ME 575
Homework #4: Computing Derivatives
Due Feb. 24 at 11:50 p.m. Rev 2.0

Note: The description of the assignment is subject to change slightly. However, I don't anticipate changes to the requested tasks. Indeed, I may add something on Automatic Differentiation. I am posting this now to provide it for this who wish to get started early.

Overview
For this assignment you will optimize a 10-bar truss problem where you will provide derivatives to the optimizer. Our goals are to learn several approaches for numerically computing derivatives, understand their advantages and disadvantages, and apply these methods to a problem to better understand the need for accurate gradients. You will use software already developed for truss analysis.

Truss Specification
Everyone will optimize the same 10-bar truss. This is a problem taken from the literature that is well known. The description of the problem (along with the solution) is given in the Appendix. The truss analysis software (***Truss.m***) reads in a data file (***Data.m***) that describes the particular problem. These files will be provided to you.

Truss Derivatives
Before we can optimize the truss we need to provide the following derivatives:
1) Derivatives of weight (objective) with respect to cross-sectional areas (design variables), i.e.

$$\frac{\partial w}{\partial a_i} \quad i = 1,2,...n$$

2) Derivatives of stress (constraints) with respect to cross-sectional areas, i.e.,

$$\frac{\partial \sigma_j}{\partial a_i} \quad i = 1,2,...n; j = 1,2,...m$$

There are *m* constraints (*m*=10), and *n* variables (*n*=10) so this will be a 10 by 10 matrix.

The truss analysis function is already provided that returns weight and stress for given cross-sectional areas (***Truss.m***). There is also a driver program to call this routine as a stand-alone program (***StandAloneTruss.m***), a data file which describes the 10-bar truss (***Data.m***), and a file (***OptimizeTruss.m***) that is already setup to optimize the truss. All files are available under Content > MATLAB Examples on Learning Suite. Executing ***OptimizeTruss.m*** will optimize the truss using ***fmincon*** and built-in derivatives. You will modify this file to add your own derivatives to the code.

You should compute the derivatives of the objective (weight) and the constraints (stress) using the following methods:

(a) A forward difference approach
(b) A central difference approach
(c) The complex-step approach

Truss Optimization

After you have developed the gradients, optimize the truss using ***fmincon***. Previously we have used the default solver, which is the Interior Point method. The ***fmincon*** optimizer has three other solvers, and you might want to experiment to find which works best on this problem. Solve this optimization problem, from the same starting point, using all the derivative methods.

Report on the following:
a) Is this a problem that could benefit from some simple scaling? Report on what you learn here.
b) Provide a listing giving your MATLAB code for each of the methods (just the code that shows the method and how implemented) with comments. Provide a brief explanation of how you integrated derivatives into the code and discuss any difficulties encountered. Indicate for each method how it is calculated and what the expected error is.
c) Evaluate the errors of the derivatives of the various methods at the starting point. Are these in line with expectations? How did you decide on a good perturbation? What are the relative merits of the approaches?
d) Provide a table giving,

- The number of function calls and iterations required by each derivative method to reach the optimum. You should keep track of function calls using the gobal variable ***"nfun"***
- If possible, report the time of execution. (I tried the "tic/toc" function of MATLAB but wasn't sure if this was giving accurate results. If you use this function, run it several times and average.)
- The stopping criterion given by MATLAB, i.e. why did it terminate, and the final objective value. Discuss any differences you observe.

Automatic Differentiation

Apply Automatic differentiation (AD) to the Spring design problem we did for Homework 1. (Note: you do not need to optimize it—just evaluate the functions once and show how you can get function values and derivatives simultaneously). Follow the example of the Two-bar truss, using the MATLAB AD approach given in the notes. You can copy the necessary AD functions (file ***valder.m***) from Content > MATLAB on Learning Suite.

Include the following:
a) Explain briefly how this method works.
b) Include your MATLAB code
c) Discuss how AD differs from other numerical methods.

Notes/Suggestions/Things to Watch Out For

- I would start with integrating derivatives into the stand-alone program and then move from there into integrating into the optimization.

- I would first provide derivatives for the objective function but let MATLAB do the constraints. Then move on to the constraints.
- Note that "i" cannot be used as an imaginary number and as a loop counter or array index in the same function
- I would spend some time reviewing how MATLAB handles the cases when you supply your own derivatives. In this regard, I have copied some relevant parts of documentation in the attached sheet.
- There is an option in ***fmincon*** to run a check on your gradients (see ***CheckGradients***). This allows you to not only check the accuracy of your derivatives but to make sure they are in the correct order.

