Numerical Linear Algebra

Master on Fundamentals of Data science
Universitat de Barcelona

Project 1
NLA: direct methods in optimization with constraints.

Pablo Álvarez
Blai Ras

November 15, 2020

Note: this project was coded between Pablo Alvarez and me, Blai Ras. Nevertheless, I'm the author of the following paper, meaning that each one of us did a different conclusion separately.

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

Given $F : \mathbb{R}^N \to \mathbb{R}^N$, defined as

$$F(z) = F(x, \gamma, \lambda, s) = (\underbrace{Gx + g - A\gamma - C\lambda}_{F_1}, \underbrace{b - A^\top x}_{F_2}, \underbrace{s + d - C^\top x}_{F_3}, \underbrace{s \odot \lambda}_{F_4})$$

where $\odot$ denotes the element-wise product of two vectors, we need to solve $F(z) = 0$, through the Newton method. This means solving the linear system $M_{KKT}(\delta) = r$.

Newton's method for functions with more than one variables consists of starting on an intitial point z_0 and end on the next point $z_1 = z_0 + z$, where we obtain:

$$0 = F(z_0 + z) \simeq F(z_0) + J_F(z_0) \cdot \delta_z$$

In other words, solving the linear system $J_F(z_0) \cdot \delta_z = -F(z_0)$. In consequence, we just need see that $M_{KKT} = J_F(z_0)$. Let's calculate the Jacboian matrix:

$$J_F = \begin{pmatrix} \frac{\partial F_1}{\partial x} & \frac{\partial F_1}{\partial \gamma} & \frac{\partial F_1}{\partial \lambda} & \frac{\partial F_1}{\partial s} \\ \frac{\partial F_2}{\partial x} & \frac{\partial F_2}{\partial \gamma} & \frac{\partial F_2}{\partial \lambda} & \frac{\partial F_2}{\partial s} \\ \frac{\partial F_3}{\partial x} & \frac{\partial F_3}{\partial \gamma} & \frac{\partial F_3}{\partial \lambda} & \frac{\partial F_3}{\partial s} \\ \frac{\partial F_4}{\partial x} & \frac{\partial F_4}{\partial \gamma} & \frac{\partial F_4}{\partial \lambda} & \frac{\partial F_4}{\partial s} \end{pmatrix} = \begin{pmatrix} G & -A & -C & 0 \\ -A^\top & 0 & 0 & 0 \\ -C^\top & 0 & 0 & I \\ 0 & 0 & S & \Lambda \end{pmatrix} := M_{KKT}$$

In efect, we see that $M_{KKT} = J_F(z_0)$, where I denotes the identity matrix, and S, Λ the matrices with values s_i, λ_i in their diagonals, respectively.

Thus, obtaining δ_{z_k} (predictor step) reduces to solve a linear system with matrix M_{KKT} and right-hand vector $-F(z_k)$.

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

First strategy

We need to see what we obtain when we isolate the vector δ_s in the third row of M_{KKT}. Remember that in this first section we are using $A = 0$.

We have on the third row that:

$$\begin{pmatrix} O & S & \Lambda \end{pmatrix} \begin{pmatrix} \delta_x & \delta_\lambda & \delta_s \end{pmatrix}^T = -r_3 \Longleftrightarrow S\delta_\lambda + \Lambda\delta_s = -r_3 \Longleftrightarrow \delta_s = -\Lambda^{-1}(r_3 + S\delta\lambda)$$

Let's save this isolation for now. Let's focus our second row:

$$\begin{pmatrix}-C^T & 0 & I\end{pmatrix}\begin{pmatrix}\delta_x & \delta_\lambda & \delta_s\end{pmatrix}^T = -r_2 \iff -C^T\delta x + \delta s = -r_2$$

Where we now replace δs:

$$\iff -C^T\lambda x - \Lambda^{-1}r_3 - \Lambda^{-1}S\delta\lambda = -r_2$$
$$\iff -C^T\delta_x - \Lambda^{-1}S\delta_\lambda = -(r_2 + \Lambda^-1r_3) \iff (-C^T - \Lambda^{-1}S)(\delta) = -(r_2 - \Lambda^{-1}r_3)$$

This will only be possible if and only if Λ is inversible, i.e. if $\delta_i > 0\ \forall i$. In section 1.2 of the project we are told so.

