APPENDIX B

SOFTWARE

This appendix contains the Fortran code that is required to use the aircraft models given in the text and is not otherwise readily available. For the steady-state trim algorithm (Section B.1) we give the basic trimmer subroutine, part of the constraint subroutine, a cost function, and the Simplex minimization algorithm. The user must write a driver program and add additional flight-path constraints, as required. In Section B.2 a subroutine for numerical linearization is given, and the user need only add a driver program. Software for time-history simulation and control systems design is readily available from other sources and so, in the rest of this appendix, we have given only the Runge-Kutta algorithm that was used for most of the examples.

Appendix A, Appendix B, and the programs used in this book can be obtained on a floppy disc, at a nominal cost, from Dr. B. L. Stevens, 1051 Park Manor Terr., Marietta, GA 30064.

B.1 AIRCRAFT STEADY-STATE TRIM CODE

The subroutine "TRIMMER" (below) sets up a function minimization algorithm to determine a steady-state trim condition for either a 6-DoF or 3-DoF (longitudinal-only) aircraft model. The subroutine arguments are the number of degrees of freedom (NV) and the "COST" function (which must be declared "EXTERNAL" in the main program). Labeled COMMON storage is used to pass the state and control vectors to and from the main program and the cost function (and, in the case of the control vector, the aircraft model also).

The main program must initialize the state vector according to the trim condition required, and the control vector can simply be set to zero initially. It must also set the turn rate, roll rate, or pull up rate; set flags for coordinated turns or stability-axis roll;

Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems, Third Edition.
Brian L. Stevens, Frank L. Lewis, Eric N. Johnson.

