Project 1 NLA

Eddie Conti

October 2023

Contents

1 Introduction to the problem and preliminaries

The aim of the project is to investigate how to address optimization problems with some ideas from numerical linear algebra. We are interested in solving the problem

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \ & \frac{1}{2}x^TGx + g^Tx \\ \text{subject to} \ & Ax = b, \quad C^Tx \geq d, \end{aligned} \tag{1}$$

where where $G \in \mathbb{R}^{n\times n}$ is symmetric semidefinite positive, $g \in \mathbb{R}^n$, $A \in \mathbb{R}^{n\times p}$, $C \in \mathbb{R}^{n\times m}$, $b \in \mathbb{R}^p$ and $d \in \mathbb{R}^m$.
In this scenario, we use the KKT theorem in order to tackle (1). In simple terms, the equalities established by Karush, Kuhn and Tucker allow us to approach the primal problem in a new way, by taking into account the constraints, or better, a weighted sum of the active constraints. Under the hypothesis of convexity of the problem, KKT conditions guarantee that any solution is a minimum of $f(x)$. Besides, the mathematical details, if we are willing to obtain the solution x^* to (1), then we have to solve the KKT conditions. The conditions are usually 4, and essentially express the following: the gradient of the objective function at the minimum point is a linear combination of the constraints at the minimum point with the associated Lagrangian multipliers; the constraints are satisfied at the minimum point (i.e. x^* lies in the feasible region); the multipliers associated to inequality constraints are grater or equal than 0 and finally the complementary condition that "captures" only the multipliers associated to active constraints.
In problem (1) the Lagrangian is given by

$$L(x, \gamma, \lambda, s) = \frac{1}{2}x^TGx + g^Tx - \lambda^T(A^Tx - b) - \lambda^T(C^Tx - d - s)$$

using the slack variables $s \geq 0$. The optimality conditions are the following:

$$\begin{aligned} Gx + g - A\gamma - C\lambda &= 0 \\ b - A^Tx &= 0 \\ s + d - C^Tx &= 0 \\ s_i\lambda_i &= 0, \quad i = 1, \dots, m. \end{aligned}$$

Due to the last term, the previous system is nonlinear, but using Newton's method we are able to determine the solution, solving a linear system. The idea is very simple: the problem lies in the last equation

$$s_i\lambda_i = 0,$$

Because it is a product. However by using differentiation rules it splits into a sum of two terms, which is linear. In the multidimensional case with a scalar function, the Newton's method relies on

$$f(x + h) = f(x) + J(x)h + O(||h^2||),$$

where J is the Jacobian matrix. We then have to solve the linear system $J(x)h = -f(x)$. In our case, we have to determine the solution to the linear system defined by the KKT

matrix:

$$M_{KKT} = \begin{pmatrix} G & -A & -C & 0 \\ -A^T & 0 & 0 & 0 \\ -C^T & 0 & 0 & I \\ 0 & 0 & S & \Lambda \end{pmatrix}$$

and right-hand vector $-F(z_0)$, where $z = (x, \gamma, \lambda, s)$. Here, the key point is that we have constructed a vectorial function $F : \mathbb{R}^N \to \mathbb{R}^N$, $N = n + 2m + p$, whose output are the equations in the linear system above, so that we can translate the optimality conditions into a root-finding problem that can be tackled using Newton's method.
In this setting, our project starts and we are required to solve the KKT system with different methods, to compare them, to deal with the general case (both inequality and equality constraints) and to ensure that the solution verifies the feasibility condition. In simple terms, to meet the last requirement we "push" the point toward the "diagonal" of the feasible region. In the next section we delve into and analyse the various aspects of the project and in particular the required codes.

2 Details of the implementation and answers to theoretical questions

Answer to T1: As already mentioned, Newton's method is a root-finding algorithm. It's geometric features are quite straightforward since the idea is to approximate the function with its best linear approximation. The method requires to solve $J(x)h = -f(x)$. In our case the function F is defined as it follows:

$$F(x, \gamma, \lambda, s) = (Gx + g - A\gamma - C\lambda, b - A^T x, s + d - C^T x, s\lambda),$$

where $s\lambda = (s_1\lambda_1, \ldots, s_m\lambda_m)$. As a consequence, given a point x we have to evaluate $-f(x)$ which is a vector and $J(x)$ which a matrix (the Jacobian matrix) of numbers.
Therefore, $J(x)h = -f(x)$ it is a linear system of the form $Ax = b$ (here A, b does not refer to the A, b of the problem). If we compute the derivative of our function F, treating this function as a vectorial function "with 4 output" we obtain the matrix M_{KKT}:

$$(\frac{\partial F}{\partial x}, \frac{\partial F}{\partial \gamma}, \frac{\partial F}{\partial \lambda}, \frac{\partial F}{\partial s}) = \begin{pmatrix} G & -A & -C & 0 \\ -A^T & 0 & 0 & 0 \\ -C^T & 0 & 0 & I \\ 0 & 0 & S & \Lambda \end{pmatrix}.$$

This answer why the prediction steps reduces to solve a linear system with matrix M_{KKT}.