Second strategy

Now we need to start in the second row:

$$-C^T\delta_x + \delta_s = -r_2 \iff \delta_s = C^T\delta_x - r_2$$

Where we replace on the third row:

$$S\delta_s + \Lambda\delta_s = -r_3$$
$$\iff S\delta_s + \Lambda C^T\delta_s - \Lambda r_2 = -r_3$$
$$\iff \delta_s = -S^{-1}\Lambda C^T\delta_x + S^{-1}(-r_3 + \Lambda r_2)$$

And finally replace it on the first row:

$$G\delta_x - C\delta_\lambda = -r$$
$$\iff \underbrace{(G + CS^{-1}\Lambda C^T)}_{\hat{G}}\delta_x = -r_1 + \underbrace{CS^{-1}(-r_3 + \Lambda r_2)}_{\hat{-r}}$$

Again, the condition for this to happen is that S must be inversible $\iff s_i > 0\ \forall i$

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.*

Let's start remainding that now A is not zero. In this case, then, the M_{KKT} fourth row is:

$$\begin{pmatrix}0 & 0 & S & A\end{pmatrix}\begin{pmatrix}\delta_x & \delta_\gamma & \delta_\lambda & \delta_s\end{pmatrix}^T = -(-r_1 r_A r_C r_s)^T$$
$$\iff S\delta_\lambda + \Lambda\delta_S = -r_s \iff \delta_S = -\Lambda^{-1}(r_s + S\delta_\lambda)$$

We replace it on the third row:

$$\begin{pmatrix} -C^T & 0 & 0 & I \end{pmatrix} \begin{pmatrix} \delta_x & \delta_\gamma & \delta_\lambda & \delta_s \end{pmatrix}^T = -r_c$$

$$\iff -C^T\delta_x + \delta_s = r_c \iff -C^T\delta_x - \Lambda^{-1}S\delta_\lambda - \Lambda^{-1}r_s = -r_c$$

$$-C^T\delta_x - \Lambda^{-1}S\delta_\lambda = -(r_c - \Lambda^{-1}r_s) \iff \begin{pmatrix} -C^T & 0 & \Lambda^{-1}S \end{pmatrix} \begin{pmatrix} \delta_x & \delta_\gamma & \delta_\lambda \end{pmatrix}^T = -(r_c - \Lambda^{-1}r_s)$$

If we take this conclusion onto the 2 first block of the M_{KKT} we get:

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

Denote that the left matrix is known as $\hat{G}$. $\hat{G}$ is symmetric, because:

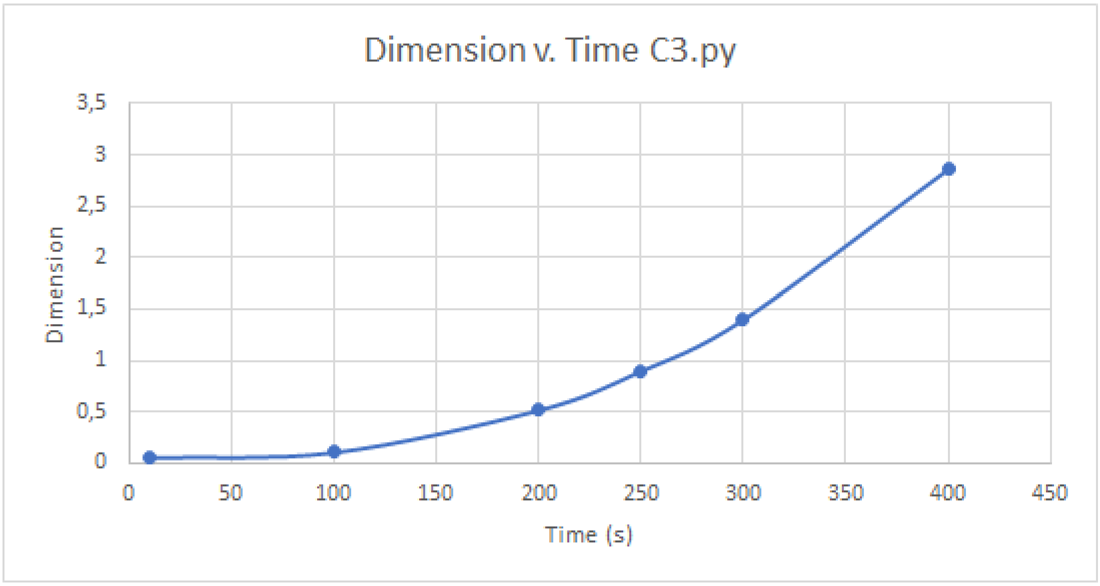
- G is symmmetric by hypotesis.
- (- Λ^{-1} S) because it's the product of two diagonal matrices, resulting in a diagonal matrix.

Results

For this section, the execution times shown are the mean of 5 executions performed, except for the table at section "C5 and C6" where 7 executions were performed.

