Optimization Labs 1, 2, and 3

1 Forewords

These Labs will be split in 3 labs of 2 hours, spread over the three two weeks of January (be careful with the time management with exams and defenses). The objective is fairly easy: given "simple" optimization problems, implement several strategies and compare them. You can use Matlab or C++ to your convenience. You can work alone, or by group up to three students at most.

You will be evaluated on the quality of the deliverables, including:

1. Technical report in pdf format in which you justify, explain, and illustrate your approaches.
2. Commented source code

Important information: the deadline is on Monday, January the 21th (last week of the semester - week OFF for you to arrange mobility issues). Please notice that you will not be able to complete these labs if you do not prepare them a minimum at home. Consequently, arrive in labs prepared (prepare your questions and suggestions) and use these two hour slots to discuss some of them with the professor and experiment ideas.

2 Function minimization in $1D$

Before working with n dimensional functions, we first need to equip ourselves with a couple of routines for $1D$ minimization. Do not worry much about the function you consider: what is important here is more the implementation and analysis of your routines rather than finding a "sufficiently" complicated function to make some methods to crash. For instance, you could consider $f(x) = -e^{-x^2} + e^{-(x+1)^2}$; high degree polynomials can also be convenient (you know the roots, easy derivatives) for methods requiring the derivatives.

Implement, at least, the following routines to determine a local minimum x^* of a function $f(x)$ over a given interval I:

1. "Brute force approach": high density sampling of the interval, then pick up the minimum: horrible and inefficient method, but it works...
2. Golden Search Method
3. Brent's method
4. Gauss-Newton's method

3 Function minimization in $2D$

3.1 Minimization of a quadratic form

You can consider any quadratic function for this exercice. Implement, illustrate and comment, minimization routines using:

1. Arbitrary line search
2. Steepest descent method
3. Powell's method
4. Conjugate gradients

3.2 Minimization of a non quadratic form

There exist well known test functions for unconstrained optimization algorithm, see
`http://en.wikipedia.org/wiki/Test_functions_for_optimization`.

For this exercice, we will consider the Goldstein-Price function defined on the domain $[-2,2]^2$ as:

$$\begin{aligned} f(x,y) = &\left(1+(x+y+1)\left(19-14x+3x^2-14y+6xy+3y^2\right)\right). \\ &\left(30+(2x-3y)^2\left(18-32x+12x^2-48y+36xy+27y^2\right)\right) \end{aligned} \tag{1}$$

This function admits a minimum at $(0,-1)$ for which $f(0,-1)=-3)$.

Implement, illustrate and comment, minimization routines using for the Goldstein-Price function using:

1. Arbitrary line search
2. Steepest descent method
3. Any Quasi-Newton method of your choice (Broyden, DFP, BFGS)

4 Function minimization in *nD*

Now, we just have tried to determine a local minimum of "simple" functions in one or two dimensions. It is very common to try to determine the best set of parameters of a given curve which best represent some real data: welcome to the wonderful world of data fitting...

We are going to work in two phases. First we will generate various data sets(with noise, missing parts, etc) for which we will know the parameters used to generate them: this will be our groundtruth. In the second part, we will then try to determine the optimal parameters which best fit those generated data.

We need to have a function with sufficient number of parameters n, let say $n > 4$, though we would like to have a function which is sufficiently understandable for this work to be "geometrically" meaningful and interpretable. One class of curves which is simple enough and for which there exist numerous techniques is the ellipse.

4.1 Groundtruth data creation

An ellipse centered at the origin without rotation can be defined in polar coordinates as:

$$\begin{pmatrix} x(\theta,a) \\ y(\theta,b) \end{pmatrix} = \begin{pmatrix} a\cos(\theta) \\ b\sin(\theta) \end{pmatrix}, \tag{2}$$

where a and b represent the length of its semi axis.

1. What is the parametric representation of an ellipse rotated by an angle θ_0 then translated by a vector $T = (T_x, T_y)$?
2. Implement a routine to generate n points randomly sampled on such ellipse
3. Upgrade your previous routine to add noise to the data

4.2 Data fitting using Levenberg-Marquardt

Now, you are able to generate sets of noisy 2D point clouds which correspond to ellipses. Assuming that we are searching for a rotated and translated ellipse, we are now going to try to determine the parameters which best fit the previously created data. In order to do so, we first need to identify all the parameters to be retrieved.

1. What are the parameters to be determined?
2. How can you characterize the notion of "good fit"? In other words, propose one cost function to be optimized.
3. Implement Levenberg-Marquardt algorithm. Analyse the behavior of the algorithm (noise, initial guess, internal parameters, approximate or exact derivatives, etc).

5 Annexes, tricks, and suggestions

1. Be sure that your functions $f_k(x)$ return not only a result, but also their gradient: some methods require first (and sometimes second) order derivatives.

2. If derivatives cannot be analytically expressed: approximate them. For instance, have a look at the five point stencil technique for first and second order derivatives:
`http://en.wikipedia.org/wiki/Five-point_stencil`

3. Design your code cleverly: be sure that for each method you can pass a pointer to a function and the search domain. If you are not familiar with function handles in matlab, see
`http://www.mathworks.fr/fr/help/matlab/ref/function_handle.html`

4. Do not copy and paste existing source codes: I will detect it (and you will fail). Rather add a reference to show how inspiring these sources have been... You can also compare your own implementations to other ones available online, and that is much more interesting!