I

MATLAB/SIMULINK Programs for Flight/Ground Manoeuvres and Gust/Turbulence Encounters

In this appendix, some sample programs for calculation of equilibrium and dynamic manoeuvres, gusts and turbulence, and taxiing and landing are presented.

I.1 RIGID AIRCRAFT DATA

Before proceeding with calculation of results from different manoeuvres and gust/turbulence encounters, the basic aircraft data need to be entered for the rigid or flexible aircraft as appropriate. To save space, the aircraft data will simply be specified for a sample case; clearly, in a normal programme, the reader may choose to enter some parameters via a dialogue and to introduce loops to cover multiple cases (e.g. of air speed). Note that units used throughout for defining data are in N, kg, m, s and rad. Wherever possible, the symbols used in the code below *Pgm_I1_Rigid_Aircraft_Data* aim to match those used in the main text, with the use of the underscore symbol _ indicating a subscript to follow; therefore, for example, m_W in MATLAB is equivalent to m_W in the text.

```
% Data for Symmetric Aircraft
close all;  clear all

% Mass and Dimensions

m = 10000; W = m * 9.81; m_F = 0.15 * m; m_W = 0.3 * m; m_C = 0.4 *m;
m_T = 0.15 * m; S_W = 30.0; S_T = 7.5; s = 7.5; c = 2.0; s_tp =
3.0; c_tp = 1.25; l_W = 0.3*c; l_T = 3.5 * c; l_A = 0.125 * c; l_E =
0.125 * c; l_WM = l_W - l_A - l_E; l_M = 0.375 * c; l_F = (m_T *
l_T- m_W * l_WM) / m_F; l_WT = l_W + l_T; l_N = l_F; l_M = 0.375 * c;
mu = m_W / 2 / s;

% Moments of Inertia

I_y_fuse = m_F * l_F^2 + m_T * l_T^2; I_y_W = m_W * (c / 3)^2; I_y =
I_y_fuse + I_y_W + m_W * l_WM^2; l_y_W = sqrt(I_y_W / m_W); l_y =
sqrt(I_y / m);
```

Introduction to Aircraft Aeroelasticity and Loads J. R. Wright and J. E. Cooper

```
% Landing gear
C_N = 3200; C_M = 19200; K_N = 80000; K_M = 240000; l_B = l_N + l_M;

% Basic aerodynamics
a_W = 4.5; a_T = 3.2; a_E = 1.5; alpha_0 = - 0.03; C_M0 = - 0.03;
C_D = 0.1; k_epsilon = 0.35;
```

I.2 FLEXIBLE AIRCRAFT DATA

When considering the flexible aircraft, the choice in Appendix C was between three mode types, namely fuselage bending, wing bending or wing twist dominant. This part of the data entry must specify the relevant modal information – mode parameters, mode shape, modal mass, natural frequency, modal stiffness and J integrals. The example shown *Pgm_I2_Flexible_Aircraft_Data* is that for the fuselage bending mode.

```
% Additional Data for Flexible Aircraft (if required) - Example
  of Fuselage Bending dominant
kappa_e0 = 1; gamma_e0 = 0; A = 0; B = 0;

% Normalisation to 1 at wing tip trailing edge
kappa_tip = kappa_e0*(1 + A) + gamma_e0*(1 + B)*(c - c/4 - l_A);
kappa_e0 = kappa_e0/kappa_tip;

% Solution of equations to yield dominant Fuselage Bending mode shape
X = [m_F m_T; -m_F*l_F m_T*l_T]; Y = [- (m_W + m_C)*kappa_e0;
m_W*l_WM*kappa_e0]; Z = X\Y; kappa_eC = 1; kappa_eF = Z(1);
kappa_eT = Z(2); gamma_eT = 2*(kappa_eT - kappa_eC) / l_T - gamma_e0;

% Modal Mass, Natural Frequency and Modal Stiffness for
  Fuselage Bending dominant
m_e = m_F*kappa_eF^2 + m_W*kappa_e0^2 + m_C*kappa_eC^2 + m_T*
kappa_eT^2; f_e = 4.0;  omega_e = 2*pi*f_e;  k_e = omega_e^2*m_e;

% `J' Integrals for aerodynamic derivatives
J1 = gamma_e0*(1 + B / 2);
J2 = kappa_e0*(1 + A / 3) - l_A*gamma_e0*(1 + B / 2);
J3 = gamma_e0*kappa_e0*(1 + A / 3 + B / 2 + A*B / 4) - l_A*
gamma_e0^2*(1 + B + B^2 / 3);
```

This code will need revising for other mode types, using the information given in Appendix C.

