H

MATLAB/SIMULINK Programs for Flutter

In this appendix, some sample MATLAB programs are given for the calculation of the aeroelastic behaviour of a binary aeroelastic system, its response to control surface and gust/turbulence inputs, and also the addition of a simple PID control loop to reduce the gust response.

H.1 DYNAMIC AEROELASTIC CALCULATIONS

In Chapter 11 the characteristics of the flutter phenomenon were described using a binary aeroelastic system. The following code sets up the system equations, including structural damping if required, and then solves the eigenvalue problem for a range of speeds and plots the $V\omega$ and Vg trends.

```
% Pgm_H1_Calcs
% Sets up the aeroelastic matrices for binary aeroelastic model,
% performs eigenvalue solution at desired speeds and determines
the frequencies
% and damping ratios
% plots V_omega and V_g trends

% Initialize variables
clear; clf

% System parameters
s = 7.5;          % semi span
c = 2;            % chord
m = 100;          % unit mass / area of wing
kappa_freq = 5;   % flapping freq in Hz
theta_freq = 10;  % pitch freq in Hz
xcm = 0.5*c;      % position of centre of mass from nose
xf = 0.48*c;      % position of flexural axis from nose
e = xf/c - 0.25;  % eccentricity between flexural axis and aero
                    centre (1/4 chord)

velstart = 1;     % lowest velocity
velend = 180;     % maximum velocity
velinc =0.1;      % velocity increment
```

```
a = 2*pi;         % 2D lift curve slope
rho = 1.225;      % air density
Mthetadot = -1.2; % unsteady aero damping term
M = (m*c^2 - 2*m*c*xcm)/(2*xcm);   % leading edge mass term

damping_Y_N = 1;  % =1 if damping included   =0 if not included
if damping_Y_N == 1
    % structural proportional damping inclusion    C = alpha *
      M + beta * K
    % then two freqs and damps must be defined
    % set dampings to zero for no structural damping
    z1 = 0.0;               % critical damping at first frequency
    z2 = 0.0;               % critical damping at second frequency
    w1 = 2*2*pi;            % first frequency
    w2 = 14*2*pi;           % second frequency
    alpha = 2*w1*w2*(-z2*w1 + z1*w2)/ (w1*w1*w2*w2);
    beta = 2*(z2*w2-z1*w1) / (w2*w2 - w1*w1);
end

% Set up system matrices
% Inertia matrix
a11=(m*s^3*c)/3 + M*s^3/3;  % I kappa
a22= m*s*(c^3/3 - c*c*xf + xf*xf*c) + M*(xf^2*s); % I theta
a12 = m*s*s/2*(c*c/2 - c*xf) - M*xf*s^2/2;  %I kappa theta
a21 = a12;
A=[a11,a12;a21,a22];

% Structural stiffness matrix
k1 = (kappa_freq*pi*2)^2*a11;     %  k kappa   heave stiffness
k2 = (theta_freq*pi*2)^2*a22;     %  k theta   pitch stiffness
E = [k1 0; 0 k2];

icount = 0;
for V = velstart:velinc:velend       % loop for different velocities
    icount = icount +1;
    if damping_Y_N == 0;    % damping matrices
        C = [0,0; 0,0];        % =0 if damping not included
    else                    % =1 if damping included
        C = rho*V*[c*s^3*a/6,0;-c^2*s^2*e*a/4,-c^3*s*Mthetadot/8] +
            alpha*A + beta*E;
        % Aero and structural damping
    end
    K = (rho*V^2*[0,c*s^2*a/4; 0,-c^2*s*e*a/2])+[k1,0; 0,k2]; %
        aero / structural stiffness

    Mat = [[0,0; 0,0],eye(2); -A\K,-A\C];   % set up 1st order
                                              eigenvalue solution
                                              matrix
    lambda = eig(Mat);                       % eigenvalue solution

    % Natural frequencies and damping ratios
```

```
    for jj = 1:4
        im(jj) = imag(lambda(jj));
        re(jj) = real(lambda(jj));
        freq(jj,icount) = sqrt(re(jj)^2+im(jj)^2);
        damp(jj,icount) = -100*re(jj)/freq(jj,icount);
        freq(jj,icount) = freq(jj,icount)/(2*pi);    % convert
                                                       frequency to
                                                       hertz
    end
    Vel(icount) = V;
end

% Plot frequencies and dampings vs speed
figure(1)
subplot(2,1,1); plot(Vel,freq,'k');
vaxis = axis; xlim = ([0 vaxis(2)]);
xlabel ('Air Speed (m/s) '); ylabel ('Freq (Hz)'); grid

subplot(2,1,2);
plot(Vel,damp,'k')
xlim = ([0 vaxis(2)]); axis([xlim ylim]);
xlabel ('Air Speed (m/s) '); ylabel ('Damping Ratio (%)'); grid
```

