Numerical Design Optimization

Report 2: UAV Spar Optimization

Unique ID: 1059

Professor Hicken

1. Executive summary

1.1 Problem Statement

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 length of the wing is predetermined.

1.2 Overview of methods

Methods for representing, analysing stress and deflection of the spar and optimizing geometry will be needed.

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. A mix of boundary constraints, linear inequality, and nonlinear inequality constraints are used for the most efficient implementation of the methods. 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.

1.3 Overview of results

The nominal design used is a thick spar, a hollow cylinder with the maximum radius and the minimum radius allowed by the boundary constraints. The final mass of the spar is 4.9183 kg which is a 62.91% increase over the nominal value of 13.26 kg.

2. Analysis method

2.1 Assumptions

In order to simplify the problem a few assumptions are made; a planar symmetry assumption is made, this mean the the cross section of the beam is symmetric on the longitudinal axis; 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 will be shown in the report, 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). However, for the first iteration of design these assumptions are okay and as more data is collected about wing twisting, material properties, and deformations the model will be updated.

2.2 Geometric representation

The geometry is represented by 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. These will be the design variables in this study.

2.3 Stress calculations.

The spar is treated as a cantilever beam with a triangular loading on it, similar to that shown in *Figure 2.3.1*. The "area" under the force distribution is the total force that the spar needs to support, in this case that is 2.5 * g * mass/ 2 (divided by 2 because there are 2 wings). So the maximum force on the root of the spar, by geometry, is g * mass / L.

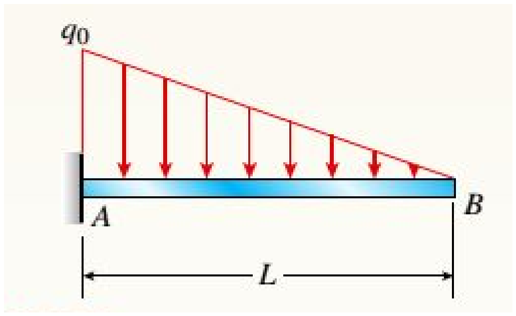

Figure 2.3.1: Triangular force distribution

The stresses and deflection are calculated using Euler-Bernoulli Beam Theory, which is shown in *Equation 2.3.1*. Here w is the vertical displacement, q(x) is the applied load, E is the applied Young's modulus, Iyy is the second moment of area with respect to the y-axis. For a circular annulus Iyy is shown in *Equation 2.3.2.*

$$\frac{d^2}{dx^2}\left(EI_{yy}\frac{d^2w}{dx^2}\right) = q, \qquad \forall x \in [0, L]$$

Equation 2.3.1: Euler-Bernoulli Beam Theory

$$\frac{\pi}{4}(r_o{}^4 - r_i{}^4) = Iyy$$

Equation 2.3.2: Second moment of area with respect to the y-axis for a circular annulus

The Euler-Bernoulli Beam equation is solved for w using boundary conditions for a cantilever beam, namely: no displacement or angular displacement at root and, in our case, no stress at tip. With these boundary conditions the Euler-Bernoulli Beam equation can be used to solve for the normal stress as a function of x. This is shown in *Equation 2.3.3,* where $-z_{max}$ is the height of cross section, or outer radius in our case.

$$\sigma_{xx}(x) = -z_{\max}E\frac{d^2w}{dx^2}$$

Equation 2.3.3: Stress of beam along x in terms of displacement

The Euler-Bernoulli beam equation is discretized and solved using the finite-element method. The solution is represented using Hermite- cubic functions.

3. Optimization

3.1 Objective Function

The objective of this study is to minimize the mass of the spar. So the objective function calculates the mass of the spar. It does this via truncated cone volume calculations between nodes. The area of the circular annulus is found at each point and then using the numerical integration function “trapz” in MatLab the integral over each annulus is taken and multiplied by the spacing between nodes which returns the volume of the spar. The volume can easily be turned into the mass by multiplying by the density. The code for the objective function is shown in *Appendix B,3: obj.*

3.2 Constraints

A variety of constraints need to be put on the system to ensure that it remains feasible to manufacture and it is safe in operating conditions. The simplelist constraints are upper and lower bound constraints, since the inner radius and thickness is being optimized, it will be an upper and lower bound on these two variables. The lower bound on thickness is 2.5 mm and for the inner radius it is 1 cm. The upper bound bound on thickness is the maximum outer radius minus the minimum inner radius, in our case that is 5cm- 1cm = 4cm and the upper bound on the inner radius is the maximum outer radius minus the minimum thickness, in our case that is 5cm - .25cm = 4.75 in.

