Optimization: Deterministic optimization

Xavier Olivé Fernández (1394407)

January 3, 2021

---

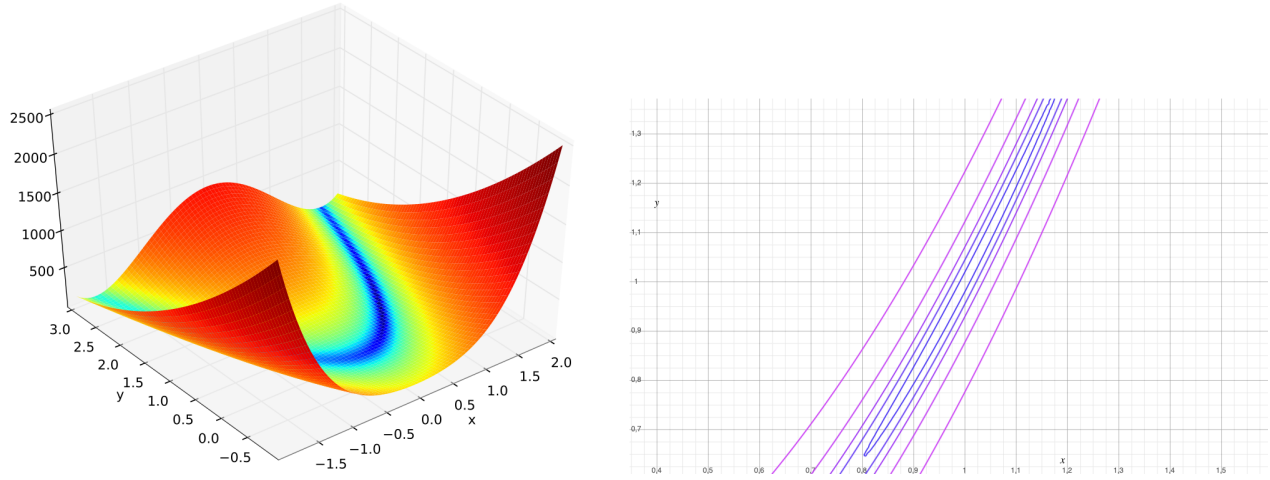
1 Introduction

The propose of this assignment is to understand an to implement some deterministic methods to solve nonlinear problems. In this case we are interested in solving The Rosenbrock's function. First we will explain the function and we will justify the methods used to find the solution.

Rosenbrock's function is:

$$f(x_1, x_2) := 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \tag{1}$$

It has a "Banana shape", where the global minimum is inside a long, narrow, parabolic shaped flat valley. To find the valley is trivial as the valley walls are very steep. To converge to the global minimum, however, is difficult [1]. We can see the shape in Figure 1. We can see it in 3 dimensions and the representation of some "cuts" over the plane by some $f(x_1, x_2)$ values fixed (to simplify we translate x_1 as x and x_2 as y).



(a) 3D [2]

(b) 2D (by cuts)

Figure 1: Representation in 3D and in 2D (by cuts) of the Rosenbrock's function

As well know this function has a global minimum at (1,1) [1] and it is **non-convex**, as it doesn't contains line segment between any two points in the set $x_1, x_2 \in C$ [3]. Here, we understand C as the part contained above the function. It is easily to see that no negative values for f are possible as it is a **quadratic function**. We can also ensure that no local minimums exist as the derivatives of the function is only 0 for the point (1, 1).

Rosenbrock's function is an example of **Non-linear least squares problem**. Least squares problems can be solved by general optimisation methods (we will use first Conjugate Gradient Method), but we shall present special methods that are more efficient (we will use Levenberg–Marquardt method). In many cases they achieve better than linear convergence, sometimes even quadratic convergence, even though they do not always need implementation of second derivatives [4]. For our case an indirect dependence with x^4 exists. This implies applying concrete changes

to the general methods expressed in the books that do not fit our case, since the Hessian of the function will not always be a positive definite matrix (it depends on the values of x_1 and x_2 at all times).

For our concrete case we want to minimize f over $\mathbb{R}^2$ starting at the point (-1.5, -1) [5]. As, usual in Rosenbrock's function we won't have any constrain.

We can say that the starting point has positive but also negative aspects as a seed. As negative aspects we can remark that it is located quite far from the minimum and this causes that more steps are needed to arrive at the wished value with the indicated tolerance. As positive aspects we can emphasize that the two values are in negative quadrants (they have the matrix symbol). This at the algebra level assumes that the Hessian is defined positively in most salts despite depending on x_1 and x_2. This gives consistency to our program.

