Numerical Design Optimization

Report 4: UAV Spar Optimization With Uncertainty

Unique ID: 1059

Professor Hicken

1. Executive summary

A spar for a 500 kg UAV needs to be optimized to be as light as possible. This spar will be used on a UAV designed for long-endurance flight. The UAV will go through a 2.5g maneuver and the spar needs to support the extra weight. The operating condition for the aircraft is dynamic and thus the force loading on the wing has some uncertainty. The design of this spar will account for uncertainty.

2. Overview of Project 2

Much of the code used in project 2 is reused here and thus for a more in-depth analysis please read report 2. An overview of methods and assumptions are below.

2.1 Methods and Analysis

The geometry is represented as a series of nodes with a thickness and an inner radius to define them. The outer radius can be simply derived by adding the inner radius and the thickness at each node. When plotted, this represents a cross section of the upper half of the spar.

The stresses and deflection are calculated using Euler-Bernoulli Beam Theory. The spar is treated as a cantilever beam with a triangular loading on it.

To optimize the geometry, an objective function and constraints need to be defined. Boundary constraints for the maximum inner and outer radius, a linear inequality to ensure the thickness remains viable, and a nonlinear inequality constraint to ensure that the stress remains under yield stress are implemented. The objective is to minimize the mass, so the objective function returns the mass. To improve the optimization gradients are supplied using complex step methods rather than the default forward step. The optimization engine used is “fmincon” found in MatLab’s Optimization Toolbox.

2.2 Overview of Assumptions

It is assumed the cross section varies smoothly from node to node; a normality assumption is made, which means plan sections that are normal to longitudinal plane before bending remain normal after bending; a linearization assumption means that we can ignore nonlinear effects due to deformations; the material is assumed to be elastic and isotropic; finally an assumption that strain energy accounts for only bending moment deformations is made. Some of these assumptions may not be valid, in particular the linearization, material and strain energy

assumptions may have problems. As shown in the report 2, the deformations are not always "small" which may create problems with the linearization assumption. The material used is carbon fiber, depending on how carbon fiber is manufactured it may not be isotropic, most carbon fiber composites fall into a "Quasi- isotropic" category, this means the strength and stiffness is equal only in plane (DragonPlate, 2020). Strain energy can also be created by torsion, and if the twisting of the wing is not accounted for in analysis, the spar may fail (Wing Twisting 2019). As more experiments are done the model may be updated to account for the additional stresses.

3. Uncertainty

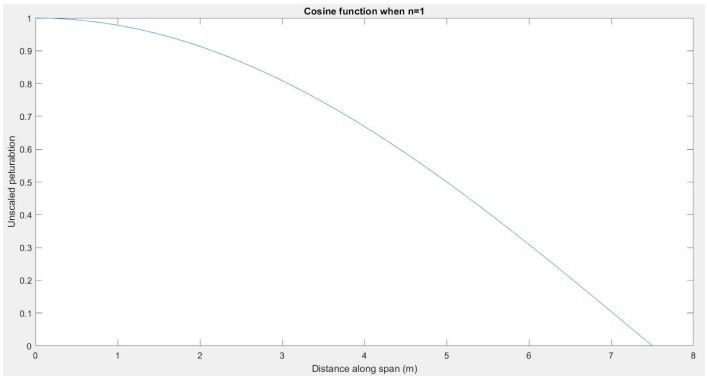
The uncertainty is accounted for by first adding a perturbation to the nominal force loading, of the form shown in *Equation 3.1* and *3.2*. As n increases, the frequency of the cosine function increases and the standard deviation of the random variable ξ_n decreases. This closely matches reality as high frequency disturbances have been shown to have lower amplitudes. A graphical representation of each disturbance as well as of an example of a new force curve is shown in Figure 3.1 and 3.2.

$$\xi_n \sim \mathcal{N}\left(0, \frac{f_{\text{nom}}(0)}{10n}\right)$$

Equation 3.1: Mean and standard deviation of the random variables

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

Equation 3.2: Cosine series



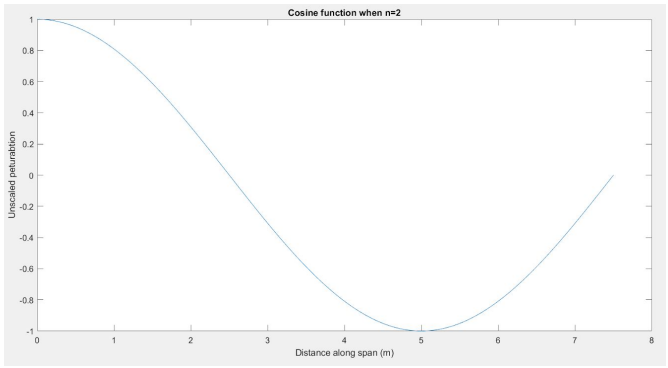
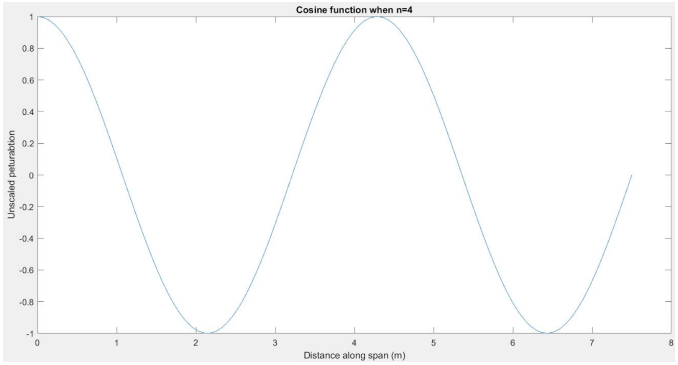