The next constraint used is a linear equality constraint. This constraint ensures that the outer radius is never too large. The optimization algorithm takes the constraint in the form of $A*x \leq b$. The “x” matrix here is the column vector of radii followed by thicknesses at each node. The correlation between the inner radius, thickness and outer radius is simply: Inner radius + thickness = Outer radius at each node. The A matrix is (Nnode, 2*Nnodes), here Nnode denotes the total number of nodes, large and contains a one in columns i and i+Nnode, so if it is the third node it would have a one in column 3 and 6. The b matrix contains only the outer radius and is a column vector of size (Nnode, 1). A representation of the A, b and x matrices are shown below in *Figure 3.1.1*. The code for the lower and upper bounds as well as the inequality constraint is found in *Appendix B.2: Bounds.*

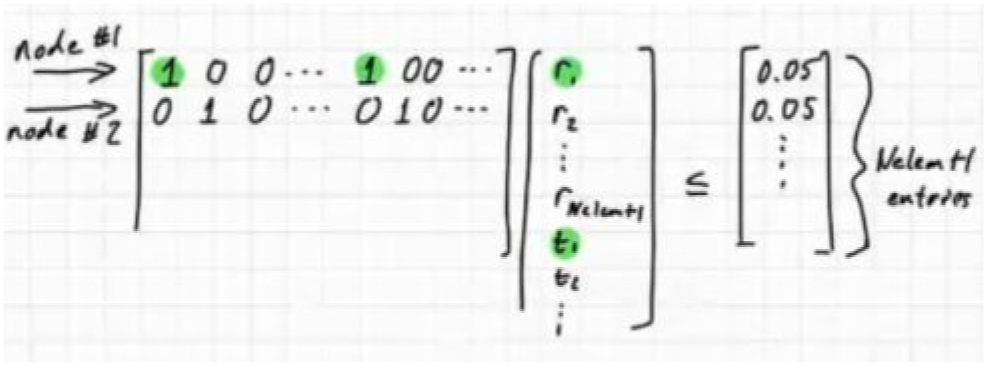

*Figure 3.2.1: Form of the A * x ≤ B constraint*

The final constraint is a nonlinear constraint to ensure that the system is not taking too much stress and thus at risk of failing. This constraint is the form of c(x) ≤ 0, where c(x) is the constraint function. To get this constraint, the stress is calculated at each node via the Euler-Bernoulli method as explained above and then the stress is divided by the ultimate tensile strength of carbon fiber and subtracted by one, the stress at any point goes over the ultimate tensile strength the constraint function will be greater than 0 and the design is infeasible and will not be considered in optimization. It is recommended to provide the Jacobian of this matrix using the complex step method of differentiation, for speed and accuracy. The Jacobian is size (2* Nnode, Nnode) and provides the partial derivative of the constraint function at each node with respect to each design variable. The code for this is shown in *Appendix B.1: Noncon.*

3.3 Convergence Study

Two convergence studies were done, both to find best the number of elements to use. The first looked at the correlation between mass of the spar and the number of elements. The results of which are shown in *Figure 3.2.1.* It is clear that the mass of spar is quite stable after the number of elements goes past 10.