H.2 AEROSERVOELASTIC SYSTEM

In Chapter 12 the inclusion of a closed loop control system was introduced via the addition of a control surface to the binary flutter model. The following code enables the response of the binary aeroelastic system subject to control surface excitation and also a vertical gust sequence to be calculated through the use of the SIMULINK function Binary_Sim_Gust_Control shown in Figure H1. The control surface input is defined as a ‘chirp’ with start and end frequencies needing to be specified, and the gust input contains both ‘1-cosine’ and random turbulence inputs. Note that the random signal is generated by specifying an amplitude variation and random phase in the frequency domain via the inverse Fourier transform. All simulations are in the time domain. Note that the feedback loop is not included here but it would be a straightforward addition to the code.

```
% Chapter B05

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    Binary Aeroelastic System plus control plus turbulence    %%
%    Define control_amp, turb_amp, gust_amp_1_minus_cos to     %%
%    determine which inputs are included                       %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all; close all

% System parameters

V = 100;          % Airspeed
s = 7.5;          % semi span
```

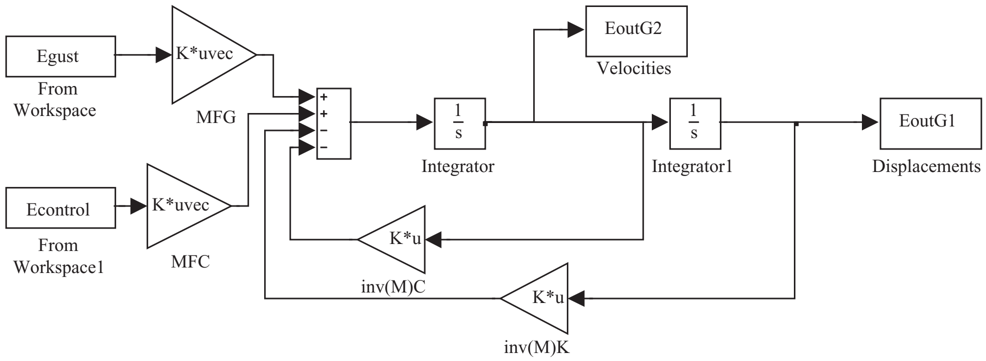

Figure H.1 SIMULINK Implementation for the open loop aeroservoelastic system.

```
c = 2;              % chord
a1 = 2*pi;          % lift curve slope
rho = 1.225;        % air density
m = 100;            % unit mass / area of wing
kappa_freq = 5;     % flapping freq in Hz
theta_freq = 10;    % pitch freq in Hz
xcm = 0.5*c;        % position of centre of mass from nose
xf = 0.48*c;        % position of flexural axis from nose
Mthetadot = -1.2;   % unsteady aero damping term
e = xf/c - 0.25;    % eccentricity between flexural axis and
                      aero centre
damping_Y_N = 1;    % =1 if damping included   =0 if not included

% Set up system matrices
a11 = (m*s^3*c)/3 ;  % I kappa
a22 = m*s*(c^3/3 - c*c*xf + xf*xf*c);   % I theta
a12 = m*s*s/2*(c*c/2 - c*xf);           % I kappa theta
a21 = a12;
k1 = (kappa_freq*pi*2)^2*a11;           % k kappa
k2 = (theta_freq*pi*2)^2*a22;           % k theta
A = [a11,a12; a21,a22];
E = [k1 0; 0 k2];
if damping_Y_N == 0;  %    =0 if damping not included
    C = [0,0; 0,0];
else
    C = rho*V*[c*s^3*a1/6,0; -c^2*s^2*e*a1/4,-c^3*s*Mthetadot/8];
end
K = (rho*V^2*[0,c*s^2*a1/4; 0,-c^2*s*e*a1/2])+[k1,0;0,k2] ;

% Gust vector
F_gust = rho*V*c*s*[s/4 c/2]';

% Control surface vector
EE = 0.1;  % fraction of chord made up by control surface
```

```
ac = a1/pi*(acos(1-2*EE) + 2*sqrt(EE*(1-EE)));
bc = -a1/pi*(1-EE)*sqrt(EE*(1-EE));
F_control = rho*V^2*c*s*[-s*ac/4 c*bc/2]';

% Set up system matrices for SIMULINK

MC = inv(A)*C;
MK = inv(A)*K;
MFG = inv(A)*F_gust;
MFC = inv(A)*F_control;

dt = 0.001;         % sampling time
tmin = 0;           % start time