Figure 3.1 and 3.2: The Cosine Function when n =1 and n=2



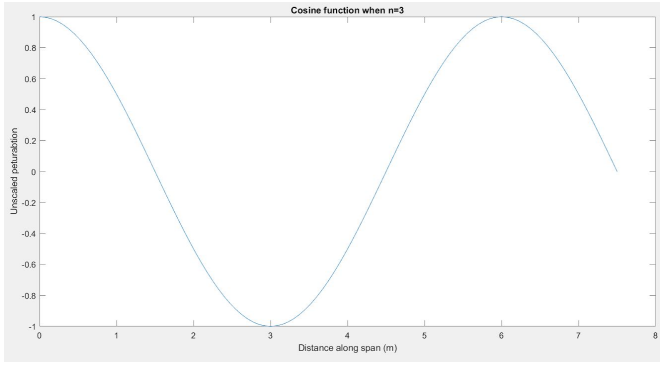


Figure 3.3 and 3.4: The Cosine Function when n =3 and n=4

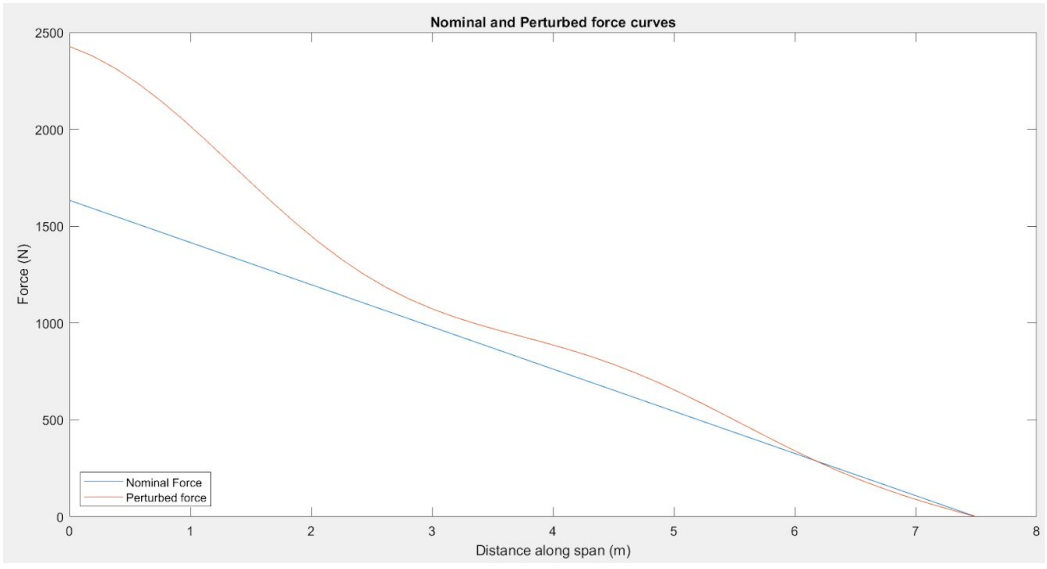


Figure 3.5: The nominal force and an example of perturbed loading

Next the standard deviation of the stress is calculated at each node using *Equation 3.3.* To minimize the risk of failure, the nonlinear constraint is modified to ensure the stress in the spar remains below the mean plus six standard deviations of the stress, shown in *Equation 3.4*.

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

Equation 3.3: Form of Standard Deviation Calculations

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

Equaison 3.4: Form of Inequality Constraint

The implementation of this method is done with multidimensional stochastic collocation using Gauss- Hermite quadrature points. Stochastic collocation samples the function and multiple predetermined points that are scaled by the mean and the standard deviation and then multiplies each function call by a corresponding weight, also predetermined. All the products are then added together to find the mean. A general equation for 3 quadrature points, to change the number of points, changes the maximum value of k, in one dimension is shown in *Equation 3.5.*

$$\mu_f(x) \approx \frac{1}{\sqrt{\pi}} \sum_{k=1}^{3} w^{(k)} f\left(\sqrt{2}\sigma\xi^{(k)} + x\right)$$

Equation 3.5: Stochastic collocation

In this case, there are four random variables, so to modify the one dimensional case, four different quadrature loops will have to be run to sample the function in 4 to the power k locations essentially running through all the points one variable at a time. The general form of multidimensional Gauss-Hermite Quadrature is shown in *Equation 3.6* and the unscaled points and weights for Gauss- Hermite quadrature are shown in *Table 3.1.*

$$E_f(x) = \left(\prod_{i=1}^{p} \frac{1}{\sqrt{\pi}}\right) \sum_{k1=1}^{m} \sum_{k2=1}^{m} \cdots \sum_{kp=1}^{m} \left(\prod_{i=1}^{p} w^{(k_i)}\right) * f(\sqrt{2} * \sigma_1 \xi^{(k_1)} + \mu_1 + \sqrt{2} * \sigma_2 \xi^{(k_2)} + \mu_2 \ldots \sqrt{2} * \sigma_p \xi^{(k_p)} + \mu_p)$$

Equation 3.6: Multidimensional Gauss-Hermite Quadrature

Table 3.1: Unscaled Gauss-Hermite Quadrature points

n	x_i	w_i
2	$\pm\frac{1}{2}\sqrt{2}$	$\frac{1}{2}\sqrt{\pi}$
3	0	$\frac{2}{3}\sqrt{\pi}$
	$\pm\frac{1}{2}\sqrt{6}$	$\frac{1}{6}\sqrt{\pi}$
4	$\pm\sqrt{\frac{3-\sqrt{6}}{2}}$	$\frac{\sqrt{\pi}}{4\left(3-\sqrt{6}\right)}$
	$\pm\sqrt{\frac{3+\sqrt{6}}{2}}$	$\frac{\sqrt{\pi}}{4\left(3+\sqrt{6}\right)}$

To ensure the code is working properly, a test comparing the mean ±6 standard deviations of the stress along the spar for the nominal design of r_{in}=0.0415m and r_{out}=0.05m is conducted using 15 elements and 2 quadrature points. Results shown in *Figures 3.6 and 3.7,* they are sufficiently similar.

