DESIGN OPTIMIZATION PROJECT FOUR REPORT

Wing Spar Design With Load Uncertainty

Unique ID: 8449

1 Executive Summary

Optimization of the annulus spar of the unmanned aerial vehicles was performed to minimize the mass of the spar while ensuring the maximum stress on the beam is lower than the material's ultimate strength by taking account of the uncertainties. Specifically in this project, the loading was uncertain, due to the uncertainties during the cruise. The load uncertainty in the analysis of the wing spar was counted by using the stochastic collocation method. As same as project 2, the Euler-Bernoulli Beam Theory was utilized to model the spar. Then the finite-element analysis was employed to discretize the Euler-Bernoulli Equation. Fmincon was again employed to find the optimal value. The optimal mass was found to be 8.6015 kg while satisfying the constraints. The improvement is 70.7 % over the nominal mass of 29.32 kg.

2 Problem Statement

As been mentioned, the goal of this project is to minimize the mass of the spar while satisfying the constraints. The spar has a length of 7.5 m with the material density of 1600 kg/m^3. Young's modulus of the spar is 70GPa. The highest stress with adding six times standard deviation should not be greater than the ultimate tensile stress, which is 600 MPa in this project. The geometric constraints were on the inner and the outer radius of the spar and the thickness. The thickness would have to be greater than 2.5 mm. The inner radius of the spar could not be smaller than 1 cm. The outer radius should be less than 5 cm.

3 Analysis Method

As the same as project 2, the analysis of the spar depends on the Euler-Bernoulli Beam Theory. In short, the displacement of the beam in the vertical direction was computed by using the Euler-Bernoulli Equation

$$\frac{d^2}{dx^2}\left(EI_{yy}\frac{d^2w}{dx^2}\right) = q(x) \tag{1}$$

where w refers to the vertical displacement, q(x) refers to the force along the spar per length, E is Young's modulus being 70GPa in this project, and I_{yy} is the second-moment of area with respect to the y-axis. The finite-element analysis was also employed to discretize the Euler-Bernoulli Equation in this project. The spar was divided into several nodes during the analysis and the calculation. The vertical displacements and the normal stress at each node along the spar were found.

Different from project 2, the loading has uncertainties, where the probability perturbation can be expressed as equation (2)

$$\delta_f(x,\xi) = \sum_{n=1}^{4}\xi_n \cos\left(\frac{(2n-1)\pi x}{2L}\right) \tag{2}$$

where the random variable ξ_n had a normal distribution $\mathcal{N}(0,\frac{f_{nom}(0)}{10n})$, in which f_{nom} denotes the nominal loading and can be found using equation (3)

$$f_{nom}(x) = \frac{2.5W}{L}(1 - \frac{x}{L}) \quad (3)$$

where W is the half of the aircraft's weight and L is the length of the spar on one side of the aircraft. In total, the loading was modeled by using equation (4).

$$f(x, \xi) = f_{nom}(x) + \delta_f(x, \xi) \quad (4)$$

After the loading was analyzed, with the Euler-Bernoulli Beam Theory and finite-element analysis, the displacement on each node can be calculated. Then the stress with uncertainties was found by using the displacement calculated from the loading with the perturbation. After that, the mean stress and the standard deviation of stress were found by applying the method that will be described in section 4.

4 Optimization Goal and Method

As mentioned above, the optimization goal is to minimize the mass of the spar while satisfying the constraints. Like project 2, for the geometric constraints, the thickness would have to be greater than 2.5 mm. The inner radius of the spar could not be smaller than 1 cm. The outer radius should be less than 5 cm. Unlike project 2, the optimization in project 4 was under uncertainty. The pressure constraint was that the mean plus 6 standard deviations of the stress needed to remain below the ultimate strength of the carbon fiber. This can be expressed by equation (5)

$$E[s(x, \xi)] + 6\sqrt{Var[s(x, \xi)]} \leq s_{yield} \quad (5)$$

where $s(x, \xi)$ is the spanwise stress distribution in the spar.

The mean and the standard deviation of the stress were found using the method called Stochastic Collocation. The mean stress can be found using equation (6)

$$E(f) \approx \frac{1}{\sqrt{\pi}} \sum_{i=1}^{m} w_i f(\sqrt{2}\sigma\xi_i + \mu) \quad (6)$$

and the standard deviation of the stress can be found using equation (7)

$$\sigma = \sqrt{E(f^2) - E(f)^2} \quad (7)$$

where w_i is the weight and ξ_i is the quadrature locations. The number of Gauss-Hermite quadrature points used in this project was 3. The data were generated by using the website called "Keisan Online Calculator"[1]. Formula $\sqrt{2}\sigma\xi_i + \mu$ was used to find the random variable $\xi_n \sim \mathcal{N}(0, \frac{f_{nom}(0)}{10n}$. Then ξ_n was used to find the probability perturbation δ_f. After that, the force and

[1] Keisan Online Calculator, https://keisan.casio.com/exec/system/1281195844

the displacement were found. However, since the number of the random variables selected was four, the equation to get the mean value of the stress needed to be written as the following

$$E(f) \approx \left(\frac{1}{\sqrt{\pi}}\right)^4 \sum_{k_1=1}^{m} w^{k_1} \sum_{k_2=1}^{m} w^{k_2} \sum_{k_3=1}^{m} w^{k_3} \sum_{k_4=1}^{m} w^{k_4} f(\sqrt{2}\sigma_1\xi^{k_1} + \mu_1, \sqrt{2}\sigma_2\xi^{k_2} + \mu_2, \sqrt{2}\sigma_3\xi^{k_3} + \mu_3, \sqrt{2}\sigma_4\xi^{k_4} + \mu_4) \quad (8)$$

The above equation can be written in MATLAB code as shown in the Appendix. After all, the mean stress and the standard deviation of stress were found for the stress constraint.

Similar to project 2, the constraints were the nonlinear inequality constraints that were put into fmincon to solve the optimization problem. In short, the complex method was applied to find the Jacobian of nonlinear inequality constraints and the gradient of the objective function.

