

Assignment - Constrained optimization

Interior Point Method applied to linear programming problems

Matteo Bunino

January 21, 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:

$$F'(x_k, \lambda_k, s_k) \cdot \begin{pmatrix} \Delta x_k \\ \Delta \lambda_k \\ \Delta s_k \end{pmatrix} + F(x_k, \lambda_k, s_k) = 0$$
$$\begin{pmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S_k & 0 & X_k \end{pmatrix} \cdot \begin{pmatrix} \Delta x_k \\ \Delta \lambda_k \\ \Delta s_k \end{pmatrix} + F(x_k, \lambda_k, s_k) = 0$$

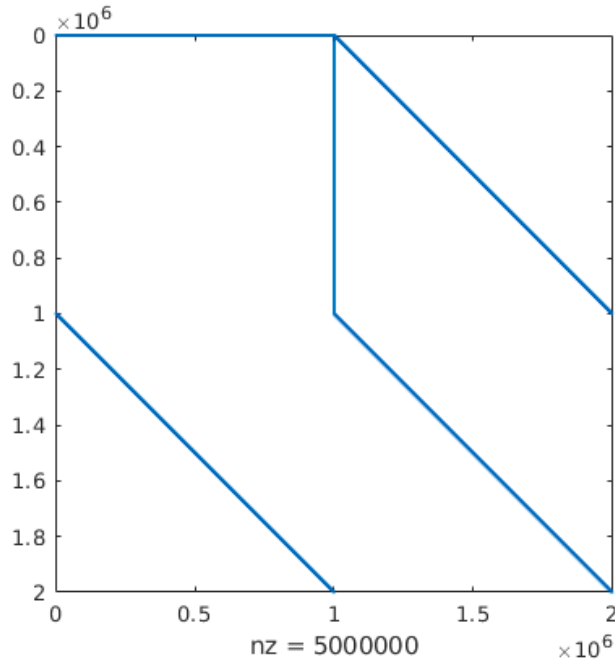
Where $X_k = \text{diag}(x_k)$ and $S_k = \text{diag}(s_k)$.

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

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

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 didn't use the LDL decomposition neither.

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:

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

Since X is invertible ($x > 0$ is guaranteed by the IPM), multiplying $-X^{-1}$:

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

S is invertible for the same reason:

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

The matrix $AS^{-1}XA^T \in \mathbb{R}^{m \times m}$ 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 motivated 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. $x = \text{ones}(n, 1)$, $s = \text{ones}(n, 1)$, $\lambda = 1$.
- b. $x = 100 * \text{ones}(n, 1)$, $s = 100 * \text{ones}(n, 1)$, $\lambda = 100$.
- c. a method showed on the Nocedal Wright book to choose a good starting configuration.

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

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. $\delta_x = \max(-1.5\min\tilde{x}_i, 0)$, $\delta_s = \max(-1.5\min\tilde{s}_i, 0)$
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. $x^0 = \hat{x} + \hat{\delta}_x e$, $\lambda^0 = \tilde{\lambda}$, $s^0 = \hat{s} + \hat{\delta}_s e$

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 (toleranza) 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 $A = LU$ ($[L,U] = \text{lu}(A)$) per L in questo modo non è una matrice triangolare. Devo usare $PA = LU$ ($[L,U,P] = \text{lu}(A)$) che per matrici sparse diventa $PAQ = LU$ ($[L,U,P,Q] = \text{lu}(A)$).