  Note that for this I set the finite difference type to central (in the code below) so we have a more accurate derivative as a check on our derivatives. Otherwise we have a relatively inaccurate method (forward difference) as the standard against which our derivatives are checked.

  Options:

```
'SpecifyObjectiveGradient',true,'SpecifyConstraintGradient',
true,...
'FiniteDifferenceType','central','CheckGradients',true)
```

- Absolute value of constraints issue
  I took the absolute value of the stresses as a way to check that both compressive (negative) and tensile (positive) stresses were less than the allowed value, and the optimizer took these just fine (see code in ***OptimizeTruss.m***). That is, the constraints have both upper and lower allowable values:

$$-25000 \le Stress \le 25000$$

  However, the absolute value function is not defined for complex variables, and this caused my derivatives to come back as zero. So you will need to change how these constraints are handled, at least for that method.

  I could have changed this in the original function, but chose not to so you could see the issues that come up.

Step 1: Write a file for the objective function and gradient.

```
function [f,gradf] = objfungrad(x)
f = exp(x(1))*(4*x(1)^2+2*x(2)^2+4*x(1)*x(2)+2*x(2)+1);
% Gradient of the objective function:
if nargout  > 1
    gradf = [ f + exp(x(1)) * (8*x(1) + 4*x(2)),
    exp(x(1))*(4*x(1)+4*x(2)+2)];
end
```

Step 2: Write a file for the nonlinear constraints and the gradients of the nonlinear constraints.

```
function [c,ceq,DC,DCeq] = confungrad(x)
c(1) = 1.5 + x(1) * x(2) - x(1) - x(2); % Inequality constraints
c(2) = -x(1) * x(2)-10;
% No nonlinear equality constraints
ceq=[];
% Gradient of the constraints:
if nargout > 2
    DC= [x(2)-1, -x(2);
        x(1)-1, -x(1)];
    DCeq = [];
end
```

`gradf` contains the partial derivatives of the objective function, `f`, returned by `objfungrad(x)`, with respect to each of the elements in `x`:

$$\nabla f = \begin{bmatrix} e^{x_1}\left(4x_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1\right) + e^{x_1}\left(8x_1 + 4x_2\right) \\ e^{x_1}\left(4x_1 + 4x_2 + 2\right) \end{bmatrix}. \quad \text{(6-58)}$$

The columns of `DC` contain the partial derivatives for each respective constraint (i.e., the `i`th column of `DC` is the partial derivative of the `i`th constraint with respect to `x`). So in the above example, `DC` is

$$\begin{bmatrix} \frac{\partial c_1}{\partial x_1} & \frac{\partial c_2}{\partial x_1} \\ \frac{\partial c_1}{\partial x_2} & \frac{\partial c_2}{\partial x_2} \end{bmatrix} = \begin{bmatrix} x_2 - 1 & -x_2 \\ x_1 - 1 & -x_1 \end{bmatrix}. \quad \text{(6-59)}$$

Since you are providing the gradient of the objective in `objfungrad.m` and the gradient of the constraints in `confungrad.m`, you *must* tell `fmincon` that these files contain this additional information. Use `optimoptions` to turn the options `SpecifyObjectiveGradient` and `SpecifyConstraintGradient` to `true` in the example's existing `options`:

```
options = optimoptions(options,'SpecifyObjectiveGradient',true,'SpecifyConstraintGrad
```

If you do not set these options to `'on'`, `fmincon` does not use the analytic gradients.

The arguments `lb` and `ub` place lower and upper bounds on the independent variables in `x`. In this example, there are no bound constraints, so set both to `[]`.

Step 3: Invoke the constrained optimization routine.

```
x0 = [-1,1];            % Starting guess
options = optimoptions(@fmincon,'Algorithm','sqp');
options = optimoptions(options,'SpecifyObjectiveGradient',true,'SpecifyConstraintGrad
lb = [ ]; ub = [ ];   % No upper or lower bounds
[x,fval] = fmincon(@objfungrad,x0,[],[],[],[],lb,ub,...
   @confungrad,options);
```

Appendix
Listing of Data.m:

```
%Ten bar truss of Venkaya
E = 1.e7;
ndof = 12;
nbc = 4;
nnode = 6;
nelem = 10;

% format is coordinate, then degree of freedom numbers
Node = [720 360 1 2; 720 0 3 4; 360 360 5 6; 360 0 7 8;...
    0 360 9 10; 0 0 11 12];

% format is for each degree of freedom (horizontal then
vertical)
force = [0; 0; 0; -100000; 0; 0; 0; -100000; 0; 0; 0; 0];

% put boundary conditions in reverse order, highest to lowest
bc = [12, 11, 10, 9];

% format is start node to end node, area
Elem = [5 3 5; 3 1 5; 6 4 5; 4 2 5; 3 4 5; 1 2 5; 5 4 5; ...
    6 3 5; 3 2 5.; 4 1 5];
```