2 Implementation

2.1 Methods

Here we will explain the basics of Step Descent methods used for that exercise. As our models are conditioned by the x^4 dependence and some modifications need to be done, they will be also explained. In general, descendent methods are iterative methods which mace steps approaching the minimum we want to find. Steps are calculated with:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k \tag{2}$$

The tolerance of our method is defined as wanted. However it is related with the gradient as we to arrive to a minimum (so we interested in arriving to a point where the gradient is near 0 i.e. near the most plane zone as possible). For our case we will use the $[\nabla f(\mathbf{x})]^T \nabla f(\mathbf{x})$ as the residues comparator so tolerance is set by a ϵ^2 dependence.

2.1.1 Conjugate Gradient Method

Conjugate irection methods are motivated by the desire to accelerate the typically slow convergence associated with steepest descent [6]. They use the direction of the step computed as:

$$\mathbf{d}_k = -\nabla f(\mathbf{x}_k) + \beta_k \mathbf{d}_{k-1}, \tag{3}$$

where the term $\beta_k \mathbf{d}_{k-1}$ can be interpreted as a "momentum" that our new direction mantains from the previous one.

In our concrete case f is a nonlinear function, and in comparison with the linear algorithm there are 3 important changes [7]:

1. The recursive formula for the residual cannot be used. In all pseudocodes found $\mathbf{r}_k$ was calculated by iterative way $\mathbf{r}_k = \mathbf{r}_{k-1} - \alpha \mathbf{A}\mathbf{d}_{k-1}$, where $\mathbf{A}$ correspond to the Hessian matrix $\mathbf{A} = \mathbf{H}(f(\mathbf{x}_0)$ for quadratic cases (when $\mathbf{A}$ has non dependence with x_1 and x_2 and it is positive-defined). Because of all that, all computation will be done with $\mathbf{r} = -\nabla f(\mathbf{x}_k)$ and $\mathbf{H}(f(\mathbf{x}_k)) = f''(\mathbf{x}_k)$.

2. It becomes more complicated to compute the step size α than in the simple quadratic case. However we can adopt the same formula with the equivalent dependences and using:

$$\alpha_k = \frac{-[(f'(\mathbf{x}_k))]^T \mathbf{d}_k}{\mathbf{d}_k^T [\mathbf{H}(f(\mathbf{x}_k))] \mathbf{d}_k} \tag{4}$$

3. There are several different choices for β. In our case we will follow Fletcher–Reeves formula which is the easiest one:

$$\beta_k^{FR} = \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{r}_{k-1}^T \mathbf{r}_{k-1}} \tag{5}$$

2.1.2 Levenberg–Marquardt Method

The Levenberg-marquardt algorithm is first shown to be a blend of neutral steepest-descent and Gauss-Newton iteration [8]. Away from the minimum, in regions of negative curvature, the Gauss-Newton approximation is not very good. In such regions, a simple steepest-descent step is probably the best plan [9]. Instead of using (2), we will use a modification:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - (\mathbf{H} + \lambda \mathbf{I})^{-1} \nabla f(\mathbf{x}_k), \tag{6}$$

where we can reinterpret $\alpha \equiv (\mathbf{H} + \lambda \mathbf{I})^{-1}$ and $\mathbf{d}_k \equiv -\nabla f(\mathbf{x}_k)$. We can see that in this case parameters α and β are not necessary. Here we use λ. When it is small $\mathbf{H}$ approximates the Gauss-Newton Hessian. When λ is large, the inverse term is close to the identity, causing steepest-descent steps to be taken [9]. Here $-(\mathbf{H} + \lambda \mathbf{I})^{-1} \nabla f(\mathbf{x}_k)$ can be understood as the new variable $\mathbf{h}_k$ which is used to actualization of the step ad has the meaning of the direction taken in for this method.

This method is a trust region method and so, there is an area centered at $\mathbf{x}$ where the model is sufficiently accurate. This trust-region method uses an areas where the Taylor expansion is valid for every step. λ (*dumpingt parametre*) is the variable which controls the trust region size. The damping parameter influences both the direction and the size of the step, and this leads us to make a method without a specific line search. The updating of λ is controlled in each step by the gain ratio [8]:

$$\rho = \frac{f(\mathbf{x}_k) - f(\mathbf{x}_k + \mathbf{h}_k)}{L(0) - L(\mathbf{h}_k)} \equiv \frac{f(\mathbf{x}_k) - f(\mathbf{x}_{k+1})}{\frac{1}{2}\mathbf{h}_k^T(\lambda \mathbf{h}_k - f'(\mathbf{x}_k))}, \tag{7}$$

where L indicates the expression that approximates the function $f(\mathbf{x}_k)$ in the finite region near $\mathbf{x}_k$ (the trust-region which is updated at each iteration using heuristics). It is obtained by a Taylor series expansion of f around $\mathbf{x}_k$ point [8].

