Project 1 NLA:

Direct methods in optimization with constraints

Dafni Tziakouri

05/11/2023

2 Solving the KKT system

T1: Show that the predictor steps reduces to solve a linear system with matrix MKKT.

We are given a function $\mathrm{F}: R^N \to R^N, \mathrm{N} = \mathrm{n} + \mathrm{p} + 2\mathrm{m}$, where

$$\mathrm{F}(\mathrm{z}) = \mathrm{F}(\mathrm{x}, \gamma, \lambda, \mathrm{s}) = \frac{1}{2}\mathrm{x}^{\mathrm{T}}\mathrm{G}\mathrm{x} + \mathrm{g}^{T}\mathrm{x} - \gamma^{\mathrm{T}}(\mathrm{A}^{\mathrm{T}}\mathrm{x} - \mathrm{b}) - \lambda^{\mathrm{T}}(\mathrm{C}^{\mathrm{T}}\mathrm{x} - \mathrm{d} - \mathrm{s})$$

and to solve the following optimization problem $\mathrm{F}(\mathrm{z}) = 0$ we will use a Newton method.

We will start with a random point $\mathrm{z}_0 = (\mathrm{x}_0, \gamma_0, \lambda_0, \mathrm{s}_0)$.

By using the Newton's method we want to iteratively improve the guess for z. So, in each iteration, we compute a correction term δ_z using the formula:

$$\mathrm{F}(\mathrm{z}_0 + \delta_z) \approx \mathrm{F}(\mathrm{z}_0) + \mathrm{J}_{\mathrm{F}}(\mathrm{z}_0)\,\delta_z ,$$

where $\mathrm{J}_{\mathrm{F}(\mathrm{z}_0)}$ is the Jacobean matrix of F with respect to z.

Therefore, we need to solve the following equation $-\mathrm{F}(\mathrm{z}_0) = \mathrm{J}_\mathrm{F}(\mathrm{z}_0)\delta_\mathrm{z}$. This equation represents the KKT conditions for the optimization problem.

The matrix MKKT id defined as:

$$M_{KKT} = \mathrm{J}_\mathrm{F}(\mathrm{z}_0) = \begin{bmatrix} \frac{\partial F_1}{\partial x} & \frac{\partial F_1}{\partial \gamma} & \frac{\partial F_1}{\partial \lambda} & \frac{\partial F_1}{\partial s} \\ \frac{\partial F_2}{\partial x} & \frac{\partial F_2}{\partial \gamma} & \frac{\partial F_2}{\partial \lambda} & \frac{\partial F_2}{\partial s} \\ \frac{\partial F_3}{\partial x} & \frac{\partial F_3}{\partial \gamma} & \frac{\partial F_3}{\partial \lambda} & \frac{\partial F_3}{\partial s} \\ \frac{\partial F_4}{\partial x} & \frac{\partial F_4}{\partial \gamma} & \frac{\partial F_4}{\partial \lambda} & \frac{\partial F_4}{\partial s} \end{bmatrix} = \begin{bmatrix} G & -A & -C & 0 \\ -A^T & 0 & 0 & 0 \\ -C^T & 0 & 0 & I \\ 0 & 0 & S & \Lambda \end{bmatrix}$$

We need to solve the linear system:

$\mathrm{M}_{\mathrm{KKT}} * \delta_\mathrm{z} = -\mathrm{F}(\mathrm{z}_0)$, where $\delta_\mathrm{z} = \mathrm{x}_{k+1} - \mathrm{x}_\mathrm{k}$ the step between the points of the iterations.

After obtaining δ_z, we will update our guess for z as:

$\mathrm{z}_{\mathrm{k+1}} = \mathrm{z}_\mathrm{k} + 0.95 * \alpha_\mathrm{k} * \delta_\mathrm{z}$, here a_k is a step size that we can adapt or reduce in each iteration to ensure convergence.

To sum up $\mathrm{M}_{\mathrm{KKT}}$ is vital in the Newton method for optimization as it captures how constraints and objectives respond to variable changes. By solving the linear system $\mathrm{M}_{\mathrm{KKT}} * \delta_\mathrm{z} = -\mathrm{F}(\mathrm{z}_0)$ is pivotal for reaching a solution that meets the KKT conditions and optimizes the problem.

C1: Write down a routine function that implements the step-size substep**.**

It can be found in the attached Python file, under the “C1” tag.

2.1 Inequality constrains case (i.e. with A = 0)

C2: Write down a program that, for a given n, implements the full algorithm for the test problem. Use the numpy.linalg.solve function to solve the KKT linear systems of the predictor and corrector substeps directly.