I.3 FLIGHT CASE DATA

In this section, the data relevant to the aircraft flight case are generated *Pgm_I3_Flight_Case_Data*; once again, a dialogue could be used and loops introduced.

```
% Data for Flight Case-specify EAS (and relative density
  if not at sea level)
V0 = 150;                    % EAS
```

```
rho0 = 1.225;                % Sea level density
root_sigma  = 0.8;           % Relative density at altitude
V = V0 / root_sigma;         % Convert to TAS for gusts / ground
                               manoeuvres
```

I.4 AERODYNAMIC DERIVATIVE CALCULATION

The way that manoeuvres and gusts/turbulence have been treated in this book is via the use of rigid and elastic aircraft aerodynamic derivatives, either for inertial or body fixed axes systems; the results seen in Appendix B show that most of the derivatives are the same whereas some are different for each of the two cases. In this section, code is included for calculating the longitudinal derivatives in inertial axes, first for the rigid and then the flexible aircraft, including gust derivatives. The derivatives relative to body fixed axes (in this case wind axes, as used in Chapters 14 and 15) are also included below *Pgm_I4_Aero_Derivatives* where they differ to the inertial axes expressions.

```
% Derivatives evaluated using Equivalent Air Speed and sea level
  air density

% Aerodynamic Derivatives (inertial axes)- Heave DoF
Z_0 = -0.5*rho0*V0^2*[- S_W*a_W + S_T*a_T*k_epsilon]*alpha_0;
Z_alpha = -0.5*rho0*V0^2*[S_W*a_W + S_T*a_T*(1 - k_epsilon)];
Z_q = -0.5*rho0*V0*S_T*a_T*l_T;
Z_eta = -0.5*rho0*V0^2*S_T*a_E;
Z_zdot = -0.5*rho0*V0*(S_W*a_W + S_T*a_T*(1 - k_epsilon));
Z_gW = -0.5*rho0*V0*S_W*a_W;
Z_gT = -0.5*rho0*V0*S_T*a_T*(1 - k_epsilon);

% Aerodynamic Derivatives (inertial axes)- Pitch DoF
M_0W = 0.5*rho0*V0^2*S_W*c*C_M0 -0.5*rho0*V0^2*S_W*a_W*l_W*alpha_0;
M_0T = -0.5*rho0*V0^2*S_T*a_T*k_epsilon*l_T*alpha_0;
M_0 = M_0W + M_0T;
M_alpha = 0.5*rho0*V0^2*[S_W*a_W*l_W - S_T*a_T*(1 - k_epsilon)*l_T];
M_q = -0.5*rho0*V0*S_T*a_T*l_T^2;
M_eta = -0.5*rho0*V0^2*S_T*a_E*l_T;
M_zdot = 0.5*rho0*V0*(S_W*a_W*l_W - S_T*a_T*l_T*(1 - k_epsilon));

M_gW = 0.5*rho0*V0*S_W*a_W*l_W;
M_gT = -0.5*rho0*V0*S_T*a_T*l_T*(1 - k_epsilon);

% Additional aerodynamic derivatives for wind axes- Rigid DoF
(if required)
Z_w = -0.5*rho0*V0*(S_W*a_W + S_T*a_T*(1 - k_epsilon)+ S_W*C_D);
M_w = 0.5*rho0*V0*(S_W*a_W*l_W - S_T*a_T*l_T*(1 - k_epsilon));

% Aerodynamic Derivatives-Elastic DoF (if required)
Z_e = 0.5*rho0*V0^2*(-S_W*a_W*J1 - S_T*a_T*gamma_eT);
Z_edot = -0.5*rho0*V0*S_T*a_T*kappa_eT;
M_e = 0.5*rho0*V0^2*(S_W*a_W*l_W*J1 - S_T*a_T*l_T*gamma_eT);
M_edot = -0.5*rho0*V0*S_T*a_T*l_T*kappa_eT;
```

```
Q_0 = 0.5*rho0*V0^2*(S_W*a_W*J2 - S_T*a_T*k_epsilon*kappa_eT)
*alpha_0;
Q_alpha = 0.5*rho0*V0^2*(-S_W*a_W*J2 - S_T*a_T*(1 - k_epsilon)
*kappa_eT);
Q_q = -0.5*rho0*V0*S_T*a_T*l_T*kappa_eT;
Q_eta = -0.5*rho0*V0^2*S_T*a_E*kappa_eT;
Q_e = 0.5*rho0*V0^2*(-S_W*a_W*J3 - S_T*a_T*gamma_eT*kappa_eT);
Q_zdot = 0.5*rho0*V0*(-S_W*a_W*J2 - S_T*a_T*(1 - k_epsilon)
*kappa_eT);
Q_edot = -0.5*rho0*V0*S_T*a_T*kappa_eT^2;
Q_gW = -0.5*rho0*V0*S_W*a_W*J2;
Q_gT = -0.5*rho0*V0*S_T*a_T*kappa_eT;

% Additional aerodynamic derivative for wind axes-Elastic DoF
Q_w = 0.5*rho0*V0*(- S_W*a_W*J2 - S_T*a_T*(1 - k_epsilon)
*kappa_eT);
```