$\rho > 0$ indicates that we are inside the trust region and we can decrease λ so that the next Levenberg-Marquardt step is closer to the Gauss-Newton step. For that cases:

$$\lambda_{k+1} = \lambda_k \cdot max(\frac{1}{3}, 1 - (2\rho - 1)^3), \tag{8a}$$

If ρ is negative, then we increase λ with the twofold aim of getting closer to the steepest descent direction and reducing the step length [4].

$$\lambda_{k+1} = 3\lambda_k, \tag{8b}$$

2.2 Code

The code that allows us to generate the result is divided into two different files included in the Code folder:

- `main.c` contains the main function of all the program. As all programs it has as first part of declaration of variables. Most of variables are defined type double as we will use a large number of figures for calculations. Then we have a common zone of initialization of all variables where the initial values are assigned. `main.c` uses an argument for selecting the method wanted for reaching the minimum. There are two options: `'g'` and `'l'`.
  - The first option will use **Conjugate gradient method** that will use its own loop to get the result. In this loop we calculate first α and the next step, then update the gradient and the Hessian and finally we calculate the residue got and we update the new direction. The loop output occurs when the residue is below the indicated tolerance.
  - `'l'` corresponds to the **Levenberg–Marquardt Method**. As before it consists of a `while` loop (that finishes when the residue is below the indicated tolerance) where the direction and the step done are calculated. Then it uses variable ρ to check if the step is accepted or not. For the first case the gradient, the Hessian and the residue are updated and λ is reassigned with a lower value. In the case the step is rejected, we enlarge λ and we repeat again the loop with the new value

For both loops each step is stored in the file `"Coordenades.txt"` that is created in the directory of usage of the program. There's also and `exit` command when no specific method is selected.

The program finishes with a printout of the results on the screen.

- `detlibrary.h` contains all the necessary include calls to standard C libraries, the defined macros (the given initial point (x_{1_0}, x_{2_0}) on (`INITIALx_1`, `INITIALx_2`), ϵ which will define the error in `epsilon` and the replacement factor `RR` for λ in the Levenberg–Marquardt Method) and all necessary functions that are called both by the main function in `main.c`. This functions are:
  - `solve_Rosen_function()` Calculates the value of the function for the position (x_1, x_2) given.
  - `Gradient_calc()` Calculates the value of the gradient for the position (x_1, x_2) given and save the values in the location of an array.
  - `Hessian_calc()` Calculates the value of the Hessian for the position (x_1, x_2) given and save the values in the location of an array.
  - `d_CGM_calc()` Calculation of the direction in which the next step will be done for the Conjugate gradient method following equation (3).
  - `alpha_calc()` Calculation of α for the Conjugate gradient method following equation (4).
  - `step_CGM()` Calculation of the new step following (2) in the indicated way for Conjugate gradient method.
  - `p_LM_calc2()` Calculation of second term of the expression (6): $-(\mathbf{H} + \lambda\mathbf{I})^{-1}\nabla f(\mathbf{x}_k)$ that we can identificate as $\mathbf{h}_k$ and that has a function of the direction for the next step.
  - `step_LM()` Calculation of the new step following (6) in the indicated way for Levenberg–Marquardt Method.
  - `rho_calculation()` Calculation of ρ for the Levenberg–Marquardt Method following equation (7).

NOTE 1: we do not use any `malloc` or any memory space allocation as we do not need high memory spaces (no lists, no structures and the largest array is 2x2).
NOTE 2: In the same code there are some notes that that explain and justify the commands used.

