Project 1 NLA: direct methods in optimization with constraints

Jaume Sánchez

October 2023

Contents

1 Introduction

This report corresponds to the study carried out solving the first Numerical Linear Algebra project. In this, you can find both theoretical answers and the explanation and analysis of the results of the practical part (code part). In order to better understand the practical part, we will first solve the theoretical part.

2 Theoretical questions

T1: Show that the predictor steps reduces to solve a linear system with matrix M_{KKT}.

Proof. Let us first define F as follows, $F : \mathbb{R}^N \longrightarrow \mathbb{R}^N$,

$$F(z) = F(x, \gamma, \lambda, s) = (Gx + g - A\gamma - C\lambda, b - A^T x, s + d - C^T x, s\lambda) = (F_1, F_2, F_3, F_4) \tag{1}$$

where $N = n + p + 2m$.

Now, we will suppose that $\hat{z}$ is a zero of F, that is, $F(\hat{z}) = 0$. Computing the Taylor expansion in $\hat{z}$ of F we obtain:

$$0 = F(\hat{z}) \approx F(z_k) + J_F(z_k)(\hat{z} - z_k)$$

Where by J_F we denote the Jacobian matrix of F and where we have elided the second order terms. Note that now if we call $\delta_z = z_{k+1} - z_k$ we have that

$$0 = F(z_k) + J_F(z_k)\delta_z \iff J_F(z_k)\delta_z = -F(z_k)$$

If we are able to see that, in fact, $J_F(z_k)$ is equal to the M_{KKT} matrix, then we are done.
But, note that $J_F(z_k)$ is nothing but:

$$J_F(z_k) = \begin{pmatrix} \frac{\partial F_1}{\partial x} & \frac{\partial F_1}{\partial \gamma} & \frac{\partial F_1}{\partial \lambda} & \frac{\partial F_1}{\partial s} \\ \frac{\partial F_2}{\partial x} & \frac{\partial F_2}{\partial \gamma} & \frac{\partial F_2}{\partial \lambda} & \frac{\partial F_2}{\partial s} \\ \frac{\partial F_3}{\partial x} & \frac{\partial F_3}{\partial \gamma} & \frac{\partial F_3}{\partial \lambda} & \frac{\partial F_3}{\partial s} \\ \frac{\partial F_4}{\partial x} & \frac{\partial F_4}{\partial \gamma} & \frac{\partial F_4}{\partial \lambda} & \frac{\partial F_4}{\partial s} \end{pmatrix} = \begin{pmatrix} G & -A & -C & 0 \\ -A^T & 0 & 0 & 0 \\ -C^T & 0 & 0 & I \\ 0 & 0 & S & \Lambda \end{pmatrix}$$

So, as both matrices are the same we can conclude that the predictor steps reduces to solve a linear system with matrix M_{KKT}. □

T2: Explain the previous derivations of the different strategies and justify under which assumptions they can be applied.

Proof. Note that as in this case we do not allow equality constrains, we have that $A = 0$. With this, the system that we want to solve is converted into

$$\begin{pmatrix} G & -C & 0 \\ -C^T & 0 & I \\ 0 & S & \Lambda \end{pmatrix} \begin{pmatrix} \delta_x \\ \delta_\lambda \\ \delta_s \end{pmatrix} = \begin{pmatrix} -r_1 \\ -r_2 \\ -r_3 \end{pmatrix}$$

Let us discuss the two different strategies that are proposed.
In order to be more clear, we will rewrite the matrix system using equations

$$\left.\begin{array}{rcl} G\delta_x - C\delta_\lambda & = & -r_1 \\ -C^T\delta_x + \delta_s & = & -r_2 \\ S\delta_\lambda + \Lambda\delta_s & = & -r_3 \end{array}\right\} \tag{2}$$