I.5 EQUILIBRIUM MANOEUVRES

In this section, related to Chapter 13, the equations are set up from the aerodynamic derivatives and the solution carried out for the equilibrium manoeuvre of both the rigid and elastic aircraft *Pgm_I5_Equilibrium_Manoeuvres*. In this and later cases, the sections of MATLAB code for aircraft data, flexible aircraft data (if required), flight case and aerodynamic derivatives must be run first. Both the rigid and elastic cases may be run together. Angles are output in degrees.

```
% Load factor and steady pitch rate for equilibrium manoeuvre
  n = 1.0; q_pr = 0;

% Trim response of a rigid aircraft-requires aircraft data,
  flight case and derivative codes
% Setting-up and Solving Equations of Motion for the Rigid Aircraft

ARigid = -[Z_eta Z_alpha; M_eta  M_alpha];
CRigid = [1 ; 0]; DRigid = [Z_q; M_q]; ERigid = [Z_0; M_0];
BRigid = inv(ARigid)*(CRigid*(n*W) + DRigid*q_pr + ERigid);
Bdeg = BRigid*180 / pi;
Trim_Elevator_Rigid = Bdeg(1)
Trim_Incidence_Rigid = Bdeg(2)

% Trim response of an elastic aircraft-requires aircraft
  data, flight case, flexible mode and derivative codes

% Setting-up and Solving Equations of Motion for the Elastic
  Aircraft

AElastic = - [Z_eta  Z_alpha  Z_e;  M_eta  M_alpha  M_e;  Q_eta
Q_alpha  Q_e - k_e];
CElastic = [1 ; 0 ; 0]; DElastic = [Z_q ; M_q ; Q_q];
EElastic = [Z_0 ; M_0 ; Q_0];
BElastic = inv(AElastic)*(CElastic*(n*W) + DElastic*q_pr +
```

```
EElastic); Bdeg = BElastic*180 / pi;
Trim_Elevator_Elastic = Bdeg(1)
Trim_Incidence_Elastic = Bdeg(2)

Note that BElastic(3) yields the generalised coordinate for the
fuselage bending mode deformation in the trimmed state and so
the absolute deformation may be obtained by multiplying by the
normalised mode shape.
```

I.6 DYNAMIC MANOEUVRES

In this section, related to Chapters 14 and 15, the equations of motion are defined with respect to the body fixed (in particular wind) axes system and therefore the matrices are structured differently. Only the linearized longitudinal symmetric manoeuvre involving heave and pitch will be considered here, so variations in the fore-and-aft velocity will be ignored. These linearized equations need to be solved against time for a particular control input. In this case, the elevator input will be considered as a step on–step off pulse and will be defined using an input array. Other inputs can be substituted if required.

The first SIMULINK function *Model_I6_Dynamic_Manoeuvres* in Figure I.1 will solve the response w, q to the elevator input in moving axes and integrate $q(= \dot{\theta})$ to determine θ. This output from the moving axes equations will then be used to transform these response velocities into an inertial reference frame fixed with respect to earth, thus yielding the velocities U_E, W_E in earth fixed axes. A further SIMULINK function *Model_I6_Earth_Axes* in Figure I.2 will be used to integrate these velocities in order to determine the position coordinates X_E, Z_E of the aircraft in earth axes. The reader should refer back to Appendix G for a brief introduction to setting up the SIMULINK block diagram.

It should be possible to analyse both the rigid and elastic aircraft using the same SIMULINK models, but with different equations defined in the MATLAB code. However, the code here *Pgm_I6_Dynamic_Manoeuvres* is for the rigid aircraft.

```
% Dynamic response of a symmetric rigid aircraft to an elevator
input-requires aircraft data, flight case and derivative codes

% Simulation data
tmin = 0;  tmax = 5.0; dt = 0.005;
```

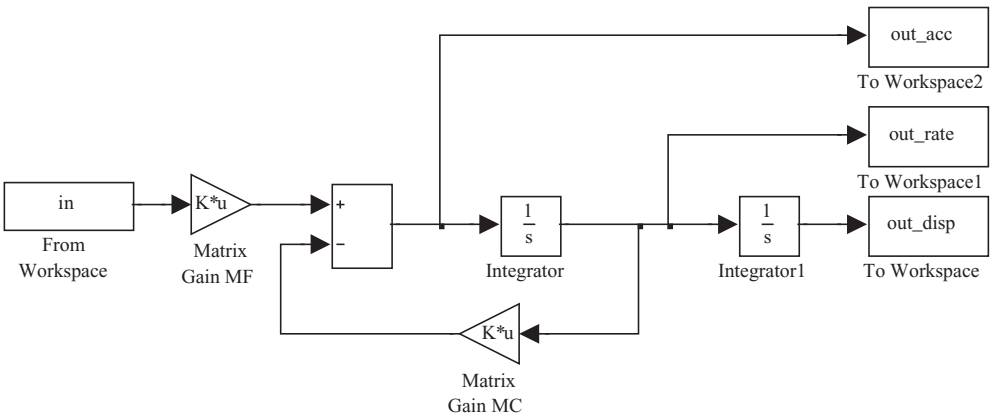

Figure I.1 SIMULINK diagram for a dynamic manoeuvre in body fixed axes – suitable for rigid and elastic aircraft.