3 Results

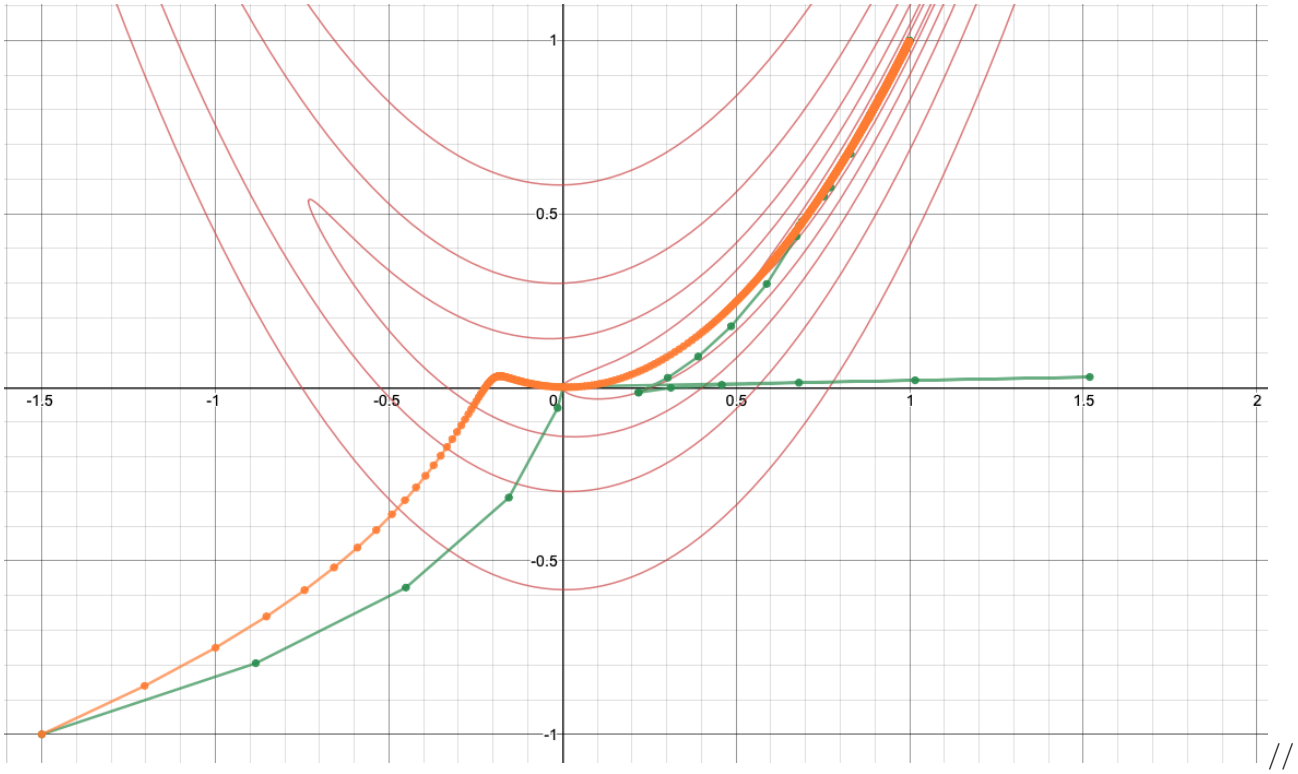
In this section we will comment the obtained results. First we will show the "path" followed for each method from the initial point to the minimum of the function. Each set of points presented in this report is available in the folder `Report_steps` in a `.txt` file.

Paths are represented Figure 2.

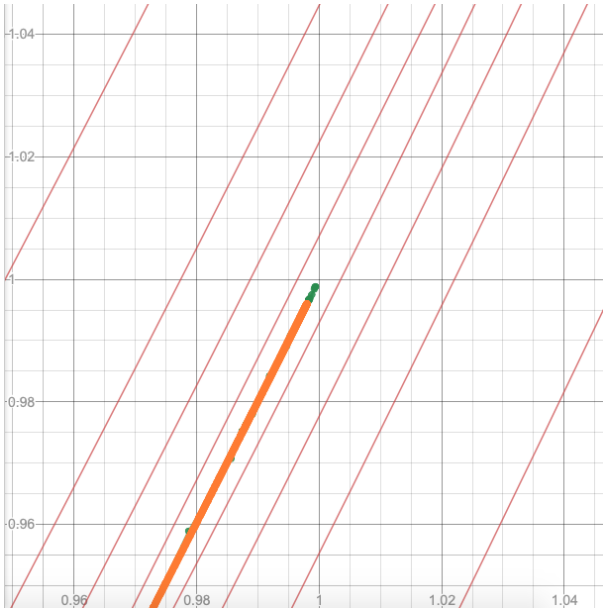
In red we indicate some cuts of the Rosenbrock function near the origin. In green and orange we represent the steps done by the Conjugate gradient method and the Levenberg–Marquardt Method respectively. We observe how the two methos converge to the point (1, 1) which is the absolute minimum of our function.

For the Conjugate gradient method (green) we see that the steps are quite pronounced and with a low number we can reach the minimum. Near the origin we can make a very large and irregular jump that stands out on the way to reach the minimum. This jump is not typical of this method especially when we have simple quadratic equations. In our case, Hessian depends on our variables x_1 and x_2 and this means that α often takes very high values near the origin that generate jumps like the ones we can see. We see how this method finds the result very quickly pro does it very abruptly.