Listing of OptimizeTruss.m:

```
 % ------------Starting point and bounds------------
    %design variables
    x0 = [5, 5, 5, 5, 5, 5, 5, 5, 5, 5]; %starting point (all
areas = 5 in^2)
    lb = [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1];
%lower bound
    ub = [20, 20, 20, 20, 20, 20, 20, 20, 20, 20]; %upper
bound
    global nfun;
    nfun = 0;

    % ------------Linear constraints------------
    A = [];
    b = [];
    Aeq = [];
    beq = [];

    % ------------Call fmincon------------
```

```
    tic;
    options = optimoptions(@fmincon,'display','iter-
detailed','Diagnostics','on');
    [xopt, fopt, exitflag, output] = fmincon(@obj, x0, A, b,
Aeq, beq, lb, ub, @con, options);
    eltime = toc;
    eltime
    xopt    %design variables at the minimum
    fopt    %objective function value at the minumum
    [f, c, ceq] = objcon(xopt);
    c
    nfun

    % ------------Objective and Non-linear
Constraints------------
    function [f, c, ceq] = objcon(x)
        global nfun;

        %get data for truss from Data.m file
        Data

        % insert areas (design variables) into correct matrix
        for i=1:nelem
            Elem(i,3) = x(i);
        end

        % call Truss to get weight and stresses
        [weight,stress] = Truss(ndof, nbc, nelem, E, dens,
Node, force, bc, Elem);

        %objective function
        f = weight; %minimize weight

        %inequality constraints (c<=0)
        c = zeros(10,1);          % create column vector
        for i=1:10
            c(i) = abs(stress(i))-25000; % check stress both
pos and neg
        end

        %equality constraints (ceq=0)
        ceq = [];
        nfun = nfun + 1;

    end
```

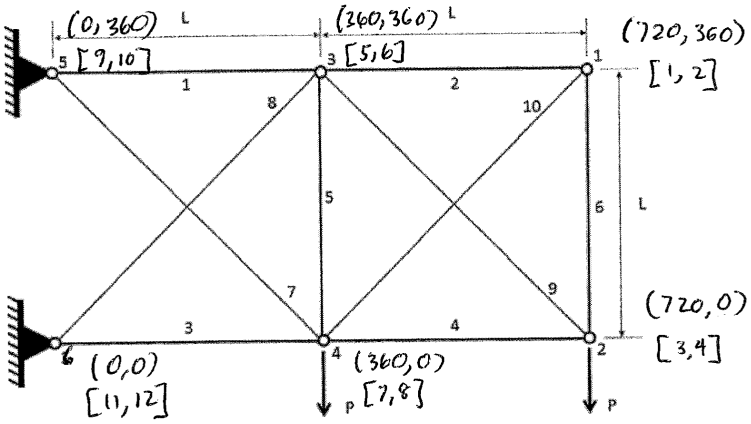
```
% ------------Separate obj/con (do not change)------------
function [f] = obj(x)
    [f, ~, ~] = objcon(x);
end
function [c, ceq] = con(x)
    [~, c, ceq] = objcon(x);
end
```

Ten Bar Truss Sizing Optimization

The ten bar truss model was developed by Venkayya for:
Venkayya, V. B. (1971). "Design of Optimum Structures," Computers & Structures, 1(1), 265-309.

The design problems for this page are described in:
Haftka, Raphael T. Elements of Structural Optimization. Springer, 1992. Pages 238, 244.

Example CoFE input files for this page are provided here:
NASTRAN_CoFE/CoFE_examples/o1_tenbarOpt/



L = 360"
P = 100 Kips

Details

- Young's modulus: 10^7 psi
- Specific mass: 0.1 lbm/in^3
- Minimum area: 0.1 in^2
- Allowable stress: +/- 25000 psi
- The ten design variables are the areas of the ten members.

Design Problem 1

Design for minimum weight with stress constraints.

Design Variable	Design from CoFE + SQP	Design in Haftka
x_1	7.94	7.94
x_2	0.1	0.1
x_3	8.06	8.06
x_4	3.94	3.94
x_5	0.1	0.1
x_6	0.1	0.1
x_7	5.74	5.74
x_8	5.57	5.57
x_9	5.57	5.57
x_{10}	0.1	0.1

Objective Function	Design from CoFE + SQP	Design in Haftka
Weight	1593.2 lb	1593.2 lb