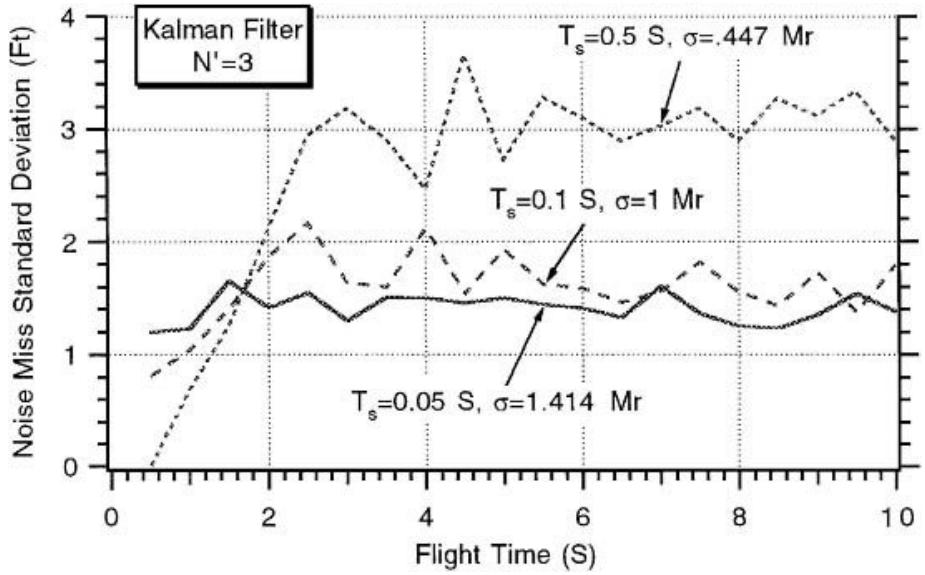

Fig. A.7 Measurement noise miss still increases with decreasing sampling rate.

a fading memory or Kalman filter is used. Both of these digital noise filters take into account information concerning the data rate so that they can both achieve good performance when the data rate changes.

BRUTE FORCE FREQUENCY RESPONSE [3]

We have seen in Chapter 22 how we can find the frequency response of a linear system by first analytically deriving the open-loop transfer function of the system under consideration and then finding its magnitude and phase as a function of frequency. Because a great deal of algebraic manipulation of the system under consideration is involved in this procedure, it is easy to make an error. This section will show that an independent check of the frequency response can be made by using a brute force simulation approach on the system under consideration.

Recall that when we are finding the frequency response of a system we are essentially finding the amplitude and phase of the steady-state sinusoidal output of a linear system driven by a sinusoidal input, as can be seen in Fig. A.8. It is important to note that for a frequency response the sinusoidal output magnitude and phase is found for different sinusoidal input frequencies.

Fig. A.8 Model for brute force frequency response method.

Here, the sinusoidal input to the linear system of Fig. A.8 is given by

$$x = \sin \omega t$$

where ω is the sinusoidal input frequency. Because the system under consideration is linear, the output in the steady state (that is, after transients have died out) must also be a sine wave and can be expressed as

$$y = A \sin(\omega t + \phi)$$

where A is the amplitude of the sinusoidal output and ϕ is the phase angle. **B**ecause the system is linear, the frequency of the system output is the same as the frequency of the input. If we multiply the steady-state system output by a sine wave of the same frequency as the input and integrate the result over a period, we obtain P or

$$P = \int_0^{2\pi/\omega} y \sin \omega t \, dt = \int_0^{2\pi/\omega} A \sin(\omega t + \phi) \sin \omega t \, dt = \frac{A\pi}{\omega} \cos \phi$$

Similarly, if we multiply the steady-state system output by a cosine wave of the same frequency as the input and integrate the result over a period, we obtain Q or

$$Q = \int_0^{2\pi/\omega} y \cos \omega t \, dt = \int_0^{2\pi/\omega} A \sin(\omega t + \phi) \cos \omega t \, dt = \frac{A\pi}{\omega} \sin \phi$$

From the two preceding equations we can see that P and Q are related to the magnitude and phase of the system output according to

$$\sqrt{P^2 + Q^2} = \frac{A\pi}{\omega}$$

$$\phi = \tan^{-1} \frac{Q}{P}$$

In other words, information concerning the magnitude and phase of the system's steady-state sinusoidal output because of a sinusoidal input can be obtained from P and Q. To make the two preceding relationships useful we must first figure out a way to determine when we are in steady state (that is, transients have died out). If we integrate over a period and evaluate P_0, and then integrate again over another period and evaluate P_1, the difference is

$$\Delta P_1 = P_1 - P_0$$

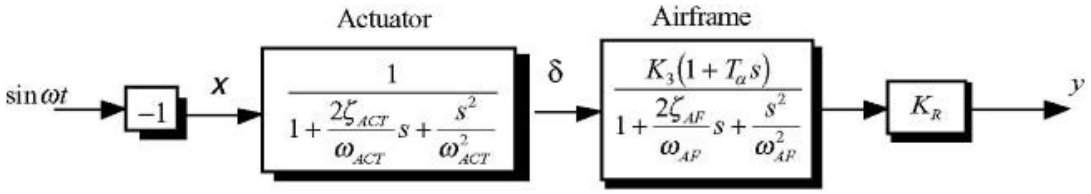

Fig. A.9 Computing frequency response by brute force.

If the difference is very small or zero we know we are in steady state. If not, we evaluate other differences until we are in steady state or

$$\Delta P_2 = P_2 - P_1$$

$$\vdots$$

$$\Delta P_n = P_n - P_{n-1}$$

To see if the brute-force frequency response technique works, let us again consider the rate gyro flight control system that was originally presented in Fig. 22.11 of Chapter 22. If we break the loop at the actuator (as was done in Fig. 22.15) the open-loop transfer function can be obtained by brute force from Fig. A.9. In this diagram, we will first choose a sinusoidal input frequency and then evaluate P and Q when steady state is reached. Another input frequency will be chosen and the process will be repeated. Enough input frequencies will be chosen to compute a proper frequency response.

To simulate Fig. A.9, we must first convert the transfer functions of the block diagram to differential equations. If

$$x = -\sin \omega t$$

we have already shown that by using the chain rule from calculus that the transfer function for the actuator can be converted to the differential equation

$$\ddot{\delta} = \omega_{ACT}^2 \left(x - \delta - \frac{2\zeta_{ACT}}{\omega_{ACT}} \dot{\delta} \right)$$

while the airframe transfer function becomes

$$\ddot{e} = \omega_{AF}^2 \left(\delta - e - \frac{2\zeta_{AF}}{\omega_{AF}} \dot{e} \right)$$

and the system output becomes

$$y = K_R K_3 (e + T_\alpha \dot{e})$$

The computerized method for finding the brute-force frequency response appears in Listing A.4. Notice that the integration interval is small as in other simulations of the rate gyro flight control system. Error criteria are set to

ensure that P is in steady-state. For example, we assume that it takes at least 20 s for transients to die out. At the lower frequencies it will take longer for the transients to die out, and therefore we have an additional error criteria (that is, test when differences in P over a period are sufficiently small). We can see from Listing A.4 that the "For loop" increments the sinusoidal input frequency in an intelligent manner. The resultant magnitude and phase of the system output for each input frequency is printed out and also written to a file.