In orange we have Levenberg–Marquardt Method. This method is much slower but very smooth and robust when converging to the desired point. For this specific case we can identify how it first approaches the valley and once it has found it, it advances to the desired point, the minimum.



(a) General view



(b) Ampliation near the minimum point (1, 1)

Figure 2: Computed steps by the Conjugate gradient method (green) and the Levenberg–Marquardt Method (orange) when solving the Rosenbrocks function minimization problem.

In Figure 2b we have an enlargement of the surroundings of the minimum point (1, 1). We can see how it turns out that the first method is closer however this one is more irregular and the way to converge is not so robust. In fact, it is normal for the second method not to be as close to the minimum point as the first. It should be noted that the stipulated tolerance is the same for both methods and is linked to the residue of the square of the gradient. Therefore, the tolerance is related to a smooth access and steps suffered to the desired point and following this logic the second method is further away as the steps taken to arrive are more small and smooth.

With a visual view, we can also check definitly if (-1.5,-1) is a good seed point. As negative aspects we can remark that it is located quite far from the minimum and this causes that more steps are needed to arrive at the wished value with the indicated tolerance. As said before it was so for form the minimum. In the case of the Levenberg-Marquardt method, about 2,000 are needed. In addition, the point is located on a very steep slope and this, although it makes us access the valley very quickly, ends up being an unwanted sharp jump for the Conjugate Gradient method. However, a part of the Hessian advantage we can emphasize the fact that the point is sufficiently facing the sought solution (following in a sufficiently straight line the area of the valley where the minimum is).

Now we compare all this observations in a table for the two methods:

	Conjugate Gradient	Levenberg–Marquardt
Steps	63	2001
Final (x_1, x_2)	(0.999388043, 0.998777089)	(0.998037820, 0.996069237)
Final $f(\mathbf{x})$	$0.375 \cdot 10^{-6}$	$3.861 \cdot 10^{-6}$
Time (ms)	0.0930	1.7770

Table 1: Results obtained for each method. We see the total amout of steps, the postion where we arrived, the value of Rosenbrock function in that point and the time spend

Here we can compare the results obtain and check the affirmations done in the graphical part by numerical data. Like we said, Levenberg-Marquardt is much slower (21 times in time) and 30 less amount of steps are needed to get the result with the same tolerance for both. For the position got we can see how they are very similar but with Conjugate Gradient is a little bit better. That was justified before by the graphical analysis.

4 Conclusions

In this assignment, two Descent methods have been applied to find the absolute minimum of Rosenbrock function (a non-convex function). To do so, we have used Conjugate Gradient method and Levenberg-Marquardt method. For both cases we have achieved our goal of finding the point (1,1) that corresponds to this absolute minimum, so the principal objective of the assignment has been fulfilled.

In addition, by analyzing the results we were able to check the pros and cons of using each of the methods. While the Conjugate Gradient is much faster, Levenberg-Marquardt is smoother and more robust. We have also evaluated the pros and cons of the starting point, which although it presents some problem of distance, has some advantages over Hessian or position.

References

[1] Wikipedia: the free encyclopedia. *Rosenbrock's function*, 2020. URL: `https://en.wikipedia.org/wiki/Rosenbrock_function`

[2] Wikipedia: the free encyclopedia. *Rosenbrock's function*, 2009. URL: `https://en.wikipedia.org/wiki/Rosenbrock_function#/media/File:Rosenbrock_function.svg`

[3] Lieven Vandenberghe. *Convex Optimization: Tutorial lectures, Machine Learning Summer*. School University of Cambridge, 2009.

[4] K. Madsen, H.B. Nielsen, O. Tingleff. *Methods for non-linear least squares problems*. Informatics and Mathematical Modelling - Technical University of Denmark, 2004.

[5] Lluís Alsedà. *Deterministic optimization assignment*.

[6] David G. Luenberger, Yinyu Ye. *Linear and Nonlinear Programming (p 277-297)*. Springer Science+Business Media, 2008. ISBN: 978-0-387-74502-2.

[7] Jonathan R. Shewchuk. *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. Carnegie Mellon University, 1994.

[8] Ananth Ranganathan. *The Levenberg-Marquardt Algorithm*. 2004.

[9] A. Zisserman. *Optmization*. 2013. URL: https://www.robots.ox.ac.uk/~az/lectures/opt/lect1.pdf