C3

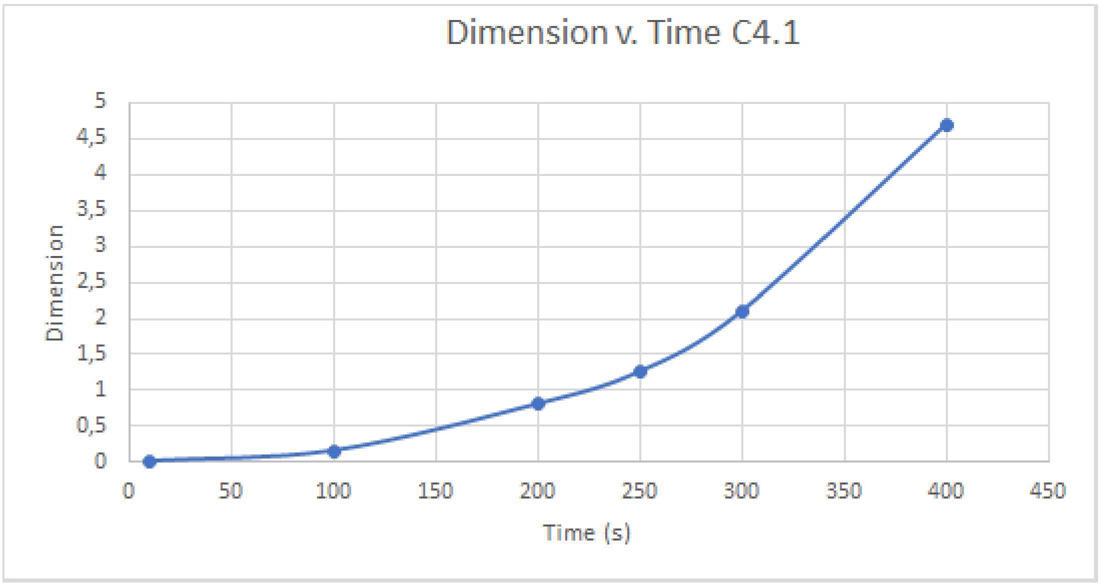
C3.py allows us to test C1 and C2. After multiple tests, I can observe that the number of iterations rarely changes: we usually get 12 for dimensions around 1-10 and 13 to 14 iterations for dimensions around 10-400 or more iterations. The execution time, nevertheless, increases almost exponencially, as we can see below. I stopped testing at dimension 400 due to that it didn't make sense to continue increasing the dimension after seeing that it grows exponentially.



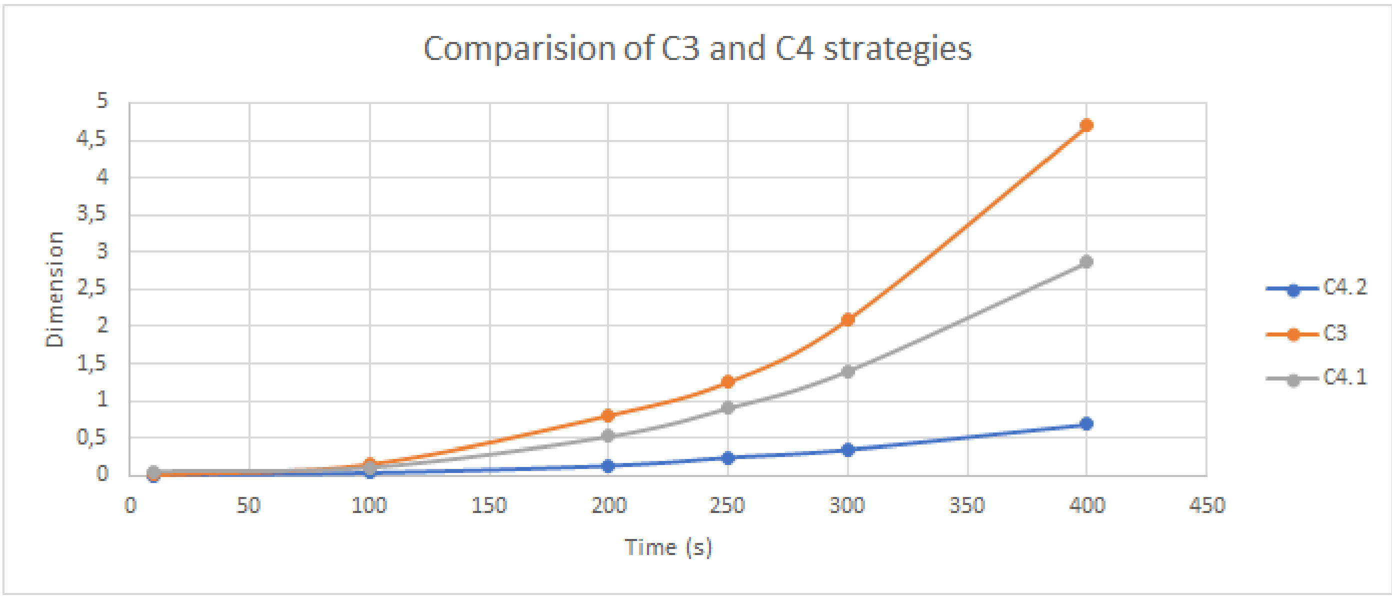
If we focus the precision of the results, I get exaclty the same minimum, with the same digit precision: the dimension does not matter.