and pass these through a common block ("CONSTRNT") to the constraint routine. A Simplex routine (given below) is used for function minimization, and it returns the coordinates of the cost function minimum in the Simplex vector S. The cost function is then called once more to set the state and control vectors to their final values, and these values are passed through COMMON to the main program to be placed in a data file. Subroutine "SMPLX" can easily be replaced by "ZXMWD" from the IMSL library, or "AMOEBA" from "Numerical Recipes" if desired. The author is indebted to Dr. P. Vesty for this Simplex routine.

```
SUBROUTINE TRIMMER (NV, COST)
PARAMETER (NN=20, MM=10)
EXTERNAL COST
CHARACTER*1 ANS
DIMENSION S(6), DS(6)
COMMON/ STATE/ X(NN)
COMMON/ CONTROLS/ U(MM)
COMMON/ OUTPUT/ AN,AY,AX,QBAR,AMACH ! common to aircraft
DATA RTOD /57.29577951/
S(1)= U(1)
S(2)= U(2)
S(3)= X(2)
IF(NV .LE. 3) GO TO 10
S(4)= U(3)
S(5)= U(4)
S(6)= X(3)
10     DS(1) = 0.2
       DS(2)= 1.0
       DS(3)= 0.02
IF(NV .LE. 3) GO TO 20
       DS(4)= 1.0
       DS(5)= 1.0
       DS(6)= 0.02
20     NC= 1000
WRITE(*,'(1X,A,$)')'Reqd. # of trim iterations (def. = 1000) : '
READ(*,*,ERR=20) NC
SIGMA = -1.0
CALL SMPLX(COST,NV,S,DS,SIGMA,NC,F0,FFIN)
FFIN = COST(S)
IF (NV .GT. 3) THEN
WRITE(*,'(/11X,A)')'Throttle     Elevator,     Ailerons,     Rud-
der'
WRITE(*,'(9X,4(1PE10.2,3X),/)') U(1), U(2), U(3), U(4)
WRITE(*,99)'Angle of attack',RTOD*X(2),'Sideslip angle',RTOD*X(3)
WRITE(*,99) 'Pitch angle', RTOD*X(5), 'Bank angle', RTOD*X(4)
WRITE(*,99) 'Normal acceleration', AN, 'Lateral acceln', AY
WRITE(*,99) 'Dynamic pressure', QBAR, 'Mach number', AMACH
ELSE
WRITE(*,'(/1X,A)')' Throttle     Elevator     Alpha  Pitch'
WRITE(*,'(1X,4(1PE10.2,3X))')U(1),U(2),X(2)*RTOD,X(3)*RTOD
WRITE(*,'(/1X,A)')'Normal acceleration Dynamic Pressure Mach '
WRITE(*,'(5X,3(1PE10.2,7X))') AN,QBAR,AMACH
END IF
```

```
WRITE(*,99)'Initial cost function ',F0,'Final cost function',FFIN
99     FORMAT(2(1X,A22,1PE10.2))
40     WRITE(*,'(/1X,A,$)') 'More Iterations ? (def= Y) : '
READ(*,'(A)',ERR= 40) ANS
IF (ANS .EQ. 'Y'.OR. ANS .EQ. 'y'.OR. ANS .EQ. '/') GO TO 10
IF (ANS .EQ. 'N'.OR. ANS .EQ. 'n') RETURN
GO TO 40
END
FUNCTION CLF16 (S)   ! F16 cost function (see text)
PARAMETER (NN=20)
REAL S(*), XD(NN)
COMMON/STATE/X(NN) ! common to main program
COMMON/CONTROLS/THTL,EL,AIL,RDR  ! to aircraft
THTL = S(1)
EL = S(2)
X(2)= S(3)
AIL = S(4)
RDR = S(5)
X(3) = S(6)
X(13)= TGEAR (THTL)
