POLITECNICO
DI TORINO

Predictor-Corrector Interior Point Method for a Linear Programming problem

---

Course: Numerical optimization for large scale problems and Stochastic Optimization

Professors: Sandra Pieraccini, Enrico Bibbona

Students: Andrea Rubeis, Nicola Scarano

Academic year 2020-2021

Index

Abstract

In this report we present our implementation of the Predictor-Corrector Interior Point Method and we discuss the result obtained from its application on a linear programming problem.

Theoretical background

$$\text{(LP)} \quad \begin{matrix} \min c^T x \\ s.t. Ax = b \\ x \geq 0 \end{matrix} \qquad \text{(LD)} \quad \begin{matrix} max\ b^T \lambda \\ s.t.\ \ c = s + A^T \lambda \\ s \geq 0 \end{matrix} \text{ with } \begin{matrix} A \in \mathbb{R}^{mxn}, \text{ full rank} \\ b \in \mathbb{R}^m \\ c \in \mathbb{R}^n \\ \lambda \in \mathbb{R}^n \\ s \in \mathbb{R}^n \\ x \in \mathbb{R}^n \end{matrix}$$

We call the objective function $f(x)$ for LP case and $q(\lambda, s)$ for the LD case.

Considering the LP problem, we can write its Lagrangian function $\mathcal{L}(x, \lambda, s) = c^T x - s^T x - \lambda(Ax - b)$ and its KKT conditions which are the same for the LD problem and they can be written as:

$$\left\{ \begin{matrix} \mathrm{c} - \mathrm{s} - \mathrm{A}^T\lambda = 0 \\ \mathrm{b} - \mathrm{Ax} = 0 \\ \mathrm{x}^T\mathrm{s} = 0 \\ \mathrm{s} \geq 0 \\ \mathrm{x} \geq 0 \end{matrix} \right. \quad \left. \begin{matrix} (\nabla_x \mathcal{L}(\mathrm{x}, \lambda, \mathrm{s}) = 0) \\ (\nabla_\lambda \mathcal{L}(\mathrm{x}, \lambda, \mathrm{s}) = 0 \\ s^T \mathrm{g(x)} = 0 \\ \mathrm{s} \geq 0 \\ \mathrm{g(x)} \geq 0 \end{matrix} \right\} \rightarrow \left\{ \begin{matrix} \mathrm{F(x, \lambda, s)} = 0 \\ \mathrm{s} \geq 0 \\ \mathrm{x} \geq 0 \end{matrix} \right\} \text{where } \mathrm{F(x, \lambda, s)} = \begin{bmatrix} \mathrm{A}^T\lambda + \mathrm{s} - \mathrm{c} \\ Ax - b \\ XSe \end{bmatrix} \quad (XSe = x^T s),$$

We replace $x^T s$ with XSe to transform the linear system $\mathrm{F(x, \lambda, s)}$ in a square one, by using the diagonal matrices X,S and multiply them by the canonical vector e, so we can now solve it using the Newton method for non linear problems.

In the IPM we want to solve the non linear system $\mathrm{F(x, \lambda, s)} = 0$ by taking two steps: *predictor* and *corrector* step. In the first one we want to find the point $(\Delta \mathrm{x}_k{}^{aff}, \Delta \lambda_k{}^{aff}, \Delta \mathrm{s}_k{}^{aff})$ which has to minimize as much as possible XSe satisfying the two equalities $\mathrm{A}^T\lambda + \mathrm{s} - \mathrm{c} = 0$ and $Ax - b = 0$.

$$F'(\mathrm{x}_k, \lambda_k, \mathrm{s}_k) \begin{bmatrix} \Delta \mathrm{x}_k{}^{aff} \\ \Delta \lambda_k{}^{aff} \\ \Delta \mathrm{s}_k{}^{aff} \end{bmatrix} = -\mathrm{F}(\mathrm{x}_k, \lambda_k, \mathrm{s}_k) \rightarrow \begin{bmatrix} 0_{\mathrm{n \times m}} & A^T & I_{\mathrm{n}} \\ A & 0_{\mathrm{m \times n}} & 0_{\mathrm{m \times n}} \\ S & 0_{\mathrm{n \times n}} & X \end{bmatrix} \begin{bmatrix} \Delta \mathrm{x}_k{}^{aff} \\ \Delta \lambda_k{}^{aff} \\ \Delta \mathrm{s}_k{}^{aff} \end{bmatrix} = \begin{bmatrix} \mathrm{c} - \mathrm{s} - \mathrm{A}^T\lambda_k \\ b - A\mathrm{x}_k \\ -\mathrm{x}_k \mathrm{s}_k e \end{bmatrix}$$

Once we solved this linear system for $\Delta \mathrm{x}_k{}^{aff}, \Delta \lambda_k{}^{aff}, \Delta \mathrm{s}_k{}^{aff}$, we choose the steplengths α_P^{aff} and α_D^{aff} such that both $x_{k+1} = x_k + \alpha_P^{aff} \Delta x_k{}^{aff}$ and $s_{k+1} = s_k + \alpha_D^{aff} \Delta s_k{}^{aff}$ are strictly positive.