5 Results and Discussions

The optimal mass in project 4 is 8.6015 kg with the inner and outer radius as shown in figure (1).

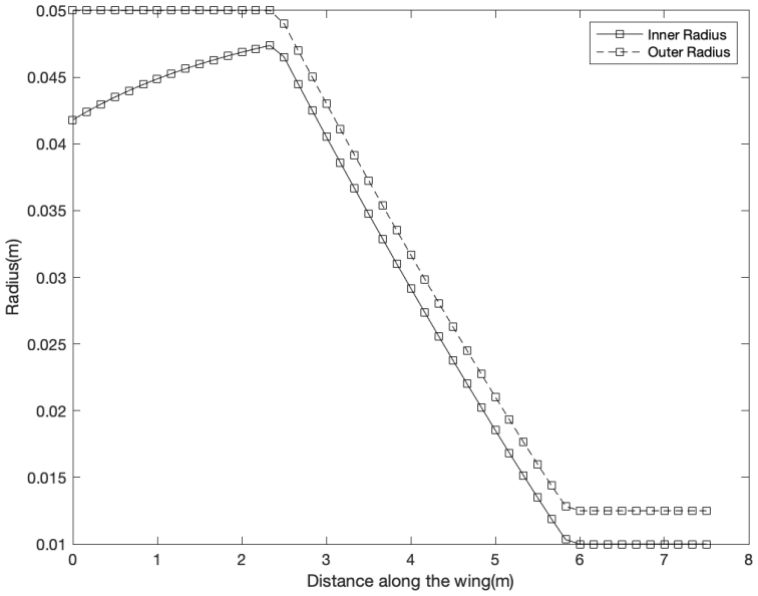


Figure (1) Optimal Spar For $N_{elem} = 45$ with Uncertainties

Project 2 deterministic optimal mass was 4.914 kg with inner and outer radius as shown in figure.

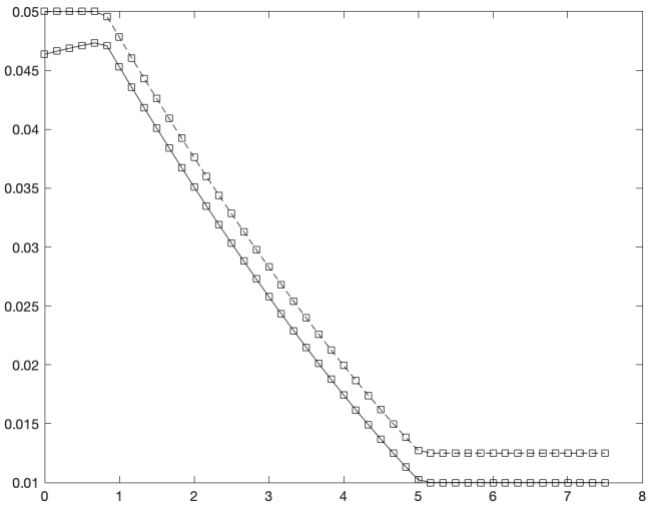


Figure (2) Optimal Spar For $N_{elem} = 45$ without Uncertainties

Both the cases have the same number of elements, 45. However, as shown in figure (1) and figure (2), the thickness of the spar with counting the uncertainties in project 4 is greater than that of in project 2. Due to the uncertainties, the total loading accounted on each node in project 4 is greater than the loading in project 2 so the inner and outer radius has to increase to hold the load and keep the stress (mean stress + 6 * standard deviation stress) under the constraint, which then causes the thickness to be greater than the deterministic optimal thickness in project 2. From figure (1) and figure (2), it was also obvious that the length of the thicker part of the spar from the root is longer in project 4 than that of in project 2. That was also due to the fact that the requirement for the spar to hold the load was higher in project 4 than that of in project 2. In order to maintain the stress (mean stress + 6 * standard deviation stress) below the ultimate stress of the material, the thickness had to be greater than the case that is without uncertainties. The figure of optimal stress distribution of project 2 and project 4 is shown in figure (3) and figure (4).

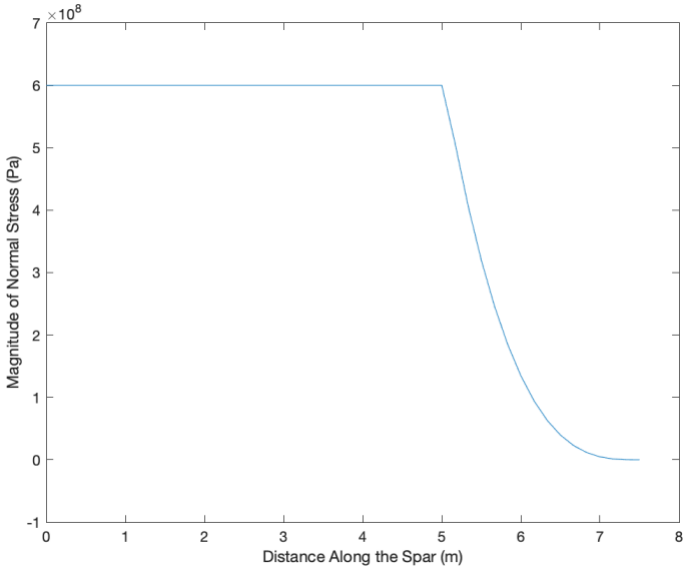


Figure (3) Optimal Stress Distribution Excluding Uncertainties For 45 Elements For Project 2

