Numerical optimisation – Practice exercises

Zs. Németh and L. Lócsi

May 20, 2020

Exercises worth 1 point each. Each student is assigned to solve 4 exercises from both parts. A minimum of 2+2 points is required to pass; and the practice mark is based on the total score: 4 points = passed(2); 5 points = average(3); 6 points = good(4) and 7-8 points = excellent(5).

Part I

1. Suppose that the units of the plane $\mathbb{R}^2$ represent kilometers. From our country house at $(0,0)$, we want to go to the mart at point $(10,10)$ (to buy surgical mask and toliet paper), taking minimal time. There are three kinds of terrain on the way. The area

$$F = \{(x,y) : 2 < x < 5, y \in \mathbb{R}\}$$

is covered by forest, where we can move at a reduced speed of 3 km/h. Then,

$$S = \{(x,y) : 5 \le x \le 7, y \in \mathbb{R}\}$$

is a swamp, where passage is only possible at 1 km/h. All other terrain is regular grassy area N, where we can proceed at 5 km/h.
Formalize the underlying optimization problem, i.e. introduce variables and constraints, explain what they represent and give an objective function (i.e. the time we take) to minimize. Which known algorithm would you recommend to solve the problem with? Give the answer to this problem rounded to minute precision.

2. Suppose that there are some given number $m \in \mathbb{N}$ of immobile electric charges at some points of the plane. We want to place some given $n \in \mathbb{N}$ number of further charged particles (not including the fixed charges) along a circle centered at the origin, in such a way that the sum potential of the system is minimized.
Supposing that all charges are equal, formalize this optimization problem. Which known algorithm would you recommend to solve the problem with? Demonstrate an approximate solution for a parameter setting of your choice.

3. In Epidemistan the new Tiara Bacteria is claiming its victims, and scientists have found out that the most effective way to prevent the disease: to have the input of vitamins F, G, H and I for the inhabitants of the country as balanced as possible (i.e. the same amount of each). Now these vitamins are present in three fruits of the country: abbles, balalas and caranbolas. One pound of abbles contain no vitamin F, but 1 unit of vitamin G and I, and 5 units of vitamin H. One pound of balalas contain 4 units of vitamin F, 2 units of vitamin G and I, but no vitamin H. And one pound of caranbolas contain 1 units of both vitamins F and H, and 3 units of vitamins G and I. What would be the optimal proportion of these fruits to eat each day, given that a person is only advisied to consume one pound of fruit cocktail? Formalize the question as an optimization problem. Which known algorithm would you recommend to solve the problem with?

4. Implement a program for the visualization of the Armijo line search method in case of $\mathbb{R} \to \mathbb{R}$ functions. Show the effect of the parameters c and ϱ, plot the objective function, the line corresponding to c and the points examined by the algorithm. Find some nice test functions.

5. Modify the implementation of the gradient descent method, such that the plots also include the further points examined by the line search method underneath, and also the examined line should be visualised.

6. * Implement a program that plots a given $\mathbb{R}^2 \to \mathbb{R}$ function as a 3D surface, and its second order approximations (Taylor-polynomials) at a given point, step-by-step in each step of Newton's method (without line search and the positive definite correction, i.e. based on `Newton_noLS.m`).

7. * Implement the unimodal Fibonacci line search algorithm, an improvement to the zero-order unimodal line search algorithm (`zero_unimodal.m`).

8. Implement the unimodal first-order bisection line search algorithm.

9. Implement a non-unimodal line search method, which terminates when a point satisfying the Wolfe conditions is found. You may use the code for the Armijo conditions (`Armijo_LS.m`) as reference.

10. Modify the gradient descent algorithm (`grad_descent.m`) to approximate a local minimizer along the search direction at each iteration (instead of a point satisfying "only" the Armijo conditions). You may assume that the objective is differentiable any number of times.

11. * Implement a system of linear equations solver to iteratively approximate the solution of $Ax = b$, ($A \in \mathbb{R}^{n\times n}$ invertible, $b \in \mathbb{R}^n$) using the gradient descent algorithm (`grad_descent.m`) on a suitable objective. If possible, also try to avoid using line search by moving to the exact directional minimum at each iteration instead.