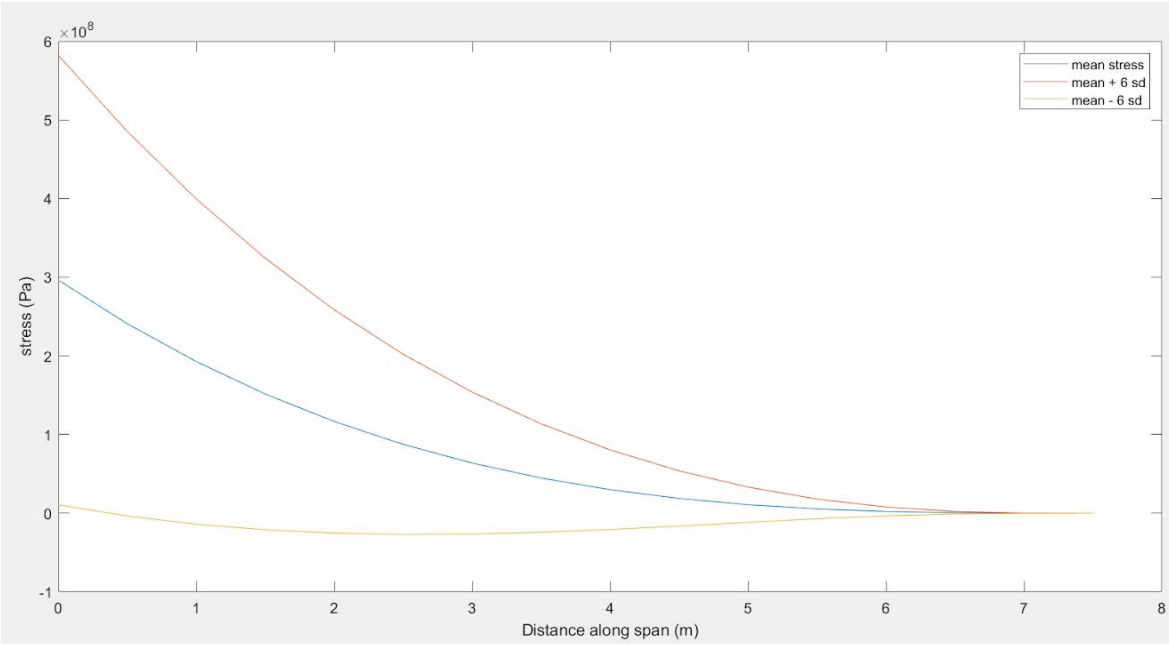


Figure 3.6: Mean ±6 standard deviations as done by author

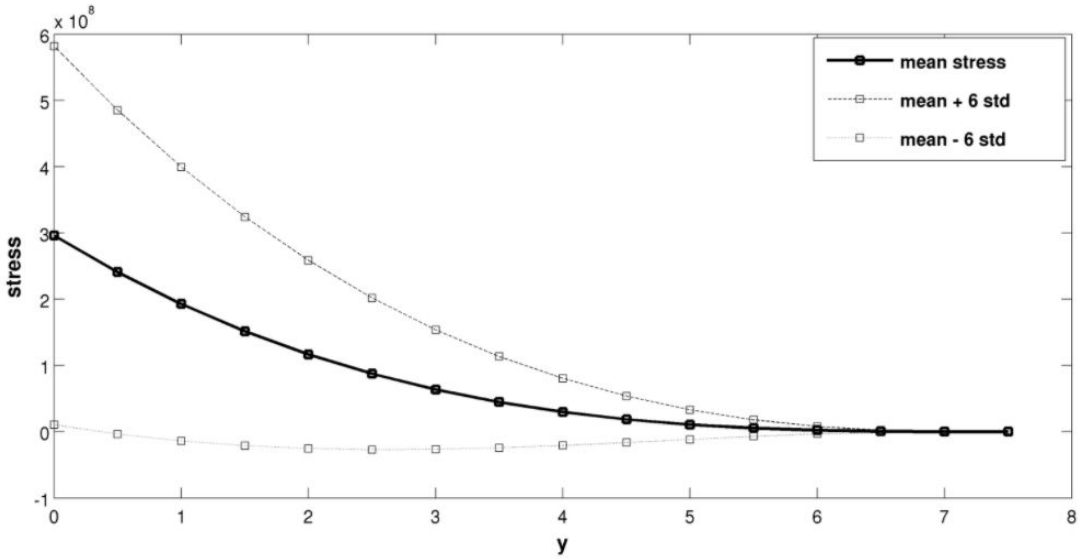


Figure 3.7: Nominal mean ±6 standard deviations

4. Convergence

A convergence study is done, to find the ideal number of quadrature points. As the number of quadrature points increases so does the accuracy of the standard deviation, however the computational cost and thus time increases with it as well. To show the effect of quadrature points on accuracy and cost the standard deviation at the root of the spar and the computational time are compared to the number of points, one quadrature point was not considered as there would be no standard deviation and thus a 100% error. The results are shown in *Figures 4.1 and 4.2.*