CALL CONSTR (X)
CALL   F (TIME,X,XD)
CLF16 = XD(1)**2 + 100.0*( XD(2)**2 + XD(3)**2 )
&      + 10.0*( XD(7)**2 + XD(8)**2 + XD(9)**2 )
RETURN
END
SUBROUTINE CONSTR (X) ! used by COST, to apply constraints
DIMENSION X(*)
LOGICAL COORD, STAB
COMMON/CNSTRNT/RADGAM,SINGAM,RR,PR,TR,PHI,CPHI,SPHI,COORD,STAB
C common to main program.
CALPH = COS(X(2))
SALPH = SIN(X(2))
CBETA = COS(X(3))
SBETA = SIN(X(3))
IF (COORD) THEN
! coordinated turn logic here
ELSE IF (TR .NE. 0.0) THEN
! skidding turn logic here
ELSE   ! non-turning flight
X(4)= PHI
D = X(2)
IF(PHI .NE. 0.0) D = -X(2) ! inverted
IF( SINGAM .NE. 0.0 ) THEN ! climbing
SGOCB = SINGAM / CBETA
X(5)= D + ATAN( SGOCB/SQRT(1.0-SGOCB*SGOCB)) ! roc constraint
ELSE
X(5) = D     ! level
END IF
X(7)= RR
X(8)= PR
IF (STAB) THEN     ! stab.-axis roll
X(9)= RR*SALPH/CALPH
```

```
ELSE
X(9) = 0.0   ! body-axis roll
END IF
END IF
RETURN
END
SUBROUTINE SMPLX(FX,N,X,DX,SD,M,Y0,YL)
C This simplex algorithm minimizes FX(X), where X is (Nx1).
C DX contains the initial perturbations in X. SD should be set
   according
C to the tolerance required; when SD<0 the algorithm calls FX M
   times
REAL X(*), DX(*)
DIMENSION XX(32), XC(32), Y(33), V(32,32)
C
NV=N+1
DO 2 I=1,N
DO 1 J=1,NV
1     V(I,J)=X(I)
2     V(I,I+1)=X(I)+DX(I)
Y0=FX(X)
Y(1)=Y0
DO 3 J=2,NV
3     Y(J)=FX(V(1,J))
K=NV
4     YH=Y(1)
YL=Y(1)
NH=1
NL=1
DO 5 J=2,NV
IF(Y(J).GT.YH) THEN
YH=Y(J)
NH=J
ELSEIF(Y(J).LT.YL) THEN
YL=Y(J)
NL=J
ENDIF
5     CONTINUE
YB=Y(1)
DO 6 J=2,NV
6     YB=YB+Y(J)
YB=YB/NV
D=0.0
DO 7 J=1,NV
7     D=D+(Y(J)-YB)**2
SDA=SQRT(D/NV)
IF((K.GE.M).OR.(SDA.LE.SD)) THEN
SD=SDA
M=K
YL=Y(NL)
DO 8 I=1,N
8      X(I)=V(I,NL)
```

```
RETURN END IF
DO 10 I=1,N XC(I)=0.0
DO 9 J=1,NV
9      IF(J.NE.NH) XC(I)=XC(I)+V(I,J)
10     XC(I)=XC(I) /N
DO 11 I=1,N
11     X(I)=XC(I)+XC(I)-V(I,NH)
K=K+1
YR=FX(X)
IF(YR.LT.YL) THEN
DO 12 I=1,N
12     XX(I)=X(I)+X(I)-XC(I)
K=K+1
YE=FX(XX)
IF(YE.LT.YR) THEN
Y(NH)=YE
DO 13 I=1,N
13           V(I,NH)=XX(I)
ELSE
Y(NH)=YR
DO 14 I=1,N
14     V(I,NH)=X(I)
END IF
GOTO 4
ENDIF
Y2=Y(NL)
DO 15 J=1,NV
15     IF((J.NE.NL).AND.(J.NE.NH).AND.(Y(J).GT.Y2)) Y2=Y(J)
IF(YR.LT.YH) THEN
Y(NH)=YR
DO 16 I=1,N
16     V(I,NH)=X(I)
IF(YR.LT.Y2) GO TO 4
ENDIF
DO 17 I=1,N
17 XX(I)=0.5*(V(I,NH)+XC(I))
K=K+1
YC=FX(XX)
IF(YC.LT.YH) THEN
Y(NH)=YC
DO 18 I=1,N
18     V(I,NH)=XX(I)
ELSE
DO 20 J=1,NV
DO 19 I=1,N
19     V(I,J)=0.5*(V(I,J)+V(I,NL))
20     IF(J.NE.NL) Y(J)=FX(V(1,J))
K=K+N
ENDIF
GO TO 4
END
```