LISTING A.4 BRUTE-FORCE FREQUENCY RESPONSE PROGRAM

```
% Program runs very slowly
n=0;
ZACT=.7;
WACT=150;
K3=-1.89;
TA=.457;
ZAF=.058;
WAF=25.3;
KR=.1;
PI=3.1416;
H=.0001;
for I=2:160
        W=10^(.025*I-1);
        PERIOD=2.*PI/W;
        T=0.;
        S=0.;
        E=0.;
        ED=0.;
        DEL=0.;
        DELD=0.;
        P=0.;
        Q=0;
        PPREV=0;
        QPREV=0;
        DELP=0;
        DELQ=0;
        DELPOLD=0;
        DELQOLD=0;
        DELDELP=100;
        DELDELQ=100;
        while ~(T>20. & abs(DELDELP)<.0001)
                EOLD=E;
                EDOLD=ED;
                DELOLD=DEL;
                DELDOLD=DELD;
```

```
        POLD=P;
        QOLD=Q;
        STEP=1;
        FLAG=0;
        while STEP<=1
        if FLAG==1
        STEP=2;
                E=E+H*ED;
                ED=ED+H*EDD;
                DEL=DEL+H*DELD;
                DELD=DELD+H*DELDD;
                P=P+H*PD;
                Q=Q+H*QD;
                T=T+H;
            end
            X=-sin(W*T);
            DELDD=WACT*WACT*(X-DEL-2.*ZACT*DELD/WACT);
            EDD=WAF*WAF*(DEL-E-2.*ZAF*ED/WAF);
            Y=KR*K3*(E+TA*ED);
            PD=Y*sin(W*T);
            QD=Y*cos(W*T);
            FLAG=1;
        end
        FLAG=0;
        E=.5*(EOLD+E+H*ED);
        ED=.5*(EDOLD+ED+H*EDD);
        DEL=.5*(DELOLD+DEL+H*DELD);
        DELD=.5*(DELDOLD+DELD+H*DELDD);
        P=.5*(POLD+P+H*PD);
        Q=.5*(QOLD+Q+H*QD);
        S=S+H;
        if (S>=(PERIOD-.0001))
        S=0.;
            DELP=P-PPREV;
            DELQ=Q-QPREV;
            PPREV=P;
            QPREV=Q;
            DELDELP=DELPOLD-DELP;
            DELDELQ=DELQOLD-DELQ;
            DELPOLD=DELP;
            DELQOLD=DELQ;
        end
end
PHASE=57.3*atan2(DELQ,DELP);
if PHASE>90.
        PHASE=PHASE-360;
```

```
        end
        GAIN=10.*log10((DELP^2+DELQ^2)*W*W/(PI*PI));
        n=n+1;
        ArrayW(n)=W;
ArrayPHASE(n)=PHASE;
ArrayGAIN(n)=GAIN;
end
figure
plot(ArrayW,ArrayGAIN),grid
xlabel('Frequency (r/s)')
ylabel('Gain (db)')
figure
plot(ArrayW,ArrayPHASE),grid
xlabel('Frequency (r/s)')
ylabel('Phase (deg)')
clc
output=[ArrayW',ArrayGAIN',ArrayPHASE'];
save datfil.txt output /ascii
disp 'simulation finished'
```

Listing A.4 was run for the nominal case, and Fig. A.10 displays the open-loop frequency response obtained by the method of brute force. By comparing this open-loop response to that of Fig. 22.16, we can see that both responses are identical in magnitude and phase. However, the analytic frequency response method of Chapter 22 (namely, Listing 22.2) yields the answers faster because numerical integration techniques are not involved. It is important to note that when using the analytical frequency response method, it is easier to make mistakes in setting up the program. On the other hand, the brute-force method has a longer computer running time (the running time in MATLAB is more than

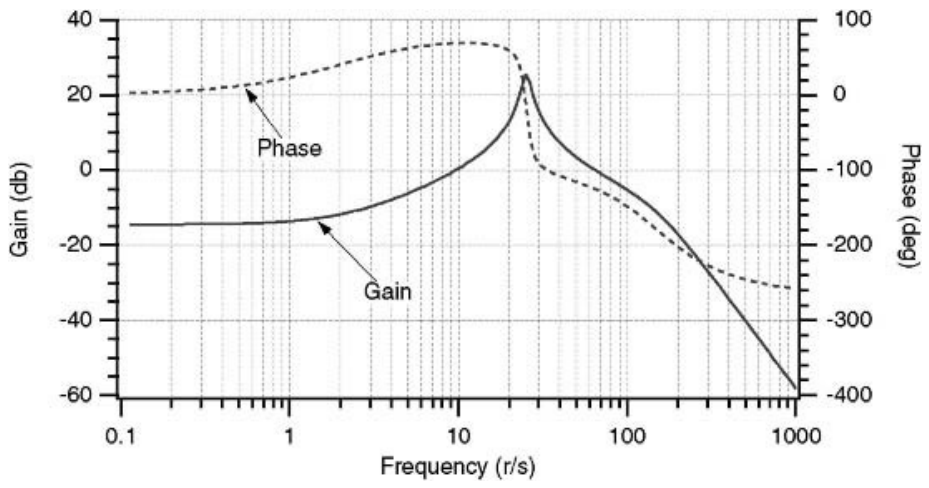

Fig. A.10 Brute-force method results are identical to those of Fig. 22.16.

TABLE A.1 BOTH METHODS ARE NUMERICALLY IDENTICAL

	Frequency domain		Time domain	
ω, r/s	Gain, db	Phase, deg	Gain, db	Phase, deg
0.112	−14.46	2.85	−14.46	2.85
0.501	−14.25	12.5	−14.25	12.5
3.981	−7.91	58.0	−7.91	58.0
33.4966	11.53	−100.43	11.53	−100.45
149.624	−11.30	−179.5	−11.30	−179.4
1000	−58.1	−257.85	−58.1	−258.53

15 minutes while in FORTAN it is less than a minute) because numerical integration is involved for each input frequency, but there is less chance of making a mistake in setting up the program because it is similar to simulation.

Table A.1 shows, for selected frequencies, how the frequency and time domain approaches compare. It can be assumed that the frequency domain answers are exact and the time domain answers are approximate. We can see that the gain is identical for both methods, but at the higher frequencies the phase angle obtained by the time domain method is in slight error.

In summary, we can say that the brute-force method for finding the open-loop frequency response of a system can be used as a useful check on analytically derived answers.

MINIMUM ENERGY TRAJECTORIES

We derived the hit equation in Chapter 11. The formula for the required velocity for an impulsive missile to travel a certain distance was given by

$$V = \sqrt{\frac{gm(1 - \cos\phi)}{a\cos\gamma[\cos\gamma - \cos(\phi + \gamma)]}}$$

where ϕ was related to the distance to be traveled, gm was a gravitational constant, a was the radius of the Earth, and γ was the flight path angle.

A minimum energy trajectory is one in which the velocity to travel a certain distance is minimized. To minimize the velocity in the preceding expression, we set the derivative of the velocity with respect to the flight path angle to zero or

$$\frac{dV}{d\gamma} = 0$$

After much algebra one can show that the flight path angle that yields minimum energy trajectories γ_{ME} is given by [4]

$$\gamma_{ME} = \frac{\pi}{4} - \frac{\phi}{4}$$

Recall that when we use Lambert guidance we specify where we are, where we want to go, and the amount of time it takes to get there. Therefore, in future uses of Lambert guidance for minimum energy trajectories we would like to specify the flight time that yields the appropriate flight path angle γ_{ME}. Recall in Chapter 11 we showed that the formula for the flight time was given by

$$t_F = \frac{a}{V\cos\gamma}\left\{\frac{\tan\gamma(1-\cos\phi)+(1-\lambda)\sin\phi}{(2-\lambda)\left[\dfrac{1-\cos\phi}{\lambda\cos^2\gamma}+\dfrac{\cos(\gamma+\phi)}{\cos\gamma}\right]}\right.$$
$$\left.+\frac{2\cos\gamma}{\lambda\left(\dfrac{2}{\lambda}-1\right)^{1.5}}\tan^{-1}\left[\frac{\sqrt{\frac{2}{\lambda}-1}}{\dfrac{\cos\gamma}{\tan\frac{\phi}{2}}-\sin\gamma}\right]\right\}$$

Listing A.5 programs the preceding formulas for the minimum energy flight path angle, velocity, and flight time. We can see that a loop appears in which the downrange distance to be traveled is varied from 100 to 20,000 km in steps of 100 km. Listing A.12 then calculates the flight time that corresponds to each minimum energy trajectory.