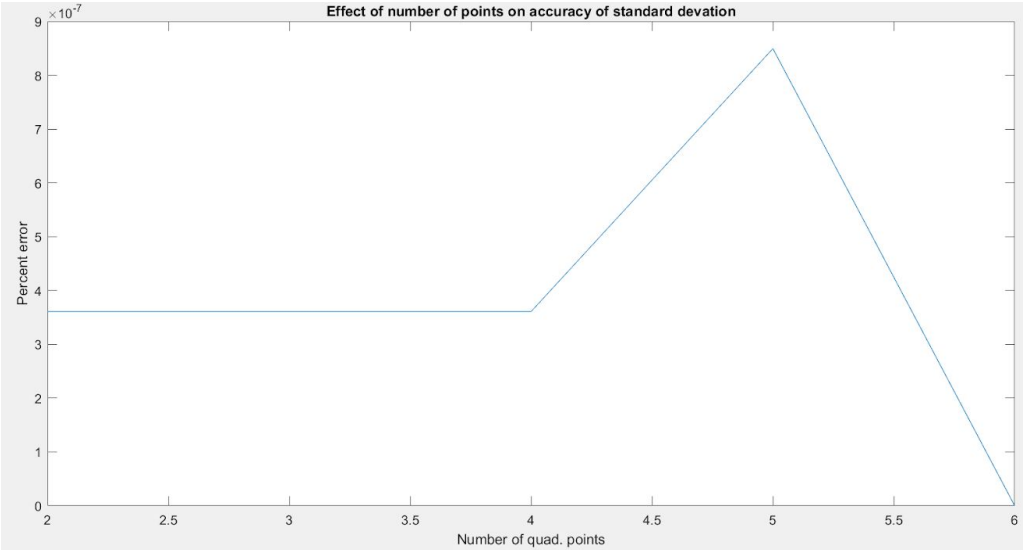


Figure 4.1: Effect of number of points on accuracy of standard deviation

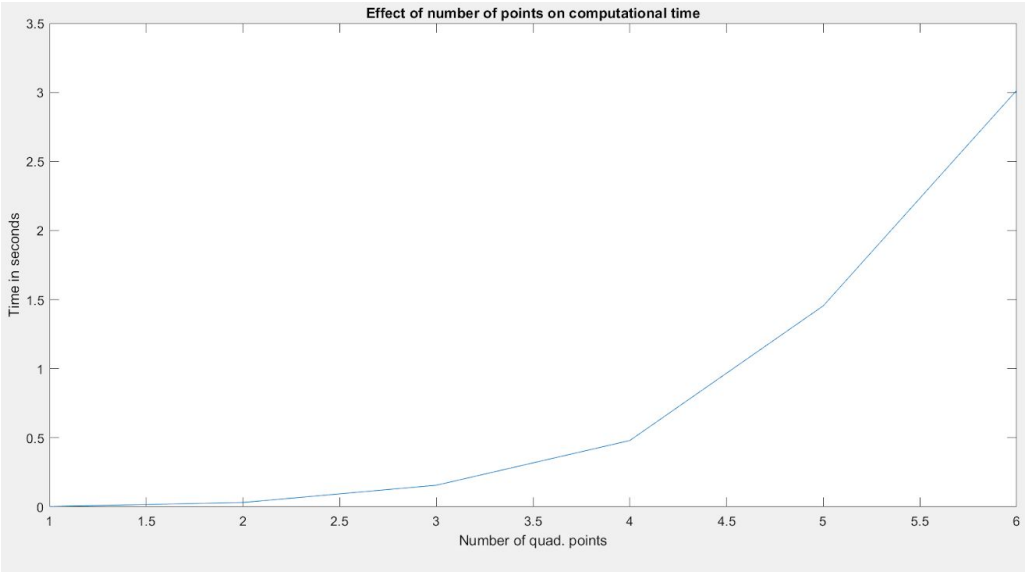


Figure 4.2: Effect of number of points on computational time

From these studies, the optimal number of points is 2 as the percent error versus 6 points is on the order of 10^{-7} and the computational time is very good, 0.0319 seconds. All convergence code is in *Appendix B.6: convergence.*

5. Optimization Algorithm

The optimization method used is fmincon in MatLab's Optimization Toolbox. This method is used because it can handle constrained optimization problems like the one presented. Within fmincon the algorithms considered were the default, "sqp" and "active-set". The algorithm chosen is "active-set" because it more reliably converged to a clean solution, regardless of initial condition and did so much faster than the default. The initial guess is the nominal design.

6. Results

The code described above is run with 35 elements, the optimized mass is 8.5918 kg. The optimized geometry is shown in *Figure 4.1*, for reference the optimized geometry without uncertainty is shown in *Figure 4.2,* the stress distribution with uncertainty along the spar is shown in *Figure 4.3*, the deflection of the spar at the nominal force loading is shown in *Figure 4.4,* and the first order optimality is shown in *Figure 4.5* to show to the result reached is a local minimum and is the best design for this application.