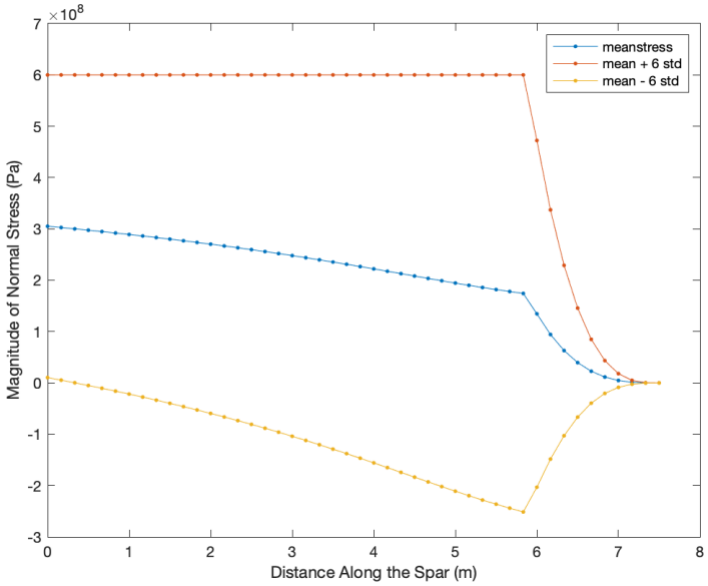


Figure (4) Optimal Stress Distribution Including Uncertainties For 45 Elements For Project 4

In figure (4), the mean stress and the mean stress with ± 6 * standard deviation were shown. The maximum stress along the spar was the mean stress + 6 * standard deviation. In both cases in figure (3) and figure (4), the maximum stress was the ultimate stress of the material, 600 MPa, which means that in both cases, the spar was fully optimized with the mass while satisfying the constraint. So, along each node, the stress was at the ultimate stress that the material can hold with the smallest thickness, meaning the smallest mass. However, the mean stress was almost a half of the maximum stress, which means for the sake of safety, in reality, the spar should be built with a structural strength that should have a mean stress around a half of the maximum stress. The radius should be greater than the ideal case that was without uncertainties otherwise the design is not robust.

When not counting the uncertainties, the maximum stress was ultimate material strength along the spar until 5 meters. While in the case with counting the uncertainties, the maximum stress was ultimate material strength along the spar until 6 meters. The increment of 1 meter along the spar was also due to the reason that was mentioned above that the loading the spar needs to bear is greater when counting the uncertainties.

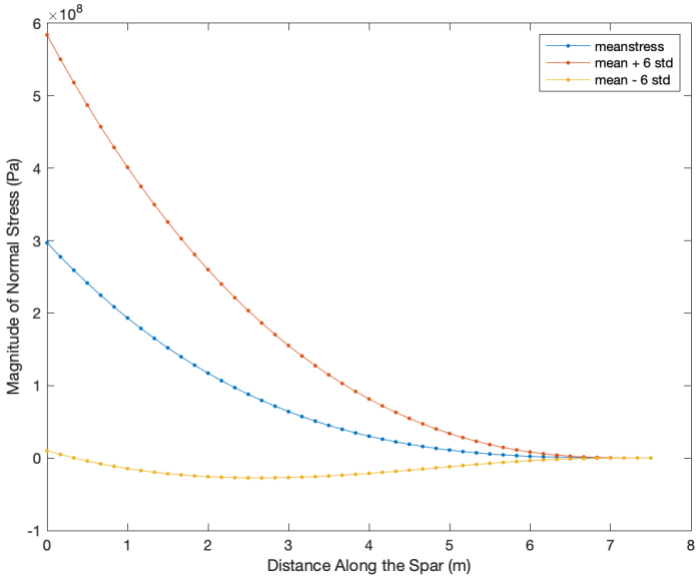


Figure (5) Nominal Stress Distribution Including Uncertainties For 45 Elements

Figure (5) above shows the nominal stress distribution by counting the uncertainties. It was clearly showing that different from the optimal stress distribution as shown in figure (4), the max stress along the spar didn't reach the ultimate material strength, which means the spar can still be optimized and reduce the weight.

Figure (6) shows a convergence history of the optimization process while satisfying all the constraints. The low value of the first-order optimality proves that the local minimum was found and the optimal mass was a good solution to the problem.

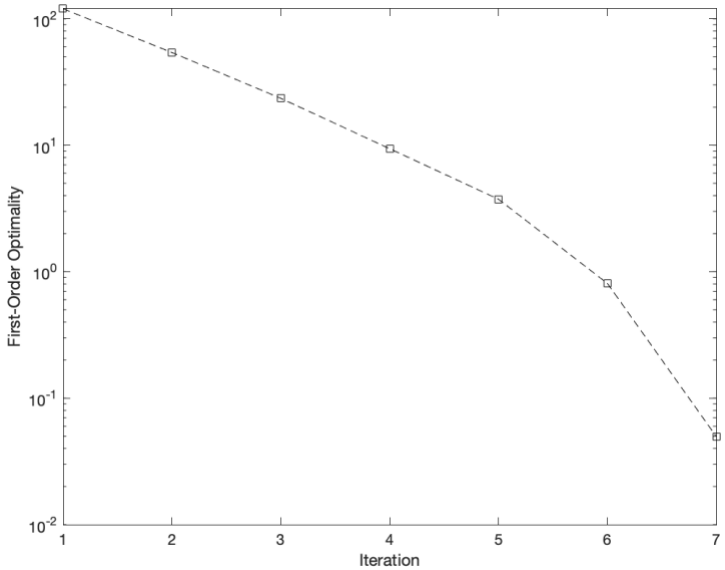


Figure (6) Convergence History For 45 Elements

A study of the optimization history was conducted through figure (7) below. The percentage error between the optimal masses of 8.5925 kg of 25 of the number of elements, and 8.616 kg of 45 of the number of elements was 0.26%, which is below the tolerance value of 1%. For the purpose of

saving the computational cost, 25 was chosen to be the number of elements for the further study of the effect of the different number of quadrature points.