Strategy 1

According to strategy 1, if we isolate δ_s from the third row we obtain $\delta_s = \Lambda^{-1}(-r_3 - S\delta_\lambda)$. Note that, in order to isolate δ_s we have to assure that Λ^{-1} is invertible. As Λ is a diagonal matrix with strictly positive numbers on it (because we are in the feasibility region) we have that, in fact, the inverse matrix of Λ is well defined.
Then, we substitute the previous expression into the second row and we finally obtain

$$\left.\begin{array}{rcl} G\delta_x - C\delta_\lambda &=& -r_1 \\ -C^T\delta_x - \Lambda^{-1}S\delta_\lambda &=& -r_2 + \Lambda^{-1}r_3 \end{array}\right\} \iff \begin{pmatrix} G & -C \\ -C^T & -\Lambda^{-1}S \end{pmatrix}\begin{pmatrix} \delta_x \\ \delta_\lambda \end{pmatrix} = \begin{pmatrix} -r_1 \\ -r_2 + \Lambda^{-1}r_3 \end{pmatrix}$$

Note that, if we want to apply the LDL^T factorisation to the previous matrix we need the matrix to be symmetric.
An easy computation shows that, being A, B, C, D block matrices:

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}^T = \begin{pmatrix} A^T & C^T \\ B^T & D^T \end{pmatrix}$$

So, in our case we have that

$$\begin{pmatrix} G & -C \\ -C^T & -\Lambda^{-1}S \end{pmatrix}^T = \begin{pmatrix} G^T & ((-C)^T)^T \\ (-C)^T & (-\Lambda^{-1}S)^T \end{pmatrix} = \begin{pmatrix} G & -C \\ -C^T & -\Lambda^{-1}S \end{pmatrix}$$

The last equality is due to the fact that G is symmetric by definition and $\Lambda^{-1}S$ is a product of diagonal matrices, which is already a diagonal matrix. Hence, the matrix is already symmetric (without any assumptions).

Strategy 2

Let us focus now on the second strategy, whose first action consists on isolating δ_s from the second row. Again, following (2) we obtain that $\delta_s = -r_2 + C^T\delta_x$. Now, substituting this expression into de third row we obtain $\delta_\lambda = S^{-1}(-r_3 + \Lambda r_2) - S^{-1}\Lambda C^T\delta_x$.

Similarly to the first strategy, we need to compute the inverse matrix of S. Arguing in the same way as before, S^{-1} exists because S is a diagonal matrix with strictly positive numbers. Finally, we substitute this last expression into the first row ($G\delta_x - C\delta_\lambda = -r_1$) obtaining, $(G + CS^{-1}\Lambda C^T)\delta_x = -r_1 + CS^{-1}(-r_3 + \Lambda r_2)$. Both sides of the equality are well defined, so we just now have to check if the left side matrix is symmetric and positive definite, as we want to apply Cholesky factorization.

$$(G + CS^{-1}\Lambda C^T)^T = G^T + (CS^{-1}\Lambda C^T)^T = G^T + (C^T)^T\Lambda^T(S^{-1})^T C^T$$

As Λ and S^{-1} are diagonal matrices, we have that the matrices commute and $\Lambda^T = \Lambda$. Recall that G is symmetric by definition and, obviously $(C^T)^T = C$. With that, we have that the last expression becomes

$$G^T + (C^T)^T\Lambda^T(S^{-1})^T C^T = G + CS^{-1}\Lambda C^T$$

So the matrix is symmetric. Let us now check that the matrix is also positive definite, that is, for all $x \in \mathbb{R}^n \setminus \{0\}$ we have that

$$x^T(G + CS^{-1}\Lambda C^T)x > 0 \iff x^T Gx + x^T(CS^{-1}\Lambda C^T)x > 0$$

As G is semidefinite positive it suffices to prove that the matrix $CS^{-1}\Lambda C^T$ is positive definite. Let us rename $D := S^{-1}\Lambda$, a diagonal matrix with strictly positive numbers and let $\sqrt{D} = \sqrt{d_i}, 1 \leq i \leq m$ where d_i are the coefficients on the diagonal of D.
So,

$$x^T(CS^{-1}\Lambda C^T)x = x^T CDC^T x = x^T C\sqrt{D}\sqrt{D}C^T x = ||x^T C\sqrt{D}||^2 \geq 0$$