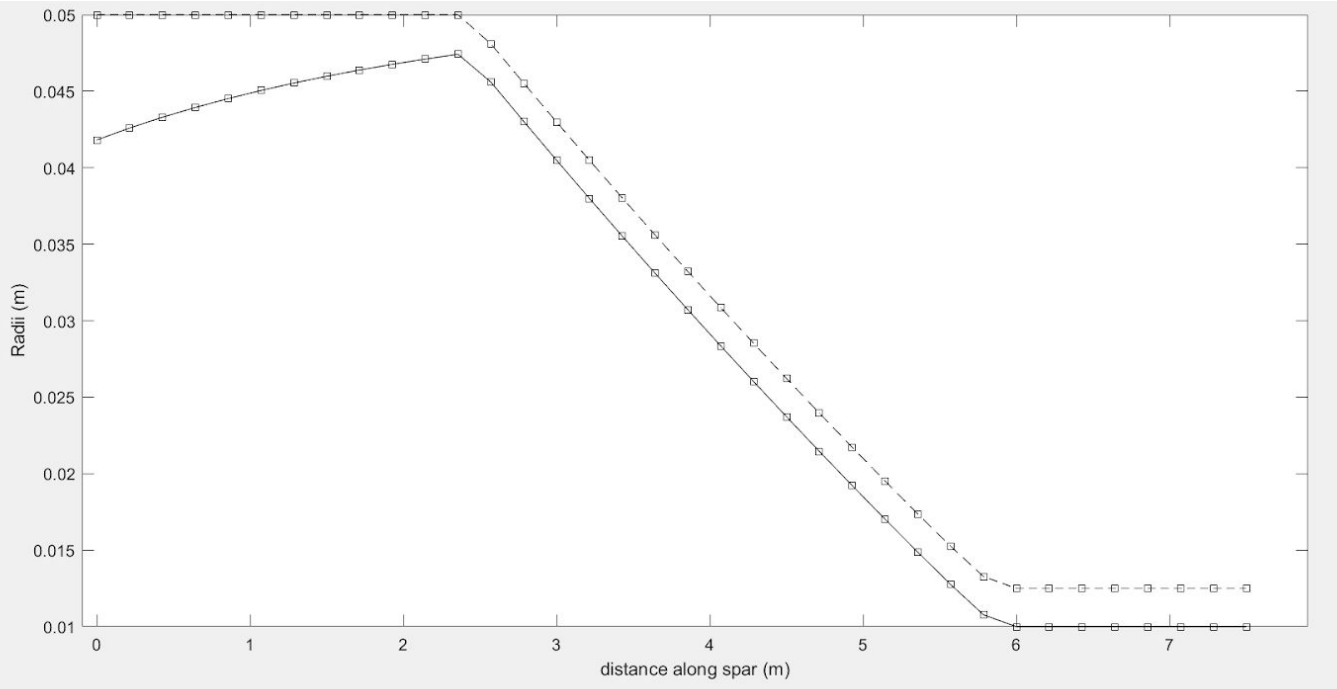


Figure 6.1: Representation of Geometry with Inner and Outer Radii

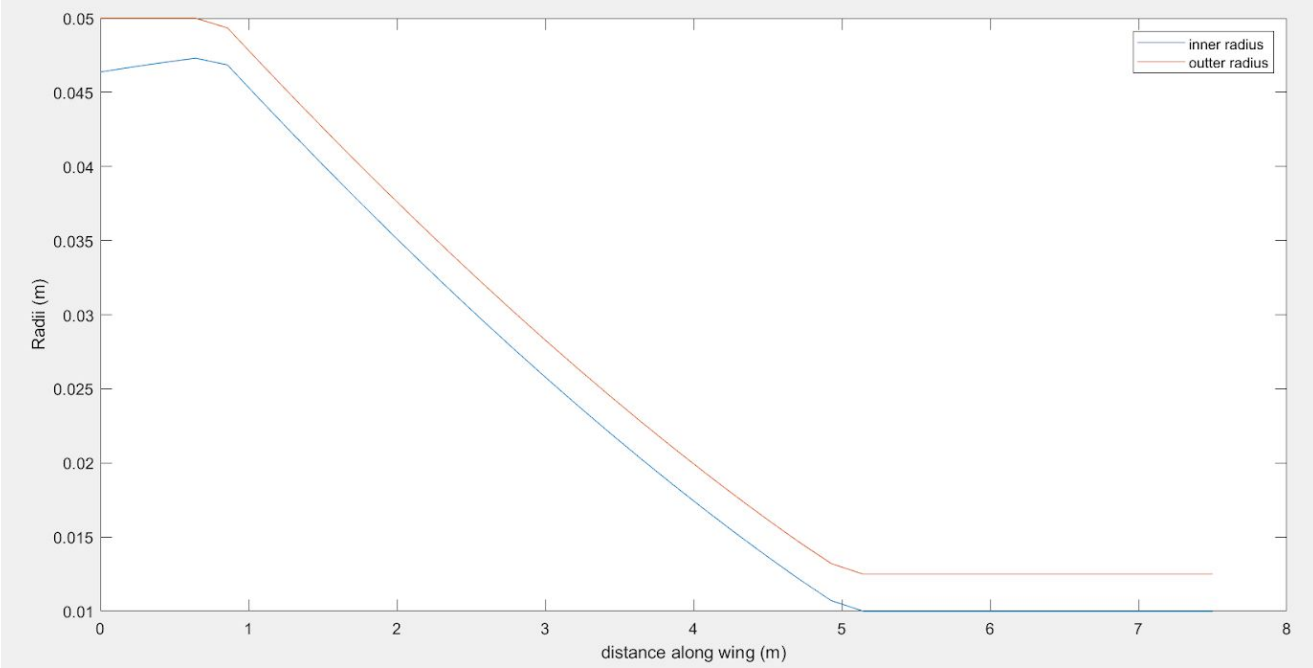


Figure 6.2: Original Geometry without uncertainty

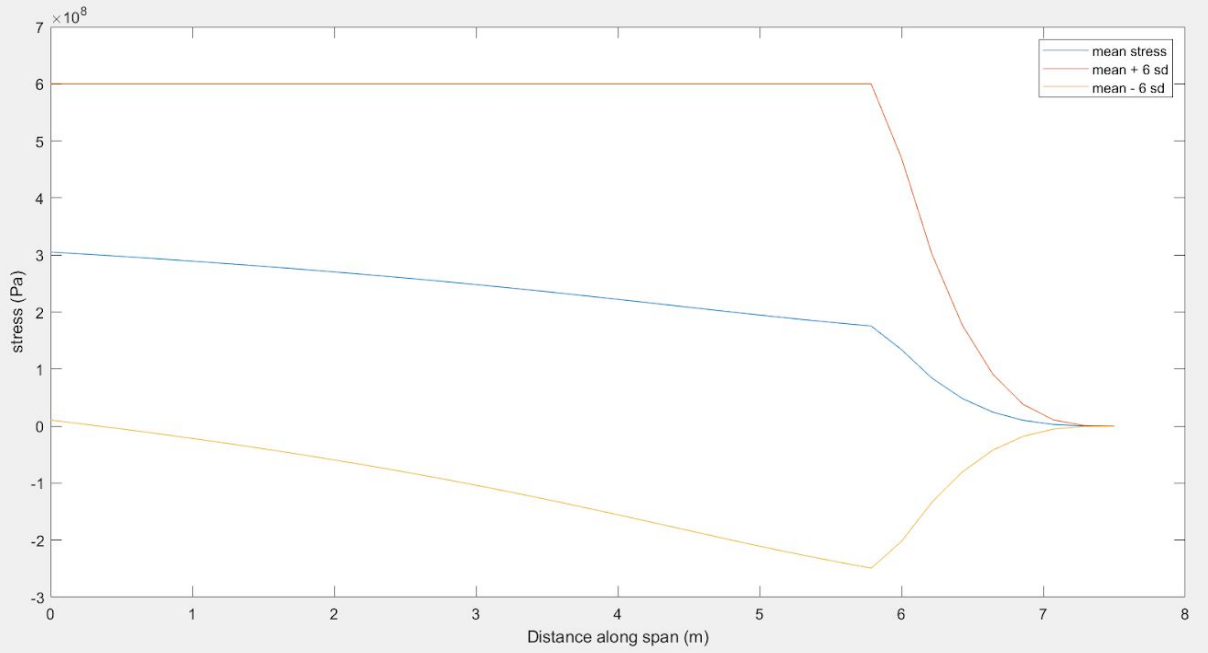


Figure 6.3: Stress Distribution Along the Length of the Spar, with uncertainty

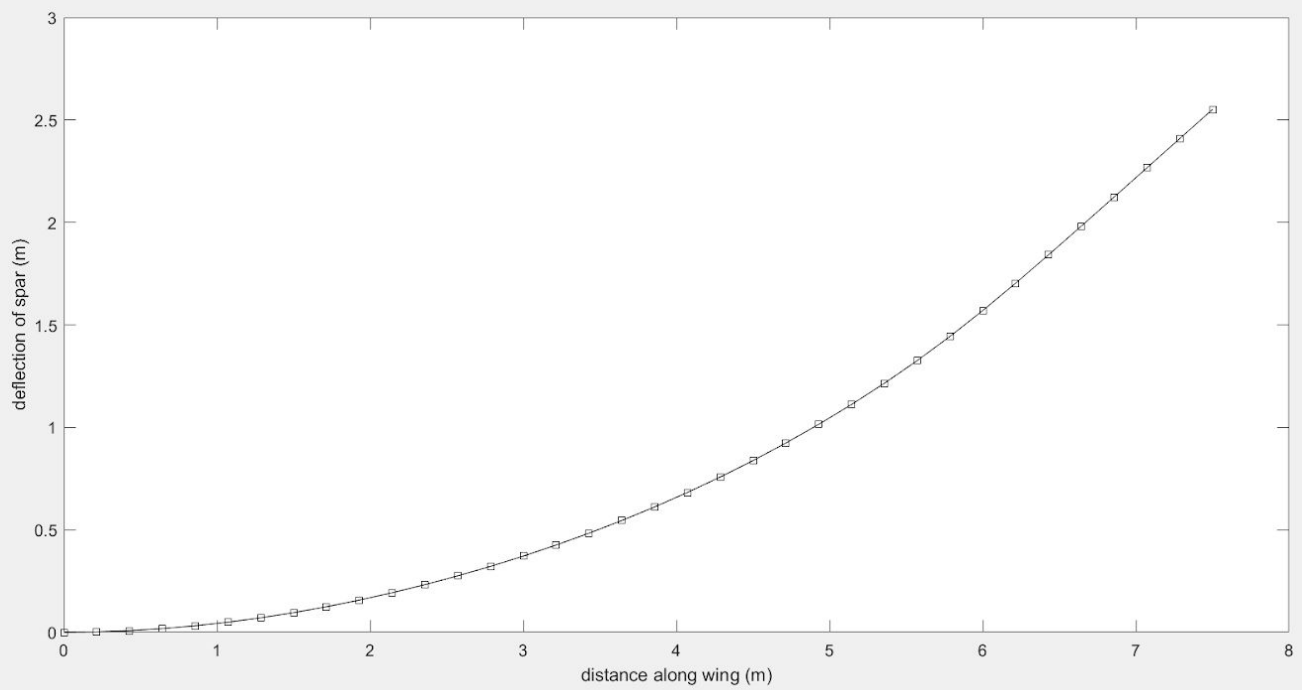


Figure 6.4: Deflection of Spar due to Bending Moment

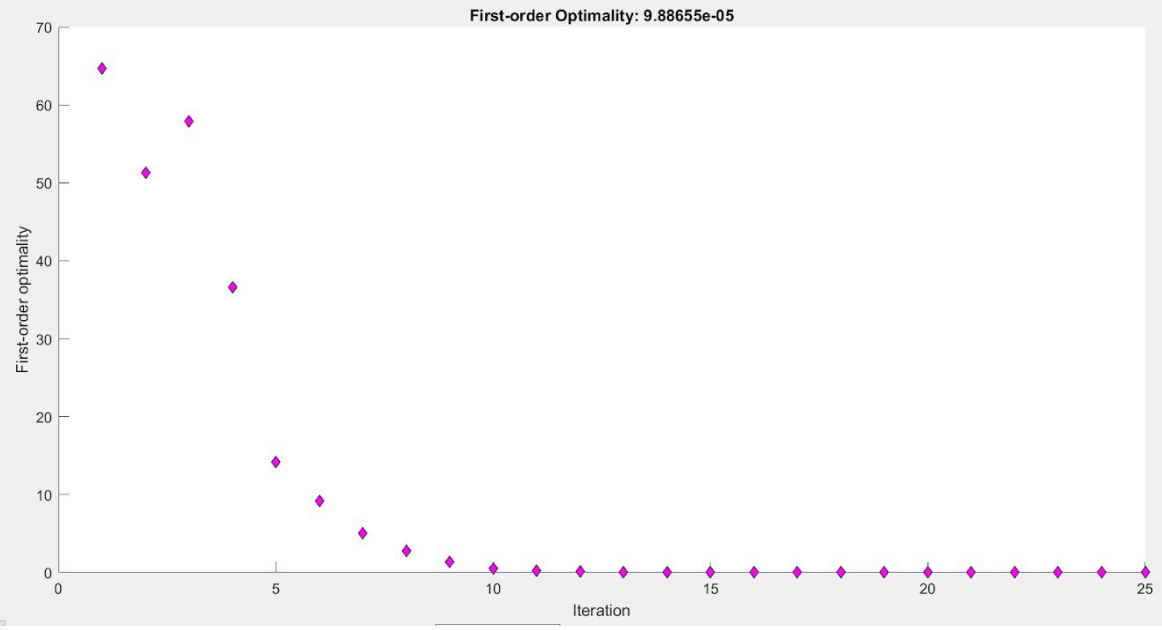


Figure 6.5: Plot of First Order Optimality Vs Iterations

7. Conclusions and Discussion of the Results

The optimized mass of 8.5918 kg is a 70.7% decrease over the nominal mass of 29.32 kg; this is up from 4.9168 kg, the optimized mass without uncertainty.

The optimal stress distribution curve would be one where the stress is held constant at 600 MPa through the length of the spar. However due to the constraints put in place to ensure the final design is manufacturable, the stress along the beam is held constant at 600 MPa until it is not feasible anymore, this is the point where it flatlines at the minimum inner radius and thickness. The linear decrease in the middle portion of the spar also makes sense because on average the force distribution linearly decreases along the length of the spar. The deflection curve shows why the linearization assumption may not be valid in this case as the deformation is quite large ~37% of the length of the spar.