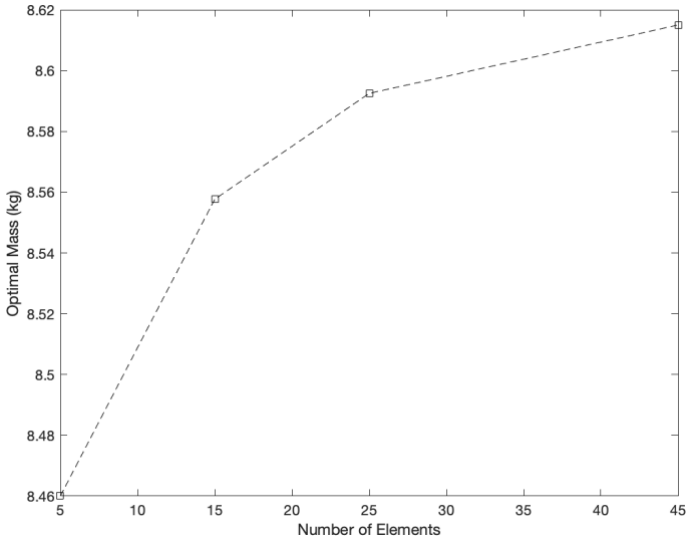
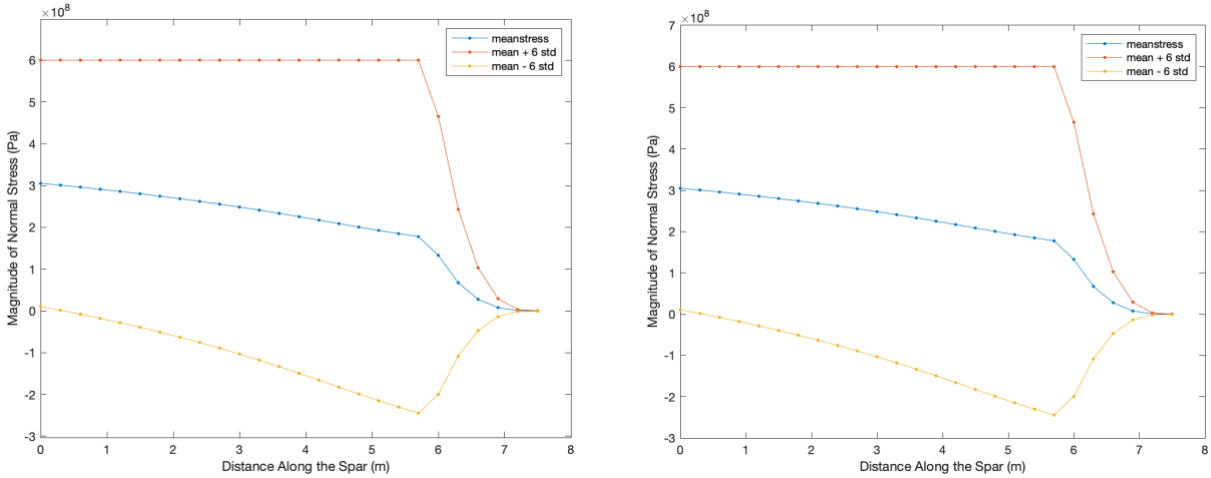


Figure (7) Optimal History For 3 Quadrature Points

All the said solutions above to the optimization problem with uncertainties came from the selection of 3 as the number of the quadrature points. In order to find the most proper amount of quadrature points, the investigation was conducted using the figures as shown below in figure (8).



(a) 3 quadrature points (b) 4 quadrature points

Figure (8) Stress Distribution with 3 and 4 Quadrature Points For 25 Elements

Due to the extremely large computational cost (greater than 30 minutes running time) of using 2 quadrature points, this selection was discarded. Figure 8a showed the stress distribution along the spar with 3 quadrature points when the number of elements was 25 with uncertainties. Similarly, figure 8b showed the stress distribution with 4 quadrature points, respectively. One can easily find

that both cases have similar results. However, by using the "tic toc" function in MATLAB, the running time of the case of using 4 quadrature points was found to be 481 seconds, which is about 8 minutes. In comparison, the case of using 3 quadrature points only took 187 seconds, which is about 3 minutes. In this case, due to the computational cost, 3 quadrature points were selected as the final decision of the amount of the quadrature points.

In all, with the selection of 45 finite elements and 3 quadrature points, the optimal mass found was 8.6015 kg, corresponding to 84.38 N while satisfying all of the constraints. The optimal mass had an improvement of 70.7 % over the nominal mass of 29.32 kg.

6 Reference

[1] "Nodes and Weights of Gauss-Hermite Calculator." *High Accuracy Calculation for Life or Science.*, https://keisan.casio.com/exec/system/1281195844.

Appendix:

Following codes were written or modified by myself

Appendix A: opt_spar.m

```
% minimize wing spar weight subject to stress constraints at manuever
clear all;
close all;

tic

Nelem = 45;
L = 7.5; % semi-span in meters
rho = 1600; % density of standard carbon fiber, kg/m^3
yield = 600e6; % tensile strength of standard carbon fiber, Pa
E = 70e9; % Young's modulus, Pa
W = 0.5*500*9.8; % half of the operational weight, N
force = (2*(2.5*W)/(L^2))*[L:-L/Nelem:0].'; % loading at manueuver

% define function and constraints
fun = @(r) SparWeight(r, L, rho, Nelem);
nonlcon = @(r) WingConstraints(r, L, E, yield, Nelem);
lb = 0.01*ones(2*(Nelem+1),1);
up = 0.05*ones(2*(Nelem+1),1);
A = zeros(Nelem+1,2*(Nelem+1));
b = -0.0025*ones(Nelem+1,1);
for k = 1:(Nelem+1)
    A(k,k) = 1.0;
    A(k,Nelem+1+k) = -1.0;
end
```

```
% define initial guess (the nominal spar)
r0 = zeros(2*(Nelem+1),1);
r0(1:Nelem+1) = 0.04625*ones(Nelem+1,1);
r0(Nelem+2:2*(Nelem+1)) = 0.05*ones(Nelem+1,1);

options = optimset('GradObj','on','GradConstr','on', 'TolCon', 1e-4, ...
    'TolX', 1e-8, 'Display','iter', 'Algorithm', 'active-set', ...
    'OutputFcn', 'optimplotfirstorderopt');
[ropt,fval,exitflag,output] = fmincon(fun, r0, A, b, [], [], lb, up, ...
    nonlcon, options);

% plot optimal radii
r_in = ropt(1:Nelem+1);
r_out = ropt(Nelem+2:2*(Nelem+1));
x = [0:L/Nelem:L].';
figure
plot(x, r_in, '-ks');
hold on;
plot(x, r_out, '--ks');
xlabel('Distance along the wing(m)')
ylabel('Radius(m)')
legend('Inner Radius','Outer Radius')

% display weight and stress constraints
[f,~] = fun(ropt)
[c,~,~,~] = nonlcon(ropt)

toc
```