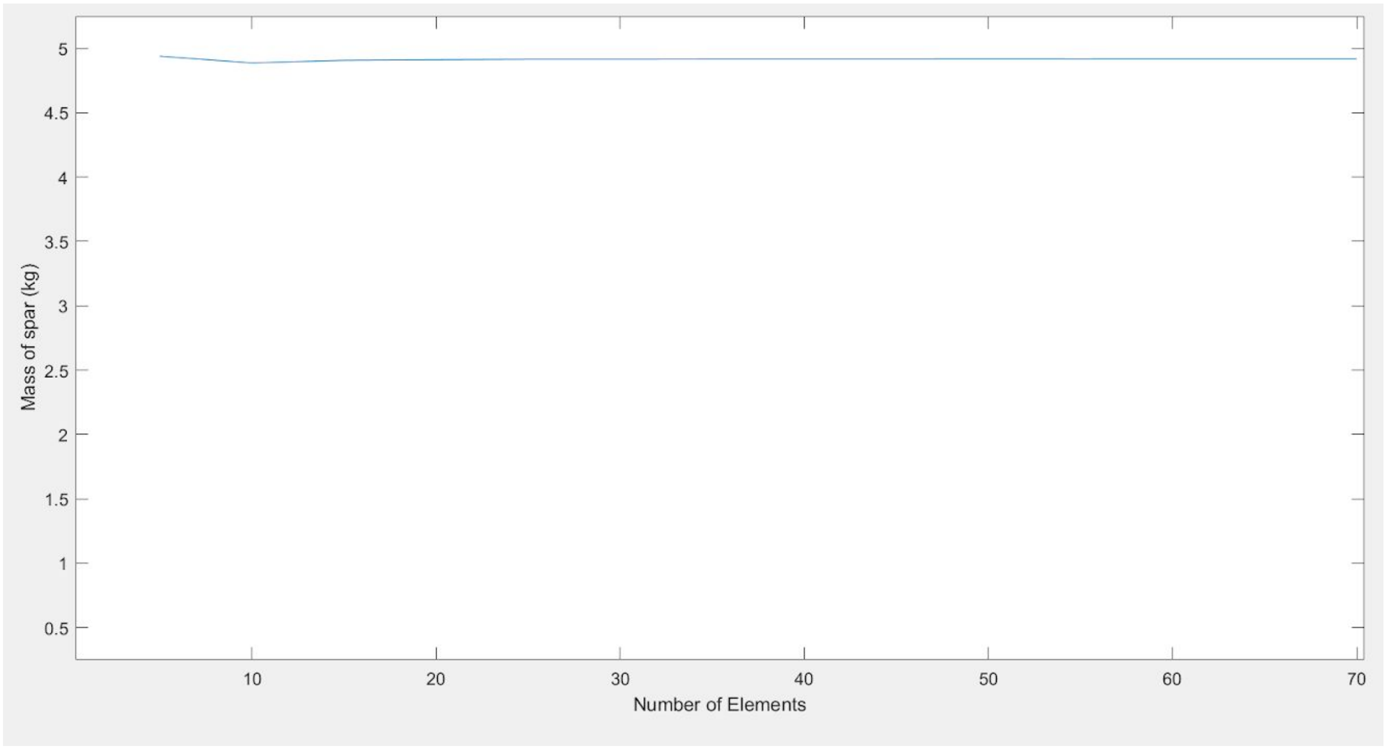

Figure 3.3.1: Mesh convergence of number of elements vs Mass of Spar

The second convergence study looks at the effect of the number of elements on the stress at the tip. Ideally the stress at the tip should be zero, however in practice ,due to error in numerical integration, it will not be exactly 0. As the number of nodes increases, the error reduces, to make it clearer the stress at the tip is divided by the stress at the root to relatively show how much error there is. The result of this convergence study is shown in *Figure 3.2.2*. It is clear that the normalized stress at the tip goes to zero as the number of elements goes past 30.

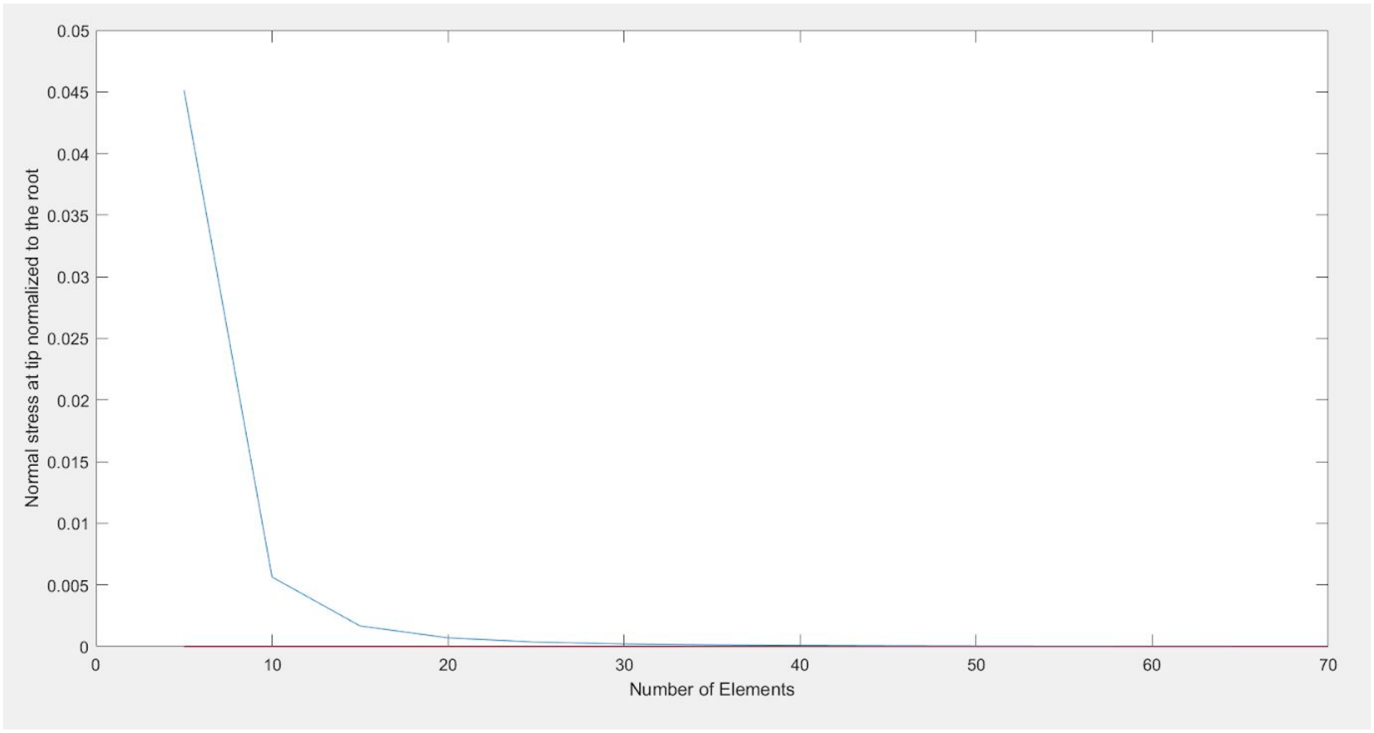

Figure 3.3.2: Mesh convergence of number of elements vs normalized of stress at tip

All convergence code is in *Appendix B.6: mesh_conv.*

3.4 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 a thick spar.

