ME46060 “Engineering Optimization – Concepts and Applications” Exercise 4

Subject: “Valve spring”

Objectives

- Solution of an unconstrained problem by means of a self made steepest descent algorithm using the Matlab line search algorithm fminbnd. Visualization of the optimization process.
- Learn how to use Matlab algorithm fminunc to solve unconstrained problems.

Course material

- Sections 4.5 and 4.6 of Chapter 4, Section 6.3 of Chapter 6 of Papalambros and Wilde [1]

Note: In [1] the “steepest descent method” is called “gradient method”.

Introduction

This exercise considers a similar **unconstrained** optimization problem as in exercise 3 (now the components of the objective function are squared instead of absolute valued). That is the valve spring involving two design objectives. On the one hand we want to come as close as possible to a specified value for the spring stiffness, and on the other hand we want to reach a specified value for the lowest eigenfrequency of the spring. Mathematically the problem is defined as follows:

Minimize $f(\mathbf{x}) = ((k-ktarget)/ktarget)^2 + w*((freq1-frtarget)/frtarget)^2$ (1)

Subject to the set constraint:

$0.02 \le x(1) \le 0.04$ (m)

$0.002 \le x(2) \le 0.005$ (m)

where:

- k = the axial spring stiffness
- *ktarget* = target value for the spring stiffness (= 10000 N/m).
- *freq1* = lowest eigenfrequency
- *frtarget* = target value for lowest eigenfrequency (= 300 Hz).
- w = weighing factor between the two objective function components.

Row vector **x** represents the design variables, being the coil diameter D (= x(1)) and the wire diameter d (= x(2)) respectively. The other parameters are given in the file springparam1.m.

The problem is visualized using the file springw4ex1.m (generating Figure 1, valid for w = 1.0). It can immediately be seen that the solution of problem (1) is at the intersection of the contours k = 10000 N/m and *freq1* = 300 Hz. Last week we applied line searches within the design space along **given** search directions. This week we regard **computed** search directions, which are applied step by step and in a loop.
The line search is defined as:

Minimize $f(alpha)$ (2)

subject to $\mathbf{x} = \mathbf{x_q} + alpha*\mathbf{S_q}$, $0 \le alpha \le alpha_{max}$

where:

- $\mathbf{x_q}$ = row vector representing the initial point of the line search.
- $\mathbf{S_q}$ = row vector representing the line search direction.
- *alpha* = scalar representing the step size along the search direction.
- $alpha_{max}$ = upper bound of alpha such that the design point **x** does not violate the set constraint in (1).

The function *f*(*alpha*) is computed by the (given) file springobjw4.m. In this function *alpha* is the (one-dimensional) design "vector"; *xq*, *sq*, *ktarget*, *frtarget* and *w* are extra input parameters.
Note: springobjw4.m uses the given files springanalysis1.m and springparams1.m.

In the exercises 4.1 and 4.2 we use one dimensional line searches (as described above) to build a steepest descent optimization process. In exercise 4.3 the Matlab algorithm fminunc is used to solve the problem.

Exercise 4.1: Steepest descent line searches using the algorithm fminbnd

In springw4ex1.m the point $\mathbf{x_q}$ = [0.022 0.0035] is used as initial point. Run springw4ex1.m and carry out some steepest descent line searches. Look at the result in Figure 1, and (in the Matlab command window) at the applied search directions (sq) and the stepsize (alphaq). Do you trust the results?

The remedy is to scale the search vector. Implement the following scaling in springw4ex1 (name the modified program springw4an1.m): first normalize the search vector to unity length, and next scale it to a reasonable length in the D-d-design space. Furthermore, choose a proper upper bound for alphaq as input variable of fminbnd.

Rerun the program for several steps and interpret the results. Why are the subsequent steepest descent search directions in the picture not orthogonal (as it should be)? You see a zigzag optimization path; can you explain that?

Exercise 4.2: Build your own steepest descent optimization algorithm.

Extend the program springw4an1 to springw4ex2 in order to automate the optimization process. Therefore modify the while-loop to a while-loop over a specified maximum number of optimization cycles. Also implement (within the loop) the following convergence criterion: the optimization process should be terminated if the (absolute) difference between the current objective function value and the previous value is smaller than 1.0e-6.

Run your program and interpret the results. How many optimization cycles are necessary for convergence?

Exercise 4.3: Solution the unconstrained problem using the algorithm fminunc.

Write a program named springw4ex3.m which uses the Matlab algorithm fminunc to solve the problem of the previous exercises. For computation of the objective function use the (given) file s_objw43.m (why cannot we use springobjw4.m as we did in the exercises 4.1 and 4.2?). Use the anonymous function approach discussed in Exercise 3 to pass the additional parameters for s_objw43.m. Use the problem visualization from springw4ex1.m. However, now the optimization path cannot be visualized. Only plot the initial point and the final solution in Figure 1. Adjust the algorithm for **medium-scale optimization** and **BFGS Hessian update** (see documentation). Also set option 'Display' to 'iter'. Check the default value of option 'Tolfun'.

Run the program and interpret the results.

Now modify 'Hessupdate" to 'steepdesc', and carry out a steepest descent optimization run. Compare the results with those of your own steepest descent program made in exercise 4.2.

References

[1] Papalambros, P.Y. and Wilde, D.J., *Principles of optimal design: modeling and computation*, 3rd edition, Cambridge University Press, 2017.