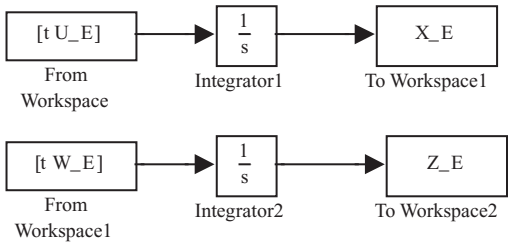

Figure I.2 SIMULINK diagram to integrate velocities in earth axes.

```
% Time increment - will need to be smaller for the elastic
  taircraft (~0.002s)
t = [tmin: dt: tmax]'; [N, dummy] = size(t); eta = zeros(N, 1);

% On / off pulse input to elevator
tpulse = 1.0; npulse = tpulse / dt + 1; eta_in = 2.0;
eta_in = eta_in*pi / 180;
eta(1:npulse) = eta_in*ones(npulse, 1);

% Aircraft initial condition
U_e = V0;  W_e = 0;

% Flight mechanics linearised equations of motion
M = [m 0;0 I_y]; C = [-Z_w -(m*U_e + Z_q); -M_w -M_q];
F = [Z_eta; M_eta];
MC = inv(M)*C; MF = inv(M)*F;

% Simulation of response in body fixed (wind) axes to yield w,q
and then theta
in = [t, eta];
sim('Model_I6_Dynamic_Manoeuvres');

% Output variables
w = out_rate(:,1);            % Downwards velocity
q = out_rate(:,2);            % Pitch rate
wdot = out_acc(:,1);          % Downwards acceleration
theta = out_disp(:,2);        % Integral of q (i.e. theta)
alpha = w / V0;               % Incidence
gamma = theta - alpha;        % Flight path angle - perturbation
az = wdot - q*U_e;            % Normal acceleration at CoM

% Response plots- body fixed axes
figure(1); subplot(311); plot(t, q*180/pi,'k-', t, alpha
*180/pi, 'k:')
title('Pitch Rate and Incidence Response for Elevator Input')
xlabel('Time (s)'); ylabel('Pitch Rate (deg/s), Incidence
(deg)'); legend('Pitch Rate', 'Incidence')
```

```
subplot(312); plot(t, az/9.81, 'k-')
title('Normal Acceleration for Elevator Input'); xlabel
('Time (s)'); ylabel('Normal Acceleration (g)')
subplot(313); plot(t, theta*180/pi, 'k-', t, gamma*180/pi, 'k:')
title('Pitch and Flight Path Angles for Elevator Input')
xlabel('Time (s)');  ylabel('Pitch and Flight Path Angles (deg)');
legend('Pitch Angle', 'Flight Path Angle')

% Transform velocities related to CoM from body fixed to earth
  fixed axes

% Centre of Mass
U_E = U_e + W_e*theta;
W_E = -U_e*theta + W_e + w;

% Tailplane
U_E_tp = U_e + W_e*theta + w.* theta;
W_E_tp = -U_e*theta + W_e + w + l_T*q;

% Response plots - velocities in earth axes
figure(2); plot(t, w, 'k-', t, W_E, 'k:')
title('Vertical Velocity Response (relative to wind / earth axes)
for Elevator Input')
xlabel('Time (s)');  ylabel('Velocity (m/s)')
legend('Vertical velocity relative to wind axes w', 'Vertical
velocity relative to earth axes WE')
figure(3);  plot(t, W_E, t, W_E_tp, 'k-')
title('Vertical Velocity Response (relative to earth axes) for
Elevator Input')
xlabel('Time (s)'); ylabel('Velocity WE (m/s)'); legend
('Centre of mass', 'Tailplane')

% Integrate to yield CoM position coordinates from velocities
in earth axes
sim('Model_I6_Earth_Axes');

% Response plots - CoM position in earth axes
figure(4);  plot(X_E, -Z_E, 'k-')
title('CoM Flight Profile in Earth Axes following Elevator Input')
xlabel('Horizontal Displacement XE (m)'); ylabel('Vertical
Displacement  ZE (m)')
```

Note that when setting up this and other SIMULINK models involving the transfer of data to and from the workspace, the following settings were used:

(a) From Workspace: set sample time dt, select 'Interpolation'.
(b) To Workspace: set sample time dt and decimation 1; select 'Array'.

Also, apart from the landing simulation, all initial conditions for the integrators are set to zero.

I.7 GUST RESPONSE IN THE TIME DOMAIN

In this section, related to the treatment of gusts in the time domain in Chapter 16, the equations of motion for the gust response of a rigid aircraft are set up and the response to a '1-cosine' gust is obtained using SIMULINK, set up for the second-order equations of motion referred to inertial axes; the gust time history is defined as an input array. The program is *Pgm_I7_Gust_Time* and the SIMULINK function is *Model_I7_Gust_Time*.