C4

In this case the number of iterations with each strategy is 14 around dimensions from 1 to 350, and then it switches to 15 in dimensions 350-400. Time, as C3, increases exponentially as we grow the dimension of the problem:



Nevertheless, we see an improvement of the execution time if we start comparing with the strategy number two:



And the solution found (the minimum) keeps exaclty as good as it was.

C5 and C6

Now we add the A matrix and start reading the matrices from files. Remember that in this problem we are working with dimension 100. The results obtained are exactly the ones expected: $1.15907181 \cdot 10^4$ for the "optpr1" folder and $1.08751156 \cdot 10^6$. This base is always constant and then the following decimals vary a little.

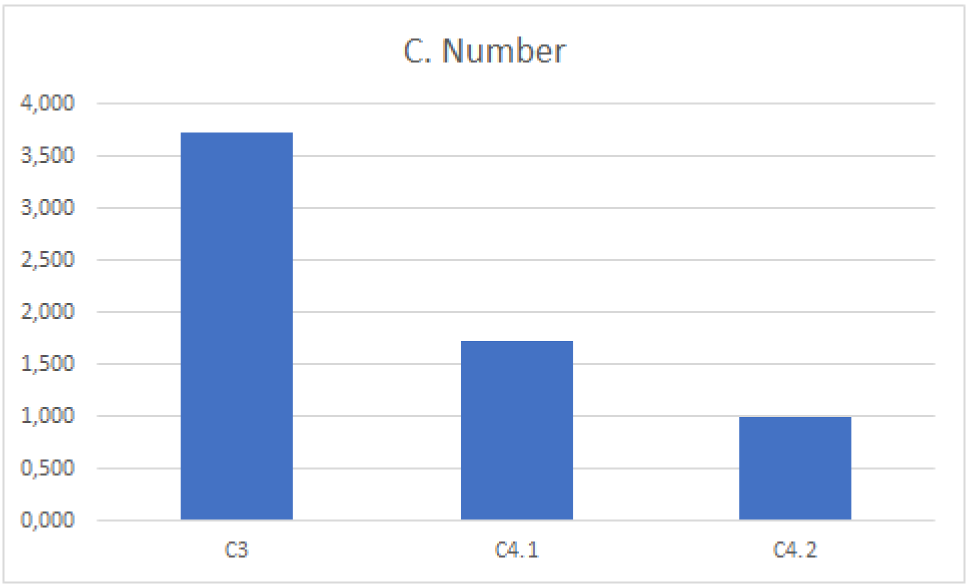
There's some issue here with the execution time: the C6 implementation (which we prove it to be better than C5) lasts longer than C5. That's because of the function *np.linalg.solve()*, which we are forced to use instead of *np.linalg.solve_trianglular()*. The first one takes longer than the second one but surprisely the second one does not output the correct matrices. Those are the mean execution times for both problems:

	Time (s)		Iterations	
	Optpr1	Optpr2	Optpr1	Optpr2
C5	0,2	127,23	21	28
C6	2,52	304,28	26	32

Condition Number

The condition number tells us how good conditioned a linear system is. This means that the closest it is to 1, the more "good" conditioned is. Let's say we solved a linear system with high condition number. If we then change an epsilon more the coefficients of our system (like adding or substracting 0.001), the new linear system found will have solutions "far away" from the previous ones. If the condition number is close to one, this new solutions will be close to the original ones. Let's check the condition number for every exercice:

In this case, we look at C3 and C4 with every strategy, and we see that we get closer to one:



But then, looking at C5 and C6...:

Code	C. Number
C3	3,726
C4.1	1,732
C4.2	1
C5, optpr1	1246556,347
C5, optpr2	8759727442
C6, optpr1	1246398,695
C6, optpr2	8759726976

We see that we get far away. This means that although we compute the solutions faster, our problem starts getting "bad conditioned".

Difficulties

I found one major difficulty, and it was related to the execution time. Suddenly, after coding one strategy, I got an execution time higher than the code without that strategy. At first I though it was a coding error or something like that, but in fact it was the way things we're created. For example: inversing a matrix can be done with *p.linalg.inv(matrix)* but it takes longer than doing: *np.diag(1/matrix).*

Another example was de *np.dot().* Matrix multiplication can be done concatenating this function, but you can actually optimize it in order to perform some multiplications first and some second.

Finally, as commented in the results section, *np.linalg.solve()* heavily increases the execution time if comparing to *np.linalg.solve_ trianglular()*, and that's another issue you have to notice.