So, as we want the matrix $CS^{-1}\Lambda C^T$ to be positive definite, we just have to impose that for all $x \in \mathbb{R}^n \setminus \{0\}$

$$||x^T C\sqrt{D}||^2 \neq 0$$

Note that it suffices to impose that because the norm, by definition, is always a positive number. □

T3: Isolate δ_s from the 4th row of M_{KKT} and substitute into the 3rd row. Justify that this procedure leads to a linear system with a symmetric matrix.

Proof. In this section, now $A \neq 0$, so now the system to solve is the following

$$\begin{pmatrix} G & -A & -C & 0 \\ -A^T & 0 & 0 & 0 \\ -C^T & 0 & 0 & I \\ 0 & 0 & S & \Lambda \end{pmatrix} \begin{pmatrix} \delta_x \\ \delta_\gamma \\ \delta_\lambda \\ \delta_s \end{pmatrix} = \begin{pmatrix} -r_L \\ -r_A \\ -r_C \\ -r_s \end{pmatrix}$$

Where, $F(z_0) = (r_L, r_A, r_C, r_s)$. From the fourth row, we have that

$$S\delta_\lambda + \Lambda\delta_s = -r_s \iff \delta_s = \Lambda^{-1}(-r_s - S\delta_\lambda)$$

Substituting the latter into the third row, we obtain

$$\begin{pmatrix} G & -A & -C \\ -A^T & 0 & 0 \\ -C^T & 0 & -\Lambda^{-1}S \end{pmatrix} \begin{pmatrix} \delta_x \\ \delta_\gamma \\ \delta_\lambda \end{pmatrix} = \begin{pmatrix} -r_L \\ -r_A \\ -r_C + \Lambda^{-1}r_s \end{pmatrix}$$

Let us see if the previous matrix is symmetric:

$$\begin{pmatrix} G & -A & -C \\ -A^T & 0 & 0 \\ -C^T & 0 & -\Lambda^{-1}S \end{pmatrix}^T = \begin{pmatrix} G^T & (-A^T)^T & (-C^T)^T \\ -A^T & 0 & 0 \\ -C^T & 0 & (-\Lambda^{-1}S)^T \end{pmatrix} = \begin{pmatrix} G & -A & -C \\ -A^T & 0 & 0 \\ -C^T & 0 & -\Lambda^{-1}S \end{pmatrix}$$

Using same arguments than before, we have just seen that the matrix is symmetric. Hence, this is obviously a linear system with a symmetric matrix. □

3 A little explanation

Once answered the theoretical questions, I will explain the structure of what remains about this memory. First of all, the delivery consists on two scripts: *C2-C4.py* and*C5-C6.py*. The former, answers the questions **C1**,**C2**,**C3** and **C4**. To do that, I have implemented 8 functions; the first one, *Newton_step*, answers the **C1** question of the project. In fact, this function was given and it computes the step-size correction. Then we can find the function F, that actually returns the output of the function defined in **T1** (see 1). Then, we can see three functions : *create_MKKT_c2*, *create_MKKT_c4_1* and *create_MKKT_c4_2*; these functions allows us to create the M_{KKT} initial matrix on each exercise $(C2, C4_1, C4_2)$, where the subindexes denote the strategy we follow, that is, strategy one or strategy 2. (Note that the last function will be also used to update the matrix). Finally, we find three more functions: *c2*, *c4_1* and *c4_2*; these functions will solve the Newton method algorithm, using the corresponding strategies, that is: without strategy, using LDL^T descomposition and using Cholesky descomposition, respectively. In other words, these function are the most important function of this script.

To finish this script, we can find a little main that answers all the questions in the project such as: computation time, number of iterations, precision of the results and condition number of the matrices using all the eight functions that we have just described.