Note that the distance origin is at the aircraft tail, and at $t = 0$ the aircraft wing is about to enter the gust, as illustrated diagrammatically below (aircraft moving right to left). The simulation ends when the wing reaches the end of the total air space allocated.

```
% Gust response of a rigid aircraft in the time domain-requires
  aircraft data, flight case and derivative codes

% Gust profile                    ***************
% Air space     ----------------------------------------
% Aircraft      W+++++++T                        W+++++++T

% Set-up time array and length of simulation
tmin = 0; tmax = 8.0;
dt = 0.005;           % Time increment - will need to be smaller
                        for the elastic aircraft (~0.002s)
t = [tmin: dt: tmax]'; [Nsim, dummy] = size(t);

% Set-up distance array and size for simulation (important
  to use TAS not EAS)
x_g = V*t;                      % Distance array
dx = V*dt;                      % Distance increment
Nb = round(l_WT / dx+1);        % No of points between wing/tailplane
N = Nsim + Nb - 1;              % No of points for total air space

% Gust velocity profile (1-cosine)
delta_wgt = 5.0;                % Max gust velocity TAS
L_g = 250.0;                    % Gust length
Nd = round(L_g / dx + 1);       % No of points for gust
Nbd = Nb + Nd - 1;              % No of points for a/c and gust
wg = zeros(N, 1);
wg_W = zeros(N, 1);
wg_T = zeros(N, 1);
wg(Nb:Nbd) = (delta_wgt / 2)*
(1 - cos(2*pi*x_g(1:Nd) / L_g));  % Gust velocity array at centre
                                    of mass
wg_W(1:Nsim) = wg(Nb:N);
wg_T(1:Nsim) = wg(1:Nsim);        % Gust velocity array at wing and
                                    tailplane
% Equations of motion for rigid aircraft (elastic is similar
  but with an additional row / column for elastic mode
M = [m 0; 0 I_y]; CC = - [Z_zdot Z_q; M_zdot M_q]; KK = -
[0 Z_alpha; 0 M_alpha];
FW = [Z_gW;  M_gW]; FT = [Z_gT;  M_gT];
```

```
MI = inv(M); MC = MI*CC; MK = MI*KK; MFW = MI*FW; MFT = MI*FT;

% Simulation to find displacements and velocities of response in
  inertial axes
inW = [t, wg_W(1:Nsim)];
inT = [t, wg_T(1:Nsim)];   % Input arrays at wing and tail[lane
sim('Model_I7_Gust_Time')  % Solve equations via SIMULINK model

% Displacements, velocities and accelerations at CoM, nose and
  tail (all Nsim x 1 arrays)
z_C = out_disp(:,1); zdot_C = out_rate(:,1); zddot_C = out_acc(:,1);
theta = out_disp(:,2); theta_dot = out_rate(:,2); theta_ddot =
out_acc(:,2);
z_F = z_C - l_F*theta; zddot_F = zddot_C - l_F*theta_ddot;
z_T = z_C + l_T*theta; zddot_T = zddot_C + l_T*theta_ddot;

% Plot responses
subplot(211); plot(t, theta*180/pi, 'k-', t, theta_dot *180/pi, 'k:')
title('Pitch Response for Rigid Aircraft in Gust')
xlabel('Time (s)'); ylabel('Pitch (deg) and Pitch Rate (deg/s)');
legend('Pitch Angle', 'Pitch Rate')
subplot(212); plot(t, zddot_C/9.81, 'k-', t, zddot_T/9.81, 'k:')
title('Nose/CoM/Tail Heave Acceleration for Rigid Aircraft in Gust')
xlabel('Time (s)'); ylabel('Heave Acceleration (g)'); legend
('CoM', 'Tailplane')
```

The SIMULINK diagram for the rigid gust case in the time domain is shown in Figure I.3; the same model may be used for the elastic aircraft, with the matrices simply being defined differently in the MATLAB code.

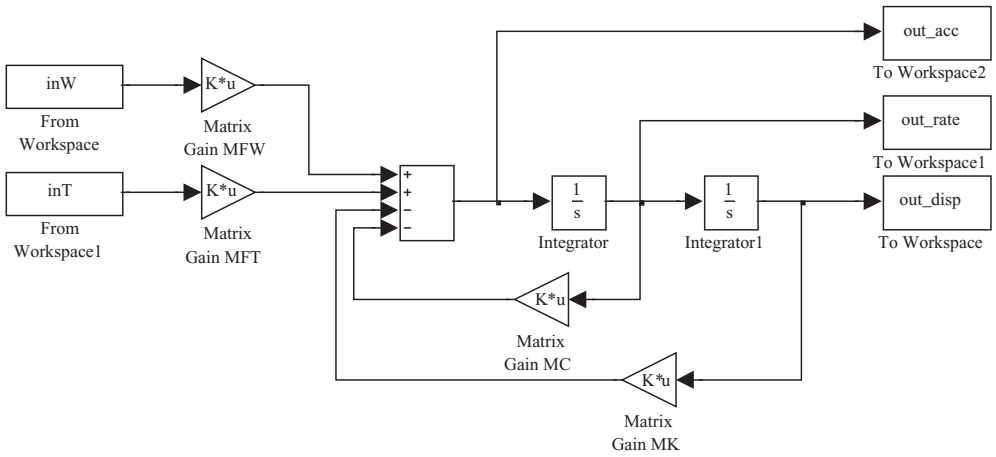

Figure I.3 SIMULINK diagram for the time domain gust response.