4. Results

The code described above is run with 35 elements. The optimized mass is 4.9168 kg, the optimized geometry is shown in *Figure 4.1*, the stress distribution along the spar is shown in *Figure 4.2*, the deflection of the spar is shown in *Figure 4.3,* and the first order optimality is shown in *Figure 4.4* to show to the result reached is a local minimum and is the best design for this application.

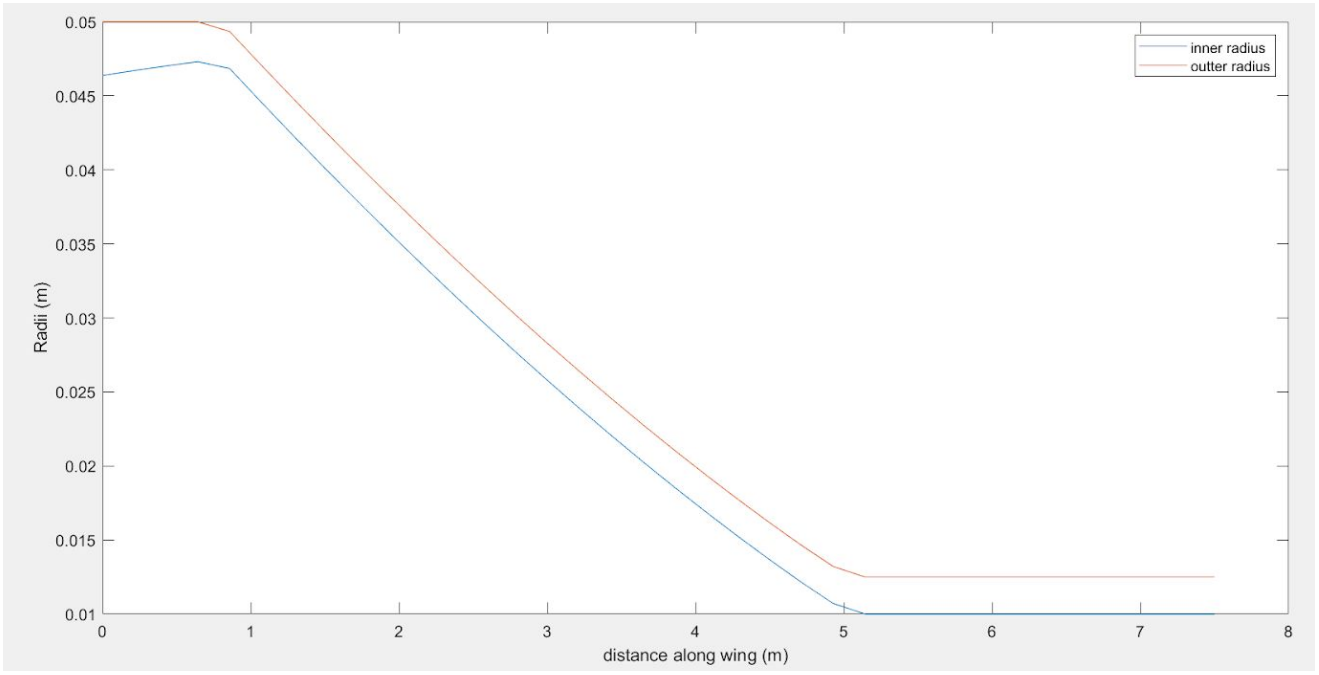

Figure 4.1: Representation of Geometry with Inner and Outer Radii

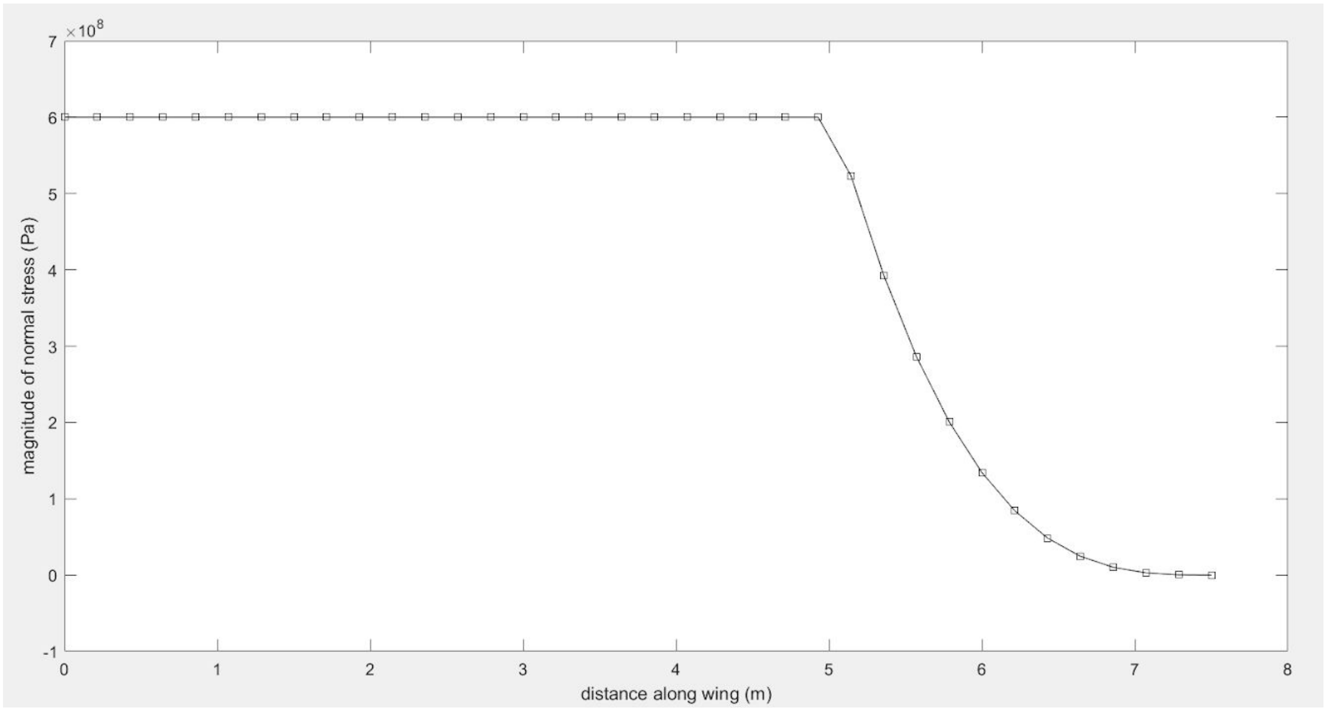

Figure 4.2: Stress Distribution Along the Length of the Spar

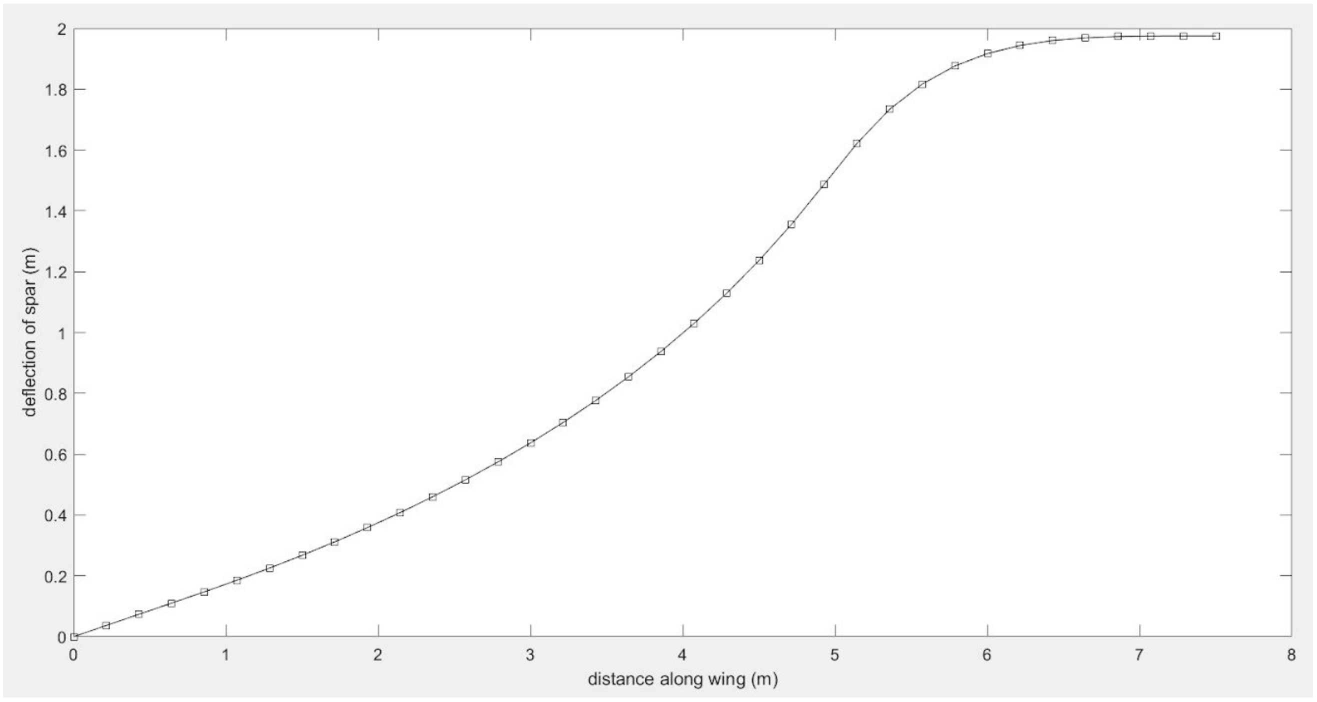

