G

MATLAB/SIMULINK Programs for Vibration

It is beyond the scope of this book to present a detailed treatment of the use of MATLAB and SIMULINK, but instead a few key programs are presented which the reader can use as a template for trying other related calculations. It is assumed that a basic knowledge of the packages will be obtained from other sources.

G.1 FORCED RESPONSE OF AN SDoF SYSTEM

In Chapter 1, the example of an SDoF system excited by a single cycle of a square wave was considered. The response was obtained in three ways, namely using superposition (essentially simple convolution), numerical integration and via the frequency domain (using the Fourier Transform).

G.1.1 Superposition (Essentially Convolution)

The superposition approach treats the double pulse excitation as the combination of three steps. The response to a single step is used as a constituent building block in generating the overall response. Clearly, different inputs such as square waves of different numbers of cycles and periods may be explored by simple modifications of the code.

The program *Pgm_G11_Superposition* is shown below. The theory in Chapter 1 will be required to follow the program.

```
% Response of SDoF system to single cycle of a square wave
clear all; close all;

% System parameters
f_nat = 2; period = 1 / f_nat; w_nat = 2 * pi * f_nat; zeta = 0.05;
mass = 1000; stiff = w_nat^2 * mass; force = 1000;
w_dpd = w_nat* sqrt(1-zeta^2); psi = atan2(sqrt(1 - zeta^2), zeta);

% Data parameters
T = 8; dt = 0.01; t = [0:dt:T]; [dummy,nt] = size(t);

% Response to step force
for it = 1:nt
   a = exp(-zeta * w_nat * t(it)) / sqrt(1 - zeta^2);
   b = sin(w_dpd * t(it) + psi);
   s(it)= force / stiff * (1 - a * b);
end
```

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

```
% Response to square wave using superposition
% Function p shows 'shape' of excitation force
pulse_width = period / 2; npulse = round(pulse_width / dt);
for it = 1 : npulse
   x(it) = s(it); f(it) = 10;
end
for it = npulse + 1 : 2 * npulse
   x(it) = s(it) - 2 * s(it - npulse);
   f(it) = - 10;
end
for it = 2 * npulse + 1 : nt
   x(it) = s(it) - 2 * s(it - npulse) + s(it - 2 * npulse);
   f(it) = 0;
end

% Plot response in mm
plot(t,x*1000,'k-',t,f,'k:'); axis([0 T -25 25]);
xlabel('Time (s)'); ylabel('Response to Double Pulse (mm)')
```

G.1.2 Numerical Integration

The numerical integration approach uses the SIMULINK package called from MATLAB. The SIMULINK diagram shown in Figure G.1 is built by rewriting the differential equation of motion in terms of the acceleration, so typically

$$\ddot{x} = -\frac{c}{m}\dot{x} - \frac{k}{m}x + \frac{f(t)}{m}. \tag{G.1}$$

The three right-hand side terms are added in the sum block to provide the acceleration, the integrators generate velocity and displacement and the feedback terms provide the inputs to the sum block and so on. The force array corresponding to the required time values is calculated in the MATLAB code from which the SIMULINK function *Model_G12_Num_Integration* is called. The result of the integration is returned to the workspace for plotting, etc. Clearly, other excitation types may be explored by modifying this code.

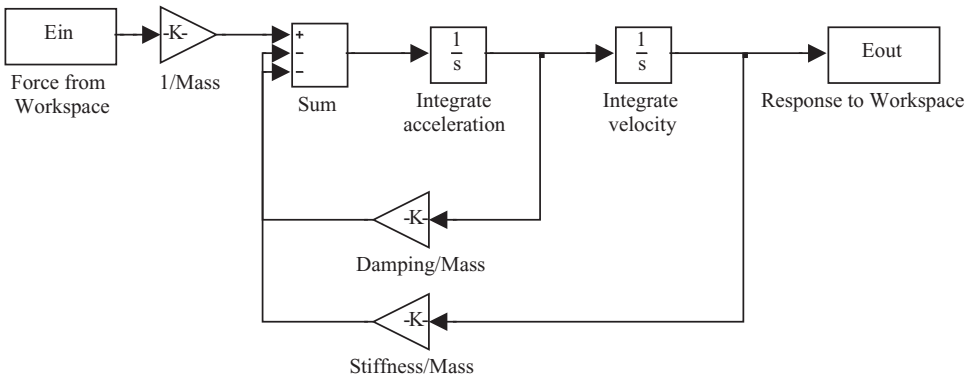

Figure G.1 SIMULINK diagram for the SDoF example.