It can be found in the attached Python file, under the “C2” tag.

C3: Write a modification of the previous program C2 to report the computation time of the solution of the test problem for different dimensions n.

It can be found in the attached Python file, under the “C3” tag.

When we applying an upper threshold of 100 iterations, dimensions of matrix (nxn) =5x5, and epsilon= 10e-16 the results are the following:

```
The computational time for the test problem is equal to:
0.05760073661804199
The minimum of the function was found: -7.5524074244333965
The real minimum is: -7.552407424433394
Iterations needed: 13
Condition number: 23.739153102014566
```

We can notice that the computational time for this program is very short, which provide us information about the efficiency or our implementation.

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

Strategy 1:

We isolate δ_s from the 3rd row of the $\mathrm{M_{KKT}}$ matrix, as:

$$\delta_s = \Lambda^{-1}(-r_3 - Sd_\lambda)$$

Therefore, $(-C^T \quad 0 \quad 1)(\delta_x\ \delta_\lambda\ \delta_s)^T = -r_2 \quad => -C^T\delta_x + \delta_s = -r_2$

Now we can substitute δ_s from the previous equality:

$$-C^T\delta_x - \Lambda^{-1}(-r_3 - Sd_\lambda) = -r_2$$

$$=> \quad -C^T\delta_x - \Lambda^{-1}(Sd_\lambda) = (-r_2 - \Lambda^{-1}r_3)$$

To solve this equation, Λ^{-1} must be available and invertible.

Strategy 2:

In a similar way, we isolate δ_s from the 2nd row of the $\mathrm{M_{KKT}}$ matrix, as:

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

and then we substitute this expression for δ_s into the 3rd row:

$$S\delta_s + \Lambda\delta_s = -r_3$$

$$=> \qquad S\delta_s + \Lambda(-r_2 + C^T\delta_x) = -r_3$$

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

Finally, we substitute these expressions into the 1st row to obtain a smaller linear system:

$$\hat{G}\delta_x = -r_1 - \hat{r}$$

This smaller system can be solved using Cholesky factorization and we assume that S is invertible and Λ is available.

C4: Write down two programs (modifications of C2) that solve the optimization problem for the test problem using the previous strategies. Report the computational time for different values of n and compare with the results in C3.

It can be found in the attached Python file, under the "C4.Strategy 1" and "C4.Strategy 2" tags.

Algorithm	N (dimension)	Iterations	Time consumption	Condition number
C3	10	13	0.08610	23.74
C4_LDL	10	13	0.02613	28917368231557.566
C4_Cholesky	10	13	0.01459	1.00

Algorithm	N (dimension)	Iterations	Time consumption	Condition number
C3	50	15	0.42031	24.33
C4_LDL	50	15	0.09993	7715279992153.37
C4_Cholesky	50	15	0.04391	1.00

Algorithm	N (dimension)	Iterations	Time consumption	Condition number
C3	100	14	0.61739	24.77
C4_LDL	100	14	0.31444	23750065446450.97
C4_Cholesky	100	14	0.10595	1.00

We notice that for all values of n, the C4_Cholesky method has the shortest computation time, making it the fastest among the three methods.

Also, the number of iterations required for convergence is relatively consistent across all three methods for a given n. This suggests that the number of iterations required doesn't vary significantly with n.

Furthermore, the C4_LDL method shows high condition numbers for all values of n. This can make the numerical solution unstable and less reliable. On the other hand, C4_Cholesky method consistently reports a condition number very close to 1, suggesting that the Cholesky factorization method used in this approach provides stable and well-conditioned results.

2.2 General case (with equality and inequality constrains)

C5: Write down a program that solves the optimization problem for the general case. Use numpy.linalg.solve function. Read the data of the optimization problems from the files (available at the Campus Virtual). Each problem consists on a collection of files: G.dad, g.dad, A.dad, b.dad, C.dad and d.dad. They contain the corresponding data in coordinate format.

It can be found in the attached Python file, under the “C5” tag.

The results obtained by the code using the vectors and matrices given in “optpr1” and “optpr2” are the following:

```
For matrices and vectors from optpr1, the obtained results are the
following:

Computation time:  1.4867184162139893
Minimum was found: 11590.718119426767
Condition number: 1.9495161760426004e+18
Iterations needed: 25
```

```
For matrices and vectors from optpr2, the obtained results are the
following:

Computation time:  345.94063329696655
Minimum was found: 1087511.5673215014
Condition number: 5.554413718885413e+18
Iterations needed: 28
```

We obtained that:

1. Optpr2 requires significantly more computational time compared to optpr1, indicating that optpr2 is more computationally intensive.