Figure 4.4: Deflection of Spar due to Bending Moment

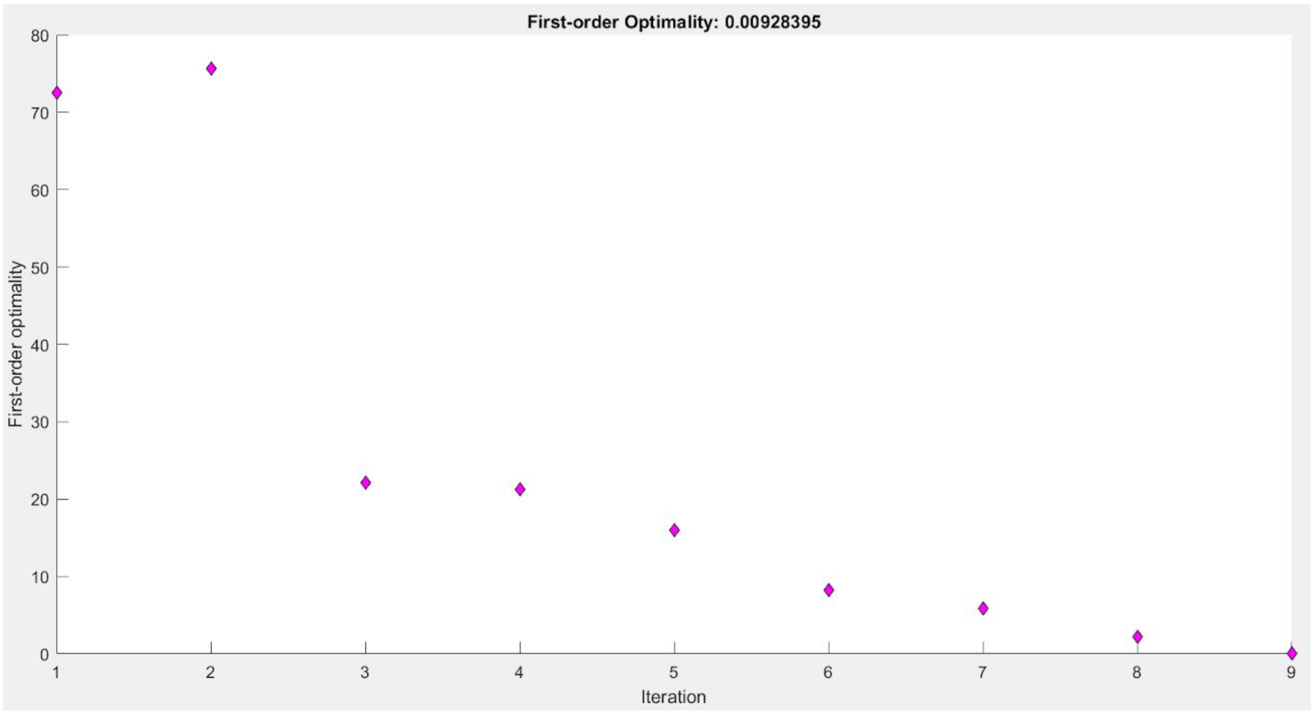

Figure 4.4: Plot of First Order Optimality Vs Iterations

5. Conclusions and Discussion of the Results

The final mass of the spar is 4.9183 kg which is a 62.91% increase over the nominal value of 13.26 kg for a thick spar.

