Project 4: Uncertain Loading of a UAV Spar

ID: 6110
MANE 4280, Design Optimization

December 11, 2018

Contents

1 Executive Summary

A spar is a type of support found in the wing of an aeroplane. Used as early as the Wright brother's days, spars are used in plane wings today to provide structural support, and to transport the forces acting on the wings to the rest of the plane.

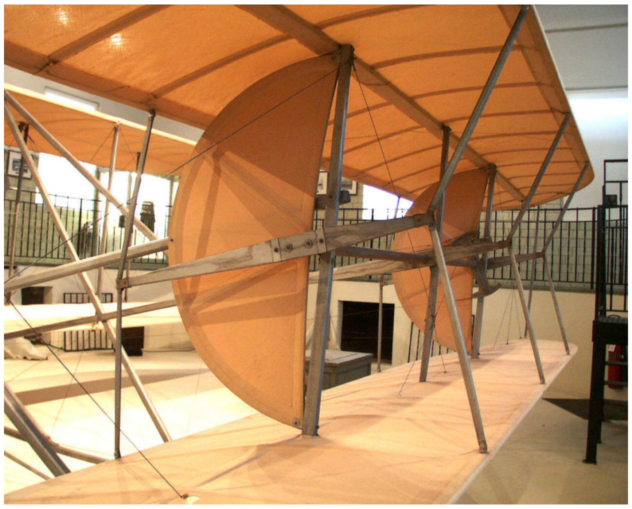


Figure 1: Spars and ribs are visible in the aerofoils of the Wright brother's Flyer[1].

A spar similar to the type explored in this project is seen in figure 2

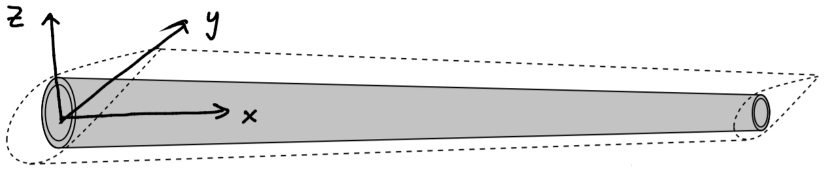


Figure 2: Diagram of spar with axis [2].

The goal of this project is to develop an optimal design for a spar that will minimize the mass under uncertain loading subject to a given set of constraints. This report will explain the analysis methods and their limitations, discuss how the geometry was parameterized, investigate the optimization methods and their results.

2 Analysis Methods

Before the analysis begins the details of the spar should be listed. It has a length of 7.5 meters. It's density is 1,600 $\frac{kg}{m^3}$[3]. A diagram of the spar is seen in figure 2. The minimum radius of the spar is one centimeter and the maximum radius is five centimeters. The thickness must be greater than 2.5 mm. The bending of the spar is modeled by Euler-Bernoulli Beam Theory using the following equations[2].

$$\frac{d^2}{dx^2}\left(EI_{yy}\frac{d^2w}{dx^2}\right) = q, \forall x \in [0, L] \tag{1}$$

Where

w is the vertical displacement in the z direction (m)
$q(x)$ is the applied load (N/m)
E is the elastic, or Youngs, modulus (Pa)
I_{yy} is the second-moment of area with respect to the y axis $(\mathrm{m})^2$

$$I_{yy} = \int\int z^2 dz dy \tag{2}$$

The spar is treated like a cantilever beam with the following boundary conditions:

$$w(x=0) = 0, \; \frac{d^2w}{dx^2}(x=L) = 0 \tag{3}$$

$$\frac{dw}{dx}(x=0) = 0, \; \frac{d^3w}{dx^3}(x=L) = 0 \tag{4}$$

This is to say that there is no vertical or angular displacement at the root and no stress at the tip. Once the vertical displacement is known, the stress may be solved by

$$\sigma_{xx}(x) = -z_{max}E\frac{d^2w}{dx^2} \tag{5}$$

The beam is discretized by the finite-element method. The finite-element equations come from the minimization of the potential energy. The solution is represented using Hermite-cubic shape functions. It is assumed that the spar does not experience any thermal effects due to the warping or the atmosphere's temperature.

The load on the spar is described by

$$f(x,\xi) = f_{\text{nom}}(x) + \delta_f(x,\xi) \tag{6}$$

Where

$$f_{\text{nom}}(x) = \frac{2.5W}{L}\left(1 - \frac{x}{L}\right) \tag{7}$$

And the probabilistic perturbation has the form

$$\delta_f(x,\xi) = \sum_{n=1}^{4} \xi_n \cos\left(\frac{(2n-1)\pi x}{2L}\right) \tag{8}$$

with

$$\xi_n \sim \mathcal{N}\left(0, \frac{f_{\text{nom}}(0)}{10n}\right) \tag{9}$$

2.1 Limitations

This model is concerned only with the stress on a spar from a turn and the spar's mass. It does not consider any other aerodynamic problems the spar may face; such as flutter for instance. The model assumes the majority of the force will be applied at the root of the spar. This would be a poor assumption if a payload is mounted along the wing. Many UAVs do have wing mounted payloads, such as missiles. Additionally the model does not take into account the manufacturability of the spar, some designs may provide lots of strength with little weight, but could never be manufactured in the real world.

3 Geometry parameterization

To simplify the analysis the spar's geometry was divided into small sections. As the sections get smaller and smaller they better approximate a spar with a shape determined by some sort of polynomial.