Appendix B: CalcUncertainStress.m

```
function [limit_stress,mean_stress,std_stress] = CalcUncertainStress...
                                                  (r_in, r_out, L, E, Nelem)
% Compute the stresses with uncertainties
% Inputs:
%   r_in - the inner radius of the annular region
%   r_out - the outer radius of the annular region
%   L - length of the beam
%   E - longitudinal elastic modulus
%   Nelem - number of finite elements to use
% Outputs:
%   limit_stress - the mean stress plus 6 times standard deviation stress
%   mean_stress - the mean stress
%   std_stress - the standard deviation stress

%-------------------------------------------------------------------------
W = 250 * 9.81; % half of the total mass
f_nom_zero = 2 * 2.5 * W / L;
rho = 1600; % density of standard carbon fiber, kg/m^3
yield = 600e6; % tensile strength of standard carbon fiber, Pa
x = [0:L/Nelem:L];
```

```
delta_f = @(xi1,xi2,xi3,xi4,x) (xi1 .* cos( pi.*x/(2*L) ) +      ...
                                xi2 .* cos( 3 * pi.*x/(2*L) ) +  ...
                                xi3 .* cos( 5 * pi.*x/(2*L) ) +  ...
                                xi4 .* cos( 7 * pi.*x/(2*L) ) );

mu1 = 0; mu2 = 0; mu3 = 0; mu4 = 0;
sigma1 = f_nom_zero/10; sigma2 = f_nom_zero/20;
sigma3 = f_nom_zero/30; sigma4 = f_nom_zero/40;

%-------------------------------------------------------------------------

% % using a 2 point Gauss - Hermite quadrature
% xq = [-0.707106781186547524400 8; 0.707106781186547524400 8];
% wq = [0.886226925452758013649 1; 0.886226925452758013649];

%-------------------------------------------------------------------------

% % using a 3 point Gauss - Hermite quadrature
xq = [-1.224744871391589049099;0;1.224744871391589049099];
wq = [0.295408975150919337883;1.181635900603677351532;...
      0.295408975150919337883]./sqrt(pi); % weight adjusted

%-------------------------------------------------------------------------

% using a 4 point Gauss - Hermite quadrature
% xq = [-1.650680123885784555883;
%       -0.524647623275290317841;
%       0.524647623275290317841;
%       1.650680123885784555883];
% wq = [0.081312835447245177143;
%       0.804914090005512836506 1;
%       0.804914090005512836506 1;
%       0.081312835447245177143 03]./sqrt(pi); % weight adjusted;

%-------------------------------------------------------------------------

f_nom = (2*(2.5*W)/(L^2))*[L:-L/Nelem:0];

%-------------------------------------------------------------------------

% nominal values
% r0 = zeros(2*(Nelem+1),1);
% r0(1:Nelem+1) = 0.0415*ones(Nelem+1,1);
% r_in = r0(1:Nelem+1);
% r0(Nelem+2:2*(Nelem+1)) = 0.05*ones(Nelem+1,1);
% r_out = r0(Nelem+2:2*(Nelem+1));

%-------------------------------------------------------------------------

% r0 = [r_in;r_out];
% w = SparWeight(r0, L, rho, Nelem);

mean_stress = zeros(1,length(x)).';
```

```
mean_stress_square = zeros(1,length(x)).';
% std_stress = zeros(1,length(x)).';

for i1 = 1:size(xq,1)
    xi1 = sqrt(2)*sigma1*xq(i1) + mu1;

    for i2 = 1:size(xq,1)
        xi2 = sqrt(2)*sigma2*xq(i2) + mu2;

        for i3 = 1:size(xq,1)
            xi3 = sqrt(2)*sigma3*xq(i3) + mu3;

            for i4 = 1:size(xq,1)
                xi4 = sqrt(2)*sigma4*xq(i4) + mu4;

                force = f_nom + delta_f(xi1,xi2,xi3,xi4,x);

                Iyy = CalcSecondMomentAnnulus(r_in, r_out);
                u = CalcBeamDisplacement(L, E, Iyy, force, Nelem);

                mean_stress = mean_stress + wq(i1)*wq(i2)*wq(i3)*wq(i4)...
                              .* CalcBeamStress(L, E, r_out, u, Nelem);

                mean_stress_square = mean_stress_square + ...
                                     wq(i1)*wq(i2)*wq(i3)*wq(i4)...
                                     .* (CalcBeamStress(L, E, r_out, u,...
                                     Nelem)).^2;

            end
        end
    end
end

std_stress = sqrt(mean_stress_square - mean_stress.^2);

limit_stress = mean_stress+std_stress.*6;
% x = [0:L/Nelem:L];
% figure;plot(x,force);
% figure;plot(x,f_nom);

%-------------------------------------------------------------------------

% figure;plot(x,mean_stress,'.-');hold on
% plot(x,mean_stress+std_stress.*12,'.-');
% plot(x,mean_stress-std_stress.*12,'.-');
% legend('meanstress','mean + 6 std','mean - 6 std')

end
```