I.8 GUST RESPONSE IN THE FREQUENCY DOMAIN

In this section, related to the treatment of gusts (i.e. continuous turbulence) in the frequency domain in Chapter 16, the equations of motion for the gust response of a rigid aircraft are set up and the power spectral density (PSD) of the response to turbulence is obtained using MATLAB *Pgm_I8_Gust_Frequency*.

```
% Gust response of a rigid aircraft in the frequency domain-
  requires aircraft data, flight case and derivative codes

% RMS gust velocity (TAS) and characteristic scale wavelength
  (2500 ft-converted to m)
sigma_g = 1; L_g = 2500/3.2808;

% Equations of motion for rigid aircraft
M = [m 0; 0 I_y]; CC = [- Z_zdot -Z_q; -M_zdot -M_q];
KK = [0 -Z_alpha; 0 -M_alpha];
FW = [Z_gW; M_gW]; FT = [Z_gT;  M_gT];

% Calculate Gust PSD and FRF for each frequency value from 0 up
  to Nyquist and set Nyquist frequency
% value to be real - note that FRF needs to be 2 dimensional
  (responses and frequency stored)
% The zero frequency value is NaN (Not a Number) because the KK
  Matrix is singular
f_nyq = 5; nt = 1024; nf = nt/2 + 1; frq = linspace(0, f_nyq, nf);
df = frq(2) - frq(1);
H = zeros(2, nf); Hddot = zeros(2, nf); i = sqrt(-1);

for ifq = 1:nf
  w = 2*pi*frq(ifq); omLg = w*L_g/V; OM(ifq) = w/V;
  num = 1 + 8/3*(1.339*omLg)^2; den = (1 + (1.339*omLg)^2)^(11/6);
  psi_gr(ifq) = sigma_g^2
  *L_g/pi*num/den;             % Gust PSD for reduced frequency
  psi_gf(ifq) = sigma_g^2
  *2*L_g/V*num/den;            % Gust PSD for true frequency
  HI = (KK - w^2*M + i*w*CC)^-1; HQG = HI*(FW + FT * exp(- w*l_
  WT/V));
  Hqg(:,ifq) = HQG;  Hqddotg(:,ifq) = - w^2*Hqg(:,ifq);
end
% Plot gust PSD against frequency (not reduced)
  figure(1); loglog(frq(1:nf), psi_gf(1:nf), 'k-');
xlabel('Frequency (Hz)'); ylabel('PSD of Gust Velocity (m/s)^2/Hz');
title('Gust PSD - Frequency')

% Convert to FRF and |FRF|^2 for centre of mass response from
  generalised (heave/pitch) responses
Hz_C = [1 0]*Hqg; Hz_C2 = (abs(Hz_C)).^2; Hzddot_C = [1 0] *
Hqddotg; Hzddot_C2 = (abs(Hzddot_C)).^2;

% Convert to FRF for front fuselage and tailplane response from
  generalised (heave / pitch) responses
```

```
Hz_F = [1 -l_F]*Hqg;  Hz_T = [1 l_T]*Hqg;

% Calculation of centre of mass response PSD from gust PSD and
  response-to-gust |FRF|^2
Pzddot_C2 = psi_gf.*Hzddot_C2;

% Calculate root-mean-square values of acceleration at centre
  of mass (convert from m/s^2 to g)
g2 = 9.81^2;  sumC = 0;
for ifq = 2:nf
  sumC = sumC + Pzddot_C2(ifq) / g2*df;
end
rmsCg = sqrt(sumC)

% Plot centre of mass response PSD against frequency
figure(2); loglog(frq(1:nf), Hzddot_C2(1:nf)/g2,'k:',frq(1:nf),
Pzddot_C2(1:nf)/g2,'k-');
xlabel('Frequency (Hz)'); ylabel('FRF^2 and Acceleration PSD
(g^2 / Hz)');
title('Frequency Domain Gust Response - Rigid Aircraft with
Heave/Pitch Model')
legend('(Acceleration per Gust Velocity FRF)^2', 'Acceleration PSD')
```

I.9 GROUND MANOEUVRES

In this section, the calculation of responses to taxiing and landing as outlined in Chapter 17 is considered.