The simulation of Listing A.5 was run, and the minimum energy flight time corresponding to each downrange to be traveled is displayed in Fig. A.11. We

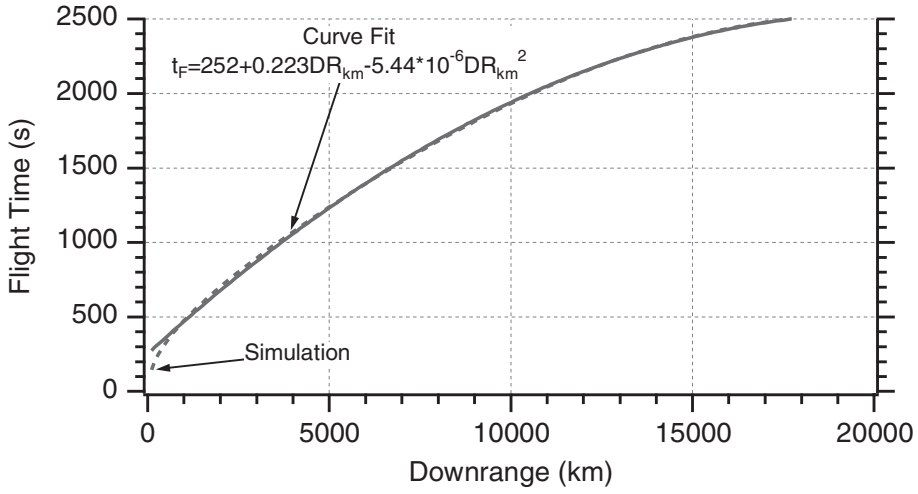

Fig. A.11 Deriving flight time formula for minimum energy trajectories.

can see that it appears that the flight time for a minimum energy trajectory is a linear function of downrange (that is, distance to be traveled in kilometers). Superimposed on the graph is the best linear least-squares curve fit to the simulation results. We can see that if the downrange to be traveled is expressed in kilometers then the minimum energy flight time in units of seconds can be expressed as

$$t_{F_{ME}} = 252 + 0.223DR_{\text{km}} - 5.44 * 10^{-6}DR_{\text{km}}^2$$

LISTING A.5 CALCULATING FLIGHT TIME FOR A MINIMUM ENERGY TRAJECTORY

```
n=0;
GM=1.4077E16;
A=2.0926E7;
CONST=sqrt(GM/A);
for DISTKM=100:100:20000,
        PHI=DISTKM*3280./A ;
        GAM=3.14159/4.-PHI/4.;
        GAMDEG=57.3*GAM;
        TOP=GM*(1.-cos(PHI));
        TEMP=A*cos(GAM)/A-cos(PHI+GAM);
        BOT=A*cos(GAM)*TEMP;
        V=sqrt(TOP/BOT);
        XLAM=A*V*V/GM;
        TOP1=tan(GAM)*(1-cos(PHI))+(1-XLAM)*sin(PHI);
        BOT1P=(1-cos(PHI))/(XLAM*cos(GAM)*cos(GAM));
        BOT1=(2-XLAM)*(BOT1P+cos(GAM+PHI)/cos(GAM));
        TOP2=2*cos(GAM);
        BOT2=XLAM*((2/XLAM-1)^1.5);
        TOP3=sqrt(2/XLAM-1);
        BOT3=cos(GAM)/tan(PHI/2)-sin(GAM);
        TEMP=(TOP2/BOT2)*atan2(TOP3,BOT3);
        TF=A*(TOP1/BOT1+TEMP)/(V*cos(GAM));
        n=n+1;
        ArrayDISTKM(n)=DISTKM;
        ArrayTF(n)=TF;
end
figure
plot(ArrayDISTKM,ArrayTF),grid
xlabel('Distance (km)')
ylabel('Flight Time (s)')
clc
output=[ArrayDISTKM',ArrayTF'];
save datfil.txt output /ascii
disp 'simulation finished'
```

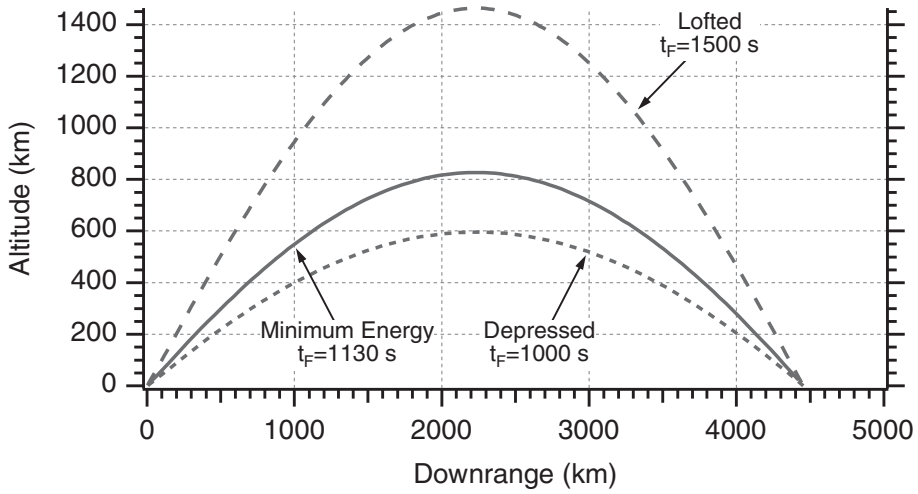

Fig. A.12 Minimum energy trajectory compared with lofted and depressed trajectories.

To test the minimum energy formula for flight time, Listing 13.3 was modified to include the new formula. We can see that the distance to be traveled is 40 deg (namely, 70 deg – 30 deg) or approximately 4500 km. In this simulation, a two-stage booster flies to its intended target according to Lambert guidance.

The first case run was for a minimum energy trajectory. In this case, the flight time turned out to be 1130 s. Trajectories corresponding to flight times of 1000 s and 1500 s were also run. We can see from Fig. A.12 that decreasing the flight time from the minimum energy value depresses the trajectory, whereas increasing the flight time from its minimum energy value lofts the trajectory.

To ensure that a flight time of 1130 s corresponds to a minimum energy trajectory, a comparison was made of the velocity profiles for each of the three trajectories. Figure A.13 shows that the final velocity for the 1000 s trajectory is

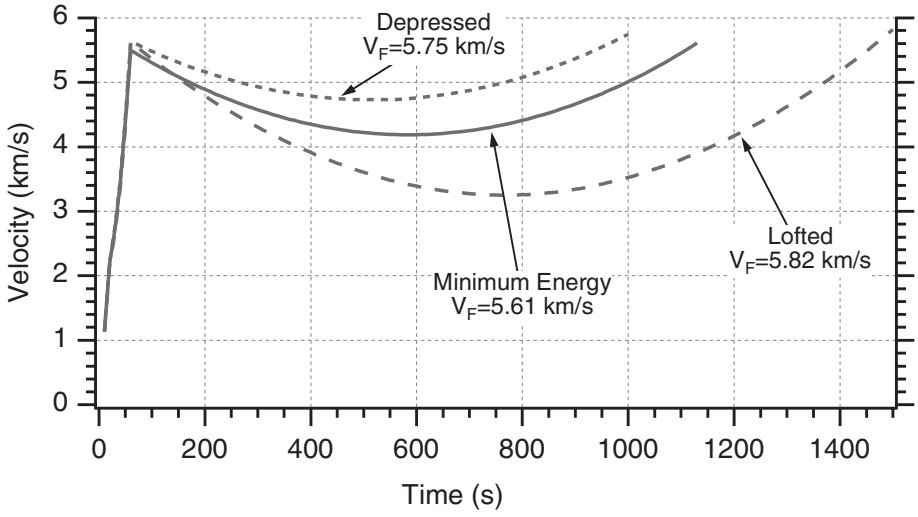

Fig. A.13 Minimum energy trajectory has least velocity at the end of the flight.

5.75 km/s, the final velocity for the 1500 s trajectory is 5.82 km/s, and the final velocity for the minimum energy trajectory is 5.61 km/s. As expected, the minimum energy trajectory yields the smallest final velocity. However, it is important to note that the velocity differential between all three trajectories is not very large.

TRAJECTORY SHAPING GUIDANCE IN THREE DIMENSIONS

In Chapter 24 we showed that the trajectory shaping guidance law in one dimension was given by

$$n_c(t) = \frac{6y + 4\dot{y}t_{\text{go}} + n_T t_{\text{go}}^2 + 2\dot{y}(t_F)t_{\text{go}}}{t_{\text{go}}^2}$$

where y was the relative position between the target and missile, $\dot{y}$ was the relative velocity between target and missile, n_T was the target acceleration, and $\dot{y}(t_F)$ was the desired relative velocity at intercept between the target and missile. In Chapter 28 we showed how several of the guidance laws presented in the text could be programmed in three dimensions. In this section we shall also show how we can convert the trajectory shaping guidance law to three dimensions. Before we can convert the preceding guidance law to three dimensions, it is first necessary to write the trajectory shaping guidance law in a slightly different form than was presented in Chapter 24. Expanding the final relative velocity term of the trajectory shaping guidance law yields

$$n_c = \frac{6y + 4\dot{y}t_{\text{go}} + n_T t_{\text{go}}^2 + 2[\dot{y}_T(t_F) - \dot{y}_M(t_F)]t_{\text{go}}}{t_{\text{go}}^2}$$

If we assume that the target velocity does not change very much during the flight, then we can say that

$$\dot{y}_T(t_F) \approx \dot{y}_T$$

and the trajectory shaping guidance law can then be rewritten as

$$n_c = \frac{6y + 4\dot{y}t_{\text{go}} + n_T t_{\text{go}}^2 + 2[\dot{y}_T - \dot{y}_M(t_F)]t_{\text{go}}}{t_{\text{go}}^2}$$