12. Implement a modified multivariable Newton method (based on `Newton.m`), which corrects the non positive definite Hessian by computing its LDL^T decomposition and changing negative and very small diagonal entries to some fixed positive δ value.

13. Implement the Polak–Ribiere conjugate gradient method. You may use the code for the Fletcher–Reeves conjugate gradient algorithm (`FR_conj_grad.m`) as reference.

14. * Implement a system of linear equations solver to iteratively compute the solution of $Ax = b$, ($A \in \mathbb{R}^{n\times n}$ symmetric positive definite, $b \in \mathbb{R}^n$) using the conjugate gradient algorithm on a suitable objective.

Part II

1. Implement the Broyden–Fletcher–Goldfarb–Shanno quasi-Newton method using exact line search. You may use the code for the Davidon–Fletcher–Powell method (`QN_DFP.m`) as reference.

2. * Implement the Broyden method using Wolfe linesearch. Be aware of numerical stability in your solution.

3. Implement the second order finite differencing method for approximating Hessians, then modify the implementation of Newton's method (`Newton.m`) such that it only relies on function values to approximate the required gradient and Hessian values.

4. Implement the derivative-free conjugate directions method in the general case (i.e. for arbitrary number of variables).

5. * Implement a constrained barrier method with logarithmic barrier. Make it able to explicitly compute the gradients and Hessians of the modified objectives based on the exact gradients and Hessians of contraint functions g_i, given as input, and use Newton's method as the unconstrained subroutine. You may use the code for the Carrol barrier (`barrier_path.m`) as reference.

6. Modify our implementation of the Carrol barrier method (`barrier_path.m`) to use a quasi-Newton method of your choice as an unconstrained subroutine. Be careful about not leaving the feasible region.

7. Modify our implementation of the Carrol barrier method (`barrier_path.m`), enabling it to find solutions closer to/right on the boundary of the feasible region: when finite differencing the Hessian, for each canonical direction check if forward differencing would evalueate the gradient outside, and if yes, try using a backward difference approximation for that column.

8. Extend the penalty method implementation `penalty_path.m`, making it able to solve generally contrained problems (GCP).

9. Implement a linear least squares solver utilizing QR decomposition. Improve the numerical stability of the method for the case of $r_{i,i} \approx 0$ entries appearing in the diagonal of the upper-triangle matrix. You may use the code for the SVD solution (`linear_LSq.m`) as reference.

10. Implement a function for fitting polynomials on a sample: the input should be the dataset $(t_j, y_j),\ j = 1 \dots m$, and the degree of the polynomial; the expected output is the coefficient vector of the polynomial and a plot of the result.

11. * Formalize the problem of fitting a circle to a noisy sample of m points on the $\mathbb{R}^2$ plane. (or in $\mathbb{C}$, if you wish) as a nonlinear least squares problem, and implement a modification of the Gauss–Newton algorithm (`Gauss_Newton.m`) to solve it. The input should be the set of sample points, and the expected output is the center and radius of the resulting circle.

12. Implement a function to visualize the Carrol barrier's effect used in inequality constrained optimization problems for functions of two variables. The input should be the objective function f, the constraints g, and the parameter ϱ. The function should create a 3D plot of

the original and the scaled objective functions. Provide some samples with bounded feasible regions (e.g. triangle, rectangle, circle or some other nice region of your choice).

13. Implement a function to visualize the square penalty's effect used in equality constrained optimization problems for functions of two variables. The input should be the objective function f, the constraints g, and the parameter ϱ. The function should create a 3D plot of the original and the modified objective functions. Provide some samples (e.g. optimizing on a line, on a circle or some other nice region of your choice).

14. Implement a function to visualize the optimization progress of the (derivative free) coordinate descent method while applied to a function of 3 variables. The input should be the objective function and the starting point. A 3D plot should be created with the points found in the optimization steps (subsequent points should be connected with line segments). Plotting the function values is not required, visualize only the trajectory of the approximate solution in the $\mathbb{R}^3$ domain. Provide a nice example where the coordinate directions are observable. You may use the code of `coord_descent.m` as a reference.