2. Optpr1 has a lower minimum objective function value than optpr2, suggesting that it is an easier optimization problem to solve.

3. The condition number of the KKT matrix for optpr2 is much higher than that of optpr1, indicating that optpr2 is a more ill-conditioned problem, which can make optimization more challenging.

4. While both problems converge within the specified tolerance, optpr2 requires slightly more iterations to reach convergence than optpr1.

T3: Isolate δs from the 4th row of MKKT and substitute into the 3rd row. Justify that this procedure leads to a linear system with a symmetric matrix.

Let's write down the equations for the general case:

$$\text{Equation 1:} \quad G\delta_x - A\delta_\lambda - C\,\delta_\lambda = -r_1$$

$$\text{Equation 2:} \quad -A\,\delta_x = -r_2$$

$$\text{Equation 3:} \quad \delta_s - C^T\delta_x = -r_3$$

$$\text{Equation 4:} \quad \Lambda\delta_s + S\,\delta_\lambda = -r_4$$

We will follow the same procedure that has been demonstrated in T2, but now isolating $\delta_s$from the 4th row instead of the 2nd and 3rd row. So, we get the following equation for δ_s:

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

If we now substitute δ_s in the 3rd row, we get the following equality:

$$-C^T\delta_x - \Lambda^{-1}(r_4 + S\delta_\lambda) = -r_3$$

If we convert this to a matrix form, we get the following result:

$$\begin{pmatrix} G & -A & -C \\ -A^T & 0 & 0 \\ -C^T & 0 & -S\Lambda^{-1} \end{pmatrix} (\delta_x \quad \delta_\lambda \quad \delta_s)^T = (-r_1 \quad -r_2 \quad \Lambda^{-1}r_4 - r_3)^T$$

In conclusion, the final matrix resulting from the procedure is indeed symmetric because is guaranteed by the properties of the matrices involved. Specifically, the matrix $-S\Lambda^{-1}$is always symmetric because it arises from the product of two diagonal matrices. As it was defined in the start of this project, matrix $G \in R^{nxn}$ is a symmetric semi definite positive matrix, and the matrix we have found matches the characteristics of matrix G, further confirming its symmetry.

C6: Implement a routine that uses LDLT to solve the optimizations problems (in C5) and compare the results.

It can be found in the attached Python file, under the “C6” tag.

The results obtained by the code using the vectors and matrices given in “optpr1” and “optpr2” are the following:

```
For matrices and vectors from optpr1, the obtained results where the
following:

Computation time:  1.833892583847046
Minimum was found: 11590.718119426772
Condition number: 6.375699758041506e+21
Iterations needed: 31


For matrices and vectors from optpr2, the obtained results where the
following:

Computation time:  545.8264486789703
Minimum was found: 1087511.5673214972
Condition number: 8.859261568310938e+22
Iterations needed: 34
```

In C5 algorithm we use the numpy.linalg.solve factorization function and in C6 algorithm the LDLT factorization function. Let's compare these two approaches:

1. In general, in C5 algorithm we had shorter computation times compared to C6 algorithm, because the additional LDLT factorization steps can be computationally expensive.

2. The minimum values found in both C5 and C6 algorithms are similar. It indicates that both methods successfully converged to the same or very close optima.

3. The condition number of the KKT matrix in C5 and C6 algorithms differs significantly. C6 algorithm typically results in a much higher condition number, this can indicate that the matrix is ill-conditioned and that small changes in the input data or the optimization problem formulation can lead to significant changes in the solution. This implies that C6 algorithm might produce less stable solutions for certain problem instances.

4. In C6 algorithm it required more iterations to converge than in C5 algorithm. This could be due to the additional LDLT factorization and the step-size correction substeps involved in C6. More iterations generally translate to longer computation times.

To conclude, both numpy.linalg.solve factorization function and LDLT factorization function are capable of solving optimization problems, but they have different characteristics.
C5 algorithm is relatively faster and might be preferred for problems where computational speed is a priority. On the other hand, C6 algorithm utilizes LDLT factorization, which can be more numerically stable for some problems. It might be preferred for problems where the condition number of the KKT matrix is a concern.

As a final project comment, we can say that we have seen that Cholesky factorization function is the fastest way to get a result, and is the most precise method with a better condition number.