It should be noted that it is possible to solve simultaneous differential equations (linear or nonlinear) using SIMULINK. A further example is shown in Appendix I where manoeuvres and gust encounters are considered.

The program *Pgm_G12_Num_Integration* is shown below. The theory in Chapter 1 will be required to follow the program.

```
% Response of SDoF to single cycle of a square wave using SIMULINK
clear all; close all

% System parameters
f_nat = 2; period = 1 / f_nat; w_nat = 2 * pi * f_nat; zeta = 0.05;
mass = 1000; damp = 2 * zeta * mass * w_nat;
stiff = w_nat^2 * mass; force = 1000;

% Simulation parameters
T = 8; dt = 0.01; t = [0:dt:T]; [dummy,nt] = size(t);

% Define excitation f(t)
pulse_width = period / 2; npulse = round(pulse_width / dt);
for it = 1 : npulse
   f(it) = force;
end
for it = npulse + 1 : 2 * npulse
   f(it) = - force;
end
for it = 2 * npulse + 1 : nt
   f(it) = 0;
end

% Run SIMULINK model for SDOF (works with column vectors in
  workspace
% blocks Ein and Eout) using variable step length solver ODE45 with
% outputs at same time intervals dt as for input - note that the
  SIMULINK file
% needs to be in the same directory as the core MATLAB program
Ein = [t',f'];
sim('Model_G12_Num_Integration');

% Access stored data for response x - convert back to row vector for
% consistency with f and t - convert to mm
x = 1000 * Eout';

% Plot response
figure(1); plot(t,x); axis([0 T -25 25]);
xlabel('Time (s)'); ylabel(`Response to Single Cycle of a Square
Wave');
```

G.1.3 Frequency Domain

The frequency domain approach uses the Fourier transform. The excitation time sequence is generated, transformed to the frequency domain, multiplied by the frequency response function (FRF) and inverse transformed to yield the response. It will be seen that special care is taken to include the negative frequency

values in the FRF in order to match the form of the Fourier transform and so allow inverse transformation to be carried out. Studying this program and its output will help the reader gain some understanding of the use of the Fourier transform in MATLAB. Clearly, other input types and analyses may be tried by suitable modification of the code.

The program *Pgm_G13_Frequency_Domain* is shown below. The theory in Chapter 1 will be required to follow the program.

```
% Response of SDoF to single cycle of a square wave using the
  Fourier Transform and
% transformation into the frequency domain and back again
clear all; close all

% System parameters
f_nat = 2; period = 1 / f_nat; w_nat = 2 * pi * f_nat; zeta = 0.05;
mass = 1000; stiff = w_nat^2 * mass; force = 1000;

% Data parameters in time and frequency domains - note that time
  vector
% stops dt short of T for FT analysis to remain a power of 2 and
  periodic
T = 8; nt = 128; dt = T / nt; t = [0:dt:T - dt];
df = 1 / T; f_nyq = 1 / 2 / dt; frq = [0:df:f_nyq]; [dummy,nf] =
size(frq);

% Define square wave cycle excitation f(t)
pulse_width = period / 2; npulse = round(pulse_width / dt);
for it = 1 : npulse
  f(it) = force;
end
for it = npulse + 1 : 2 * npulse
  f(it) = - force;
end
for it = 2 * npulse + 1 : nt
  f(it) = 0;
end

% Fourier Transform f(t) to frequency domain FF - apply scaling
factor nt
FF = fft(f,nt) / nt;
figure(1)
plot(frq(1:nf),real(FF(1:nf)),'kx',frq(1:nf),imag(FF(1:nf)),'ko');
xlabel('Frequency (Hz)'); ylabel(`Real / Imaginary Parts - FT of
x(t)');
legend('Real Part','Imaginary Part')

% Generate the Frequency Response Function (FRF) for SDoF over 0 -
  f_nyq
for ifq = 1 : nf
  w = 2 * pi * frq(ifq);
  r = w / w_nat;
  H(ifq) = (1 / stiff)/(1 - r^2 + i * 2 * zeta * r);
end
```

```
H(nf) = real(H(nf));

% Generate the FRF 'negative' frequency content (i.e. pack to nt
  complex
% numbers) to be in correct format for inverse FT
for ifq = nf + 1 : nt
   H(ifq) = conj(H(nt - ifq + 2));
end
figure(2)
plot(frq(1:nf),real(H(1:nf)),'kx',frq(1:nf),imag(H(1:nf)),'ko');
xlabel('Frequency (Hz)'); ylabel(`Real / Imaginary Parts - FRF');
legend('Real Part','Imaginary Part')

% Multiply FRF by FT of f(t)(element by element) to get XF - FT of
  x(t)
XF = H. * FF;
figure(3)
plot(frq(1:nf),real(XF(1:nf)),'kx',frq(1:nf),imag(XF(1:nf)),'ko')
xlabel('Frequency (Hz)'); ylabel(`Real /Imaginary Parts - FFT of
y(t)');
legend('Real Part','Imaginary Part');

% Generate response x(t) using the IFT of XF - apply scaling factor
nt
x = ifft(XF) * nt;

% Plot response in mm
figure(4); subplot(211);
plot(t,f,'k');
xlabel('Time (s)'); ylabel(`Excitation (N)');
subplot(212);
plot(t,x*1000,'k'); axis([0 T -25 25]);
xlabel('Time (s)'); ylabel('Response to Single Cycle of a Square
Wave (mm)');
```