I.9.1 Taxiing

In this subsection, the equations of motion for the taxiing response of a rigid aircraft are set up and the response to a '1-cosine' dip in the runway is obtained using MATLAB *Pgm_I91_Taxiing* and SIMULINK *Model_I91_Taxiing*. Note in this example that the equations are set up in first order state space form and not as second-order equations, as was the case for the gust response; either approach may be employed.

```
% Rigid aircraft taxiing over a 1-cosine dip-requires aircraft data,
  flight case but not derivative codes

% Time and distance data
tmin = 0; tmax = 10.0; dt = 0.01;  % Time increment - will need to
be smaller for the elastic aircraft (~0.002s)
t = [tmin: dt: tmax]'; [Nsim, dummy]=size(t);
x_r = V*t; dx = V*dt; Nb = round(l_B/dx + 1); N = Nsim + Nb - 1;

% Runway profile - define (1-cosine) dip - note similarity
to gust except for the need for hdot terms

%Dip                              ***************
%Runway   ------------------------------------------------------
%Aircraft    N+++++++M                                  N+++++++M
```

```
delta_h_r = 0.03; L_r = 60; Ndip = round(L_r / dx + 1); Nbd =
Nb + Ndip - 1;
h = zeros(N, 1); h_N = zeros(N, 1);  h_M = zeros(N, 1);
hdot = zeros(N, 1);  h_Ndot = zeros(N, 1); h_Mdot = zeros(N, 1);
h(Nb:Nbd) = (delta_h_r / 2)*(1 - cos(2*pi*x_r(1:Ndip) / L_r));
hdot(Nb:Nbd) = V*pi*delta_h_r / L_r*sin(2*pi*x_r(1:Ndip) / L_r);

Runway profile defined at nose and main gear positions
h_N(1:Nsim) = h(Nb:N);  h_M(1:Nsim) = h(1:Nsim);
h_Ndot(1:Nsim) = hdot(Nb:N);  h_Mdot(1:Nsim) = hdot(1:Nsim);

% Equations of motion in second order form
M = [m  0; 0  I_y];
CC = [C_N + C_M  -l_N*C_N + l_M*C_M; -l_N*C_N + l_M*C_M l_N^2*
C_N + l_M^2*C_M];
K = [K_N + K_M  -l_N*K_N + l_M*K_M; -l_N*K_N + l_M*K_M l_N^2
*K_N + l_M^2*K_M];
DC = [C_N  C_M;  -l_N*C_N l_M*C_M]; DK = [K_N K_M;  -l_N* K_N
l_M*K_M];
MC = inv(M)*CC; MK = inv(M)*K; MDC = inv(M)*DC; MDK = inv(M)*DK;

% State space equations in first order form
Null22 = zeros(2, 2); I = eye(2); C = eye(4); D = zeros(4, 4);
A = [Null22 I; -MK -MC]; B = [Null22 Null22; MDK MDC];

% State space input vector (Nsim x 4)
u = [h_N(1:Nsim) h_M(1:Nsim) h_Ndot(1:Nsim) h_Mdot(1:Nsim)];

% Simulation in state space to find displacements and velocities
in = [t, u]; sim('Model_I91_Taxiing')

% Displacements at CoM, nose and main gears (all Nsim x 1)
% Dimension of 'out' is Nsim x 4 (zc, theta, zcdot, thetadot)
z_C = out(:,1); z_N = z_C - out(:,2)*l_N; z_M = z_C + out(:,2)*l_M;

% Calculate accelerations from state space equations
% Dimension of 'outdot' is Nsim x 4 (zcdot, thetadot, zcdddot,
thetaddot)
outdot = (A*out' + B*u')'; zddot_C = outdot(:,3); theta_ddot =
outdot(:,4);
zddot_N = zddot_C - theta_ddot*l_N; zddot_M = zddot_C + theta_
ddot*l_M;

% Plot responses at nose and main gear
subplot(211); plot(t, 1000*z_N, 'k-', t, 1000*z_M, 'k:')
title('Nose/Main Gear Heave for Taxiing Rigid Aircraft')
xlabel('Time (s)'); ylabel('Heave Response (mm)'); legend
('Nose Gear', 'Main Gear')

subplot(212); plot(t, zddot_N/9.81, 'k-', t, zddot_M/9.81,'k:')
```

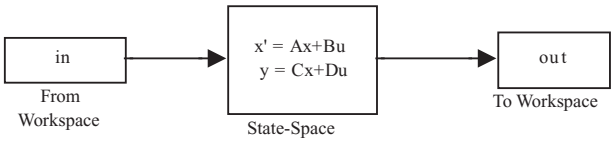

Figure I.4 SIMULINK diagram for the time domain taxiing response.

```
title('Nose/Main Gear Heave Acceleration for Taxiing Rigid
Aircraft')
xlabel('Time (s)'); ylabel('Heave Acceleration (g)'); legend
('Nose Gear', 'Main Gear')
```

The SIMULINK diagram for the rigid taxiing case is shown in Figure I.4; the same model may be used for the flexible aircraft, with the matrices simply being defined differently in the MATLAB code.

I.9.2 Landing

In this subsection, the MATLAB code *Pgm_I92_Landing* and SIMULINK model *Model_I92_Landing* that were used to generate the results for the nonlinear landing example in Chapter 17 are shown. The example included is a model of the half aircraft mass in heave motion only, the shock absorber and the tyre; look-up tables were used for the nonlinear shock absorber stiffness and damping characteristics. The SIMLINK model is shown in Figure I.5.