Appendix C: WingConstraints.m

```
function [c, ceq, dcdx, dceqdx] = WingConstraints(x, L, E, yield, Nelem)
% Computes the nonlinear inequality constraints for the wing-spar problem
% Inputs:
%   x - the DVs; x(1:Nelem+1) inner and x(Nelem+2:2*(Nelem+1) outer radius
%   L - length of the beam
%   E - longitudinal elastic modulus
%   yield - the yield stress for the material
%   Nelem - number of finite elements to use
% Outputs:
%   c, ceq - inequality (stress) and equality (empty) constraints
%   dcdx, dceqdx - Jacobians of c and ceq
%--------------------------------------------------------------------------

% assert( size(force,1) == (Nelem+1) );
assert( size(x,1) == (2*(Nelem+1)) );

c = CalcInequality(x);
ceq = [];
dcdx = zeros(2*(Nelem+1),Nelem+1);
dceqdx = [];
for k = 1:2*(Nelem+1)
    xc = x;
    xc(k) = xc(k) + complex(0.0, 1e-30);
    dcdx(k,:) = imag(CalcInequality(xc))/1e-30;
end

    function cineq = CalcInequality(x)
        % compute the displacements and the stresses
        r_in = x(1:Nelem+1);
        r_out = x(Nelem+2:2*(Nelem+1));
        [sigma,mean_stress,std_stress] = CalcUncertainStress(r_in,r_out,...
                                         L, E, Nelem);
        cineq = sigma./yield - ones(Nelem+1,1);
    end
end
```

Appendix D: Plot.m

```
opt_spar

% plot displacement
Iyyopt = CalcSecondMomentAnnulus(r_in, r_out);
uopt = CalcBeamDisplacement(L, E, Iyyopt, force, Nelem);
u_in = uopt(1:Nelem+1);
u_out = uopt(Nelem+2:2*(Nelem+1));
figure;plot(x,u_in, '-ks');hold on;
plot(x, u_out, '--ks');
xlabel('Distance along the wing(m)')
ylabel('Vertical Displacement of Spar(m)')
legend('Inner Displacement','Outer Displacement')

% plot stress distribution
figure;plot(x,c);
xlabel('Distance along the wing(m)')
ylabel('Normal Stress(MPa)')
```

```
% plot optimization history
NE = [5 15 25 45];
Mass = [8.46 8.5577 8.5925 8.615];
figure;plot(NE,Mass, '--ks');
xlabel('Number of Elements')
ylabel('Mass (kg)')

% plot First-Order Optimality Convergence History
ITR = [1 2 3 4 5 6 7];
FOO = [120 54.1 23.5 9.39 3.74 0.808 0.0498];
figure;semilogy(ITR,FOO, '--ks');
xlabel('Iteration')
ylabel('First-Order Optimality')

% plot the stress statistic for nominal spar
r_in_nominal = 0.0415*ones(Nelem+1,1);
r_out_nominal = 0.05*ones(Nelem+1,1);
[limit_stress,mean_stress,std_stress] = CalcUncertainStress(r_in_nominal...
                                        ,r_out_nominal,L, E, Nelem);
figure;plot(x,mean_stress,'.-');hold on
% plot(x,mean_stress+sqrt(std_stress).*6,'.-');
% plot(x,mean_stress-sqrt(std_stress).*6,'.-');
plot(x,mean_stress+std_stress.*6,'.-');
plot(x,mean_stress-std_stress.*6,'.-');
xlabel('Distance Along the Spar (m)')
ylabel('Magnitude of Normal Stress (Pa)')
legend('meanstress','mean + 6 std','mean - 6 std')

% plot the stress statistic for optimal spar
[limit_stress,mean_stress_opt,std_stress_opt] = CalcUncertainStress(r_in...
                                        ,r_out,L, E, Nelem);
figure;plot(x,mean_stress_opt,'.-');hold on
plot(x,mean_stress_opt+std_stress_opt.*6,'.-');
plot(x,mean_stress_opt-std_stress_opt.*6,'.-');
xlabel('Distance Along the Spar (m)')
ylabel('Magnitude of Normal Stress (Pa)')
legend('meanstress','mean + 6 std','mean - 6 std')
```

Appendix E: opt_spar_without_uncertainty_projec2.m (One thing that is needed to be aware of: All of the functions that was used in this m file was from the codes provided by Professor Hicken. Only this m file used all of the codes provided by Professor Hicken.)

```
% minimize wing spar weight subject to stress constraints at manuever
clear all;
% close all;

% carbon fiber values from http://www.performance-
composites.com/carbonfibre/mechanicalproperties_2.asp
Nelem = 45;
L = 7.5; % semi-span in meters
rho = 1600; % density of standard carbon fiber, kg/m^3
yield = 600e6; % tensile strength of standard carbon fiber, Pa
```

```
E = 70e9; % Young's modulus, Pa
W = 0.5*500*9.8; % half of the operational weight, N
force = (2*(2.5*W)/(L^2))*[L:-L/Nelem:0].'; % loading at manueuver

% define function and constraints
fun = @(r) SparWeight(r, L, rho, Nelem);
nonlcon = @(r) WingConstraints(r, L, E, force, yield, Nelem);
lb = 0.01*ones(2*(Nelem+1),1);
up = 0.05*ones(2*(Nelem+1),1);
A = zeros(Nelem+1,2*(Nelem+1));
b = -0.0025*ones(Nelem+1,1);
for k = 1:(Nelem+1)
    A(k,k) = 1.0;
    A(k,Nelem+1+k) = -1.0;
end

% define initial guess (the nominal spar)
r0 = zeros(2*(Nelem+1),1);
r0(1:Nelem+1) = 0.04625*ones(Nelem+1,1);
r0(Nelem+2:2*(Nelem+1)) = 0.05*ones(Nelem+1,1);

options = optimset('GradObj','on','GradConstr','on', 'TolCon', 1e-4, ...
    'TolX', 1e-8, 'Display','iter', 'Algorithm', 'active-set'); %,
'DerivativeCheck','on');
[ropt,fval,exitflag,output] = fmincon(fun, r0, A, b, [], [], lb, up, ...
    nonlcon, options);

% plot optimal radii
r_in = ropt(1:Nelem+1);
r_out = ropt(Nelem+2:2*(Nelem+1));
x = [0:L/Nelem:L].';
figure
plot(x, r_in, '-ks');
hold on;
plot(x, r_out, '--ks');

% display weight and stress constraints
[f,~] = fun(ropt)
[c,~,~,~] = nonlcon(ropt)
%%
% plot stress
Iyyopt = CalcSecondMomentAnnulus(r_in, r_out);
uopt = CalcBeamDisplacement(L, E, Iyyopt, force, Nelem);
sigmaopt = CalcBeamStress(L, E, r_out, uopt, Nelem);
figure;plot(x,sigmaopt);
xlabel('Distance Along the Spar (m)')
ylabel('Magnitude of Normal Stress (Pa)')
```