Because we know that

$$\dot{y} = \dot{y}_T - \dot{y}_M$$

we can also say that

$$n_c = \frac{6y + 4\dot{y}t_{\text{go}} + n_T t_{\text{go}}^2 + 2[\dot{y} + \dot{y}_M - \dot{y}_M(t_F)]t_{\text{go}}}{t_{\text{go}}^2}$$

or

$$n_c = \frac{6y + 6\dot{y}t_{\text{go}} + n_T t_{\text{go}}^2 + 2[\dot{y}_M - \dot{y}_M(t_F)]t_{\text{go}}}{t_{\text{go}}^2}$$

LISTING A.6 TRAJECTORY SHAPING GUIDANCE LAW IN THREE-DIMENSIONAL SIMULATION

```
n=0;
XNTG=4.;
VT=1000.;
VM=3000.;
RM1=0.;
RM2=10000.;
RM3=-1000.;
RT1=30000.;
RT2=10000.;
RT3=0.;
GAMFPDEG= -30.;
GAMFYDEG= 20.;
XNT=32.2*XNTG;
BETA=0.;
VT1=-VT*cos(BETA);
VT2=VT*sin(BETA);
VT3=0;
GAMFP=GAMFPDEG/57.3;
GAMFY=GAMFYDEG/57.3;
[VM1,VM2,VM3,TF]=LAUNCHLOGIC(RM1,RM2,RM3,RT1,RT2,RT3,VT1,VT2,...
                                                    VT3,VM);
H=.0001;
T=0.;
S=0.;
RTM1=RT1-RM1;
RTM2=RT2-RM2;
RTM3=RT3-RM3;
RTM=sqrt(RTM1^2+RTM2^2+RTM3^2);
VTM1=VT1-VM1;
VTM2=VT2-VM2;
VTM3=VT3-VM3;
```

```
VC=-(RTM1*VTM1+RTM2*VTM2+RTM3*VTM3)/RTM;
VM1F=VM*cos(GAMFP)*cos(GAMFY);
VM2F=VM*sin(GAMFP);
VM3F=VM*cos(GAMFP)*sin(GAMFY);
while ~(VC<0)
        if RTM<1000.
                H=.00001;
        else
                H=.0001;
        end
        BETAOLD=BETA;
        RT1OLD=RT1;
        RT2OLD=RT2;
        RT3OLD=RT3;
        RM1OLD=RM1;
        RM2OLD=RM2;
        RM3OLD=RM3;
        VM1OLD=VM1;
        VM2OLD=VM2;
        VM3OLD=VM3;
        STEP=1;
        FLAG=0;
        while STEP<=1
                if FLAG==1
                        STEP=2;
                        BETA=BETA+H*BETAD;
                        RT1=RT1+H*VT1;
                        RT2=RT2+H*VT2;
                        RT3=RT3+H*VT3;
                        RM1=RM1+H*VM1;
                        RM2=RM2+H*VM2;
                        RM3=RM3+H*VM3;
                        VM1=VM1+H*AM1;
                        VM2=VM2+H*AM2;
                        VM3=VM3+H*AM3;
                        T=T+H;
                end
                RTM1=RT1-RM1;
                RTM2=RT2-RM2;
                RTM3=RT3-RM3;
                RTM=sqrt(RTM1^2+RTM2^2+RTM3^2);
                VTM1=VT1-VM1;
                VTM2=VT2-VM2;
                VTM3=VT3-VM3;
                VC=-(RTM1*VTM1+RTM2*VTM2+RTM3*VTM3)/RTM;
                TGO=RTM/VC;
```

```
        BETAD=XNT/VT;
        XNT1=XNT*sin(BETA);
        XNT2=XNT*cos(BETA);
        XNT3=0.;
        VT1=-VT*cos(BETA);
        VT2= VT*sin(BETA);
        VT3=0.;
        VM=sqrt(VM1*VM1+VM2*VM2+VM3*VM3);
        XNC1=((6.*RTM1+6.*VTM1*TGO)/TGO^2) +2.0*(VM1-VM1F)/(TGO)+XNT1;
        XNC2=((6.*RTM2+6.*VTM2*TGO)/TGO^2) +2.0*(VM2-VM2F)/(TGO)+XNT2;
        XNC3=((6.*RTM3+6.*VTM3*TGO)/TGO^2) +2.0*(VM3-VM3F)/(TGO)+XNT3;
        XNCG=sqrt(XNC1^2+XNC2^2+XNC3^2)/32.2;
        XNCDOTVM=(XNC1*VM1+XNC2*VM2+XNC3*VM3)/VM;
        AM1=XNC1-XNCDOTVM*VM1/VM;
        AM2=XNC2-XNCDOTVM*VM2/VM;
        AM3=XNC3-XNCDOTVM*VM3/VM;
        GAMYDEG=57.3*atan2(VM3,VM1);
        GAMPDEG=57.3*atan2(VM2,sqrt(VM1^2+VM3^2));
        FLAG=1;
end
FLAG=0;
BETA=.5*(BETAOLD+BETA+H*BETAD);
RT1=.5*(RT1OLD+RT1+H*VT1);
RT2=.5*(RT2OLD+RT2+H*VT2);
RT3=.5*(RT3OLD+RT3+H*VT3);
RM1=.5*(RM1OLD+RM1+H*VM1);
RM2=.5*(RM2OLD+RM2+H*VM2);
RM3=.5*(RM3OLD+RM3+H*VM3);
VM1=.5*(VM1OLD+VM1+H*AM1);
VM2=.5*(VM2OLD+VM2+H*AM2);
VM3=.5*(VM3OLD+VM3+H*AM3);
S=S+H;
if S>=.0999
        S=0.;
        n=n+1;
        RT1K=RT1/1000.;
        RT2K=RT2/1000.;
        RT3K=RT3/1000.;
        RM1K=RM1/1000.;
        RM2K=RM2/1000.;
        RM3K=RM3/1000.;
        ArrayT(n)=T;
        ArrayRM1K(n)=RM1K;
        ArrayRM2K(n)=RM2K;
        ArrayRM3K(n)=RM3K;
        ArrayRT1K(n)=RT1K;
```

```
                ArrayRT2K(n)=RT2K;
                ArrayRT3K(n)=RT3K;
                ArrayGAMPDEG(n)=GAMPDEG;
                ArrayGAMFPDEG(n)=GAMFPDEG;
                ArrayGAMYDEG(n)=GAMYDEG;
                ArrayGAMFYDEG(n)=GAMFYDEG;
                ArrayXNCG(n)=XNCG;
   end
end
n=n+1;
                RT1K=RT1/1000.;
                RT2K=RT2/1000.;
                RT3K=RT3/1000.;
                RM1K=RM1/1000.;
                RM2K=RM2/1000.;
                RM3K=RM3/1000.;
                ArrayT(n)=T;
                ArrayRM1K(n)=RM1K;
                ArrayRM2K(n)=RM2K;
                ArrayRM3K(n)=RM3K;
                ArrayRT1K(n)=RT1K;
                ArrayRT2K(n)=RT2K;
                ArrayRT3K(n)=RT3K;
                ArrayGAMPDEG(n)=GAMPDEG;
                ArrayGAMFPDEG(n)=GAMFPDEG;
                ArrayGAMYDEG(n)=GAMYDEG;
                ArrayGAMFYDEG(n)=GAMFYDEG;
                ArrayXNCG(n)=XNCG;
figure
plot(ArrayRM1K,ArrayRM2K,ArrayRT1K,ArrayRT2K),grid
xlabel('Downrange (kft)')
ylabel('Altitude (kft)')
figure
plot(ArrayRM1K,ArrayRM3K,ArrayRT1K,ArrayRT3K),grid
xlabel('Downrange (kft)')
ylabel('Crossrange (kft)')
figure
plot(ArrayT,ArrayGAMPDEG,ArrayT,ArrayGAMFPDEG),grid
xlabel('Time (Sec)')
ylabel('Pitch Reentry Angle (deg)')
figure
plot(ArrayT,ArrayGAMYDEG,ArrayT,ArrayGAMFYDEG),grid
xlabel('Time (Sec)')
ylabel('Yaw Reentry Angle (deg)')
figure
plot(ArrayT,ArrayXNCG),grid
```

```
xlabel('Time (Sec)')
ylabel('Commanded Acceleration (g)')
axis([00,10,00,60])
clc
output=[ArrayT',ArrayRM1K',ArrayRM2K',ArrayRM3K',ArrayRT1K',...
        ArrayRT2K',ArrayRT3K',ArrayGAMPDEG',ArrayGAMFPDEG',...
        ArrayGAMYDEG',ArrayGAMFYDEG'];
save datfil.txt output -ascii
disp 'simulation finished'
RTM

function [VM1,VM2,VM3,TF]=LAUNCHLOGIC(RM1,RM2,RM3,RT1,RT2,RT3,...
                                      VT1,VT2,VT3,VM)
for TF=.1:.1:10,
        RTM1=RT1-RM1;
        RTM2=RT2-RM2;
        RTM3=RT3-RM3;
        RT1F=RT1+VT1*TF;
        RT2F=RT2+VT2*TF;
        RT3F=RT3+VT3*TF;
        THET=asin((RT2F-RM2)/(VM*TF));
        PSI=atan2(RT3F-RM3,RT1F-RM1);
        VM1=VM*cos(THET)*cos(PSI);
        VM2=VM*sin(THET);
        VM3=VM*cos(THET)*sin(PSI);
        RM1F=RM1+VM1*TF;
        RM2F=RM2+VM2*TF;
        RM3F=RM3+VM3*TF;
        RTM1F=RT1F-RM1F;
        RTM2F=RT2F-RM2F;
        RTM3F=RT3F-RM3F;
        RTMF=sqrt(RTM1F^2+RTM2F^2+RTM3F^2);
        VTM1=VT1-VM1;
        VTM2=VT2-VM2;
        VTM3=VT3-VM3;
        VC=-(RTM1F*VTM1+RTM2F*VTM2+RTM3F*VTM3)/RTMF;
        if VC<0
                break
        end
end
```

