Assignment - Constrained optimization Interior Point Method applied to linear programming problems

Matteo Bunino

January 22, 2020

1 Problem definition

The Predictor-Corrector is an Interior Point Method built to iteratively solve a nonlinear system of equations, obtained by defining the KKT condition for the pair primal-dual problem.
The linear system to solve is:

$$F(x, \lambda, s) = \begin{bmatrix} Ax - b \\ s + A^T\lambda - c \\ XSe \end{bmatrix} = 0$$

According to Newton method for nonlinear system of equation, the system can locally linearized and solved. To linearize this system is necessary to compute the Jacobian of F at each iteration and solve the linear system:

$$F'(x_k, \lambda_k, s_k) \cdot \begin{pmatrix} \Delta x_k \\ \Delta\lambda_k \\ \Delta s_k \end{pmatrix} + F(x_k, \lambda_k, s_k) = 0$$

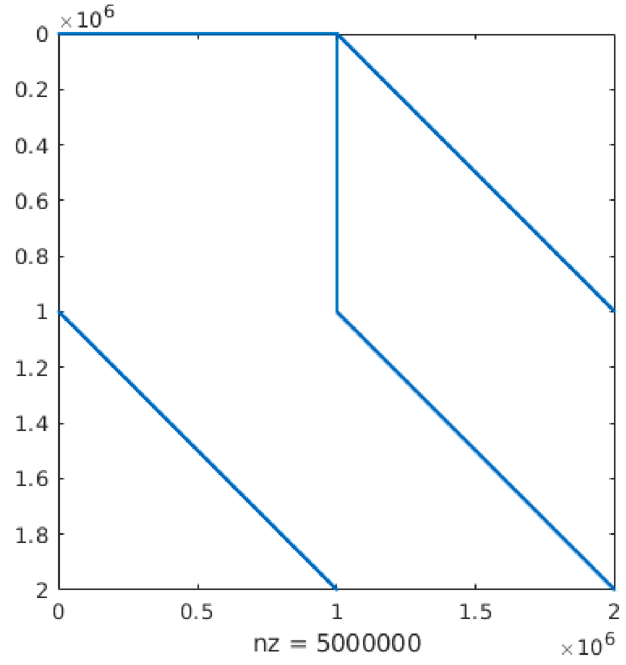
$$\begin{pmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S_k & 0 & X_k \end{pmatrix} \cdot \begin{pmatrix} \Delta x_k \\ \Delta\lambda_k \\ \Delta s_k \end{pmatrix} + F(x_k, \lambda_k, s_k) = 0$$

Where $X_k = diag(x_k)$ and $S_k = diag(s_k)$.
Interior point methods are variants of Newton method that guarantee that each iterate (x_k, λ_k, s_k) satisfy the constraint $x, s \geq 0$.

2 Problem data

$A = (1, \dots 1) \in \Re^{1*n}$,
$b = 1$,
$c \in \Re^n : c_i = a$ if i is odd, otherwise 1.
$x \in \Re^n$.

After having built the sparse jacobian of F, using the command spy() it's possible to inspect its structure:



Using the command [R,p] = chol(J) it's possible to notice, thanks to the value of p, that the matrix isn't positive definite. Hence I could not use: gradient method, conjugate gradient or Cholesky decomposition.
Since this matrix isn't symmetric, I couldn't use the LDL decomposition neither. Hence I used two different strategies: LU decomposition and system reduction.

2.1 LU decomposition

As suggested in its documentation, Matlab has a specific implementation of the command lu() which is efficient for sparse matrices: [L,U,P,Q] = lu(A).
This command makes the $PAQ = LU$ decomposition of a matrix, where L, U are triangular matrices and P, Q permutation matrices.

A linear system can be solved with this factorization the following way:

$$\begin{aligned} PA &= LUQ^{-1} \\ Ax = b \implies PAx &= Pb \\ LUQ^{-1}x &= Pb \end{aligned}$$

1. $Lz = Pb \implies z$
2. $Uy = z \implies y$
3. $Q^{-1}x = y \implies x = Qy$

The advantage of this method is that we have to factorize the jacobian of F only once per each iteration and we can use it to solve two linear systems where only the right hand vector has changed.