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 the force distribution linearly decreases along the length of the spar. The initial portion of the outer radius is at its maximum and the thickness needs to increase in order to handle the high load at the tip of the spar. The deflection curve shows why the linearization assumption may not be valid in this case as the deformation is quite large ~25% 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 October 21, 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: Noncon, Finds nonlinear constraints and jacobians

```
function [cineq, ceq, jineq, jec] = noncon(RT, L, E, force, Nelem, Uts)
% finds nonlinear constraints and their jacobians
% Inputs:
% RT: Column vector of inner radii followed by thickness at each node
% L: Length of spar
% E: elastic modulus of material
% Downwards force at each node
% Nelem: number of elements
% Uts: Ultimate tensile strength
% cineq: non linear inequality constraint
% ceq: non linear equality constraint
% jineq: jacobian for the non linear ineq constraint
% jec:" jacobian for the non linear eq constraint

    Nnode = Nelem + 1;
    [cineq, ceq] = subcon(RT); %runs subfunction
    jineq = zeros(2* Nnode, Nnode);
    h = 1e-30; % complex step, step size
    for j = 1: 2 * Nnode
```

```
        % fills the jineq matrix
        % uses complex step to find the gradient of the stress at each
        % node with respect to the inner radius and the thickness
        xc = RT;
        xc(j) = xc(j) + complex(0,h);
        jineq(j,:) = imag(subcon(xc))/h;
    end
    jec = []; % no equality constraint
    function [cineq, ceq] = subcon(R_T)
        % finds nonlinear constraints
        % R_T is RT from above
        % outputs:
        % cineq, and ceq are same as above
        Iyy = zeros(Nelem + 1, 1);
        zmax = zeros(Nelem + 1, 1);
        for i = 1:Nnode
            % Calculates the moment of inertia of the spar at each node
            % and the puts the outer radius in its own matrix
            ri = R_T(i);
            ro = ri + R_T(Nnode + i);
            zmax(i) = ro;
            Iyy(i) = pi / 4 *(ro^4 - ri^4);
            if real(Iyy(i)) < 0.0
                Iyy(i) = -Iyy(i);
            end
        end
        u = CalcBeamDisplacement(L, E, Iyy, force, Nelem);
        sigma = CalcBeamStress(L, E, zmax, u, Nelem);
        cineq = (sigma / Uts) - 1; % Ensures that the stress will stay
        % below the ultimate tensile strength
        ceq = []; % no equality constraint
    end
end
```

B.2: Bounds

```
function [Aineq, B, lb, ub] = bounds(Nnode, rmin, rmax, tmin)
% generates linear equality constraints and lower and upper bounds
% Inputs:
% Nnode: number of nodes
% rmin: minimum inner radius
% rmax: Max outter radius
% tmin: min thickness

    Aineq = eye(Nnode, 2* Nnode);
    B = zeros(Nnode,1)+rmax;
    j = 1;
    for i = Nnode+1:2*Nnode
        % populates the second half of the equality constraint matrix
        % essentially creates an identity matrix for the secong half
        Aineq(j,i) = 1;
        j = j +1;
    end
    % lower bound on inner rad and thickness
        % 1cm = rmin m ; 2.5mm = tmin
    % upper bound on inner rad and thickness
        % 5cm - 2.5mm; 5cm - 1cm
    lb1 = ones(Nnode,1)*rmin;
    lb2 = ones(Nnode,1)*tmin;
    lb = [lb1; lb2];
    ub1 = ones(Nnode,1)*(rmax - tmin);
    ub2 = ones(Nnode,1)*(rmax - rmin);
    ub = [ub1; ub2];
End
```

B.3: obj, the objective function

```
function [m, dm] = obj(rad_thic, Nnode , L, rho)
% finds the mass of the system, is our objective function that we want to
% minimize
% rad_thic: Column vector of inner radii followed by thickness at each node
% Nnode: Number of nodes
% L: Length of spar
% rho: Density of material spar is made of
% Outputs:
% m: mass of spar
% dm: the gradient of the mass of the spar

    m = subobj(rad_thic); % finds mass of spar at
    dm = zeros(2 * Nnode,1);
    h = 1e-60;
    for j = 1:2 * Nnode
        % populates the gradient matrix using complex step
        xc = rad_thic;
        xc(j) = xc(j) + complex(0,h);
        dm(j) = imag(subobj(xc))/h;
    end
    function [mass] = subobj(RT)
        % Finds the mass of the spar
        % input:
        % RT: Column vector of inner radii followed by thickness at each node
        % Output:
        % mass: Mass of spar
        area = zeros(Nnode ,1);
        s = L/ (Nnode-1); % Spacing of the nodes
        for i = 1:Nnode
            % finds the area of the anulus at each node
            ri = RT(i);
            ro = ri + RT(Nnode + i);
            area(i) = pi * (ro^2 - ri^2);
        end
        % volume can be found by taking the inetgral over each annulus and
        % multiplying by the spacing of each node
        % mass can be found by multiplying the volume by density
        mass = rho * s * trapz(area);
    end
end
```

B.4: main, executes optimization

```
function [opt, mass] = main(L, rho, E, Uts, weight, Nelem, rmin, rmax, tmin)
% MAIN function, defines the objective function and the bounderies and
% inequality constraints
% plugs them into fmincon optimization
% Inputs: L: Length of the spar
% rho: density of the material
% L: Length of spar
% E: elastic modulus of material
% weight: weight of the whole UAV
% Nelem: number of elements
% Uts: Ultimate tensile strength
% rmin: minimum inner radius
% rmax: Max outter radius
% tmin: min thickness
% Outputs:
% opt: optimized radius and thickness
% Optimized mass

%function [w] = obj(RT, Nnode , L, rho)
%function [cineq, ceq, Jineq, Jeq] = noncon(RT, L, E, force, Nelem, Uts)
%function [Aineq, B, lb, ub] = bounds(Nnode, rmin, rmax, tmin)

    Nnode = Nelem + 1;
    fun = @(RT) obj(RT, Nnode , L, rho); % creates an objective function with only
one input
    fmax = weight * 2.5 / L;
    force = linspace(fmax, 0, Nnode);%  the force distribution for this case is
triangular
```

```
    % fmax is calculated using geometetry and we need to know thr force on each
    % node
    nonlin = @(RT) noncon(RT, L, E, force, Nelem, Uts);
    [Aineq, B, lb, ub] = bounds(Nnode, rmin, rmax, tmin);
    x0 = [linspace(rmax-.01,rmin,Nnode)'; ones(Nnode, 1)*.01];
    options = optimset("algorithm", "active-set",'Display', 'iter', ...
        "GradConstr", "on", "GradObj", "on", "DerivativeCheck", "off",
"PlotFcn","optimplotfirstorderopt" );
    [opt, mass] = fmincon(fun, x0, Aineq, B, [],[], lb, ub, nonlin, options);
end
```