tmax = 10;          % end time
t = [0:dt:tmax]';  % Column vector of time instances

%%%%%% CONTROL SURFACE INPUT SIGNAL - SWEEP SIGNAL
%%%%%%%%%%%%%%%%%%%%%%%%%%%
control_amp = 5;     % magnitude of control surface sweep input
in degrees
control_amp = control_amp * pi / 180;   % radians
burst = .333;        %  fraction of time length that is chirp
signal  0 - 1
sweep_start = 1;     % chirp start freq in Hertz
sweep_end = 20;      % chirp end freq in Hertz
t_end = tmax * burst;

Scontrol = zeros(size(t));   % control input
xt = sum(t < t_end);
for ii = 1:xt
    Scontrol(ii) = control_amp*sin(2*pi*(sweep_start + (sweep_end -
sweep_start)*ii/(2*xt))*t(ii));
end

%%%%%%%%%%% GUST INPUT TERMS - "1-cosine" and/or turbulence %%%
%%%%%%%%%%%%%%%

Sgust = zeros(size(t));

%%%  1 - Cosine gust
gust_amp_1_minus_cos = 0;       % max velocity of "1 - cosine"
gust   (m/s)
gust_t = 0.05;   %  fraction of total time length that is gust 0 - 1
g_end = tmax * gust_t;
gt = sum(t < g_end);
for ii = 1:gt
    Sgust(ii) = gust_amp_1_minus_cos/2 * (1 - cos(2*pi*t(ii)/g_end));
end

%%% Turbulence input - uniform random amplitude between 0 Hz and
```

```
turb_max_freq Hz %%%
turb_amp = 0;       %   max vertical velocity of turbulence   (m/s)
turb_t = 1;         %   fraction of total time length that is
turbulence    0 - 1
t_end = tmax * turb_t;
turb_t = sum(t < t_end);  % number of turbulence time points required
turb_max_freq = 20;       % max frequency of turbulence(Hz) -
uniform freq magnitude
npts = max(size(t));
if rem(max(npts),2) ~= 0  % code set up for even number
    npts = npts - 1;
end
nd2 = npts / 2;
nd2p1 = npts/2 + 1;
df = 1/(npts*dt);
fpts = fix(turb_max_freq / df) + 1; % number of freq points that form
turbulence input

for ii = 1:fpts  % define real and imag parts of freq domain
                 % magnitude of unity and random phase
    a(ii) = 2 * rand - 1;    % real part  - 1 < a < 1
    b(ii) = sqrt(1 - a(ii)*a(ii)) * (2*round(rand) - 1);
% imag part
end

% Determine complex frequency representation with correct
frequency characteristics
tf =  (a + j*b);
tf(fpts+1 : nd2p1) = 0;
tf(nd2p1+1 : npts) = conj(tf(nd2:-1:2));
Sturb = turb_amp * real(ifft(tf));

for ii = 1:npts
    Sgust(ii) = Sgust(ii) + Sturb(ii); % "1 - cosine" plus
turbulence inputs
end

% Simulate the system using SIMULINK
Egust = [t,Sgust]; % Gust Array composed of time and data columns
Econtrol = [t,Scontrol]; % Control Array composed of time and data
columns

[tout] = sim('Binary_Sim_Gust_Control');

x1 = EoutG1(:,1)*180/pi;   % kappa - flapping motion
x2 = EoutG1(:,2)*180/pi;   % theta - pitching motion
x1dot = EoutG2(:,1)*180/pi;
x2dot = EoutG2(:,2)*180/pi;
figure(1); plot(t,Scontrol,t,Sgust)
```

```
xlabel('Time (s)'); ylabel('Control Surface Angle(deg) and
Gust Velocity(m/s)')
figure(2) ; plot(t,x1,'r',t,x2,'b')
xlabel('Time (s)'); ylabel('Flap and Pitch Angles (deg/s)')
figure(3); plot(t,x1dot,'r',t,x2dot,'b')
xlabel('Time (s)'); ylabel('Flap and Pitch Rates (deg/s)')
```