2.2 System reduction

Analyzing the structure of the block matrix, we can notice that it's possible to make some reductions. Rewriting the left hand part in a more compact way:

$$\begin{pmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S_k & 0 & X_k \end{pmatrix} \cdot \begin{pmatrix} \Delta x_k \\ \Delta\lambda_k \\ \Delta s_k \end{pmatrix} = \begin{pmatrix} r_a \\ r_b \\ r_c \end{pmatrix}$$

From the second equation is possible to to write: $\Delta s = r_b - A^T\Delta\lambda$.
Since X is invertible ($x > 0$ is guarateed by the IPM) it's possible to eliminate I from the block matrix: $R_1 = R_1 - X^{-1}R_3$ (R_i is the i-th row):

$$\begin{pmatrix} A & 0 & 0 \\ -X^{-1}S & A^T & 0 \\ S & 0 & X \end{pmatrix} \cdot \begin{pmatrix} \Delta x \\ \Delta\lambda \\ \Delta s \end{pmatrix} = \begin{pmatrix} r_a \\ r_b - X^{-1}r_c \\ r_c \end{pmatrix}$$

Now Δs can be extracted from the system: $\Delta s = X^{-1}r_c - X^{-1}S\Delta x$, obtaining:

$$\begin{pmatrix} A & 0 \\ -X^{-1}S & A^T \end{pmatrix} \cdot \begin{pmatrix} \Delta x \\ \Delta\lambda \end{pmatrix} = \begin{pmatrix} r_a \\ r_b - X^{-1}r_c \end{pmatrix}$$

Since S is invertible ($x > 0$ is guarateed by the IPM) it's possible to eliminate A from the block matrix: $R_1 = R_1 + AS^{-1}XR_2$ (R_i is the i-th row):

$$\begin{pmatrix} 0 & AS^{-1}XA^T \\ -X^{-1}S & A^T \end{pmatrix} \cdot \begin{pmatrix} \Delta x \\ \Delta\lambda \end{pmatrix} = \begin{pmatrix} r_a + AS^{-1}X(r_b - X^{-1}r_c) \\ r_b - X^{-1}r_c \end{pmatrix}$$

From which we can obtain:

$$\begin{aligned}
AS^{-1}XA^T\Delta\lambda &= r_a + AS^{-1}Xr_b - AS^{-1}r_c \\
\Delta s &= r_b - A^T\Delta\lambda \\
\Delta x &= -S^{-1}X(r_b - A^T\Delta\lambda - X^{-1}r_c) \\
&= -S^{-1}X(\Delta s - X^{-1}r_c) \\
&= S^{-1}r_c - S^{-1}X\Delta s
\end{aligned}$$

The matrix $AS^{-1}XA^T \in \Re^{mxm}$ is a real number, since in this case $m = 1$. This makes the computation of the system trivial, avoiding the problem of working with singular or indefinite matrices.

3 Results

The reduction of the Jacobian was motivate by the unfeasibility of the problem when n gets very large (1M in this case), leading to a memory error.
To solve the problem with a direct method suited for an indefinite non symmetric matrix I used the LU decomposition, which is slightly less time consuming than the Gaussian elimination and it doesn't need to be computed twice for each iteration (we have to solve two linear systems with the same coefficient matrix).

What improved the performances, indeed, was the chosen starting point. We know that it's not necessary for the starting point to exactly fulfill all the first order optimality conditions, but just $x, s > 0$.
I tried 3 starting configurations:

a. $x = ones(n, 1)$, $s = ones(n, 1)$, $\lambda = 1$.

b. $x = 100 * ones(n, 1)$, $s = 100 * ones(n, 1)$, $\lambda = 100$.

c. A method showed on the Nocedal Wright book to choose a good starting configuration. On the book it's said that the complexity of this operation is similar to the complexity of one iteration of the method.

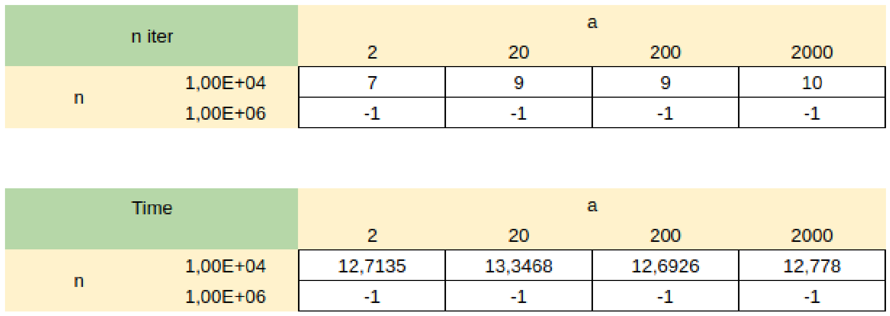
To compute the starting point:

1. $\tilde{x} = A^T(AA^T)^{-1}b$, $\tilde{\lambda} = (AA^T)^{-1}Ac$, $\tilde{s} = c - A^T\tilde{\lambda}$
2. $\delta_x = max(-1.5min_i(\tilde{x_i}), 0)$, $\delta_s = max(-1.5min_i(\tilde{s_i}), 0)$
3. $\hat{x} = \tilde{x} + \delta_x e$, $\hat{s} = \tilde{s} + \delta_s e$
4. $\hat{\delta_x} = \frac{\hat{x}^T\hat{s}}{2e^t\hat{s}}$, $\hat{\delta_s} = \frac{\hat{x}^T\hat{s}}{2e^t\hat{x}}$
5. $x^0 = \hat{x} + \hat{\delta_x}e$, $\lambda^0 = \tilde{\lambda}$, $s^0 = \hat{s} + \hat{\delta_s}e$

For all the results reported below, in the stopping criterion I used $\epsilon = 10^{-16}$.