The design variable is a vector X, with n nodes. The vector must represent the inner and outer radii of the spar at each node. The way this was implemented was as follows:

$$\underbrace{X_{(2\times n,1)}}_{\text{design variable}} = [r_1, r_2, \ldots, r_n, t_1, t_2, \ldots, t_n]^T \tag{10}$$

Where r is the inner radius of the spar at a specific node and t is the thickness at a specific node.

4 Optimization

The objective function of this problem is to minimize the mass of the spar while ensuring that the mean plus six standard deviations of the stress remains below the ultimate strength of the carbon fiber. The dependent variable is an 'X' vector. This X vector, holds the inner radius and corresponding thickness of the cylinder at each node.

$$\begin{array}{lll} \underset{X}{\text{minimize}} & \text{mass}(X) & \\ \text{subject to} & X_j \geq 0.01 \text{ meter}, & j = 1, 2, \ldots, n \\ & X_{n+j} \geq 0.0025 \text{ meter}, & j = 1, 2, \ldots, n \\ & X_j + x_{n+j} \leq 0.05\text{meter}, & j = 1, 2, \ldots, n \\ & \mu(\text{Stress}) + 6\sigma \leq \text{Stress}_{\text{yeild}}, & j = 1, 2, \ldots, n \end{array}$$

The minimization of the mass of the spar will be accomplished via the fmincon function in MATLAB and the complex step method. As the minimum mass of the spar is sought, no manipulation of the output is need (if the maximum mass was sought, the MATLAB implementation would have to search for 1/mass, as these methods traditionally seek a variable to minimize a functions output).

4.1 Optimization Methods

The optimization method used in this project was the complex step method. The complex step method is useful because it's step value is not limited by machine epsilon. As such it can take far smaller steps than Sequential Quadratic Programming (SQP)[4].

The complex step method was used to provide gradients for both the objective function (the minimization of mass) and the nonlinear constraint (the stress). As previous reports have throughly explained the complex step method, it is not described any further in this report. A more detailed description may be found in [5].

The uncertanty is implemented via stochastic collocation. Stochastic collocation operates as follows

1. Compute quadrature points, denoted $\xi^{(k)}$, that are specifically tailored to the given probability density function and the corresponding quadrature weights $w^{(k)}$.

2. Adjust weights and quadrature points via equation 11 [2]

$$\begin{aligned} w^{(k)} &= \tfrac{w^{(k)}}{\sqrt{\pi}} \\ \xi^{(k)} &= \sqrt{2}\sigma\xi^{(k)} + \mu \end{aligned} \tag{11}$$

3. Sample function f at quadrature points.

4. Compute mean as the weighted sum using equation 12 [2]

$$\mu_f \approx \sum_{k=1}^{m} w^{(k)} f\left(\xi^{(k)}\right) \tag{12}$$

Table 1 gives the first Hermite-Gauss quadrature points and weights. In the MATLAB code the Gauss Hermine quadrature locations and weights were computed using the GaussHermiteFunc code, which was developed from Geert Van Damme's Gauss-Hermite Quadrature function [6].

Table 1: Exact Hermite-Gauss quadrature points and weights [7]

n	2	3		4		5		
x_i	±0.707107	0	±1.22474	±0.524648	±1.65068	0	±0.958572	±2.02018
w_i	0.886227	1.18164	0.295409	0.804914	0.0813128	0.945309	0.393619	0.0199532

4.2 Constraints

The inner radius of the spar can be no smaller than 1 cm, and the outer radius of the spar can be no lager than 5 cm. The distance between the inner and outer radii must be lager than 2.5 mm.

To accomplish these constraints the MATLAB fmincon function used it's lower bound, upper bound, linear inequality, and non linear constraints. For more details of keeping the the spar within it's geometrical bounds please see [5]. What will be detailed here is the stress constraint under uncertain loading.

The most important constraint is that the maximum stress must be six standard deviations below the spar's ultimate tensile/compressive strength, or:

$$\mu(\text{Stress}) + 6\sigma \leq \text{Stress}_{\text{yeild}} \tag{13}$$

For a single variable Gauss-Hermine, with m = 3, the mean or expected stress is given by equation 14 [2].

$$\mu_f(x) \approx \frac{1}{\sqrt{\pi}} \sum_{k=1}^{3} w^{(k)} f\left(\sqrt{2}\sigma\xi^{(k)} + x\right) \tag{14}$$

For p uncorrelated Gaussian random variables with $\xi_i \sim \mathcal{N}(x_i, \sigma_i)_1\, i = 1, 2, \ldots, p_i$, the mean or expected stress is given by equation 15 [2].

$$\mu_f(x) = \left(\prod_{i=1}^{p} \frac{1}{\sqrt{2\pi\sigma_i}}\right) \int_{\mathbb{R}^p} f(\xi) \exp\left(-\sum_{i=1}^{p} \frac{(\xi_i - x_i)^2}{2\sigma^2}\right) d\xi \tag{15}$$

This may be approximated via equation 16 [2]

$$\mu_f(x) = \left(\prod_{i=1}^{p} \frac{1}{\sqrt{\pi}}\right) \sum_{k_1=1}^{m} \sum_{k_2=1}^{m} \cdots \sum_{k_p=1}^{m} \left(\prod_{i=1}^{p} w^{(k_i)}\right) \times \tag{16}$$
$$f\left(\sqrt{2}\sigma_1\xi^{(k_1)} + x_1, \sqrt{2}\sigma_2\xi^{(k_2)} + x_2, \ldots, \sqrt{2}\sigma_p\xi^{(k_p)} + x_p\right)$$

The variance is computed from the mean by equation 17.

$$Var(x) = \mu(x^2) - \mu(x)^2 \tag{17}$$

And the standard deviation is then computed from the variance via equation 18.

$$\sigma = \sqrt{\mu(x^2) - \mu(x)^2} = \sqrt{Var(x)} \tag{18}$$

In MATLAB this nonlinear inequality constraint is found in the stressConst function, where the following inequality constraint is passed to fmincon:

$$\sigma_{\text{current stress}} = \mu(Stress) + 6 \times \sigma \tag{19}$$
$$c = \frac{\sigma_{\text{current stress}}}{\sigma_{\text{yeild stress}}} - 1 \leq 0$$

The code that generates these matrices and all of the other codes may be seen in the appendix.

