```
% Problem 2 code and results

clear all
clc
Emu  = 29*10^6; % lb/in^2
Esig = 2*10^6; % lb/in^2

Pmu = 50000; % lb
Psig = 10000; % lb

Wmu = 1000/12; % lb/in
Wsig = 100/12; % lb/in

L = 360; % in
I = 1330; % in^4

X_i = []; % array to store x values
U_i = []; % array to store u values
B_i = []; % array to store beta values
E_i = []; % array to store error values
A_i = []; % array to store alpaha values

% First need to define the limit state function. We do not want beam
% displacement to be greater than 2 inches. Therefore following limit
% state function shall be used.
% ---------------------------------------------------------------------------
% g(P, E, W) = 2 - [(PL^3)/(48EI) + (5/385)*((W*L^4)/(E*I))] = 0
% ---------------------------------------------------------------------------
% If g(P, E, W)<0 then we have failure

% To determine failure probability 'Pf' need to get beta, which is
% beta = g_mu/g_sig

% For the HL method need to implement an iterative procedure, which will
% hopefully converge to the approximate value of beta.

% For initilal condition use the mean value poins to get beta1. After that
% let the iterative process update the inputs until beta converges. Also in
% the iterative portion of the code we will need to use the 'u' paramaters

% Code below will define initial value of beta
% ===========================================================================
% Define g_mu
g_mu = 2-((Pmu*L^3)/(48*Emu*I) + (5/385)*(((Wmu)*L^4)/(Emu*I)));

% To calculate g_sig requires that the derivative of g(P, E, W) are taken
% with respect to all the random variable (P, E, W). Then need to take the
% magnutude of the derivatives in order to obtain g_sig.

% Get derivatives
syms P1 E1 W1
```

```
g = 2-((P1*L^3)/(48*E1*I) + (5/385)*((W1*L^4)/(E1*I)));

gP = diff(g, P1);
% display('deriv gP')
% eval(subs(gP, E1, Emu))
gE = diff(g, E1);
% display('deriv gE')
% eval(subs(gE, [P1, E1, W1], [Pmu, Emu, Wmu]))
gW = diff(g, W1);
% display('deriv gW')
% eval(subs(gW1, E1, Emu))

% Now get g_sig
g_sig = sqrt((eval(subs(gP, E1, Emu))*Psig)^2 + ...
    (eval(subs(gE, [P1, E1, W1], [Pmu, Emu, Wmu]))*Esig)^2 + ...
    (eval(subs(gW, E1, Emu))*Wsig)^2);

% Now get beta
beta = g_mu/g_sig;

% Now get alphas
% For P
aP = -(eval(subs(gP, E1, Emu))*Psig)/g_sig;
% For E
aE = -(eval(subs(gE, [P1, E1, W1], [Pmu, Emu, Wmu]))*Esig)/g_sig;
% For W
aW = -(eval(subs(gW, E1, Emu))*Wsig)/g_sig;

% Calculate x's using following equation
% x_new = x_mu + beta*x_sig*alpha
X=[Pmu + beta*Psig*aP Emu + beta*Esig*aE Wmu + beta*Wsig*aW];

% Calculate u's next
U = [(X(1) - Pmu)/Psig (X(2) - Emu)/Esig (X(3) - Wmu)/Wsig];

% Store values into arrays
X_i=[X_i;X];
U_i=[U_i;U];
B_i=[B_i;beta];
E_i=[E_i;1];
A_i=[A_i;aP, aE, aW];

i=0;
% Now define while loop
while E_i(end)>0.0001 && i<100
    i=i+1;
    g = 2-((X_i(end,1)*L^3)/(48*X_i(end,2)*I) + ...
        (5/385)*((X_i(end,3)*L^4)/(X_i(end,2)*I)));

    dP = eval(subs(gP, E1, X_i(end,2)));

    dE = eval(subs(gE, [P1, E1, W1], ...
        [X_i(end,1), X_i(end,2), X_i(end,3)]));
```

```
    dW = eval(subs(gW, E1, X_i(end,2)));

    g_sig = sqrt((dP*Psig)^2 + (dE*Esig)^2 + (dW*Wsig)^2);

    % get new beta
    beta = (g - dP*Psig*U_i(end,1) - dE*Esig*U_i(end,2) - ...
        dW*Wsig*U_i(end,3))/g_sig;
    % store beta to vector
    B_i=[B_i;beta];

    % Now get alphas
    % For P
    aP = -(dP*Psig)/g_sig;
    % For E
    aE = -(dE*Esig)/g_sig;
    % For W
    aW = -(dW*Wsig)/g_sig;
    % Store alphas
    A_i=[A_i;aP, aE, aW];

    % Get new x's
    X=[Pmu + beta*Psig*aP Emu + beta*Esig*aE Wmu + beta*Wsig*aW];
    % store new x's
    X_i=[X_i;X];

    % Calculate new u's
    U = [(X_i(end,1) - Pmu)/Psig (X_i(end,2) - ...
        Emu)/Esig (X_i(end,3) - Wmu)/Wsig];
    % store new u's
    U_i=[U_i;U];

    % calculate new error
    e=abs(B_i(end) - B_i(end-1))/abs(B_i(end-1));
    % store error value into array
    E_i=[E_i;e];
end

% Get probability failure
Pf = 1 - cdf('norm', B_i(end), 0 ,1)

% Limit state function value at last iteration
g

%==========================================================================
% Next employ MCS to compare results

% Implement MCS sampling with million samples
N = 1000000;
Nvar = 3;

% Define random vector U(N, Nvar)
U=rand(N,Nvar);

% Generate samples
```

```
XP = Pmu + Psig*icdf('norm', U(:,1), 0, 1);
XE = Emu + Esig*icdf('norm', U(:,2), 0, 1);
XW = Wmu + Wsig*icdf('norm', U(:,3), 0, 1);

% Calculate surrogate response using generated samples XP, XE, XW
disp = 2-((XP*L^3)./(48*XE*I) + (5/385)*(((XW)*L^4)./(XE*I)));

% Calculate failure probability
Pf_MCS = sum(disp<0)/N
```

```
Pf =

    0.1780


g =

   2.0345e-05


Pf_MCS =

    0.1782
```

Published with MATLAB® R2014b