Numerical Optimization - Project 1, Phase 2 (team/group phase)

Deadline: Sunday, June 12, 23:59. (The deadline can be extended, but we will start with Project 2 soon)

General description:

The overall task of Project 1 is to program the four methods for unconstrained optimization: steepest descent, Newton method, a conjugate gradient method and a quasi-Newton method (we will cover conjugate gradient and quasi-Newton methods in the next lectures).

It is up to you whether you choose the line search or the trust region approach.

It is also up to you which of the (nonlinear) conjugate gradient methods and which of the quasi-Newton methods you do.

In the second phase, working in teams, you should improve your programs of the four methods by considering the following issues (all the references are to the Nocedal Wright book):

1. **stopping criteria: (4 points/10 percent)** usually, the convergence theory says what happens when an infinite number of iterates is produced. Obviously, we need to stop our algorithm at some point. The theory, however, suggest how to choose a reasonable stopping criterion. If x_k converges to a stationary point x^* of f, we would expect e.g. the following quantities to be small if we are near the solution:

$$|f(x_{k+1}) - f(x_k)|, \quad \|x_{k+1} - x_k\|, \quad \|\nabla f(x_k)\|. \tag{1}$$

   Thus, one can stop the algorithm if one (or more) of these quantities are below a chosen tolerance (e.g. $\varepsilon = 10^{-5}$).
   If f is multiplied by a number α, however, then $\|\nabla \alpha f(x_k)\| = \alpha \|\nabla f(x_k)\|$, and so the corresponding stopping criterion for αf reads e.g. $\|\nabla f(x_k)\| \leq 1$ if $\alpha = 10^{-5}$ and $\|\nabla f(x_k)\| \leq 10^{-10}$ if $\alpha = 10^5$. The same problem applies to $|f(x_{k+1}) - f(x_k)|$. Thus, one can make the stopping criteria relative e.g. by asking for $\|\nabla f(x_k)\| \leq \varepsilon \|\nabla f(x_0)\|$ or $\|\nabla f(x_k)\| \leq \varepsilon(1 + \|\nabla f(x_0)\|)$ and similarly for the others.
   Additionally, one can also impose a bound on the maximal number of iterations (e.g. 10^4 - 10^5) and/or the maximal number of function evaluations.

2. **linear systems: (2 points/5 percent)** suppose you need to solve a linear system $Ax = b$ during the algorithm (e.g. for Newton method). If A is non-singular, the solution is obviously given by $A^{-1}b$. Computing the inverse matrix A^{-1}, however, should be always avoided! Instead, one should use a suitable solver for linear systems which utilizes a factorization of matrix A, see Appendix A, page 606.

3. **derivatives: (6 points/15 percent)** what if the user only provides the objective function in a way that we can evaluate it at any point, but there is no formula defining the function and we do not know the derivatives? In Section 8.1, you can find the methods for approximating the gradient and the Hessian of a function, using only its function values.

4. **line search (initial step length): (2 points/5 percent)** See INITIAL STEP LENGTH (page 59) for suggestions how to choose the initial step length α_0.

5. **line search (the Wolfe conditions): (6 points/15 percent)** See A LINE SEARCH ALGORITHM FOR THE WOLFE CONDITIONS (page 60) and implement Algorithms 3.5 and 3.6.

6. **Newton method with Hessian modification: (6 points/15 percent)** see Section 3.4 - you can choose one of the suggested strategies.

7. **implementation of quasi-Newton methods: (2 points/5 percent)** see IMPLEMENTATION (page 142) for suggestions how to choose the initial H_0.

8. **efficient computation of quantities: (2 points/5 percent)** This is not defined very precisely, but you should always be careful whether you compute the quantities you are using efficiently. For instance, if you are re-using a quantity several times, you should not computed it repeatedly (e.g. during the line search). If you compute more complicated vector-matrix products (e.g. in quasi-Newton methods), you should think about the order in which you multiply the factors.

Score: You will get 20 points/50 percent for the "basic" submission (like in the individual phase - 4 methods that work correctly) and you can get additional points for implementing the above suggestions (up to 30 points/75 percent).

Maximal amount of the points you can get for the Project is 40 plus up to 3 bonus points, so 43 together. The bonus points can be awarded also for other reasons.

Problems to solve:

(i) 5 problems corresponding to least-squares problems (like in the individual phase - you can choose the same problems some of your teams members solved in the individual phase or you can choose different problems).

(ii) The following two functions:

$$\text{(Rosenbrock function)} \quad f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \tag{2}$$

with 5 different starting points x_0:

$$(1.2, 1.2), \quad (-1.2, 1), \quad (0, 1), \quad (-1, 0). \quad (0, -1)$$

and

$$f(x) = 150(x_1 x_2)^2 + (0.5x_1 + 2x_2 - 2)^2 \tag{3}$$

with 5 different starting points x_0:

$$(-0.2, 1.2), \quad (3.8, 0, 1), \quad (0, 0), \quad (-1, 0). \quad (0, -1).$$

What should your submission contain?

All the files containing the codes and a report.

In the beginning of the report, state clearly which of the suggestions 1. - 8. you have implemented.

Then, print all 60 runs you performed: 4 methods times (5 least-squares problems, 5 starting points for (2) and 5 starting points for (3) = 15 "problems").

After each run, provide a short summary with number of iterations, final iterate, which stopping criterion actually stopped the algorithm and the values of quantities from (1), etc (distance to a "solution" x^* for problems where you know a solution).

Moreover, for the first 5 problems (least squares), as before (individual submission), specify the function g, degree n of the polynomial and the final optimal polynomial and depict the data points (a_j, b_j) and the graphs (function g, its Taylor expansion of degree n, and the optimal polynomial).