The latter script, *C5-C6*, answers as you may expect, questions **C5** and **C6**. We can find now 9 functions. *Newton_step* is exactly the same function that we use in the first script, and F is a modification of the first F, taking into account that now we are in the general case (allowing inequality constrains), that is, $A \neq 0$. Then we find *Create_MKKT_c5* and *create_MKKT_c6*, that create the initial matrix on each exercise, $C5$ and $C6$, respectively. As we have to read some data from files, we also create two functions: *read_vec* and *read_mat* to read vectors and matrices from a file. And last but not least, we have three more functions: *c5*, *ldlt* and *c6*. The first function answers question **C5** and the last two question **C6**, note that the *ldlt* function is created because it can happen that the *ldlt* function from scipy.linalg can return a matrix D that, in some cases, is not completely diagonal.

Again, to finish this script we find the corresponding functions reading the corresponding datasets and printing the results.

4 Results

4.1 Inequality constrains case

In this section we will compare the three strategies, in terms of execution time, number of iterations, precision and the condition number. Note that all the results that we are going to comment on depend on the random variable g. This variable is a vector whose components are generated by a Gaussian distribution $N(0, 1)$. It is an obvious fact that, because of that, results may change if we take different g values, but as it would follow a similar behaviour our results will be then accurate. In order to compare them in a fair way we will always share the same g for each method while comparing all the metrics and as n, the dimension, increases.

4.1.1 Execution time

Let us first start comparing the execution time required by each method for dimensions $n = 1$ to $n = 100$. From results explained in class, we know that the first strategy (np.linalg.solve) solves the equation $Ax = b$ with an overall time complexity of $\frac{2}{3}n^3 + O(n^2)$ flops, while the other two strategies solve it in half of them. So we expect the first strategy to require more time than the other.

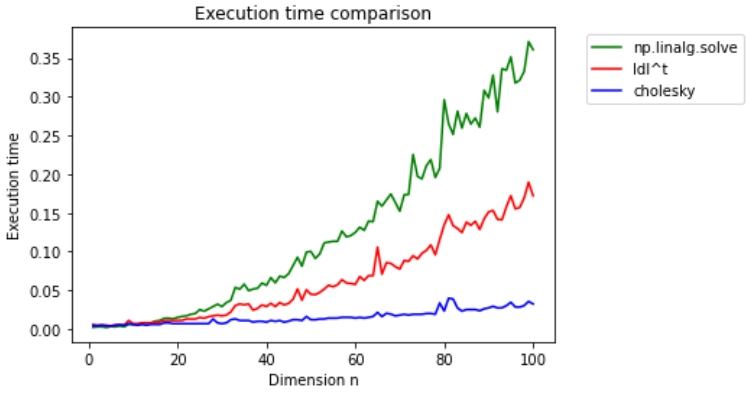

Figure 1: Execution time comparison depending on the dimension n and the chosen method.

As shown in Figure 1, the first strategy needs more time than the other two strategies. In fact, when n is not big enough ($n \approx 20$), we can see how the computation time of the three methods is very similar. However as n increases the difference also increases. For instance, at $n = 100$, the execution times were: $0.361, 0.172, 0.032$ seconds, for np.linalg.solve, ldl^t and cholesky, respectively. So, as a conclusion, the Cholesky strategy is really time efficient.