G.2 MODAL SOLUTION FOR AN MDoF SYSTEM

In Chapter 2, the modal characteristics of a two DoF system were considered and the resulting natural frequencies, modal matrix (and hence mode shapes) and modal quantities were presented. In this section, a program is presented for performing these matrix operations and for generating FRF plots. Clearly, other cases such as those explored in Chapters 3 and 4 may be considered by modifying the code.

The program *Pgm_G2_Modal_Solution* is shown below. The theory in Chapter 2 will be required to follow the program.

```
% Determination of Modal Parameters and FRFs for a 2 DoF system
clear all; close all; nmodes = 2;

% System (2 by 2) mass, stiffness and damping matrices
M = [2 0; 0 1]; K = [3000 -1000; -1000 1000]; C = [6 -2; -2 2];
```

```
% Eigenvalue solution
[vec,val] = eig(M\K);

% Sort eigenvalues / vectors into ascending order of frequency
% and find natural frequencies and mode shapes (psi is modal matrix)
for j = 1:nmodes;
  [max_vec, max_index] = max(abs(vec(:,j))); vec(:,j) = vec(:,j)./
  vec(max_index,j);
  f_nat(j) = sqrt(val(j,j)) / 2 / pi;
end
[f_nat, index] = sort(f_nat);
for j = 1:nmodes;
  psi(:,j) = vec(:,index(j));
end

% Modal matrices
MP = psi' * M * psi; CP = psi' * C * psi; KP = psi' * K * psi;

% Modal masses and dampings
for j = 1:nmodes
  modal_mass(j) = MP(j,j); modal_damping(j) = CP(j,j) / 2 / sqrt
  (MP(j,j) * KP(j,j));
end

% Write out results
f_nat, psi, MP, CP, KP, modal_mass, modal_damping

% Set up parameters for FRF generation and initialise FRF matrix
f_nyq = 10; nt = 1024; nf = nt/2 + 1; frq = linspace(0,f_nyq,nf);
H = zeros(nmodes,nmodes,nf);

% Calculate FRF for each frequency value from DC up to Nyquist
% and set Nyquist frequency value to be real - note that FRF needs to
% be a 3 dimensional array (response r / excitation e / fre-
quency nf)
for ifq = 1:nf
  w = 2 * pi * frq(ifq);
  H(:,:,ifq) = inv(K - (w^2 * M) + (i * w * C));
end
H(:,:,nf) = real(H(:,:,nf));

% Plot first row of FRF matrix - 'squeeze' function needed to bring
  matrix
% from 3 to 2 dimensions in order to plot - use `log' plot to show
FRF more clearly
subplot(211)
semilogy(frq(1:nf), abs(squeeze(H(1,1,1:nf))),'k');
xlabel('Frequency (Hz)'); ylabel(`Direct FRF 11');
subplot(212)
semilogy(frq(1:nf), abs(squeeze(H(1,2,1:nf))),'k');
xlabel('Frequency (Hz)'); ylabel(`Transfer FRF 12');
```

G.3 FINITE ELEMENT SOLUTION

In Chapter 4, the finite element method was introduced and the effect of increasing the number of elements for a uniform cantilever beam was considered. In this section, the generation and assembly of the element stiffness and mass matrices and the calculation of natural frequencies is carried out for a range of numbers of elements and also for both consistent and lumped mass representations. Clearly, other cases may be considered by modifying the code.