5 Results

The nominal mass of the spar was given to be 29.32 kg, based on an 15 element analysis with a inner radius of 4.15 cm and a outer radius of 5 cm. The resulting stress from the nominal configuration may be seen in in figure 3. The objective was to obtain an optimal mass that was 70% lower than the nominal mass.

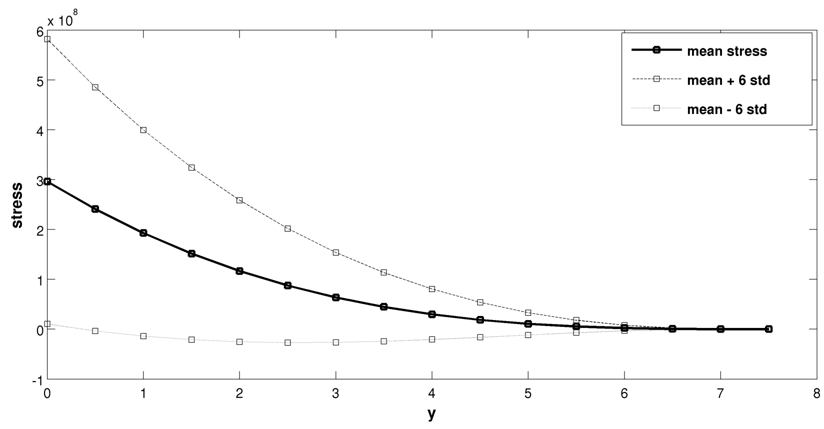


Figure 3: Stress Distribution from Nominal Configuration

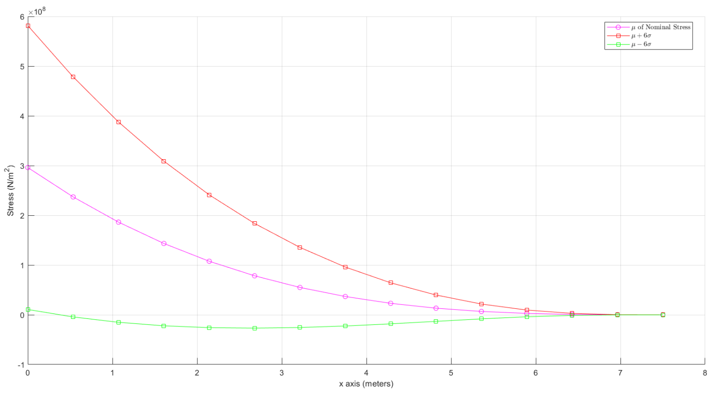
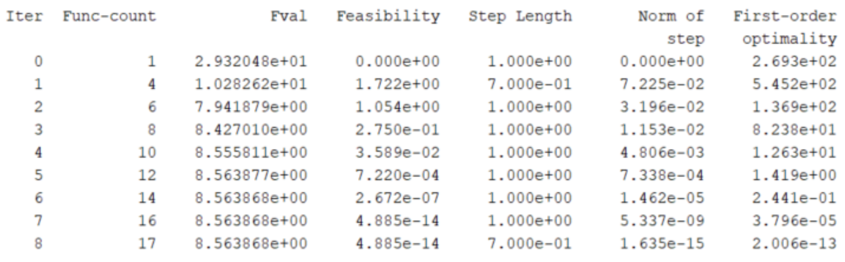


Figure 4: Stress Distribution from Nominal Configuration From Code

The MATAB code outputs the stress associated with the nominal configuration seen in figure 4. These plots are nearly identical, which means the code is working as we expect it to.

Running the MATLAB code at 15 elements and 3 collection points gave a result of 8.56387 kg, a little more than 70% less than nominal. The output from MATLAB's fmincon may be seen in figure 5 and the resulting stress and geometry may be seen in figure 6.



```
 Iter  Func-count            Fval   Feasibility   Step Length       Norm of   First-order
                                                                       step    optimality
    0           1    2.932048e+01     0.000e+00     1.000e+00     0.000e+00     2.693e+02
    1           4    1.028262e+01     1.722e+00     7.000e-01     7.225e-02     5.452e+02
    2           6    7.941879e+00     1.054e+00     1.000e+00     3.196e-02     1.369e+02
    3           8    8.427010e+00     2.750e-01     1.000e+00     1.153e-02     8.238e+01
    4          10    8.555811e+00     3.589e-02     1.000e+00     4.806e-03     1.263e+01
    5          12    8.563877e+00     7.220e-04     1.000e+00     7.338e-04     1.419e+00
    6          14    8.563868e+00     2.672e-07     1.000e+00     1.462e-05     2.441e-01
    7          16    8.563868e+00     4.885e-14     1.000e+00     5.337e-09     3.796e-05
    8          17    8.563868e+00     4.885e-14     7.000e-01     1.635e-15     2.006e-13
```