3.1 Starting configuration a

Results using LU factorization.

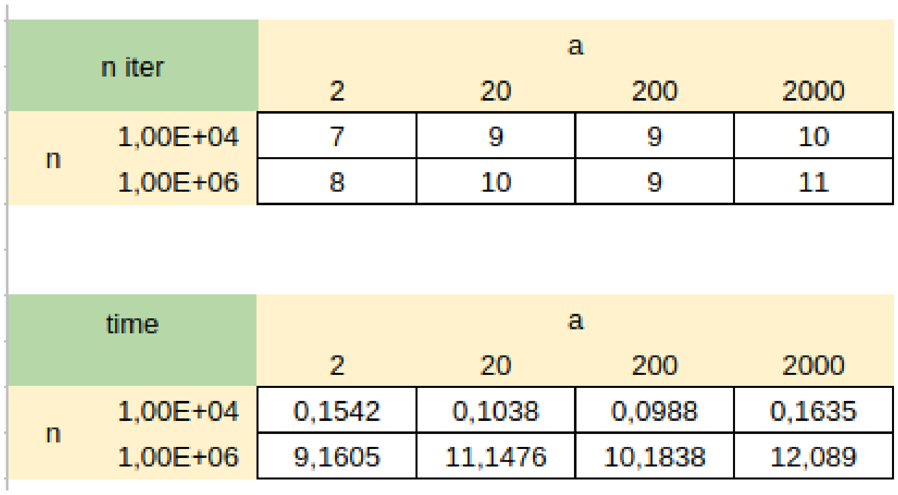


n iter		a			
		2	20	200	2000
n	1,00E+04	7	9	9	10
	1,00E+06	-1	-1	-1	-1

Time		a			
		2	20	200	2000
n	1,00E+04	12,7135	13,3468	12,6926	12,778
	1,00E+06	-1	-1	-1	-1

Figure 1: Starting point of all ones.

Results using reduction of the jacobian.



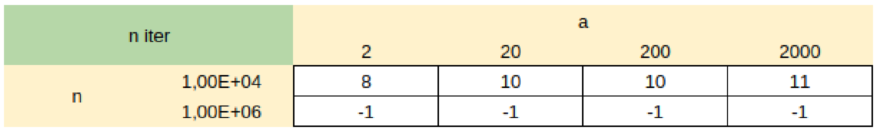
n iter		a			
		2	20	200	2000
n	1,00E+04	7	9	9	10
	1,00E+06	8	10	9	11

time		a			
		2	20	200	2000
n	1,00E+04	0,1542	0,1038	0,0988	0,1635
	1,00E+06	9,1605	11,1476	10,1838	12,089

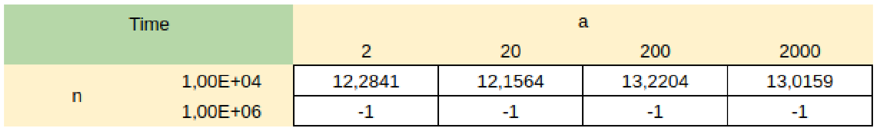
Figure 2: Starting point of all ones.

3.2 Starting configuration b

Results using LU factorization.



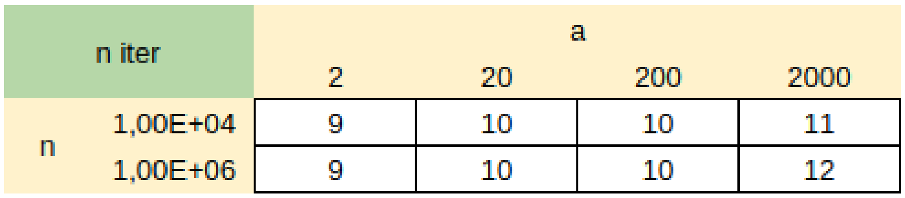
n iter		a			
		2	20	200	2000
n	1,00E+04	8	10	10	11
	1,00E+06	-1	-1	-1	-1



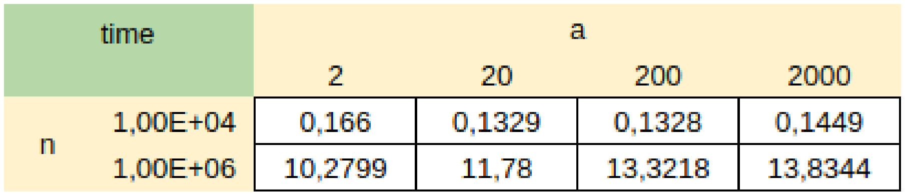
Time		a			
		2	20	200	2000
n	1,00E+04	12,2841	12,1564	13,2204	13,0159
	1,00E+06	-1	-1	-1	-1

Figure 3: Starting point of all 100.