B.2 NUMERICAL LINEARIZATION SUBROUTINE

Subroutine JACOB will calculate Jacobian matrices for the set of nonlinear state equations contained in the subroutine F (specified as an argument of JACOB). Subroutine F(TIME, X, XD) should contain "CONTROLS" and "OUTPUT" common blocks as used in the text. The argument FN is a double-precision function used to determine an approximation to each partial derivative that is required.

To calculate the *A*, *B*, *C*, *D* matrices the main program should be designed to call JACOB four times, with FN replaced in turn by each of the partial derivative functions FDX, FDU, YDX, and YDU (given below). The partial derivative functions must be declared "EXTERNAL" in the main program. The vectors X and XD are, respectively, the state vector and its derivative. The vector V must contain the equilibrium condition and should be replaced by X or U, respectively, depending on whether the partial derivatives with respect to X or U are being calculated. The array IO is used to specify the set of integers corresponding to the rows of the Jacobian matrix, and JO is used to specify the set corresponding to the columns. NR and NC are, respectively, the number of rows and the number of columns in the Jacobian matrix and the linear array ABC contains the columns of the Jacobian matrix, stacked one after the other.

The linearization algorithm chooses smaller and smaller perturbations in the independent variable and compares three successive approximations to the particular partial derivative. If these approximations agree within a certain tolerance, then the size of the perturbation is reduced to determine if an even smaller tolerance can be satisfied. The algorithm terminates successfully when a tolerance TOLMIN is reached or if a tolerance of at least OKTOL can be achieved. If the algorithm does not terminate successfully, then the successive approximations are displayed and the user is asked to decide on the value of the partial derivative.

```
SUBROUTINE JACOB (FN,F,X,XD,V,IO,JO,ABC,NR,NC)
DIMENSION X(*),XD(*),V(*),IO(*),JO(*),ABC(*)
EXTERNAL FN,F
LOGICAL FLAG, DIAGS
CHARACTER*1 ANS
REAL*8 FN,TDV
DATA DEL,DMIN,TOLMIN,OKTOL /.01, .5, 3.3E-5, 8.1E-4/
C
DIAGS= .TRUE.
PRINT '(1X,A,$)', 'DIAGNOSTICS ? (Y/N, "/"=N) '
READ(*,'(A)') ANS
IF (ANS .EQ.'/'.0R. ANS .EQ. 'N' .0R. ANS .EQ. 'n')DIAGS=.FALSE.
IJ= 1
DO 40 J=1,NC
DV= AMAX1( ABS( DEL*V(JO(J)) ), DMIN )
DO 40 I=1,NR
FLAG= .FALSE.
1     TOL= 0.1
OLTOL= TOL
TDV= DBLE( DV )
A2= 0.0
A1= 0.0
```

```
A0= 0.0
B1= 0.0
B0= 0.0
D1= 0.0
D0= 0.0
IF (DIAGS .OR. FLAG) WRITE(*,'(/1X,A8,I2,A1,I2,11X,A12,8X,A5)')
& 'Element ',I,',',J, 'perturbation','slope'
DO 20 K= 1,18 ! iterations on TDV
A2= A1
A1= A0
B1= B0
D1= D0
A0= FN(F,XD,X,IO(I),JO(J),TDV)
Bφ= AMIN1( ABS(A0), ABS(A1) )
D0= ABS ( A0 - A1 )
IF (DIAGS .OR. FLAG) WRITE(*,'(20X,1P2E17.6)') TDV,A0
IF(K .LE. 2) GO TO 20
IF (A0 .EQ. A1 .AND. A1 .EQ. A2) THEN
ANS2= A1
GO TO 30
END IF
IF (A0 .EQ. 0.0) GO TO 25
10     IF( D0 .LE. TOL*B0 .AND. D1 .LE. TOL*B1) THEN
ANS2= A1
OLTOL= TOL
IF(DIAGS .0R. FLAG) WRITE(*,'(1X,A9,F8.7)') 'MET TOL= ',TOL
IF (TOL .LE. TOLMIN) THEN
GO TO 30
ELSE
TOL= 0.2*TOL
GO TO 10
END IF
END IF
20    TDV= 0.6D0*TDV
25    IF (OLTOL .LE. OKTOL) THEN
GO TO 30
ELSE IF (.NOT. FLAG) THEN
WRITE(*,'(/1X,A)')'NO CONVERGENCE *****'
FLAG= .TRUE.
GO TO 1 ELSE
21    WRITE(*,'(1X,A,$)') 'Enter estimate : '
READ(*,*,ERR=21) ANS2
FLAG= .FALSE.
GO TO 30
END IF
30     ABC(IJ)= ANS2
IF (DIAGS) THEN
PRINT '(27X,A5,1PE13.6)','Ans= ',ANS2
PAUSE 'Press "enter"'
END IF
40     IJ= IJ+1
RETURN END
DOUBLE PRECISION FUNCTION FDX(F,XD,X,I,J,DDX)
REAL*4 XD(*), X(*)
```

```
DOUBLE PRECISION T, DDX, XD1, XD2
EXTERNAL F
TIME= 0.0
T      = DBLE( X(J) )
X(J)= SNGL( T - DDX )
CALL F(TIME,X,XD)
XD1 = DBLE( XD(I) )
X(J)= SNGL( T + DDX )
CALL F(TIME,X,XD)
XD2 = DBLE( XD(I) )
FDX = (XD2-XD1)/(DDX+DDX)
X(J)= SNGL( T )
RETURN
END
DOUBLE PRECISION FUNCTION FDU(F,XD,X,I,J,DDU)
PARAMETER (NIN=10)
REAL*4 XD(*), X(*)
COMMON/CONTROLS/U(NIN)
DOUBLE PRECISION T, DDU, XD1, XD2
EXTERNAL F
TIME= 0.0
T     = DBLE( U(J) )
U(J)= SNGL( T - DDU )
CALL F(TIME,X,XD)
XD1 = DBLE( XD(I) )
U(J)= SNGL( T + DDU )
CALL F(TIME,X,XD)
XD2 = DBLE( XD(I) )
FDU = (XD2-XD1)/(DDU+DDU)
U(J)= SNGL( T )
RETURN
END
DOUBLE PRECISION FUNCTION YDX(F,XD,X,I,J,DDX)
PARAMETER (NOP=20)
REAL*4 XD(*), X(*)
COMMON/OUTPUT/Y/(NOP)
DOUBLE PRECISION T, DDX, YD1, YD2
EXTERNAL F
TIME= 0.0
T      = DBLE( X(J) )
X(J)= SNGL( T - DDX )
CALL F(TIME,X,XD)
YD1 = DBLE( Y(I) )
X(J)= SNGL( T + DDX )
CALL F(TIME,X,XD)
YD2 = DBLE( Y(I) )
YDX = (YD2-YD1)/(DDX+DDX)
X(J)= SNGL(T)
RETURN
END
DOUBLE PRECISION FUNCTION YDU(F,XD,X,I,J,DDU)
PARAMETER (NIN=10, NOP=20)
```

```
REAL*4 XD(*), X(*)
COMMON/CONTROLS/U(NIN)
COMMON/OUTPUT/Y(NOP)
DOUBLE PRECISION T, DDU, YD1, YD2
EXTERNAL F
TIME= 0.0
T      = DBLE( U(J) )
U(J)= SNGL( T - DDU )
CALL F(TIME,X,XD)
YD1 = DBLE( Y(I) )
U(J)= SNGL( T + DDU )
CALL F(TIME,X,XD)
YD2 = DBLE( Y(I) )
YDU = (YD2-YD1)/(DDU+DDU)
U(J)= SNGL(T)
RETURN
END
```