Figure 5: Output from fmincon

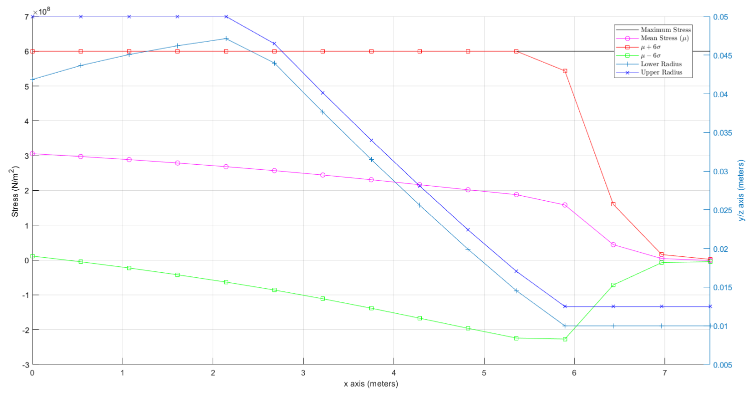


Figure 6: Spar Stress and Geometry at 15 elements, and 3 collection points.

Comparing this with the results from project 2 (optimal mass of 4.343), seen in figure 7 shows the spars have a similar shape. Both spars start at a maximum I_{yy} at the root to handle the stress (as the highest force is applied at the root). From there it decreases the mass by first shrinking the thickness, until the minimum thickness is hit. Then it decreases the mass by shrinking the inner radius while keeping the spar able to handle the stress. Eventually it hits a point where the radius and the thickness cannot be shrunk, and this the spar stays at this minimum cross section. However the spar from project 2 does not have to deal with six standard deviations of uncertainty, and can minimize it's mass far more.

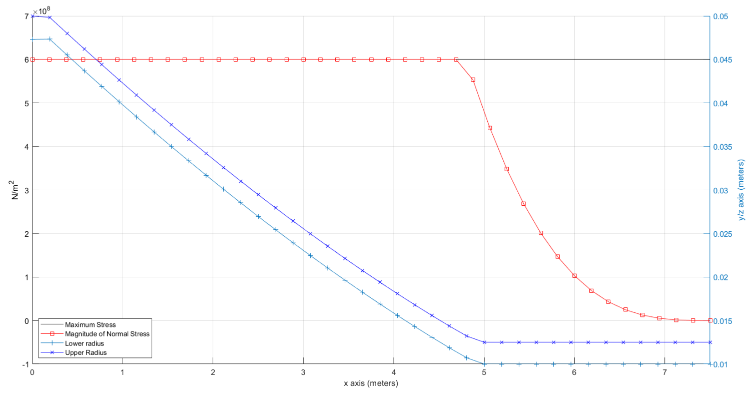


Figure 7: Geometry of spar and stress of spar from project 2.

What will happen if the code is run for more that three collection points? Running the code at six collection points provided a spar with a mass of 8.56387 kg and a geometry and stress seen in figure 8.

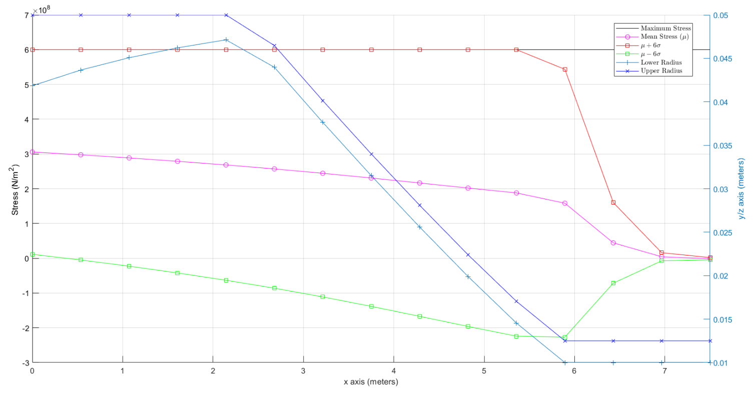


Figure 8: Spar Stress and Geometry at 15 elements, and 5 collection points.

This the results at five collection points are no different than the results at three collection points. In fact if the code is run for different numbers of elements and collection points(as seen in figure 9) the mass never changes due to the number of collection points, it only changes due to the number of elements.

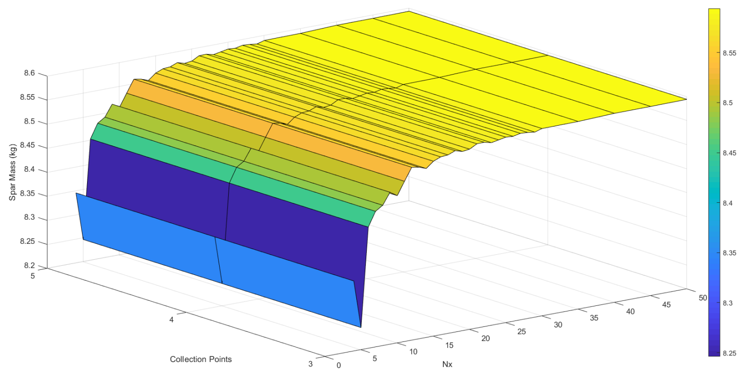


Figure 9: Mass vs number of collection points and number of elements

As a final check on convergence, we plot the displacement of the tip of the spar vs the number of nodes. This relation, seen in figure 10 shows that the displacement of the tip of the spar changes rapidly with the number of elements at first, but soon settles to a value of 0.681 meters. It stays there even though the number of elements increases.