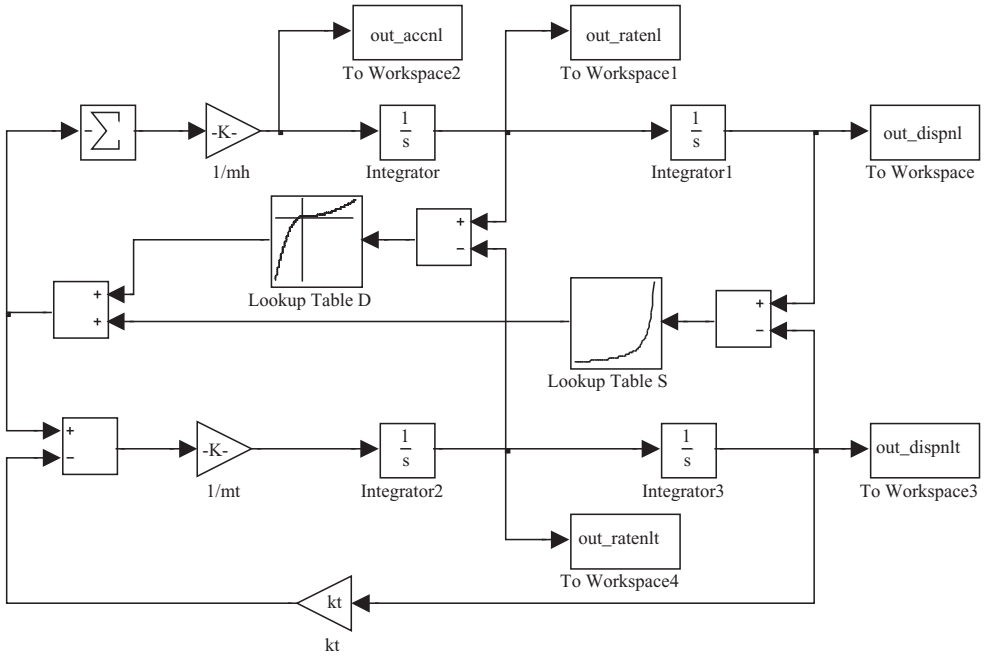

Figure I.5 SIMULINK diagram for the time domain nonlinear landing response.

```
% Rigid aircraft landing - nonlinear gas spring/tyre model -
  requires aircraft data, but not flight case/derivatives
mh = m / 2;               % Factor total mass for half aircraft
Wh = mh*9.81;
Lh = Wh;                  % Lift balances weight
W_e = 3.0;                % Vertical speed on landing (TAS)

% Single main landing gear parameters leading to non-linear
  behaviour
Pinf = 25e5; PS = 1e7; PC = 3e7; PA = 1e5; zS = 0.4; Area = Wh/PS;
Vratio = PC/Pinf;  Vinf = Vratio*Area*zS / (Vratio - 1);
zinf = Vratio/(Vratio - 1)*zS; ns = 1; nd = 1.35;

% Generate force ~ displacement variation for non-linear shock
  absorber stiffness look-up table
dz = 0.005; z = [0: dz: zS]; [dummy, nz] = size(z);
for j=1:nz
  Pd(j) = Pinf/(1 - z(j) / zinf)^nd;
  Fd(j) = (Pd(j) - PA)*Area;
end

% Generate force ~ velocity variation for nonlinear shock
absorber damping look-up table
Ccomp = 8000; Crecoil = 120000;
zdot = [-1.5: 0.005: 3]; [dummy, nzdot] = size(zdot);
for j=1:nzdot
  if zdot(j) >= 0
Fdamp(j) = Ccomp*zdot(j)^2;
  else
Fdamp(j) = - Crecoil*zdot(j)^2;
  end
end

% Tyre data for half aircraft
mt = 100;  kt = 1000e3; ct = 0;

% Run simulation for aircraft half mass on non-landing gear with
tyre mass/spring representation
tmin = 0; tmax = 0.5; dt = 0.0002; t = [tmin: dt: tmax]';
sim('Model_I92_Landing')
% Plot output
figure(1); plot(t, -out_accnl/9.81,'k-'); title('Main Gear
Deceleration for Rigid Aircraft Landing - Non-linear')
xlabel('Time (s)');  ylabel('Main Gear Deceleration (g)')

figure(2);  plot(t, out_dispnl, 'k-', t, (out_dispnl - out_
dispnlt), 'k:', t, out_dispnlt, 'k--')
title('Main Gear/Shock Absorber/Tyre Displacement Response
of Rigid Aircraft Landing - Non-linear')
xlabel('Time (s)'); ylabel('Displacement (m)'); legend('Main
Gear', 'Shock Absorber', 'Unsprung Mass')
```

```
Fgrnd = ct*out_ratenlt +
kt*out_dispnlt;              % Ground reaction force
figure(3);  plot(t, Fgrnd/Wh, 'k-'); title('Normalised Ground
Load - Rigid Aircraft Landing - Non-linear')
xlabel('Time (s)'); ylabel('Normalised Ground Load')
```

Note that the initial condition for the left hand integrator is the landing velocity We.