Results using reduction of the jacobian.



n iter		a			
		2	20	200	2000
n	1,00E+04	9	10	10	11
	1,00E+06	9	10	10	12

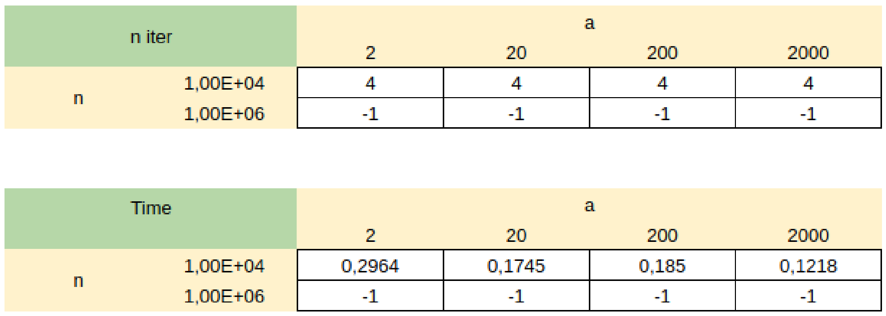


time		a			
		2	20	200	2000
n	1,00E+04	0,166	0,1329	0,1328	0,1449
	1,00E+06	10,2799	11,78	13,3218	13,8344

Figure 4: Starting point of all 100.

3.3 Starting configuration c

Results using LU factorization.

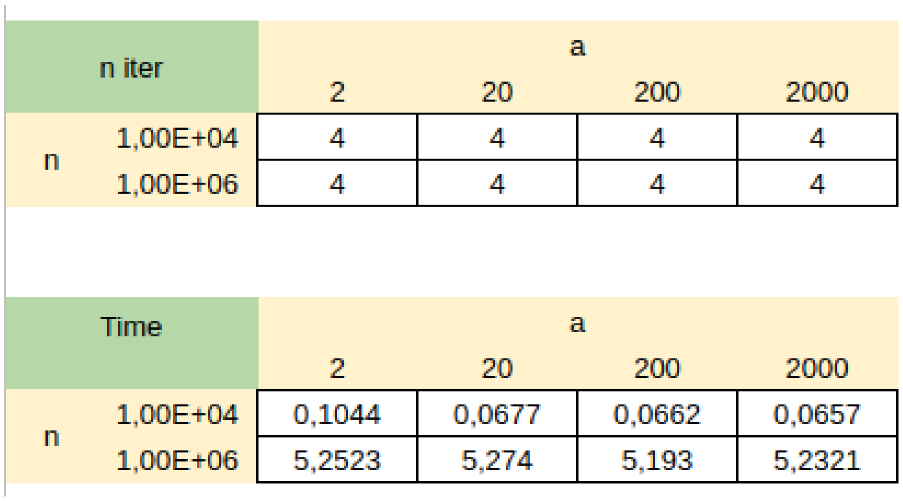


n iter		a			
		2	20	200	2000
n	1,00E+04	4	4	4	4
	1,00E+06	-1	-1	-1	-1

Time		a			
		2	20	200	2000
n	1,00E+04	0,2964	0,1745	0,185	0,1218
	1,00E+06	-1	-1	-1	-1

Figure 5: Starting point as suggested on the book.

Results using reduction of the jacobian.



n iter		a			
		2	20	200	2000
n	1,00E+04	4	4	4	4
	1,00E+06	4	4	4	4

Time		a			
		2	20	200	2000
n	1,00E+04	0,1044	0,0677	0,0662	0,0657
	1,00E+06	5,2523	5,274	5,193	5,2321

Figure 6: Starting point as suggested on the book.

4 Comments

As we can see, the only difference between the two different approaches is the execution time. This is because the method is the same, I'm just using different ways of computing it...
The worse is the starting point, the longer it takes to converge...
Code?
Quale epsilon (tolelranza) ho usato?
Complessit dello starting point dato dal libro
Aggiungi foto L, U gi presenti nella cartella code. Eventualmente rifai i calcoli per misurare i tempi.
La fattorizzazione che ho usato A = LU ([L,U] = lu(A)) per L in queto modo non una matrice triangolare. Devo usare PA = LU ([L,U,P] = lu(A)) che per matrici sparse diventa PAQ = LU ([L,U,P,Q] = lu(A)).