6 Conclusions

The MATLAB results are what would be expected. When dealing with a large uncertainty, the spar will be heavier than the spar that was developed in uncertainty free project 2. The spar has a similar geometry to project 2's spar, just scaled to handle the uncertainty. The I_{yy} must be at it's maximum near the root, then as it moves down the wing it can shrink to decrease mass. It first shrinks the thickness, and when it hits the minimum thickness it shrinks the radius. However at a certain point the radius can shrink no more and it stays at the minimum radius and thickness for the rest of the spar length. Like project 2, the stress is kept at the appropriate factor of safety for as long as possible.

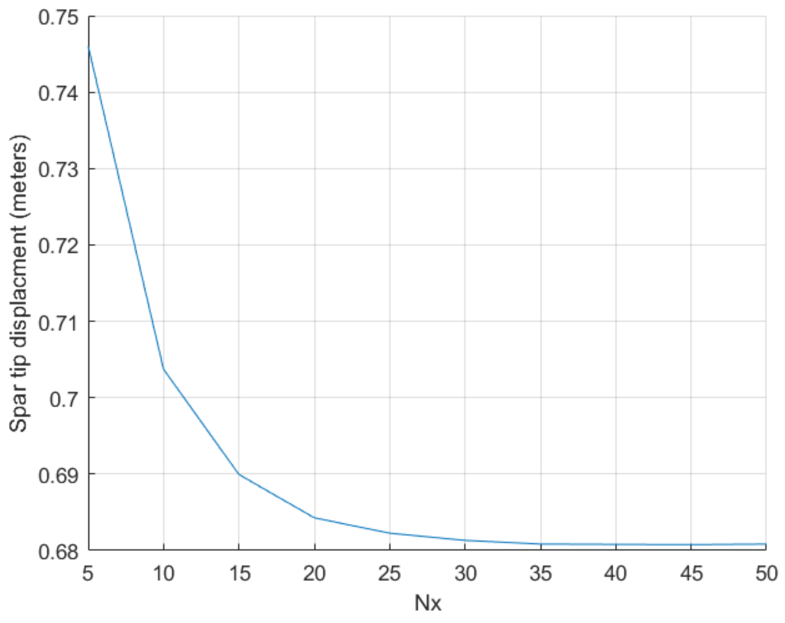


Figure 10: Spar Tip displacement

What was unexpected was that the number of collection points had no difference in the resulting mass. The only thing that changed the mass was the number of elements the spar was made up of.

Something that engineers must take into account when designing models is the run time cost. The time to run each calculation at a different number of nodes and collection points is seen in figure 11.

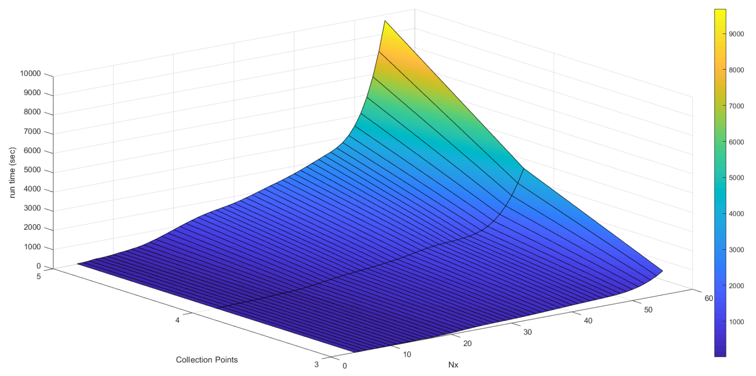


Figure 11: Time to run optimization vs the number of collection points and nodes

As the number of nodes and collection points increases, so too does the time to run the model increase. This is important as some models may take to long to run for their results to be useful. Additionally it can be seen that once the number of nodes reaches a certain number, the mass changes very little with subsequent increases. In figure 9, this point is around 40 nodes. Thus the return of accuracy over run time is so small that it is not worth running the code past this point.

7 References

References

[1] “Original 1905 wright flyer iii.” [Online]. Available: http://www.wright-brothers.org/Adventure_Wing/Hangar/Real_McCoy/Real_McCoy.htm

[2] J. Hicken, “Mane 4280 class notes,” December 2018.

[3] ——, “Mane 4280 project 4 description,” November 2018.

[4] 6110, “Mane 4280 project 1,” September 2018.

[5] ——, “Mane 4280 project 2,” October 2018.

[6] G. V. Damme. (2010, Feb) Legendre laguerre and hermite - gauss quadrature. Available: https://www.mathworks.com/matlabcentral/fileexchange/26737-legendre-laguerre-and-hermite-gauss-quadrature. [Online; accessed 1-December-2018].

[7] W.H.Beyer, *CRC Standard Mathematical Tables*, 28th ed. CRC Press, 1987.

This report was written by ID: 6110 in Latex. If desired the source code may be provided.

