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