An examination of the term in brackets of the preceding expression indicates that we are attempting to make the missile velocity $\dot{y}_M$ reach a specified value at the end of the flight. Soon we will show that this is equivalent to controlling the missile flight-path angle. By duplicating the expression for the trajectory

shaping guidance law in each of the Earth's inertial coordinates, we can express the guidance law in three dimensions by inspection as

$$n_{c_1} = \frac{6R_{TM1} + 6V_{TM1}t_{\text{go}} + n_{T1}t_{\text{go}}^2 + 2[V_{M1} - V_{M1}(t_F)]t_{\text{go}}}{t_{\text{go}}^2}$$

$$n_{c_2} = \frac{6R_{TM2} + 6V_{TM2}t_{\text{go}} + n_{T2}t_{\text{go}}^2 + 2[V_{M2} - V_{M2}(t_F)]t_{\text{go}}}{t_{\text{go}}^2}$$

$$n_{c_3} = \frac{6R_{TM3} + 6V_{TM3}t_{\text{go}} + n_{T3}t_{\text{go}}^2 + 2[V_{M3} - V_{M3}(t_F)]t_{\text{go}}}{t_{\text{go}}^2}$$

where R_{TM} and V_{TM} are relative position and velocity, respectively. In the preceding expressions, 1, 2, and 3 represent downrange, altitude, and cross range, respectively, in the Earth or inertial coordinate system. Thus the guidance commands are in each of those directions. For the trajectory shaping guidance law we want to make the total acceleration perpendicular to the missile velocity vector. This will ensure that the missile velocity will remain constant throughout the flight. A similar approach was taken in [5].

If we want to make the missile hit the target at desired flight-path angles γ_{PF} and γ_{YF}, we can say that the desired missile velocity components at the end of the flight are given by

$$V_{M1}(t_F) = V_M \cos \gamma_{\text{PF}} \cos \gamma_{\text{YF}}$$
$$V_{M2}(t_F) = V_M \sin \gamma_{\text{PF}}$$
$$V_{M3}(t_F) = V_M \cos \gamma_{\text{PF}} \sin \gamma_{\text{YF}}$$

where V_M is the total missile velocity. The trajectory shaping guidance law was implemented in the three-dimensional simulation of Listing A.6. In the nominal case of the simulation, the target is executing a constant 4-g maneuver perpendicular to the target velocity vector in the altitude-downrange plane. In addition, it is desired that the missile hit the target with a pitch flight-path angle of -30 deg and a yaw flight-path angle of 20 deg. The simulation calculates the instantaneous pitch and yaw flight-path angles, as well as the miss distance, to see if the trajectory shaping guidance law meets its objectives. A missile launch logic subroutine, valid for flight times of less than 10 s, is included to place the missile on a collision triangle (assuming no target maneuver) with the target.

The nominal case of Listing A.6 was run. The resultant altitude-downrange and crossrange-downrange trajectories of Figs A.14 and A.15 indicate that the missile is hitting the target. In addition we can see that there is much curvature to the missile, trajectory indicating that a great deal of trajectory shaping has taken place.

Figures A.16 and A.17 present the missile pitch and yaw flight-path angle profiles, respectively. We can see from both figures that the missile is meeting the

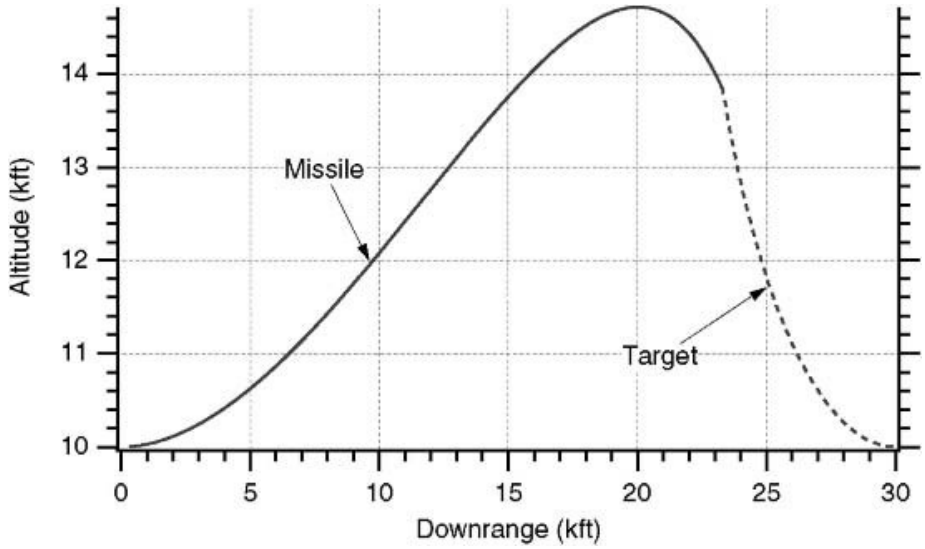

Fig. A.14 Engagement viewed in altitude-downrange plane.

desired pitch and yaw flight-path-angle objectives at the end of the flight. However we also can see from Fig. A.18 that the price paid for the trajectory shaping is a very large commanded missile acceleration. Chapter 24 goes into detail in showing how the acceleration requirements of trajectory shaping guidance can be reduced.

MODELING POISSON TARGET MANEUVER

So far in this text we have mainly considered the uniformly distributed step or weave target maneuvers for evaluating guidance system performance. We were

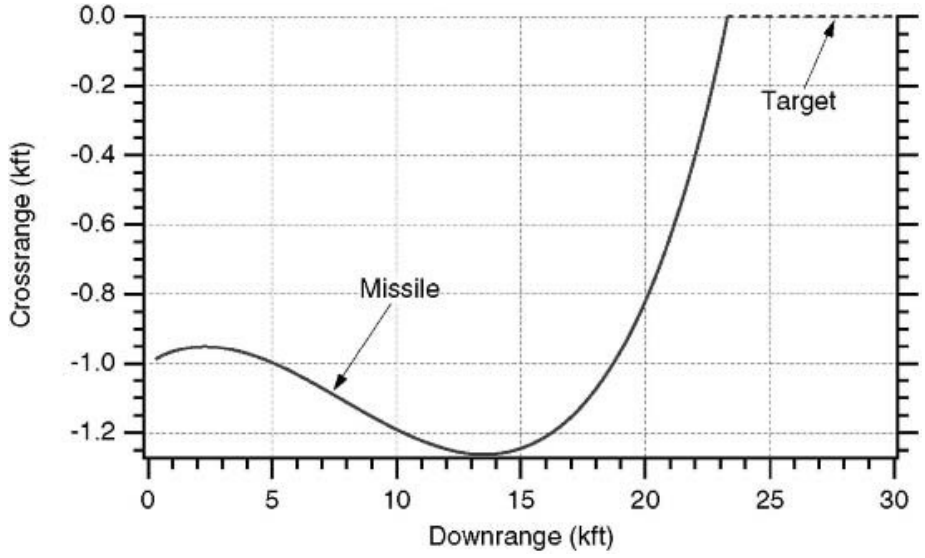

Fig. A.15 Engagement viewed in crossrange-downrange plane.