8 Appendix: MATLAB Code

Project 4 Master Code

Table of Contents

By 6110

Inital Variables and Constraints

computational values Change these for altering the analysis model

```
close all; clear all;
nodes =15; % number of nodes evaluated at
NCP=3;
NPP=4;
complex_step_size = 1e-60;
% model variables
mass =500; % kg
L = 7.5; % meter
E = 70*10^9; % Pa
Y=600 *10^6; % Yeild Stress Pa
rho = 1600.0;   % density (kg /m^3)
w = (2.5 * 9.8 *.5* 500); % 2.5 * plane weight, note that the 2.5 is
 put here to avoid as many calculations in the fmincon as possible
```

Create bounds

Creates linear bounds

```
rdist=2.5*10^-3; % thickness, m
rmin=1*10^-2; % min radi m
rmax = 5*10^-2; % max radi m


A=zeros(nodes,nodes*2); % prealloc
for i=1:nodes
    A(i, i)=1; % lower node
    A(i, i+nodes)=1; % upper node
    b(i)=rmax; % distance

    lb(1,i)=rmin; %% lb for r_low
    ub(1,i)=rmax-rdist; %% ub for r_low
    lb(1,i+nodes)=rdist; % lb for thick
    ub(1,i+nodes)=rmax-rmin-rdist; %up for think
end
```

Initalize wing object

provide the nominal spar values as initial parameter

```
r_in = 0.0415*ones(nodes,1); % nominal values of inner radii
thicc = 0.0085*ones(nodes,1); % nominal values of annular thickness
x0 = [r_in;thicc];            % concactenated design vector

wing=Spar(w,L,rho,E,Y,nodes,x0,A,b,lb,ub,NCP,NPP,complex_step_size);
```

Start fmincon

```
tic % start timer
[sprMss,sprGeo,exp,var,std,wts,pts,exitFlag, fmincOutput ]
 =fmincon_handle(wing);
fprintf('Nx=%d,Num Colc Pts = %d, Spar is %.5f kg, ',nodes,NCP,
 sprMss); % print mass
toc % output run time
makePlots(sprGeo,pts,wts,wing,exp,var,std); % create plots
```

Published with MATLAB® R2018b

Spar Class

```
classdef Spar
    %SPAR Class for Wing Spar
    %   The purpose of useing a class is to simplyfy the number of
    %   variables sent in function handles in fmincon_handle.m,
 without
    %   using global variables that cause a significant slowdown of
    %   performance. The other reason was to give the Author
 experiance
    %   with object oreinted programming in MATLAB, which is needed
 for the
    %   Author's Neural Net
    properties
        w;
        L;
        x0;
        rho;
        Nx;
        E;
        Y;
        A;
        b;
        lb;
        ub;
        NcolPts;
        NpertPts;
        step;
    end
    methods
        function obj =Spar(w,L,rho,E,Y,Nx,x0,A,b,lb,ub,NCP,NPP,step) %
 method to initilizae values.
            obj.w=w; % plane weight
            obj.L=L; % spar length
            obj.rho=rho; % density
            obj.E=E; % Youngs mod
            obj.Y=Y; % yeild stress
            obj.Nx=Nx; % number of nodes
            obj.x0=x0; % inital geometry
            obj.A=A; % bound
            obj.b=b; % bound
            obj.lb=lb; % lower bound
            obj.ub=ub; % upper bound
            obj.NcolPts=NCP; % Number of Collection points
            obj.NpertPts=NPP;
            obj.step=step; % step size
        end
    end
end
```

