Optimization Homework #4

Ryan Day

February 21, 2019

1 Truss Optimization

1.1 Scaling

As far as I can tell, scaling the constraints would be useful if the constraints were an order of magnitude higher than the design variables. The design variables are all around the same order of magnitude,, so I don't see the use of scaling there.

1.2 Matlab Code implementation

The matlab code is included below the table. I used a function that took in x, then perturbed the function with a step depending on which type (forward, central, or complex), and then calculated the result. It was not too hard to implement this function. I simply added it to the end of obj and con in order to get the derivatvies. However, I had to make one change in order to get the complex function to work. This change was changing the inequality constraint from using the abs function to taking the square root of the value squared. This did the same thing as abs but could be used with complex variables.

1.3 Expected Errors of the derivatives

I expected the errors of the derivative to be greatest for the forward, then central, then complex. The merits of the forward method is it is only takes one function call per derivative. Central method is slightly more accurate, but takes twice as many function calls. Both of these methods have subtractive error which makes it so you can't have the step size be too small. The complex step method does not have this subtractive error which makes it so you can have an extremely small step size, but then you have to make sure your function can handle complex numbers. In addition to this, the computations of the complex step can take longer than the forward step because of the included complex numbers. I decided on the perturbation 1e-6 because this would make it fairly accurate with each method. I could have lowered the perturbation for the complex step to make it even more accurate since complex does not have error from losing significant digits in subtraction, but I wanted to compare each method with the same perturbation. Because of this I used 1e-6 for each derivative approximation method.

1.4 Table and stopping criteria

	# Function calls	# Iterations	Avg Time execution	Final Objective value
No Derivatives supplied	495	19	0.573	1.5932e+03
Forward method	991	19	0.6320	1.5932e+03
Central method	3139	39	1.114	1.5932e+03
Complex method	991	19	0.932	1.5932e+03

The execution time was fastest with fmincon with no supplied derivatives because they only call the derivative function once instead of twice. The forward method and complex method had the same number of function calls as expected since they both call the objective function just once to calculate derivatives. The complex method took longer than the forward method because it had to deal with complex numbers. The central method took about twice as many function calls and so took a lot longer than any other method.

Stopping Criterion for no derivatives supplied:

Optimization stopped because the relative changes in all elements of x are less than options.StepTolerance = 1.000000e-10, and the relative maximum constraint violation, 0.000000e+00, is less than options.ConstraintTolerance = 1.000000e-06.

Stopping Criterion for other methods:

Forward:

Optimization completed: The relative first-order optimality measure, 3.928372e-08, is less than options.OptimalityTolerance = 1.000000e-06, and the relative maximum constraint violation, 0.000000e+00, is less than options.ConstraintTolerance = 1.000000e-06.

Central:

Optimization completed: The relative first-order optimality measure, 7.607382e-07, is less than options.OptimalityTolerance = 1.000000e-06, and the relative maximum constraint violation, 0.000000e+00, is less than options.ConstraintTolerance = 1.000000e-06.

Complex:

Optimization completed: The relative first-order optimality measure, 1.512774e-09, is less than options.OptimalityTolerance = 1.000000e-06, and the relative maximum constraint violation, 0.000000e+00, is less than options.ConstraintTolerance = 1.000000e-06.

The stopping criterion was only different for the fmincon function w

```matlab

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

options = optimoptions(@fmincon,'display','iter-detailed','Diagnostics','on',...
    'SpecifyObjectiveGradient',true,'SpecifyConstraintGradient',true);
[xopt, fopt, exitflag, output] = fmincon(@obj, x0, A, b, Aeq, beq, lb, ub, @con,
    options);

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) = sqrt((stress(i))^2)-25000; % check stress both pos and neg
        end

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

    % ------------Separate obj/con You may wish to change------------
    function [f, grad] = obj(x)
        [f, c, ~] = objcon(x);
        [grad,~] = findGrad(x,f,c);
    end
    function [c, ceq, cgrad, ceqgrad] = con(x)
        [f, c, ceq] = objcon(x);
        [~, cgrad] = findGrad(x,f,c);
        ceqgrad = ceq;
    end

    function [grad, cgrad] = findGrad(x,fo,co)
        % Define method of numerical differentiation
        type = "a"; % "forward", "central", or "complex"
        % Define step size
        h = 1e-6;

        [~,sizex] = size(x);
        grad = zeros(sizex,1);
        [nc,~] = size(co);
        cgrad = zeros(nc,nc);
        for index=1:sizex
            if (type=="forward" || type=="central")
                xf = x;
                xf(index) = x(index) + h;
                [f_f,c_f,~] = objcon(xf);
                if(type=="forward")
                    grad(index) = (f_f-fo)/h;
                    for j = 1:nc
                        cgrad(index,j) = (c_f(j)-co(j))/h;
                    end
                else
                    xb = x;
                    xb(index) = x(index) - h;
                    [f_b,c_b,~] = objcon(xb);
                    grad(index) = (f_f-f_b)/(2*h);
                    for j = 1:nc
                        cgrad(index,j) = (c_f(j)-c_b(j))/h;
                    end
                end
            else
                xI = x;
                xI(index) = x(index)+1j*h;
                [f_im,c_im,~] = objcon(xI);
                grad(index) = imag(f_im)/h;
                for  k= 1:nc
                    cgrad(index,k) = imag(c_im(k))/h;
                end
            end
        end
    end
```