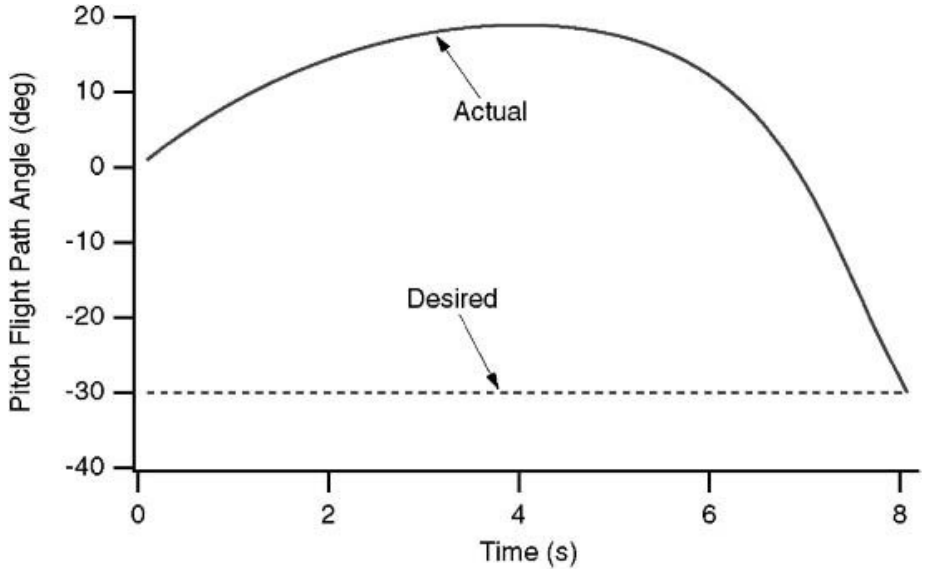

Fig. A.16 Trajectory shaping guidance law enables missile to achieve pitch flight-path-angle objective.

able to find shaping filter equivalents for these maneuvers so that the method of adjoints could be utilized to efficiently evaluate guidance system performance. Another maneuver that is often used in missile guidance system analysis is the Poisson (or random telegraph signal) target maneuver. The Poisson square wave is defined as a maneuver of amplitude $+\beta$ or $-\beta$. The length of time for which the maneuver $n_T(t)$ remains in either position is random [6]. In particular,

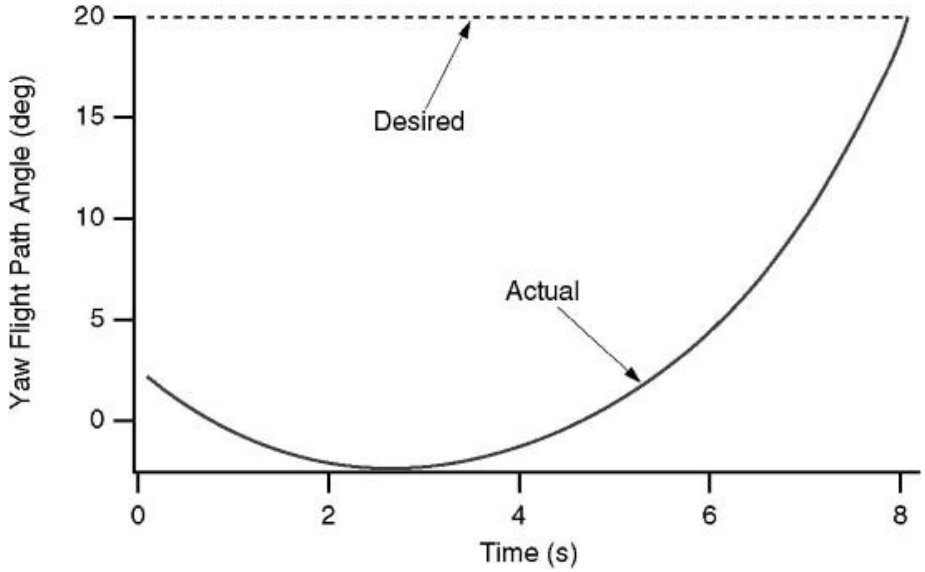

Fig. A.17 Trajectory shaping guidance law enables missile to achieve yaw flight-path-angle objective.

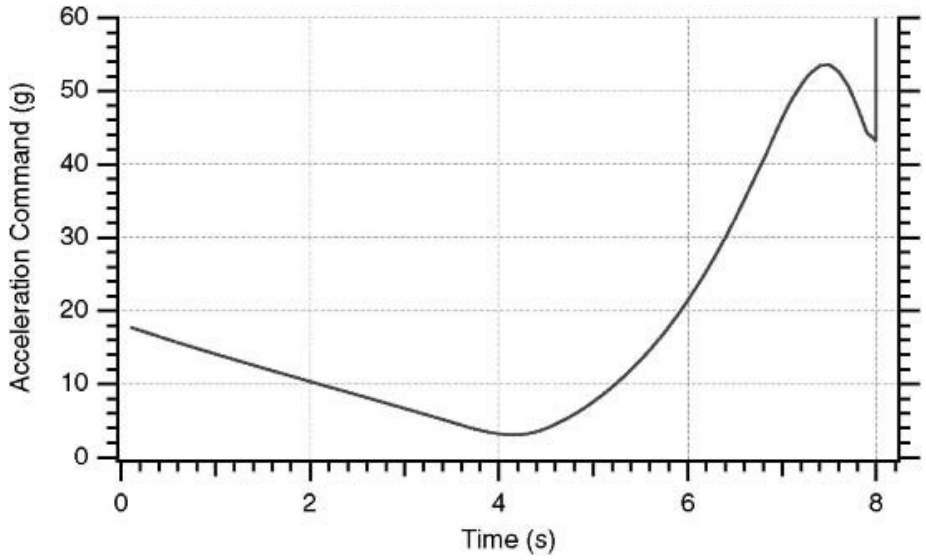

Fig. A.18 Great deal of acceleration is required to make trajectory shaping guidance law work.

the number of times the maneuver changes sign (zero crossings) is given by the Poisson distribution. If *P(k)* represents the probability of k sign changes in t_F seconds, then we can say that

$$P(k) = \frac{(\nu t_F)^k e^{-\nu t_F}}{k!}$$

where k is the number of sign changes and ν is the average number of zero crossings per second. This type of maneuver can be very stressing to a missile guidance system and is similar in shape to the vertical-S maneuver of Chapter 6. A typical realization of a *3-g* Poisson target maneuver ($\upsilon = 0.5\ \mathrm{s}^{-1}$, $\beta = 3\ g$, and $t_F = 20\ \mathrm{s}^{-1}$) is shown in Fig. A.19.

The two steps in simulating a Poisson square wave are in a forward simulation are as follows:

1. Determine the initial sign of the square wave from a Gaussian random number generator with zero mean and unity standard deviation.
2. The length of time between sign changes $\Delta\tau$ is determined by squaring and adding two Gaussian distributions with zero mean and standard deviation given by

$$\sigma = \frac{1}{\sqrt{2\nu}}$$

The second step for making the Poisson target maneuver is displayed in Fig. A.20.

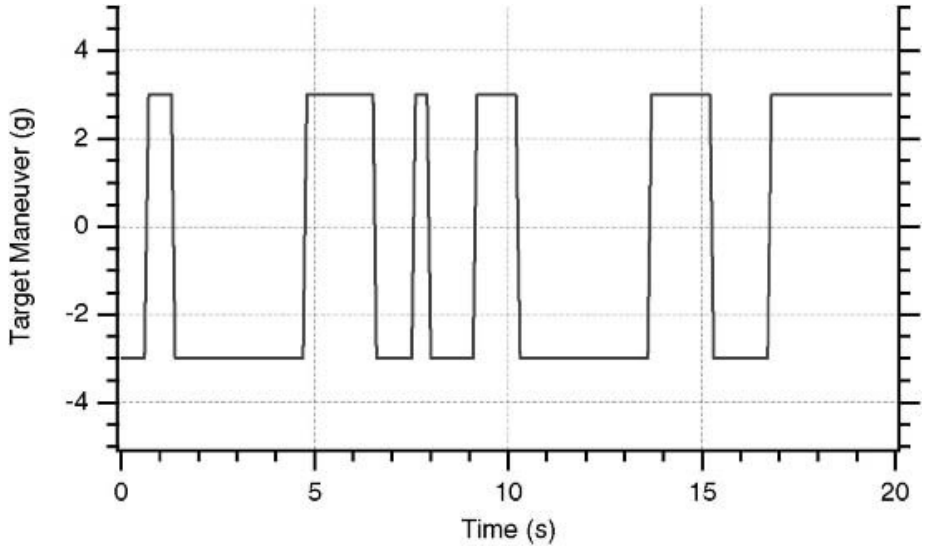

Fig. A.19 Poisson square wave target maneuver.