Published with MATLAB® R2018b

fmincon_handle

By 6110, main function to run fmincon

```
function [sprMss,sprGeo,exp,sigmaVar_final,std_final,wts,pts,exitFlag,
 fmincOutput ] = fmincon_handle(wing)
% note return wts, pts for debugging
[pts, wts] = get_pts_wts(wing);

% define anon funcs for fmincon
obj_Fun     = @(desVar) getMass(desVar,wing);
nlcon   = @(desVar) StressConst(desVar,pts,wts,wing);

optns = optimoptions(@fmincon,...
    'Display','iter-detailed',...
    'Algorithm', 'sqp',...
    'GradObj', 'on',...
    'SpecifyObjectiveGradient', true, ...
    'SpecifyConstraintGradient', true, ...
    'OptimalityTolerance', 1e-10, ...
    'StepTolerance', 1e-10, ...
    'ConstraintTolerance', 1e-10,....%);
    'FiniteDifferenceType', 'central', ...
    'FiniteDifferenceStepSize', 5e-4,...
    'CheckGradient',false);
[ sprGeo,sprMss,exitFlag, fmincOutput ]  =
 fmincon(obj_Fun,wing.x0,wing.A,wing.b,[],
[],wing.lb,wing.ub,nlcon,optns);

[exp,sigmaVar_final,~,std_final,wts,pts] =
 handleMakePlots(sprGeo,pts,wts,wing);


    function [pts, wts] = get_pts_wts(wing)
        % Returns Gauss quadrature points and weights. Note that it is
 done
        % before fmincon, not during to speed up run time.
        % Inputs:
        %   wing - spar object that has Spar Length and number of
 collection points
        % Outputs:
        %   pts - GQ points
        %   wts - GQ weights at points

  %---------------------------------------------------------------------------
        fNom0 = 2*wing.w/wing.L;
        mu = zeros(1,4);%

        sigma(1:4)=fNom0.*1./(10*(1:4));

        [xi, w] =GaussHermite(wing.NcolPts); % get points and weights
        pts = zeros((wing.NcolPts^4),4); % prealloc
        wts = zeros((wing.NcolPts^4),1); % prealloc

        i = 1;
        for i1=1:wing.NcolPts
            xi1 = sqrt(2)*sigma(1)*xi(i1) + mu(1); % first layer
            for i2=1:wing.NcolPts
                xi2 = sqrt(2)*sigma(2)*xi(i2) + mu(2);  % second layer
                for i3=1:wing.NcolPts
                    xi3 = sqrt(2)*sigma(3)*xi(i3) + mu(3);  % third
layer
                    for i4=1:wing.NcolPts
                        xi4 = sqrt(2)*sigma(4)*xi(i4) + mu(4);  %
fourth layer
                        pts(i,:) = [xi1, xi2, xi3, xi4]; % get points
                        wts(i) = w(i1)*w(i2)*w(i3)*w(i4); % get
weights
                        i = i+1; % increment i
                    end
                end
            end
        end
        fprintf('pts and wts created\n');
    end

    function [sigmaExp, sigmaVar] = getStress(pts,wts,wing,desVar)
        % Caculates expected (mean) and variance of stress.
        % Inputs:
        %   wing - spar object that has values for Spar
        %   pts - GQ points
        %   wts - GQ weights at points
        %   desVar - current design variable
        % Outputs:
        %   sigmaExp - Expectation of stress
        %   sigmaVar - Variance of stress

 %--------------------------------------------------------------------------
        [r_out, ~, Iyy] = get_RO_RI_Iyy(desVar); % get r_out, Iyy
        sigmaExp = 0; % prealloc
        sigmaVar = 0; % prealloc
        for n=1:(wing.NcolPts^4)
            forceBeam = getForce(pts(n,:),wing); % get force on bream
            dispBeam =
CalcBeamDisplacement(wing.L,wing.E,Iyy,forceBeam,wing.Nx-1); % get
beam displacment
            sigmaBeam =
CalcBeamStress(wing.L,wing.E,r_out,dispBeam,wing.Nx-1); % get stress
on beam
            sigmaExp = sigmaExp + wts(n)*sigmaBeam; % get mean
            sigmaVar = sigmaVar + wts(n)*(sigmaBeam).^2;% get var
        end

    end

    function [mass,Jacob] = getMass(desVar,wing)
```

```
       % Function to calculate mass of spar, objective function of
fmincon
       % Inputs:
       %   wing - spar object that has values for Spar
       %   desVar - design variable
       % Outputs:
       %   Jacob - grad for complex step
       %   mass - mass of spar

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

       mass = massFun(desVar,wing); % get mass from mass function
       Jacob = zeros(2*wing.Nx,1); % prealloc for speed

       for i=1:2*wing.Nx % complex step function
           ej = zeros(size(desVar));
           ej(i) = ej(i) + 1j*wing.step;
           x_plus=desVar+ej;
           f_plus=massFun(x_plus,wing);
           Jacob(i) = imag(f_plus)/wing.step;
       end

       function mass = massFun(desVar,wing) % mass function
           % Nested Function for spar mass, used in complex step
           % Inputs:
           %   desVar - design variable
           %   wing - spar object that has values for Spar
           % Outputs:
           %   mass - mass of spar

 %---------------------------------------------------------------------------
           [r_in,r_out]= getRoutRinFunc(desVar); % get r_in, r_out
           secArea = r_out.^2 - r_in.^2; % get section area
           mass = trapz(secArea)*pi*wing.rho*wing.L/(wing.Nx-1); %
integrate volumed with trapz
       end

   end

   function [c,emptArr,Jacob,emptArr2] =
StressConst(desVar,pts,wts,wing)
       % Function for stress inequality constraint, uses comples step
with gradient
       % Inputs:
       %   desVar - design variable
       %   wing - spar object that has values for Spar
       %   pts - GQ points
       %   wts - GQ weights at points
       % Outputs:
       %   c - non linear constraint for stress
       %   emptArr - empty array to return for fmincon formating
       %   Jacob - Gradiant of constraint
       %   emptArr - empty array to return for fmincon formating
 %-----------------------------------------------------------------------------
       emptArr=[]; % because you cannot have [] in an output
       emptArr2=emptArr; % and you cannot repete an output
       c = ineqFun(desVar,wing,pts,wts); % get ineq const
       grad = zeros(wing.Nx,2*wing.Nx); % prealloc

       for i=1:2*wing.Nx % stress constraing gradiant
           ej = zeros(size(desVar));
           ej(i) = ej(i) + 1j*wing.step;
           x_plus=desVar+ej;
           f_plus=ineqFun(x_plus,wing,pts,wts);
           grad(:,i) = (imag(f_plus)/wing.step);

       end
       Jacob = grad'; % return gradiant

       function c = ineqFun(desVar,wing,pts,wts)
           % Nested Function for inequality constraint, used in
complex step
           % Inputs:
           %   desVar - design variable
           %   wing - spar object that has values for Spar
           %   pts - GQ points
           %   wts - GQ weights at points
           % Outputs:
           %   c - non linear constraint for stress

 %-----------------------------------------------------------------------------
           [sigmaExp, sigmaVar] = getStress(pts,wts,wing,desVar); %
run calcStress
           sigma = getStandardDev(sigmaExp,sigmaVar); % get standard
deveiation
           c = (sigmaExp + 6*sigma)./wing.Y - 1; % compute c
       end

   end

   function standardDev = getStandardDev(sigmaExp,sigmaVar)
       % Function to compute get standard deviation
       % Inputs:
       %   sigmaExp - expected value
       %   sigmaVar - variance
       % Outputs:
       %   standardDev - standard deviation

 %-----------------------------------------------------------------------------
        standardDev = sqrt(sigmaVar - sigmaExp.*sigmaExp); % get stdv
   end


   function [force] = getForce(pts,wing)
       % Function to compute force at points
       % Inputs:
        %   wing - spar object that has values for Spar
        %   pts - GQ points
        % Outputs:
        %   force - force at points on spar

  %------------------------------------------------------------------------
        delta_F = 0; % prealloc
        x = linspace(0,wing.L,wing.Nx); % create linspace
        forceNomX = (2*wing.w/wing.L)*(1-x./wing.L); % nominal force
        for n=1:wing.NpertPts % for this code it is 4, but for other
 problems you might want more than 4 pts
            delta_F = delta_F + pts(n)*cos(((2*n-1).*pi.*x)/
(2*wing.L)); % from project discription
        end
        force = forceNomX + delta_F; % return force
    end

    function [sigmaExp,sigmaVar,desVar,sigma,wts,pts] =
 handleMakePlots(desVar,pts,wts,wing) % function gives everything
 needed for plots
        % Function to get variables used in plots
        % Inputs:
        %   desVar - design variable
        %   wing - spar object that has values for Spar
        %   pts - GQ points
        %   wts - GQ weights at points
        % Outputs:
        %   sigmaExp - expected value of stress
        %   sigmaVar - variance of stress
        %   sigma- standard deviation of stress
        %   desVar - design variable
        %   pts - GQ points
        %   wts - GQ weights at points

  %------------------------------------------------------------------------
        [sigmaExp,sigmaVar] = getStress(pts,wts,wing,desVar);  % get
 expected value and variance
        sigma = getStandardDev(sigmaExp,sigmaVar); % get standard
 deviation
    end

    function [r_in,r_out]= getRoutRinFunc(desVar)
        % Function to get r_out, r_in from design variable
        % Inputs:
        %   desVar - design variable
        % Outputs:
        %   r_in - inner radius
        %   r_out - outer radius

  %------------------------------------------------------------------------
        lnDV=length(desVar); % get length of designVariable
        r_in = desVar(1:lnDV/2); % get r_inner
        r_out = r_in+ desVar(lnDV/2+1:end); % get r_outer
    end
    function [r_out, r_in, Iyy] = get_RO_RI_Iyy(desVar)
        % function to get r_out, r_in, and Iyy
        % Inputs:
        %   desVar - design variable
        % Outputs:
        %   r_out - outer radius
        %   r_in - inner radius
        %   Iyy - Second moment of inertia

  %-------------------------------------------------------------------------
        [r_in,r_out]= getRoutRinFunc(desVar); % run function to get
 r_out, r_in
        Iyy = (pi/4).*(r_out.^4-r_in.^4); % calculate Iyy
    end

    function [x, w] = GaussHermite(n)
        % Function to determines the abscisas (x) and weights (w) for
 the
        % Gauss-Hermite quadrature of order n>1, on the interval [-
INF, +INF]. Credit to Geert Van Damme (geert@vandamme-iliano.be),
 please see referances section
        % Inputs:
        %   n - Gauss-Hermite quadrature of order, must be >1
        % Outputs:
        %   x - location of GQ points
        %   w - weight of GQ points

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

        i   = 1:n-1;
        sqrtNodesHalf   = sqrt(i/2); % (nuber of nodes /2)^(1/2)
        CMat  = diag(sqrtNodesHalf,1) + diag(sqrtNodesHalf,-1); % CMat
 created so that det(xI-CM)=L_n(x), with L_n the Hermite polynomial
        % Also, CM will be symmetrical.

        [V, L]   = eig(CMat); % get L, a diagonal matrix
        [~, indx] = sort(diag(L)); % sort and get indexes
        V       = V(:,indx)'; w = sqrt(pi) * V(:,1).^2; % get weights
        w=w./sqrt(pi); % adjust weight
    end

end
```