Appendix

A. Works Cited

Carbon Fiber 101: What do Isotropic, Quasi-Isotropic, and Anisotropic Mean? (2019, November 25). Retrieved October 21, 2020, from https://dragonplate.com/carbon-fiber-101-what-do-isotropic-quasi-isotropic-and-anisotropic-mean

Fmincon. (2006). Retrieved October 21, 2020, from https://www.mathworks.com/help/optim/ug/fmincon.html

Hicken, J. (2020). Numerical Design Optimization Lecture Slides. Retrieved December 10, 2020, from https://dragonplate.com/carbon-fiber-101-what-do-isotropic-quasi-isotropic-and-anisotropic-mean

The Mystery of Wing Twisting. (2018, July 24). Retrieved October 21, 2020, from https://aerospaceengineeringblog.com/the-mystery-of-wing-twisting/

B. Source code

B.1 Opt_spar: runs the optimization

```
% minimize wing spar weight subject to stress constraints at maneuver
clearvars;
close all;

% carbon fiber values from
http://www.performance-composites.com/carbonfibre/mechanicalproperties_2.asp
Nelem = 35;
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_nom = (2*(2.5*W)/(L^2))*[L:-L/Nelem:0].'; % loading at maneuver
delta1 = 0;
x = [0:L/Nelem:L].';
```

```
% define function and constraints
fun = @(r) SparWeight(r, L, rho, Nelem);
c1 = cos((2-1)* pi * x / (2*L));
c2 = cos((2*2-1)* pi * x / (2*L));
c3 = cos((2*3-1)* pi * x / (2*L));
c4 = cos((2*4-1)* pi * x / (2*L));
c = [c1,c2,c3,c4];
n = 2;
nonlcon = @(r) WingConstraints(r, L, E, force_nom, yield, Nelem, c,n);
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) = 0.0415*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', 'sqp',... "PlotFcn","optimplotfirstorderopt");
%, '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));
figure
plot(x, r_in, '-ks');
hold on;
plot(x, r_out, '--ks');
ylabel('Radii (m)')
xlabel('distance along spar (m)')
% display weight and stress constraints
[f,~] = fun(ropt);
[c,~,~,~] = nonlcon(ropt);
Iyy = CalcSecondMomentAnnulus(r_in, r_out);
[u] = CalcBeamDisplacement(L, E, Iyy, force_nom, Nelem);
[sigma] = CalcBeamStress(L, E, r_out, u, Nelem);

figure
plot(x,sigma,'ks-')
```

```
xlabel('distance along wing (m)')
ylabel('magnitude of normal stress (Pa)')

figure
plot(x,u(1:2:2*Nelem+1),'ks-')
xlabel('distance along wing (m)')
ylabel('deflection of spar (m)')

figure
[stress,sd] = standard_dev(2, r_in, r_out);
figure
plot(x, stress)
hold on
plot(x, stress + 6*sd);
hold on
plot(x, stress - 6*sd);
hold on
legend({'mean stress','mean + 6 sd','mean - 6 sd'},'Location', 'northeast')
xlabel('Distance along span (m)')
ylabel('stress (Pa)')
```

B.2 standard_dev: calculates the mean stress and standard deviation at a design

```
function  [stress,sd] = standard_dev(n, r_in, r_out)
% n is number of quadrature points

if n == 1
    point = [0];
    w = [sqrt(pi)]/ sqrt(pi);
elseif n == 2
    point = [-0.707106781186547524400 8, 0.707106781186547524400 8];
    w = [0.886226925452758013649 1, 0.886226925452758013649 1]/ sqrt(pi);
elseif n==3
    point = [-1.224744871391589049099,0, 1.224744871391589049099];
    w = [0.295408975150919337883, 1.181635900603677351532,
0.295408975150919337883]/ sqrt(pi);
elseif n == 4
    point = [-sqrt((3-sqrt(6))/2),sqrt((3-sqrt(6))/2), -sqrt((3+sqrt(6))/2),
sqrt((3+sqrt(6))/2)];
    w = 1/sqrt(pi)*[sqrt(pi)/ (4 * (3- sqrt(6))), sqrt(pi)/ (4 * (3- sqrt(6))),...
    sqrt(pi)/ (4 * (3+ sqrt(6))), sqrt(pi)/ (4 * (3 + sqrt(6)))];
elseif n == 5
    point = [-2.02018287, -0.9585724646, 0, 0.9585724646, 2.02018287];
    w = [0.01995324206, 0.3936193232,0.9453087205,0.3936193232, 0.01995324206]/
sqrt(pi);
elseif n == 6
    point = [-2.350604974, -1.335849074, -0.4360774119, 0.4360774119, 1.335849074,
2.350604974];
```

```
    w = [0.00453000991, 0.1570673203, 0.7246295952, 0.7246295952, 0.1570673203,
0.00453000991]/ sqrt(pi);
end


Nelem = 35;
L = 7.5; % semi-span in meters
E = 70e9; % Young's modulus, Pa
W = 0.5*500*9.8; % half of the operational weight, N
x = [0:L/Nelem:L].';
force = (2*(2.5*W)/(L^2))*[L:-L/Nelem:0].'; % nominal loading
% precomputing the cosine functions
c1 = cos((2-1)* pi * x / (2*L));
c2 = cos((2*2-1)* pi * x / (2*L));
c3 = cos((2*3-1)* pi * x / (2*L));
c4 = cos((2*4-1)* pi * x / (2*L));
c = [c1,c2,c3,c4];

% compute the displacements and the stresses

Iyy = CalcSecondMomentAnnulus(r_in, r_out);
stress = zeros(Nelem+1,1);
stress_sq = zeros(Nelem+1,1);
f0 = force(1);
% precomputing the standard deviations
sig1 =  f0 / (10 * 1);
sig2 =  f0 / (10 * 2);
sig3 =  f0 / (10 * 3);
sig4 =  f0 / (10 * 4);
for i1 = 1:n
    p1 = point(i1) *(2)^.5 * sig1; % scale the points with the standard deviation
    % since the mean is 0 it is not added here.
    for i2 = 1:n
        p2 = point(i2) *(2)^.5 * sig2;
        for i3 = 1:n
            p3 = point(i3) *(2)^.5 * sig3;
            for i4 = 1:n
                p4 = point(i4) *(2)^.5 * sig4;
                delta_f = p1* c(:,1) + p2 *c(:,2) + p3 * c(:,3) + p4*c(:,4);
                force_p = force + delta_f; % find the perturbed force
                u = CalcBeamDisplacement(L, E, Iyy, force_p, Nelem);
                s = CalcBeamStress(L, E, r_out, u, Nelem);
                stress = stress + w(i1) * w(i2) * w(i3)* w(i4) * s; % quadrature
for the mean
                stress_sq = stress_sq + w(i1) * w(i2) * w(i3)* w(i4) * s.^2;%
quadrature for the mean squared
                % used in finding the standard deviation
```

```
            end
        end
    end
end
sd = (stress_sq - stress.^2).^.5;
end
```