Following codes were given

Appendix F – HermitBasis.m(given):

```
function [N] = HermiteBasis(xi, dx)
% Evaluate the cubic Hermitian shape functions at point xi
% Inputs:
%   xi - point at which to evaluate the shape functions
%   dx - element length
% Outputs:
%   N - 4x1 vector containing the shape functions at xi
%--------------------------------------------------------------------------
if (dx <= 0.0)
    error('element length must be strictly positive');
end
if (xi < -1.0) | (xi > 1.0)
    error('shape functions must be evaluated in the interval [-1,1]');
end
N = zeros(size(xi,1),4);
% this can be done more efficiencly by precomputing some of the common
% terms, but this is adequate for an academic code.
N(:,1) = 0.25.*((1 - xi).^2).*(2 + xi);
N(:,2) = 0.125*dx.*((1 - xi).^2).*(1 + xi);
N(:,3) = 0.25*((1 + xi).^2).*(2 - xi);
N(:,4) = -0.125*dx.*((1 + xi).^2).*(1 - xi);
end
```

Appendix G – GaussQuad.m(given):

```
function [Integral] = GaussQuad(func,n)
% Integrates func over [-1,1] using n-point Gauss quadrature
% Inputs:
%   func - function handle of the form [y] = func(x), where y can be array
%   n - number of points to use for quadrature
% Outputs:
%   Integral - result of integrating func numerically
%--------------------------------------------------------------------------
if n == 1
    Integral = 2.0*func(0.0);
elseif n == 2
    Integral = func(-1/sqrt(3)) + func(1/sqrt(3));
elseif n == 3
    Integral = (8/9).*func(0.0) + (5/9).*(func(-sqrt(3/5)) +
func(sqrt(3/5)));
else
    error('GaussQuad is only implemented for n =1,2, or 3');
end
end
```

Appendix H – DhermitBasis.m (given):

```
function [dN] = DHermiteBasis(xi, dx)
% Evaluate the first derivative of the Hermitian functions at point xi
% Inputs:
```

```
%   xi - point at which to evaluate the shape functions
%   dx - element length
% Outputs:
%   dB - 4x1 vector containing the shape functions derivatives at xi
%--------------------------------------------------------------------------
if (dx <= 0.0)
    error('element length must be strictly positive');
end
if (xi < -1.0) | (xi > 1.0)
    error('shape functions must be evaluated in the interval [-1,1]');
end
dN = zeros(size(xi,1),4);
% this can be done more efficiencly by precomputing some of the common
% terms, but this is adequate for an academic code.
dN(:,1) = -0.5.*(1 - xi).*(2 + xi) + 0.25*(1 - xi).^2;
dN(:,2) = -0.25*dx.*(1 - xi).*(1 + xi) + 0.125*dx.*(1 - xi).^2;
dN(:,3) = 0.5*(1 + xi).*(2 - xi) - 0.25.*(1 + xi).^2;
dN(:,4) = -0.25*dx.*(1 + xi).*(1 - xi) + 0.125*dx.*(1 + xi).^2;
dN = dN./dx;
end
```

Appendix I – D2hermitBasis.m (given):

```
function [B] = D2HermiteBasis(xi, dx)
% Evaluate the second derivative of the Hermitian functions at point xi
% Inputs:
%   xi - point at which to evaluate the shape functions
%   dx - element length
% Outputs:
%   B - 4x1 vector containing the shape function derivatives at xi
%--------------------------------------------------------------------------
if (dx <= 0.0)
    error('element length must be strictly positive');
end
if (xi < -1.0) | (xi > 1.0)
    error('shape functions must be evaluated in the interval [-1,1]');
end
B = zeros(size(xi,1),4);
B(:,1) = 6.*xi./dx;
B(:,2) = 3.*xi - 1;
B(:,3) = -6.*xi./dx;
B(:,4) = 3.*xi + 1;
B(:,:) = B(:,:)./dx;
end
```

Appendix J – CalcElemStiff.m (given):

```
function [Aelem] = CalcElemStiff(E, IL, IR, dx)
% Compute the element stiffness matrix using Hermitian cubic shape funcs.
% Inputs:
%   E - longitudinal elastic modulus
%   IL - moment of inertia at left side of element
```

```
%   IR - moment of inertia at right side of element
%   dx - length of the element
% Outputs:
%   Aelem - the 4x4 element stiffness matrix
%-------------------------------------------------------------------------
if (IL <= 0.0) || (IR <= 0.0) || (E <= 0.0) || (dx <= 0.0)
    error('Inputs must all be strictly positive');
end
Aelem = GaussQuad(@Integrad, 2);
return
%=========================================================================

    function [Int] = Integrad(xi)
        % compute the integrand needed for the element stiffness matrix
        B = D2HermiteBasis(xi,dx);
        MI = LinearMomentInertia(xi);
        Int = (0.5*E*dx).*B.'*B.*MI;
    end

    function [MI] = LinearMomentInertia(xi)
        % evaluate the linear moment of inertia at point xi
        MI = (IL*(1-xi) + IR*(1+xi))*0.5;
    end

end
```