Regarding code **C1** and **C2** we created 5 functions that performs different component of the general code:

- "solKKT": uses np.linalg.solve to solve the KKT system in order to obtain the step;
- "Newton-step": is the aldready provided code for the computation of α;
- "step-size-substep" computes effectively the parameters μ and σ;

- "creatematrix" creates the matrix associated to the KKT system. Particular attention is devoted to the dimension of each component;
- "evalfunc" evaluates the vectorial function $F : \mathbb{R}^N \to \mathbb{R}^N$ at a specific point.
- "test-problem" generates the test problem and all the variables.

With these function we implemented for a given n the full algorithm for the test problem. In **C3** we just took the computation time for $n \in \{10, 20, \dots, 100\}$ and we plotted the computation time as a function of size n.

Answer to T2: Since the matrix $A = 0$, the matrix M_{KKT} becomes

$$M_{KKT} = \begin{pmatrix} G & -C & 0 \\ -C^T & 0 & I \\ 0 & S & \Lambda \end{pmatrix},$$

which is a 3×3 block matrix. We denote $(-r_1, -r_2, -r_3)$ the right-hand vector. The variable is $\delta = (\delta_x, \delta_\lambda, \delta_s)$ and so the last equation is:

$$S\delta_\lambda + \Lambda\delta_s = -r_3 \implies \delta_s = \Lambda^{-1}(-r_3 - S\delta_\lambda).$$

If we substitute in the second equation we obtain

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

which can be rewritten as

$$-C^T\delta_x - \Lambda^{-1}S\delta_\lambda = -r_2 + \Lambda^{-1}r_3.$$

Hence we "removed" the variable δ_s and we end up with the following linear system

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

In this case we can use the LDLT factorization because we have a symmetric matrix. In order to prove it is sufficient to observe that for a block matrix

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

In our case,

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

which concludes the symmetry of the matrix M and so the possibility to use the LDLT factorization.
For the second strategy we isolate δ_s from the second row

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

and we substitute into the third row and isolate δ_λ:

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

To conclude, if we plug this value in the first row, this leads to

$$\begin{aligned} G\delta_x - C\delta_\lambda &= -r_1 \\ G\delta_x - C(S^{-1}(-r_3 + \Lambda r_2) - S^{-1}\Lambda C^T\delta_x) &= -r_1 \\ G\delta_x + CS^{-1}\Lambda C^T\delta_x &= -r_1 + CS^{-1}(-r_3 + \Lambda r_2) \\ \tilde{G}\delta_x &= -r_1 - \tilde{r} \end{aligned}$$

where $\tilde{G} = G + CS^{-1}\Lambda C^T$ and $\tilde{r} = -CS^{-1}(-r_3 + \Lambda r_2)$. We can apply Cholesky factorization to $\tilde{G}$: to do so we have to prove that the matrix is positive semi-definite and symmetric. We recall that S^{-1} and Λ are diagonal matrix and also that CC^T is a symmetric semi-definite matrix: the symmetry is trivial, while for the second part it is enough to observe that $x^TCC^Tx = (C^Tx)(C^Tx) \geq 0$. Let us call D the diagonal matrix $S^{-1}\Lambda$ whose entries are non-negative (due to the optimality conditions, $\lambda_i, s_i \geq 0$). Let us investigate the nature of CDC^T: since the entries of D are positive we can "take the square root" and so CDC^T=HH^T, where we distributed to C and C^T the square root of the elements of D. As a consequence, HH^T represents a symmetric semi-definite positive matrix M. Now, if we add $G + M$ it is immediate to conclude that it is a symmetric (it is sum of symmetric matrices) positive semi-definite matrix:

$$\underbrace{x^TGx}_{\geq 0} + \underbrace{x^TMx}_{\geq 0} \geq 0.$$

For code **C4** we implement the two strategies and computed the computation time and compared with the computation time in **C3**. We defined two functions that compute the two strategies following the procedure describe in the project. We tried to define by hand an LDLT factorization (the code is at the end of the python project) but then the computation time of the strategy 1 was terrible compared to the ordinary strategy and strategy 2. The problem is that python does not access to this function quickly as the ones already implemented in libraries. As a consequence, we used the scipy.ldl, we computed the solution and the computation time for each strategies and plotted. The graph shows that overall the ordinary strategy requires more time compared to strategy 1 and strategy 2. The second one is the best strategy in regard to computation time. We also stored the number of iteration for each n (for strategy 1 and 2 are almost the same), the relative error of the result.
At the end of the code we provided the trend of the condition number of the matrix involved in the system for both strategies. We set $n = 50$ and the trend shows that for strategy 1 when we are close to the solution the condition number increases, while for strategy 2 the condition number increase during the first iterations and then decrease when we are close to the solution, showing a better stability.

Most of the code **C5** was already included in the first code because we treated the problem in its generality so without assuming $A = 0$. Then we solved the problem with the files and the initial conditions provided. We only changed a few things in order to adapt the

function to work with sparse matrices: for example, instead of np.hstack we used hstack from scipy.sparse, or instead of np.dot we used .dot. Since the final matrix M_{KKT} is stored as sparse matrix, to take advantage from the structure we used scipy.sparse.linalg.spsolve to solve the linear system (we observed that it is faster than np.linalg.solve). As a consequence, the code is just a repetition of the first code with small adjustments. For example for the first case we need 25 iterations to get the solution. Also for the second dataset the code produces the expected solution of $1087511,56$ with 32 iterations.

The number of iterations is motivated by the effect of the central path method, which slows down the convergence to the solution, that counterbalance the quadratic convergence. Furthermore, we stored (for computational time matters) the condition number of the matrix in the first case: the graph shows that the condition number increases as we get closer to the solution.

Answer to T3: From the 4th row of M_{KKT} we isolated δ_s

$$S\delta_\lambda + \Lambda\delta_s = -r_4 \implies \delta_s = \Lambda^{-1}(-r_4 - S\delta_\lambda).$$

If we plug into the third row we obtain

$$\begin{aligned} -C^T\delta_x + \delta_s = -r_3 \implies -C^T\delta_x + \Lambda^{-1}(-r_4 - S\delta_\lambda) &= -r_3 \\ -C^T\delta_x - \Lambda^{-1}S\delta_\lambda &= -r_3 + \Lambda^{-1}r_4. \end{aligned}$$

The arising matrix is

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

which is clearly symmetric (it is the same reasoning provided in answer T2).

To conclude, we describe code **C6**: here, we are required to use the LDLT factorization to solve the two test problems. For stability reason, we have to work on the 2×2 blocks in d arising from scipy.linalg.ldl. We should detect the 2×2 blocks. To do so, since we are willing to solve a linear system, we defined a function that detects the blocks (it checks if the element next to the diagonal or under the diagonal is $> eps$) and solve the 2×2 system, while for the other elements it computes the ordinary solution dividing by the element in the diagonal. The problem is that in this case LDLT does not maintain the structure. In order to solve this problem, we applied the permutation matrix to L: if we compute $L[perm, :] = PL$, where P is a permutation matrix, then we end up with a lower triangular matrix. As a consequence, the linear system $Lt = z$ should be switched to $PLt = Pz$. They have the same solution but the last one takes advantage from the structure of the matrix. For the L^T part we have to operate the following:

$$L^T(P^TP)x = b \implies (PL)^T(Px) = b$$

because $(PL)^T$ is upper triangular. In this case, you solve two linear systems: one to obtain the solution with $(PL)^T$ and then to collect the solution x. We implemented this solution and printed the number of iterations. We stored also the condition numbers of d and lu and graphed the condition number of the former.

For both **C5** and **C6** we stored the computation time, the LDLT requires less time (even though we have to perform a lot of operations since we lose the structure of the matrix: we have indeed to perform permutation and to call another function that solves the 2x2 blocks in the diagonal).

2.1 Code C6, second dataset

For the second given dataset the code **C6** fails to solve the problem. Even though the code is the same code used before (which works) and the strategy3 works for the first dataset, here probably due to the high dimensionality of the problem, the solution does not converge. We checked the norm of ra, rc, rl and μ and they do not go to 0 as we should expect. We tried by tackling the problem in another way using the ldl factorization and then by using scipy.linalg.solve (see strategytest function) and in this case we have convergence but with 8 min of computation time. From here, we tried to modify this code to understand the problem: if we use all the code from strategy 3 except for the diagonal part (so we use scipy.linalg.solve) we converge in 30 iterations and in 5 min and 49 seconds.
Apparently the problem relies in the diagonal resolution, even though the two strategies that we implemented (fastblock and solve-system-with-2x2-blocks) work in the first dataset: by checking the trend we saw that the solution at each iteration is always far from the expected solution. It is still unusual the computation time, most probably it is due to the matrices coming from the ldl factorization.