B.4 WingConstraint: Cineq and Ceq as well as the jacobian

```
function [cineq, ceq, dcdx, dceqdx] = WingConstraints(r, L, E, force, yield, Nelem,
c, n)
% Computes the nonlinear inequality constraints for the wing-spar problem
% Inputs:
%   r - 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
%   force - nominal force per unit length along the beam axis x
%   yield - the yield stress for the material
%   Nelem - number of finite elements to use
%   c: The Cosine functions
%   n: number of quad points to be used
% Outputs:
%   cineq, ceq - inequality (stress) and equality (empty) constraints
%   dcdx, dceqdx - Jacobians of cineq and ceq
%--------------------------------------------------------------------------
assert( size(force,1) == (Nelem+1) );
assert( size(r,1) == (2*(Nelem+1)) );

[cineq] = CalcInequality(r, n);
ceq = [];
dcdx = zeros(2*(Nelem+1),Nelem+1);
dceqdx = [];

for k = 1:2*(Nelem+1)
    xc = r;
    xc(k) = xc(k) + complex(0.0, 1e-30);
    xc = CalcInequality(xc, n);
    dcdx(k,:) = imag(xc)/1e-30;
end

    function [cineq] = CalcInequality(r, n)
        if n == 1
            point = [0];
            w = [sqrt(pi)]/ sqrt(pi);
        elseif n == 2
            point = [-0.707106781186547524400 8, 0.707106781186547524400 8];
            w = [0.886226925452758013649 1, 0.886226925452758013649 1]/ sqrt(pi);
        elseif n==3
```

```
            point = [-1.22474487139158904909,0, 1.22474487139158904909];
            w = [0.29540897515091933788, 1.18163590060367735153,
0.29540897515091933788]/ sqrt(pi);
        elseif n == 4
            point = [-sqrt((3-sqrt(6))/2),sqrt((3-sqrt(6))/2),
-sqrt((3+sqrt(6))/2), sqrt((3+sqrt(6))/2)];
            w = 1/sqrt(pi)*[sqrt(pi)/ (4 * (3- sqrt(6))), sqrt(pi)/ (4 * (3-
sqrt(6))),...
            sqrt(pi)/ (4 * (3+ sqrt(6))), sqrt(pi)/ (4 * (3 + sqrt(6)))];
        end
        % compute the displacements and the stresses
        r_in = r(1:Nelem+1);
        r_out = r(Nelem+2:2*(Nelem+1));
        Iyy = CalcSecondMomentAnnulus(r_in, r_out);
        stress = zeros(Nelem+1,1);
        stress_sq = zeros(Nelem+1,1);
        f0 = force(1); % force at root
        sig1 =  f0 / (10 * 1); % the standard devations of each random variable
        sig2 =  f0 / (10 * 2);
        sig3 =  f0 / (10 * 3);
        sig4 =  f0 / (10 * 4);
        % four for loops for four random variables
        for i1 = 1:n
            p1 = point(i1) *(2)^.5 * sig1; % scale the points with the standared
devation
    % since the mean is 0 it is not added here.
            for i2 = 1:n
                p2 = point(i2) *(2)^.5 * sig2;
                for i3 = 1:n
                    p3 = point(i3) *(2)^.5 * sig3;
                    for i4 = 1:n
                        p4 = point(i4) *(2)^.5 * sig4;
                        delta_f = p1* c(:,1) + p2 *c(:,2) + p3 * c(:,3) + p4 *
c(:,4);
                        force_p = force + delta_f; % find the perturbed force
                        u = CalcBeamDisplacement(L, E, Iyy, force_p, Nelem);
                        s = CalcBeamStress(L, E, r_out, u, Nelem);
                        stress = stress + w(i1) * w(i2) * w(i3)* w(i4) * s; %
quadrature for the mean
                        stress_sq = stress_sq + w(i1) * w(i2) * w(i3)* w(i4) *
s.^2; % quadrature for the mean squared
                                    % used in finding the standard deviation

                    end
                end
            end
        end
```

```
        sd = (abs(stress_sq - stress.^2)).^.5;
        cineq = (stress + 6 * sd)/yield - 1;
    end
end
```

B.5 Convergence

```
% this is test code to ensure that the standard deviation matches the
% nominal
close all
clearvars
max_p = 6;
times = zeros(1, max_p);
stresses = zeros(1, max_p);
sds = zeros(1, max_p);

for n = 1:max_p
    r_in = ones(Nelem+1,1) * .0415;
    r_out = ones(Nelem+1,1) * .05;
    [stress,sd] = standard_dev(n, r_in, r_out);
    sds(n) = sd(1);
    f = @() standard_dev(n, r_in, r_out);
    times(n) = timeit(f);
end
percent_error = [0,0,0,0,0];
for i = 2:max_p
percent_error(i-1) = abs(sds(6) - sds(i)) / sds(6) * 100;
end
figure
plot([1:max_p], times)
title('Effect of number of points on computational time')
xlabel('Number of quad. points')
ylabel('Time in seconds')
figure
plot([2:max_p], percent_error)
title('Effect of number of points on accuracy of standard deviation')
xlabel('Number of quad. points')
ylabel('Percent error')
```

B.6: Test

```
clearvars
close all

Nelem = 15;
L = 7.5; % semi-span in meters
x = [0:L/Nelem:L].';
r_in = ones(Nelem+1,1) * .0415;
```

```
r_out = ones(Nelem+1,1) * .05;
[stress,sd] = standard_dev(2, r_in, r_out);
figure
plot(x, stress)
hold on
plot(x, stress + 6*sd);
hold on
plot(x, stress - 6*sd);
hold on
legend({'mean stress','mean + 6 sd','mean - 6 sd'},'Location', 'northeast')
xlabel('Distance along span (m)')
ylabel('stress (Pa)')
```