4.1.2 Number of iterations

Regarding the number of iterations, we have the following plots:

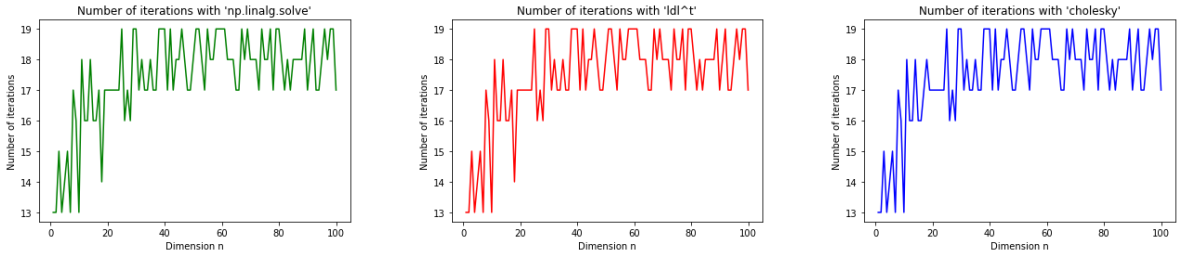

Figure 2: Number of iterations comparison depending on the dimension n and the chosen method.

They all present a very similar behaviour, although as we have seen in the previous section, some do it faster than others.

4.1.3 Precision

We know that the solution of the test problem is the following: $x = -g$. So, in order to measure the precision of our results, what we have done is to calculate the norm (euclidean) of the vector $x + g$.

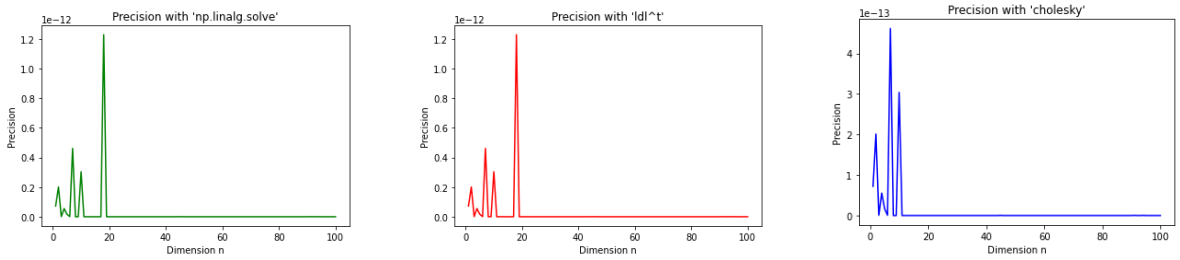

Figure 3: Precision comparison depending on the dimension n and the chosen method.

As expected, in the way that we have programmed the general method, the precision is quite high (the error is very low) in all methods, reaching a precision of up to the 13th decimal in the Cholesky strategy.

4.1.4 Condition number

Another interesting way to compare the three methods is doing it by comparing the condition number of the matrices in which we are applying the methods, that is, if the matrix is good or bad conditioned. We say that a matrix is well conditioned the closer it is to 1.
In order to do that, we first check the initial values and the last values. That is, the condition number at the first iteration and also at its last iteration, in each method.

As we can see in Figure 4, the best initial condition numbers are obtained by the Cholesky strategy while the *worse* (there are not actually bad) are obtained by the np.linalg solve strategy.
Also, in Figure 5, we can see how, again, the worse results are obtained by the np.linalg solve strateg, as the condition number is around 24, and the best results are obtained by the Cholesky strategy.
Let us have a look at what is happening in the ldl^t strategy. In other words, let us study the condition number of the L and D matrices.

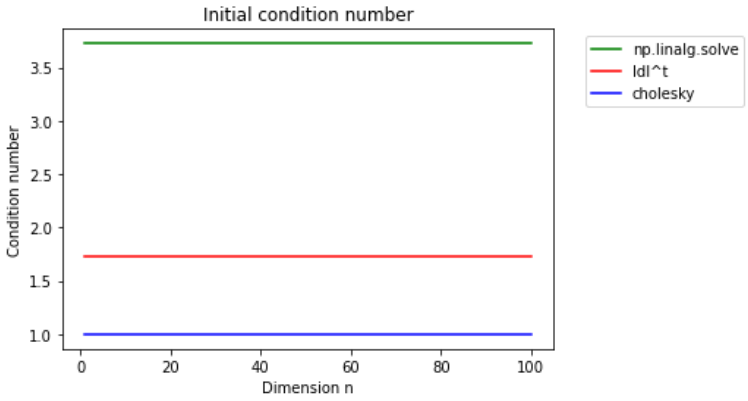

Figure 4: Initial condition number depending on the dimension n and the chosen method.

