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) CalcMass(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);
[ sprGeo,sprMss,exitFlag, fmincOutput ]  =
 fmincon(obj_Fun,wing.x0,wing.A,wing.b,[],[],wing.lb,[],nlcon,optns);

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


    function [pts, wts] = get_pts_wts(wing) %% This is imporatnt
        % This is done at the beginning, as it takes a long time to
 run
        % DO not use it in fmincon!!
        fnom0 = 2*wing.w/wing.L;
        mu = zeros(1,4);%
        sigma = fnom0*[1/10, 1/20, 1/30, 1/40];

        [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] = CalcStress(pts,wts,r_out,Iyy,wing)
    sigmaExp = 0; % prealloc
    sigmaVar = 0; % prealloc
    for n=1:(wing.NcolPts^4)
        forceBeam = CalcForce(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] = CalcMass(desVar,wing)

    mass = massFun(desVar,wing); % get mass
    Jacob = zeros(2*wing.Nx,1); % prealloc
    step = 1e-60; % step size

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

    function mass = massFun(desVar,wing) % mass function
        [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 constraing
    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
        step = 1e-60; % step size

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

        end
        Jacob = grad'; % return gradiant

        function c = ineqFun(desVar,wing,pts,wts) % function for c
inequality
            [r_out, ~, Iyy] = get_RO_RI_Iyy(desVar); % get r_out, Iyy
            [sigmaExp, sigmaVar] =
CalcStress(pts,wts,r_out,Iyy,wing); % run calcStress
            sigma = sqrt(sigmaVar - sigmaExp.*sigmaExp); % get stdv
            c = (sigmaExp + 6*sigma)./wing.Y - 1; % compute c
        end

    end

    function [force] = CalcForce(pts,wing) % compute force at points
        delt = 0; % prealloc
        x = linspace(0,wing.L,wing.Nx); % create linspace
        forceInit = (2*wing.w/wing.L)*(1-x./wing.L); % inital force
        for n=1:obj.NpertPts
            delt = delt + pts(n)*cos(((2*n-1).*pi.*x)/(2*wing.L)); %
from project discription
        end
        force = forceInit + delt; % return force
    end

    function [sigmaExp,sigmaVar,desVar,sigma,wts,pts] =
handleMakePlots(desVar,pts,wts,wing) % function gives everything
needed for plots
        [r_out, ~, Iyy] = get_RO_RI_Iyy(desVar); % get r_out, Iyy
        [sigmaExp,sigmaVar] = CalcStress(pts,wts,r_out,Iyy,wing);
        sigma = sqrt(sigmaVar - sigmaExp.*sigmaExp); % get stdv
    end

    function [r_in,r_out]= getRoutRinFunc(desVar) % function to get
r_out, r_in
        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
```

```
        [r_in,r_out]= getRoutRinFunc(desVar);
        Iyy = (pi/4).*(r_out.^4-r_in.^4);
    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].
        % works for n>=2
        % Credit to Geert Van Damme (geert@vandamme-iliano.be)
        % See referances section
        if n<2
            error('Warning Number of Collection points below 2');
        end

        i   = 1:n-1;
        a   = sqrt(i/2);
        CM  = diag(a,1) + diag(a,-1);

        % CM is such that det(xI-CM)=L_n(x), with L_n the Hermite
 polynomial
        % under consideration. Moreover, CM will be constructed in
 such a way
        % that it is symmetrical.
        [V, L]   = eig(CM);
        [x, ind] = sort(diag(L));
        V       = V(:,ind)';
        w       = sqrt(pi) * V(:,1).^2;
        w=w./sqrt(pi); % adjust weight
    end

end
```

Published with MATLAB® R2018b