In Chapter 4, Fig. 4.15 modeled a single-time-constant proportional navigation guidance system whose only error source was a uniformly distributed target maneuver. A Monte Carlo simulation of that guidance system appeared in Listing 4.5. Listing A.7 modifies Listing 4.5 by replacing the uniformly distributed step target maneuver with the Poisson target maneuver. Statements that pertain to the modeling of the Poisson target maneuver have been highlighted in boldface. In addition we can see from Listing A.7 that 1000 runs are made (rather than 50 runs in Listing 4.5) for each flight time and that the flight time

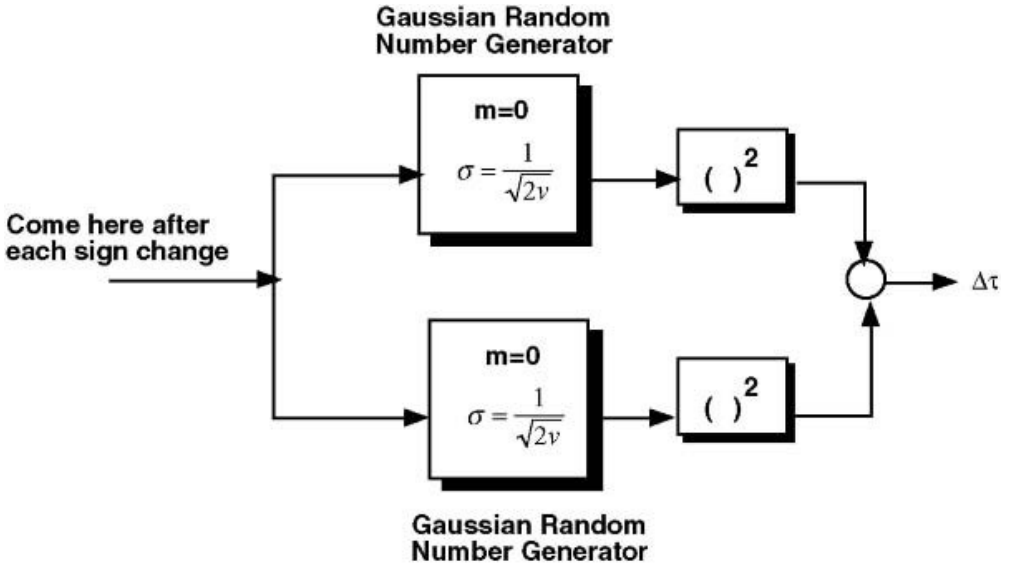

Fig. A.20 Simulating Poisson target maneuver.

is incremented every 0.2 s (rather than every second in Listing 4.5) in order to get better accuracy.

LISTING A.7 MONTE CARLO SIMULATION OF SINGLE-TIME-CONSTANT GUIDANCE SYSTEM DRIVEN BY POISSON TARGET MANEUVER

```
count=0;
VC=4000;
XNT=96.6;
VM=3000;
XNP=3;
TAU=1;
RUN=1000;
BETA=96.6;
XNU=.5;
for TF=.2:.2:10,
        Z1=0;
        for I=1:RUN
                QFIRST=1;
                SIG=1./sqrt(2.*XNU);
                PZ=uniform;
                PZ=PZ-.5;
                if PZ > 0
                        COEF=1;
                else
                        COEF=-1;
                end;
                XNT=COEF*BETA;
                DELT=9999.;
                TNOW=0;
        Y=0;
        YD=0;
        T=0;
        H=.01;
        S=0;
        XNC=0;
        XNL=0;
        while T<=(TF - 1e-5)
        if QFIRST==1
                                XNOISE1=SIG*randn;
                                XNOISE2=SIG*randn;
                                DELT=XNOISE1^2+XNOISE2^2;
                                QFIRST=0;
                                TNOW=T;
                        end;
```

```
                    if T>=(DELT+TNOW)
                      XNT=-XNT;
                      QFIRST=1;
                    end
          YOLD=Y;
          YDOLD=YD;
          XNLOLD=XNL;
          STEP=1;
          FLAG=0;
          while STEP <=1
          if FLAG==1
          Y=Y+H*YD;
          YD=YD+H*YDD;
          XNL=XNL+H*XNLD;
          T=T+H;
          STEP=2;
  end
  TGO=TF-T+.00001;
  RTM=VC*TGO;
  XLAMD=(RTM*YD+Y*VC)/(RTM^2);
  XNC=XNP*VC*XLAMD;
  XNLD=(XNC-XNL)/TAU;
  YDD=XNT-XNL;
  FLAG=1;
 end;
 FLAG=0;
 Y=.5*(YOLD+Y+H*YD);
 YD=.5*(YDOLD+YD+H*YDD);
 XNL=.5*(XNLOLD+XNL+H*XNLD);
 S=S+H;
end;
Z(I)=Y;
Z1=Z(I)+Z1;
XMEAN=Z1/I;
          end;
          SIGMA=0;
          Z1=0;
Z2=0.;
          for I=1:RUN
          Z1=(Z(I)-XMEAN)^2+Z1;
 Z2=Z(I)^2+Z2;
          if I==1
          SIGMA=0;
          RMS=0.;
          else
          SIGMA=sqrt(Z1/(I-1));
```

```
        RMS=sqrt(Z2/(I-1));
        end
        end;
        count=count+1;
        ArrayTF(count)=TF;
        ArrayRMS(count)=RMS;
end;
figure
plot(ArrayTF,ArrayRMS)
title('Shaping filter Monte Carlo results')
xlabel('Time')
ylabel('RMS Miss (ft)')
clc
output=[ArrayTF',ArrayRMS'];
save datfil.txt output -ascii
disp 'simulation finished'
```

The shaping filter equivalent of a Poisson target maneuver can be represented by white noise u_s through a low-pass filter with time constant $1/2\nu$ as shown in Fig. A.21 [6]. In this figure the white-noise input has spectral density Φ_s given by

$$\Phi_s = \frac{\beta^2}{\nu}$$

and the initial condition on the integrator has value β (to ensure that the standard deviation of the shaping filter output is β at all times).

Because the network of Fig. A.21 is driven by both an impulse and white noise, we can take its adjoint. The resultant adjoint block diagram of the Poisson target maneuver appears in Fig. A.22. In this figure we are using the same notation as the adjoint of the single-time-constant homing loop of Fig. 4.17. The outputs

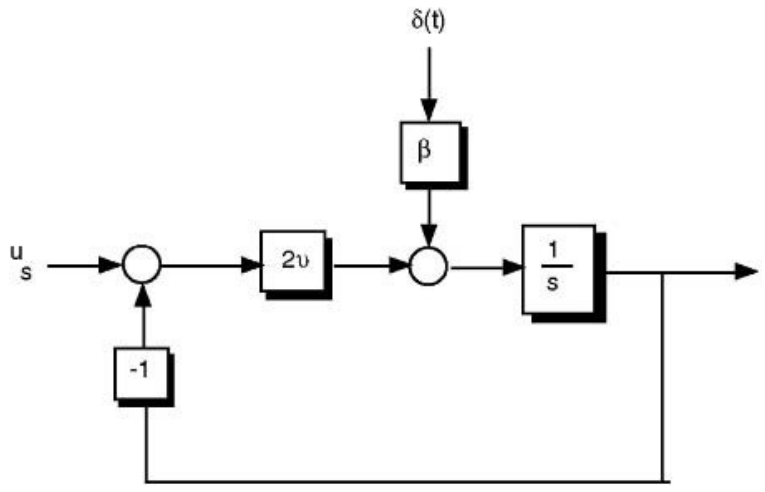

Fig. A.21 Shaping filter equivalent of Poisson target maneuver.