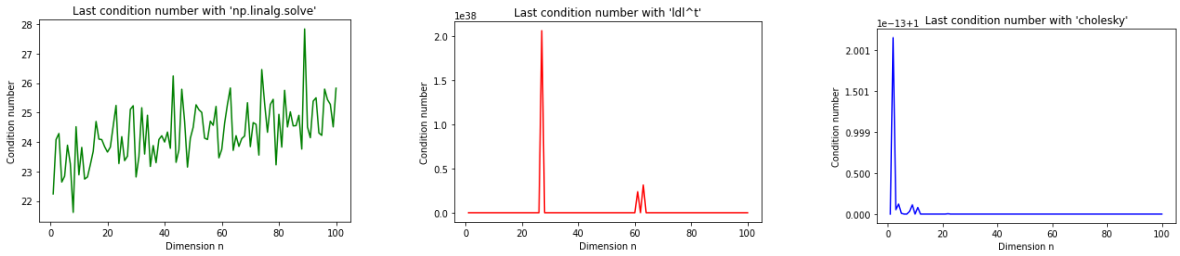

Figure 5: Last condition number depending on the dimension n and the chosen method.

As we can see in Figure 6, the bad conditioned results are due to the matrix D.
As expected, the last condition numbers of the L matrix in the Cholesky decomposition are good, that is, really close to 1. We can see that in Figure 7.

4.2 General case

In order to finish this section, we will answer questions **C5** and **C6**.In the first dataset (*optpr1*) where $n = 100, p = 50$ and $m = 200$ we obtained the following results: (see Table 1)

Method	Execution time (s)	Iterations
np.linalg.solve	0.1289	23
ldl^t	0.1225	25

Table 1: Results for the first dataset

While in the second dataset (*optpr2*) where $n = 1000, p = 500$ and $m = 2000$ we obtained the following results. (see Table 2)

I expected the ldl^t strategy to go faster in the first dataset, in fact, I think that it must be some mistake. Anyway, we can see in the second dataset how the second strategy improves the execution time by more than half.

Figure 6: Condition number comparison between L and D in ldl^t depending on the dimension n

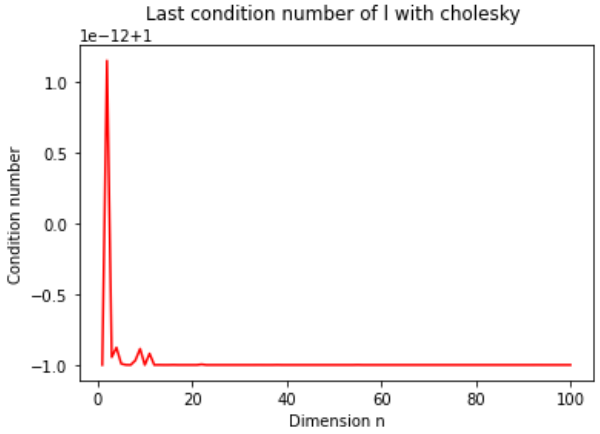

Figure 7: Last condition number of L in Cholesky depending on the dimension n.

5 Dificulties

In my opinion, the most tricky part was when I wanted to use de ldl^t function implemented just by libraries from pyhton in the last exercise. The problem was that the diagonal matrix could be a diagonal matrix per blocks (2x2) and not a normal diagonal matrix as before. Another point to highlight was the permutation vectors, it was not difficult but you have to realize how to deal with it. On the other hand, I would like to highlight that this practice has been interesting and has allowed us to apply theoretical concepts and put them into practice.

Method	Execution time (s)	Iterations
np.linalg.solve	71.2791	30
ldl^t	23.9234	29

Table 2: Results for the second dataset