Published with MATLAB® R2018b

makePlots Function

By 6110

```
function [] = makePlots(sparGeo,~,~,wing,exp,var,~ )
%MAKEPLOTS Function to make plots
%   Makes colorful diagram and then a figure of stress and gemonetry
    r_low = sparGeo(1:wing.Nx);
    r_up = r_low + sparGeo(wing.Nx+1:2*wing.Nx);
    std = sqrt(var - exp.*exp);
    x = linspace(0,wing.L,wing.Nx)';
    hfig =  findobj('type','figure');
    nfig = length(hfig);
    cArr=[0.7098    0.8392    0.9686]; % color

    figure(nfig+1)
    hold on
    pup=area(x,r_up); % upper
    pup.FaceColor=[.84 .84 .84]; % color
    plow=area(x,r_low); % inside area
    plow.FaceColor=cArr; % color
    pNup=area(x,-r_up); % lower
    pNup.FaceColor=[.84 .84 .84]; % color
    pNlow=area(x,-r_low); % inside lower area
    pNlow.FaceColor=cArr; % color
    ylabel(' y/z axis (meters)'); xlabel('x axis (meters)')
    ylim([-0.05 0.05])
    grid on
    % title('Spar Diagram')

    figure(nfig+2)
    hold on
    plot([x(1),x(end)],[600*1e6,600*1e6],'-k')
    plot(x,exp,'-om')
    plot(x,exp+6*std,'-sr');     plot(x,exp-6*std,'-sg')
    xlabel('x axis (meters)'); ylabel('Stress (N/m^2) '); xlim([ 0
 7.5])
    grid on
    yyaxis right
    ylim([.01 .05])
    plot(x,r_low,'-+')
    plot(x,r_up,'b-x')
    % strT=sprintf('Spar shape vs Stress at %d nodes',wing.Nx);
 title(strT)
    legend('Maximum Stress','Mean Stress ($\mu$)','$
\mu +6\sigma$','$\mu-6\sigma$','Lower Radius','Upper
 Radius','Location','best','Interpreter','latex')
    ylabel('y/z axis (meters)')
end
```

Published with MATLAB® R2018b