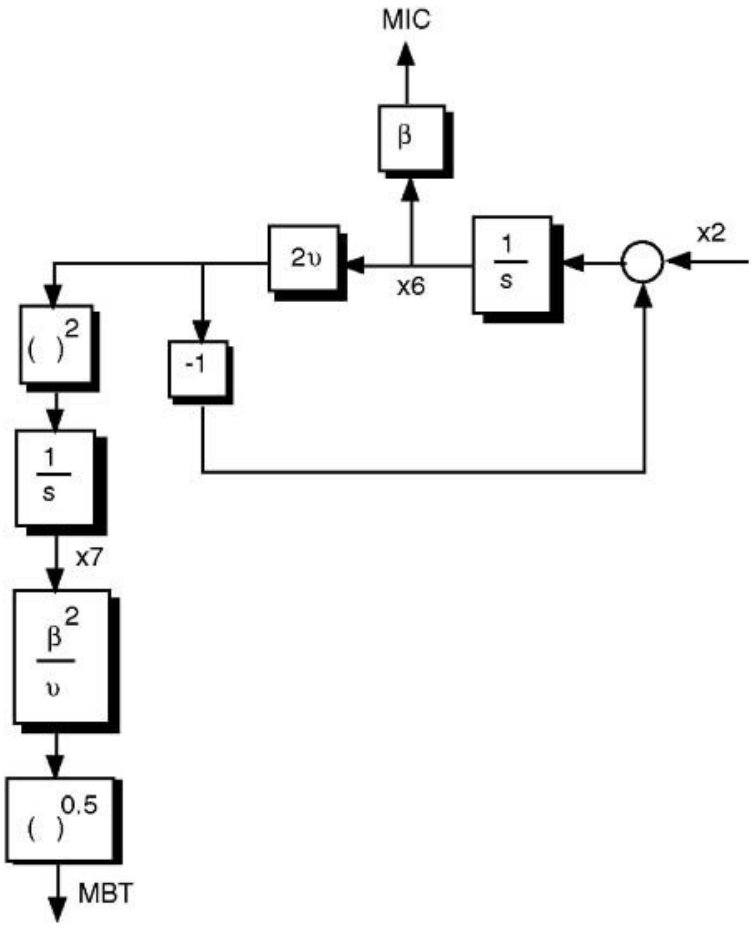

Fig. A.22 Adjoint of Poisson target maneuver.

MBT and MIC of Fig. A.22 represent the adjoint outputs of the miss caused by noise and miss caused by random initial condition on the shaping filter, respectively. Therefore the total rms miss caused by the Poisson target maneuver is given by

$$\text{RMS} = \sqrt{\text{MBT}^2 + \text{MIC}^2}$$

The adjoint of the single-time-constant proportional navigation guidance system was taken, and the resultant adjoint simulation appears in Listing A.8. Only a few modifications to the original adjoint simulation of Listing 4.6 were required, and these statements are highlighted in boldface.

The nominal cases of Listing A.7 (Monte Carlo code) and Listing A.8 (adjoint code) were run, and the rms miss distance vs flight results as a result of the Poisson target maneuver are displayed in Fig. A.23. We can see that the Monte Carlo results, which required 50,000 runs (50 flight times multiplied by 1000 runs per flight time), are identical to the single-run adjoint results.

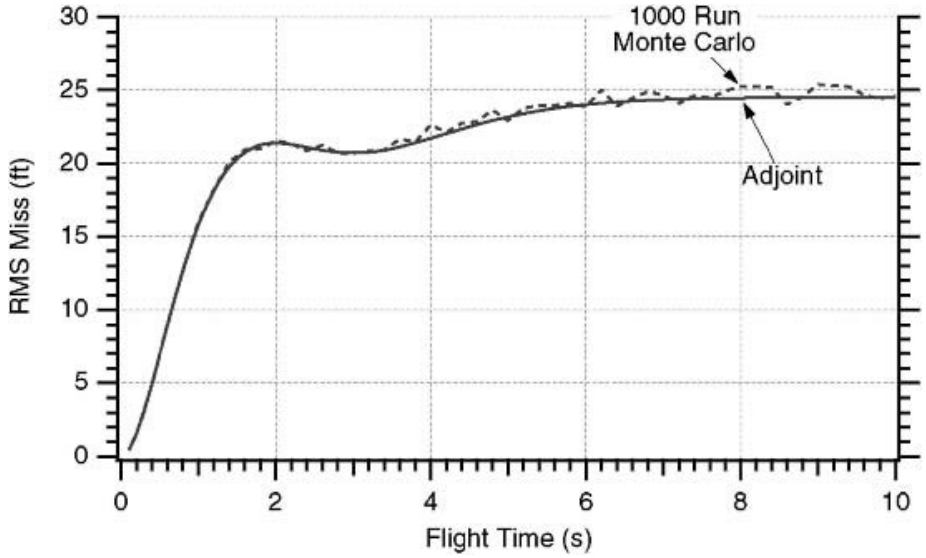

Fig. A.23 Adjoint and Monte Carlo models agree for Poisson target maneuver disturbance.

LISTING A.8 ADJOINT OF SINGLE-TIME-CONSTANT GUIDANCE SYSTEM DRIVEN BY POISSON TARGET MANEUVER

```
count=0;
XNT=96.6;
XNP=3;
TAU=1;
TF=10;
T=0;
S=0;
TP=T+.00001;
X1=0;
X2=0;
X3=1;
X4=0;
X5=0.;
X6=0;
X7=0;
H=.01;
XNU=.5;
BETA=96.6
while TP <= (TF - 1e-5)
        STEP=1;
        FLAG=0;
        S=S+H;
        X1OLD=X1;
        X2OLD=X2;
```

```
X3OLD=X3;
X4OLD=X4;
X5OLD=X5;
X6OLD=X6;
X7OLD=X7;
while STEP <=1
        if FLAG==1
                STEP=2;
                X1=X1+H*X1D;
                X2=X2+H*X2D;
                X3=X3+H*X3D;
                X4=X4+H*X4D;
                X5=X5+H*X5D;
                X6=X6+H*X6D;
                X7=X7+H*X7D;
                TP=TP+H;
        end;
        X1D=X2;
        X2D=X3;
        Y1=(X4-X2)/TAU;
        TGO=TP+.00001;
        X3D=XNP*Y1/TGO;
        X4D=-Y1;
        X5D=X1*X1;
        X6D=X2-2.*XNU*X6;
        X7D=(2.*XNU*X6)^2;
        FLAG=1;
end;
FLAG=0;
X1=(X1OLD+X1)/2+.5*H*X1D;
X2=(X2OLD+X2)/2+.5*H*X2D;
X3=(X3OLD+X3)/2+.5*H*X3D;
X4=(X4OLD+X4)/2+.5*H*X4D;
X5=(X5OLD+X5)/2+.5*H*X5D;
X6=(X6OLD+X6)/2+.5*H*X6D;
X7=(X7OLD+X7)/2+.5*H*X7D;
S=S+H;
if S>=.000999
        S=0.;
                XMBT=BETA*sqrt(X7/XNU);
                XIC=BETA*X6;
                RMS=sqrt(XMBT^2+XIC^2);
        count=count+1;
        ArrayTP(count)=TP;
        ArrayRMS(count)=RMS;
end;
```

```
end
figure
plot(ArrayTP, ArrayRMS),grid
title('Adjoint model using shaping filter approach')
xlabel('Flight Time (S)')
ylabel('RMS Miss (Ft)')
%axis([00,10,00,30])
clc
output=[ArrayTP',ArrayRMS'];
save datfil.txt output /ascii
disp('Simulation Complete')
```

REFERENCES

[1] Nesline, F. W., and Zarchan, P., "A New Look at Classical Versus Modern Homing Guidance," *Journal of Guidance and Control*, Vol. 4, No. 1, 1981, pp. 78–85.

[2] Bucco, D., "Adjoints Revisited: A Software Tool to Facilitate Their Application," *Proceedings of 1997 AIAA Guidance and Control Conference*, Washington, DC.

[3] *Advanced Continuous Simulation Language (ACSL) Reference Manual*, Mitchell and Gauthier Associates (MGA), Inc., Concord, MA, 1991, pp. A62–A69.

[4] Regan, F. J., and Anandakrishnan, J. M., *Dynamics of Atmospheric Re-Entry*, AIAA, Washington, DC, 1993, pp. 154–156.

[5] Ohlmeyer, E. J., and Phillips, C. A., "Generalized Vector Explicit Guidance," *Journal of Guidance, Control, and Dynamics*, Vol. 29, No. 2, 2006, pp. 261–268.

[6] Schwartz, M., *Information Transmission, Modulation, and Noise*, McGraw–Hill, New York, 1959, pp. 441–448.