B.3 RUNGE-KUTTA INTEGRATION

This subroutine implements "Runge's fourth-order rule" as described in Chapter 3. Its arguments are the subroutine F containing the nonlinear state equations, the current time TT, the integration time step DT, the state and state derivative vectors XX and XD, and the number of state variables NX. Subroutine F should be declared EXTERNAL in the main program unit.

```
SUBROUTINE RK4(F,TT,DT,XX,XD,NX)
PARAMETER (NN=30) ! same as main prog.
REAL*4 XX(*),XD(*),X(NN),XA(NN)
CALL F(TT,XX,XD)
DO 1 M=1,NX
XA (M)=XD(M)*DT
1       X(M)=XX(M)+0.5*XA(M)
T=TT+0.5*DT
CALL F(T,X,XD)
DO 2 M=1,NX
Q=XD(M)*DT
X(M)=XX(M)+0.5*Q
2      XA(M)=XA(M)+Q+Q
CALL F(T,X,XD)
DO 3 M=1,NX
Q=XD(M)*DT
X(M)=XX(M)+Q
3      XA(M)=XA(M)+Q+Q
TT=TT+DT
CALL F(TT,X,XD)
DO 4 M=1,NX
4      XX(M)=XX(M)+(XA(M)+XD(M)*DT)/6.0
RETURN
END
```

B.4 OUTPUT FEEDBACK DESIGN

Output feedback design is not an easy problem. Finding the optimal output feedback gains to minimize a quadratic performance index (PI),

$$J = \frac{1}{2}\int_0^\infty (x^{\mathrm{T}}Qx + u^{\mathrm{T}}Ru)\, dt, \tag{B.7.1}$$

involves solving coupled nonlinear matrix design equations of the form (Chapter 5)

$$0 = \frac{\partial H}{\partial S} = A_c^{\mathrm{T}}P + PA_c + C^{\mathrm{T}}K^{\mathrm{T}}RKC + Q \tag{B.7.2}$$

$$0 = \frac{\partial H}{\partial P} = A_c S + SA_c^{\mathrm{T}} + X \tag{B.7.3}$$

$$0 = \frac{1}{2}\frac{\partial H}{\partial K} = RKCSC^{\mathrm{T}} - B^{\mathrm{T}}PSC^{\mathrm{T}}, \tag{B.7.4}$$

where

$$A_c = A - BKC, \qquad X = x(0)x^{\mathrm{T}}(0)$$

In the design of tracking systems, the equations are even worse.

We have used two general approaches to solving such equation sets. In the first, the PI J is computed based on (B.7.2) using

$$J = \frac{1}{2}\mathrm{tr}(PX) \tag{B.7.5}$$

The Simplex routine in Section B.1 was used to minimize J. In the second approach, a gradient algorithm (e.g., Davidon-Fletcher-Powell) was used.* There, the gradient $\partial J/\partial K$ is computed using all three design equations (B.7.2) to (B.7.4).

*Press, W. H., B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes,* New York: Cambridge, 1986.