Appendix K – CalcElemLoad.m (given):

```
function [belem] = CalcElemLoad(qL, qR, dx)
% Compute the element load vector for Hermitian cubic shape functions
% Inputs:
%   qL - force per unit length at left end of element
%   qR - force per unit length at right end of element
%   dx - element length
% Outputs:
%   belem - the 4x1 element load vector
%-------------------------------------------------------------------------
if (dx <= 0.0)
    error('element length must be strictly positive');
end
belem = GaussQuad(@Integrad, 3);
return
%=========================================================================

    function [Int] = Integrad(xi)
        % compute the integrand needed for the element load vector
        N = HermiteBasis(xi,dx);
        F = LinearForce(xi);
        Int = 0.5*dx.*F.*N.';
    end

    function [F] = LinearForce(xi)
        % evaluate the linear moment of inertia at point xi
```

```
        F = (qL*(1-xi) + qR*(1+xi))*0.5;
    end

end
```

Appendix L – CalcBeamStress.m (given):

```
function [sigma] = CalcBeamStress(L, E, zmax, u, Nelem)
% Computes (tensile) stresses in a beam based on Euler-Bernoulli beam theory
% Inputs:
%   L - length of the beam
%   E - longitudinal elastic modulus
%   zmax - maximum height of the beam at each node
%   u - displacements (vertical and angle) at each node along the beam
%   Nelem - number of finite elements to use
% Outputs:
%   sigma - stress at each node in the beam
%
% Assumes the beam is symmetric about the y axis
%--------------------------------------------------------------------------

% loop over the elements and compute the stresses
sigma = zeros(Nelem+1,1);
dx = L/Nelem;
for i = 1:Nelem
    xi = [-1;1]; % stress is linear for the FEM here
    d2N = D2HermiteBasis(xi, dx);
    sigma(i:i+1) = E*zmax(i:i+1).*(d2N*u((i-1)*2+1:(i+1)*2));
end

end
```

Appendix M – CalcBeamDisplacement.m (given):

```
function [u] = CalcBeamDisplacement(L, E, Iyy, force, Nelem)
% Estimate beam displacements using Euler-Bernoulli beam theory
% Inputs:
%   L - length of the beam
%   E - longitudinal elastic modulus
%   Iyy - moment of inertia with respect to the y axis, as function of x
%   force - force per unit length along the beam axis x
%   Nelem - number of finite elements to use
% Outputs:
%   u - displacements (vertical and angle) at each node along the beam
%
% Uses a cubic Hermitian finite-element basis to solve the Euler-Bernoulli
% beam equations.  The beam is assumed to lie along the x axis, with the
% force applied transversely in the xz plane.
%--------------------------------------------------------------------------
if (L <= 0) || (E <= 0) || (Nelem <= 0)
    error('The inputs L, E, and Nelem must all be positive');
end
```

```
% Some Notes:
% 1) We construct a dense stiffness matrix and then convert it to a sparse
% matrix, because the problem is small.
% 2) There are Nelems and Nelem+1 nodes, but the DOF at the root are fixed;
% therefore, since there are 2 DOF per node, the system matrix is 2*Nelem
% by 2*Nelem.
A = zeros(2*Nelem,2*Nelem);
b = zeros(2*Nelem,1);

% loop over the interior elements
dx = L/Nelem;
for i = 2:Nelem
    Aelem = CalcElemStiff(E, Iyy(i), Iyy(i+1), dx);
    A((i-2)*2+1:i*2,(i-2)*2+1:i*2) = ...
        A((i-2)*2+1:i*2,(i-2)*2+1:i*2) + Aelem;
    belem = CalcElemLoad(force(i), force(i+1), dx);
    b((i-2)*2+1:i*2) = b((i-2)*2+1:i*2) + belem;
end
% handle the root element
Aelem = CalcElemStiff(E, Iyy(1), Iyy(2), dx);
A(1:2,1:2) = A(1:2,1:2) + Aelem(3:4,3:4);
belem = CalcElemLoad(force(1), force(2), dx);
b(1:2) = b(1:2) + belem(3:4);

% solve for the displacements
u = zeros(2*(Nelem+1),1);
Asp = sparse(A);
u(3:2*(Nelem+1)) = Asp\b;

end
```

Appendix N – CalcSecondMomentAnnulus.m

```
function [Iyy] = CalcSecondMomentAnnulus(r_inner, r_outer)
% Computes the second-moment of area for an annular region
% Inputs:
%   r_inner - the inner radius of the annular region
%   r_outer - the outer radius of the annular region
% Outputs:
%   Iyy - the second-moment of area
%-------------------------------------------------------------------------
Iyy = pi.*(r_outer.^4 - r_inner.^4)./4;

% this guards against violations in the thickness constraint
for i = 1:size(Iyy,1)
    if real(Iyy(i)) < 1e-12
        Iyy(i) = 1e-12;
    end
end

end
```

Appendix O – SparWeight.m

```
function [w, dwdx] = SparWeight(x, L, rho, Nelem)
% Estimate the weight of the wing spar
% Inputs:
%   x - the DVs; x(1:Nelem+1) inner and x(Nelem+2:2*(Nelem+1) outer radius
%   L - length of the beam
%   rho - density of the metal alloy being used
% Outputs:
%   w - the weight of the spar
%   dwdx -
%-------------------------------------------------------------------------
assert( size(x,1) == (2*(Nelem+1)) );

w = Weight(x);
dwdx = zeros(2*(Nelem+1),1);
for k = 1:2*(Nelem+1)
    xc = x;
    xc(k) = xc(k) + complex(0.0,1e-30);
    dwdx(k) = imag(Weight(xc))/1e-30;
end

    function f = Weight(x)
        r_in = x(1:Nelem+1);
        r_out = x(Nelem+2:2*(Nelem+1));
        y = r_out.^2 - r_in.^2;
        f = trapz(y)*pi*rho*L/Nelem;
    end
end
```