3 Further contents related to problem (1)

In this short section we show that a more general class of problems reduce to (1).

Let us consider the ENLP (Equality constrained Non-Linear Programming problem) problem defined by

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & f(x) \\ \text{subject to} \quad & c(x) = 0, \end{aligned}$$

where c is a vector of m non-linear functions that is assumed to be continuous and differentiable and f is a differentiable function. From the KKT conditions, we are looking for x^*, λ^* solutions to the non-linear system

$$r(x,\lambda) = \begin{pmatrix} \nabla f(x) - J_c(x)^T\lambda \\ -c(x) \end{pmatrix} = 0$$

which expresses the conditions for finding a point of stationarity for the Lagrangian function of the ENLP problem. We can use the Newton's method applied to the multidimensional case, by initiating the following iterative process from an initial iterate (x_0, λ_0):

$$\begin{pmatrix} x^{k+1} \\ \lambda^{k+1} \end{pmatrix} = \begin{pmatrix} x^k \\ \lambda^k \end{pmatrix} + \begin{pmatrix} p_x^k \\ p_\lambda^k \end{pmatrix}$$

where p_x^k, p_λ^k are the solutions to the linear system

$$J_r(x^k, \lambda^k) \begin{pmatrix} p_x^k \\ p_\lambda^k \end{pmatrix} = -r(x^k, \lambda^k),$$

where we denoted with J_r the Jacobian matrix of $r(x, \lambda)$ In other words, we have the following

$$\begin{pmatrix} W_k & -J_c(x^k)^T \\ -J_c(x^k) & 0 \end{pmatrix} \begin{pmatrix} p_x^k \\ p_\lambda^k \end{pmatrix} = - \begin{pmatrix} \nabla f(x^k) - J_c(x^k)^T \lambda^k \\ -c(x^k) \end{pmatrix} \tag{2}$$

with

$$W_k = \nabla^2 f(x^k) - \sum_{i=1}^{m} \lambda_i^k \nabla^2 c_i(x^k).$$

From the first equation in (2) we obtain

$$\begin{aligned} W_k p_x^k - J_c(x^k)^T p_\lambda^k &= -\nabla f(x^k) + J_c(x^k)^T \lambda^k \\ W_k p_x^k - J_c(x^k)^T p_\lambda^k - J_c(x^k)^T \lambda^k &= -\nabla f(x^k) \\ W_k p_x^k - J_c(x^k)^T (p_\lambda^k + \lambda^k) &= -\nabla f(x^k) \\ W_k p_x^k - J_c(x^k)^T \lambda^{k+1} &= -\nabla f(x^k). \end{aligned}$$

Since the Lagrange multipliers are not involved in the second equation of (2), we can substitute λ^{k+1} to p_λ^k and obtain the system:

$$\begin{pmatrix} W_k & -J_c(x^k)^T \\ -J_c(x^k) & 0 \end{pmatrix} \begin{pmatrix} p_x^k \\ \lambda^{k+1} \end{pmatrix} = \begin{pmatrix} -\nabla f(x^k) \\ c(x^k) \end{pmatrix}.$$

It is immediate to observe, since the arising KKT system is the same, that each step of Newton's method is equivalent to the following problem:

$$\begin{aligned} \min_{p_x} \quad & \frac{1}{2} p_x^T W_k p_x + p_x^T \nabla f(x^k) \\ \text{subject to} \quad & c(x^k) + J_c(x^k) p_x = 0, \end{aligned}$$

which is of the form of problem (1) without inequality constraints. The same can be done in the case of

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & f(x) \\ \text{subject to} \quad & c(x) \geq 0, \end{aligned}$$

which reduces to

$$\begin{aligned} \min_{p_x} \quad & \frac{1}{2} p_x^T W_k p_x + p_x^T \nabla f(x^k) \\ \text{subject to} \quad & c(x^k) + J_c(x^k) p_x \geq 0. \end{aligned}$$

As a consequence, we can deduce that solving problem (1) allow us to tackle a more general class of problem and this gives greater importance and relevance to the problem.