B.5: runner, runs optimization

```
clearvars
close all
% This script runs the optimization code
% all the parameters are defined here
L = 7.5; %meter
rho = 1600; %kg/ m^3
E = 70e9; %Pa
Uts = 600e6; %Pa
mass = 500; %kg, max mass of the UAV
weight = 9.81 * mass; %N
Nelem = 35;
rmin = .01;
rmax = .05;
tmin = .0025;
% the optimization code is run here:
[RT,mass] = main(L, rho, E, Uts, weight, Nelem, rmin, rmax, tmin);
% RT is the optimized inner radius and thicknesses at each node; mass is the
optimized mass
ri = RT(1:Nelem+1); % seperates the column matrix for the inner radius
ro = RT(1:Nelem+1) + RT(Nelem+2:2*(Nelem+1)); % generates the column matrix
% for the outer matrix by adding the thickness to the inner radius
x = linspace(0,L, Nelem+1);
Nnode = Nelem + 1;
figure(2);
plot(x, ri)
hold on
plot(x,ro)
xlabel('distance along wing (m)')
ylabel('Radii (m)')
legend("inner radius","outter radius")
fmax = weight * 2.5 / L;
force = linspace(fmax, 0, Nnode)'; % the force distribution for this case is
triangular
% fmax is calculated using geometetry and we need to know thr force on each
% node
zmax = zeros(Nnode,1);
Iyy = zeros(Nelem + 1, 1);
weight = mass  * 9.81;
    for i = 1:Nnode
        % generates a matrix of moment of inertia
        ri = RT(i);
        ro = ri + RT(Nnode + i);
        zmax(i) = ro;
        Iyy(i) = pi / 4 *(ro^4 - ri^4);
    end
[u] = CalcBeamDisplacement(L, E, Iyy, force, Nelem);
[sigma] = CalcBeamStress(L, E, zmax, u, Nelem);

figure(3)
plot(x,sigma,'ks-')
xlabel('distance along wing (m)')
ylabel('magnitude of normal stress (Pa)')

figure(4)
plot(x,u(2:2:2*Nnode),'ks-')
xlabel('distance along wing (m)')
ylabel('deflection of spar (m)')

%function [w] = obj(RT, Nnode , L, rho)
%function [cineq, ceq, Jineq, Jeq] = noncon(RT, L, E, force, Nelem, Uts)
%function [Aineq, B, lb, ub] = bounds(Nnode, rmin, rmax, tmin)
```

B.6 mesh_conv, mesh convergence

```
clearvars
close all
% This script performs a convergance study
% all the parameters are defined here
L = 7.5; %meter
rho = 1600; %kg/ m^3
E = 70e9; %Pa
Uts = 600e6; %Pa
mass = 500; %kg, max mass of the UAV
weight = 9.81 * mass; %N
Nelem = 30;
rmin = .01;
rmax = .05;
tmin = .0025;
elems = 5:5:70;
% the convergance code is run here:
masses = zeros(size(elems,2));
```

```
sigma_end = zeros(size(elems,2));
fmax = weight * 2.5 / L;

count = 0;
for i = elems
    count = count + 1;
    Nnode = i + 1;
    [RT,mass] = main(L, rho, E, Uts, weight, i, rmin, rmax, tmin);
    masses(count) = mass;
    force = linspace(fmax, 0, Nnode)';
    zmax = zeros(Nnode,1);
    Iyy = zeros(Nnode, 1);
    for j = 1:Nnode
        % generates a matrix of moment of inertia
        ri = RT(j);
        ro = ri + RT(Nnode + j);
        zmax(j) = ro;
        Iyy(j) = pi / 4 *(ro^4 - ri^4);
    end
    [u] = CalcBeamDisplacement(L, E, Iyy, force, i);
    [sigma] = CalcBeamStress(L, E, zmax, u, i);
    sigma_end(count) = sigma(Nnode)/sigma(1);
end

figure(1);
plot(elems, masses)
xlabel('Number of Elements')
ylabel('Mass of spar (kg)')


figure(2)
plot(elems,abs(sigma_end))
xlabel('Number of Elements')
ylabel('Normal stress at tip normalized to the root')
```