The program *Pgm_G3_FE_Solution* is shown below. The theory in Chapter 4 will be required to follow the program.

```
% FE model of a 'beam' with increasing number of elements
clear all; close all; nelement = 10;

% Loop around increasing numbers of elements
for jelement = 1:nelement
   nw = 4 + (jelement - 1) * 2;

   % Initialise matrices
   kw = zeros(nw); %Overall beam stiffness matrix - unconstrained
   mw = zeros(nw); %Overall beam mass matrix - unconstrained

   % Parameters for beam and element
   s = 10; L = s / jelement; E = 70e9; I = 2e-4; A = 0.04; rho = 2500;
   b = rho * A * L;

   % Element stiffness data and matrix
   k1 = 12 * E * I / L^3; k2 = 6 * E * I / L^2; k3 = 2 * E * I / L;
   k4 = 4 * E * I / L;
   k = [k1 k2 -k1 k2; k2 k4 -k2 k3; -k1 -k2 k1 -k2; k2 k3 -k2 k4];

   % Mass data
   m1 = 156 * b / 420; m2 = 22 * b * L / 420; m3 = 54 * b / 420; m4 =
   13 * b * L / 420;
   m5 = 4 * b * L^2 / 420; m6 = 3 * b * L^2 / 420; m7 = b / 2; m8 =
   b * L^2 / 24;

   % Element consistent mass matrix (comment out lumped mass matrix)
   m = [m1 m2 m3 -m4; m2 m5 m4 -m6; m3 m4 m1 -m2; -m4 -m6 -m2 m5];

   % Element lumped mass matrix (comment out consistent mass matrix)
   % m = diag([m7 m8 m7 m8]);

   % Beam overall stiffness matrix
   kw(1:4,1:4) = k(1:4,1:4);
   if jelement > 1
     for i = 2:jelement
      j = 2 * i - 1; jj = j + 3;
      kw(j:jj, j:jj) = kw(j:jj, j:jj) + k(1:4,1:4);
     end
   end

   % Beam overall mass matrix
   mw(1:4,1:4) = m(1:4,1:4);
```

```
   if jelement > 1
     for i = 2:jelement
      j = 2 * i - 1; jj = j + 3;
      mw(j:jj, j:jj) = mw(j:jj, j:jj) + m(1:4,1:4);
     end
   end

   % Select structure stiffness / mass matrices to account for the
   fixed end

   % Origin at tip (comment out origin at root)
   % nwf = nw - 2; kwf = kw(1:nwf,1:nwf); mwf = mw(1:nwf,1:nwf);

   % Origin at root (comment out origin at tip)
   kwf = kw(3:nw,3:nw); mwf = mw(3:nw,3:nw);

   % Solve for eigenvalues
   [r,la] = eig(kwf,mwf); [las,n] = sort(diag(la)); fn = sqrt(las)/2/
   pi;

   % disp('Natural frequencies and no of elements');
   % disp(sprintf('%.0f n',jelement)); disp(sprintf('%.3f n',fn));
   [nfn,dummy] = size(fn);
   for jf = 1:nfn
      if jf < 4
       fstore(jf,jelement) = fn(jf);
      end
    end
end
fstore(3,1) = NaN; fstore
element = [1 2 3 4 5 6 7 8 9 10];
hold on; axis([0 10 0 50]);
xlabel('Number of Elements'); ylabel(`Natural Frequencies (Hz)')
plot(element,fstore(1,:),'kx-')
plot(element,fstore(2,:),'kx:')
plot(element,fstore(3,:),'kx-.')
legend('Mode 1','Mode 2','Mode 3')

% Exact natural frequencies (ref Blevins)
lambdaL = [1.8751, 4.69409, 7.85476];
for j = 1:3
  fexact(j) = (lambdaL(j))^2 / 2 /pi * sqrt(E * I / rho / A / s^4);
end
disp('Exact natural frequency'); disp(sprintf('%.3f n',fexact));

% Add exact values to plot
x = [0 10]; y1 = [fexact(1) fexact(1)]; y2 = [fexact(2) fexact(2)];
y3 = [fexact(3) fexact(3)]; line(x,y1); line(x,y2); line(x,y3);

% Note - figure edited to give exact frequency a grey dashed line at
  0.25
% font and the data lines at 1.5 font
```