To ensure this, for every index $i \in \{1, \dots, n\}$ we want to find α_P^{aff} such that $(x_k)_i + \alpha_P^{aff} (\Delta x_k{}^{aff})_i > 0$. Since in the predictor step we move from one feasible iteration to another one, we already know that $(x_k)_i$ is positive because it is feasible, if the $(\Delta x_k{}^{aff})_i$ just computed is positive, then we can take any positive value

of α_P^{aff}(tipically $\alpha_P^{aff} = 1$), on the other hand if it is negative we have to solve the inequality $x_k + \alpha_P^{aff} {\Delta x_k}^{aff} > 0 \rightarrow \alpha_P^{aff} < -\frac{x_k}{{\Delta x_k}^{aff}}$

So, in order to have an α_P^{aff} such that for all the components we have a new feasible step, we have to take the minimum between 1 ($({\Delta x_k}^{aff})_i$ positive) and the minimum among all the values computed ($({\Delta x_k}^{aff})_i$ negative). Same reasoning can be done for α_D^{aff} to find the feasible iteration $s_{k+1} = s_k + \alpha_D^{aff} {\Delta s_k}^{aff}$

$$\alpha_P^{aff} = min\left\{1, min\left\{-\frac{(x_k)_i}{\left({\Delta x_k}^{aff}\right)_i} | i = 1, \dots, n, \left({\Delta x_k}^{aff}\right)_i < 0\right\}\right\}$$

$$\alpha_D^{aff} = min\left\{1, min\left\{-\frac{(s_k)_i}{\left({\Delta s_k}^{aff}\right)_i} | i = 1, \dots, n, \left({\Delta s_k}^{aff}\right)_i < 0\right\}\right\}$$

After that we compute a new measure of centrality: ${\mu_k}^{aff} = \frac{\left(x_k + \alpha {\Delta x_k}^{aff}\right)\left(s_k + \alpha {\Delta s_k}^{aff}\right)}{n}$ and $\sigma_k = \left(\frac{{\mu_k}^{aff}}{\mu_k}\right)^3$ because we need to compute the corrector step, which means pushing towards $x_{k+1}s_{k+1} = {\Delta \mathrm{x}_k}^{aff} {\Delta \mathrm{s}_k}^{aff}$=0 and compute the new optimal step by solving the linear system which merges the predictor and corrector steps:

$$\begin{bmatrix} 0_{\mathrm{n}\times\mathrm{m}} & A^T & I_{\mathrm{n}} \\ A & 0_{\mathrm{m}\times\mathrm{n}} & 0_{\mathrm{m}\times\mathrm{n}} \\ S & 0_{\mathrm{n}\times\mathrm{n}} & X \end{bmatrix} \begin{bmatrix} {\Delta \mathrm{x}_k}^{new} \\ {\Delta \lambda_k}^{new} \\ {\Delta \mathrm{s}_k}^{new} \end{bmatrix} = \begin{bmatrix} \mathrm{c} - \mathrm{s} - \mathrm{A}^T\lambda_k \\ b - A\mathrm{x}_k \\ -\mathrm{x}_k\mathrm{S}_k\mathrm{e} - {\Delta \mathrm{x}_k}^{aff} {\Delta \mathrm{s}_k}^{aff} e + \sigma_k\mu_k e \end{bmatrix}$$

As we have done before, we take $\widehat{\alpha_P}$ and $\widehat{\alpha_D}$ to ensure that also $x_{k+1} = x_k + \widehat{\alpha_P}\, {\Delta x_k}^{new}$ and $s_{k+1} = \mathrm{s}_k + \widehat{\alpha_D} {\Delta \mathrm{s}_k}^{new}$ are positive. In the end we take α_P and α_D as $\alpha_P = min\{1, \eta_k \widehat{\alpha_P}\}$ and $\alpha_D = min\{1, \eta_k \widehat{\alpha_D}\}$ where η_k is a costant in the interval $[0.9, 1)$.

Problem and method

Problem specifications

We must find the minimum of a linear programming problem LP where the objective function is linear $f = c^T x$, with $c, x \in \mathbb{R}^n$. The function is subject to the following constraints:

- the sum of even-numbered unknowns x_i should be 1;
- the sum of odd-numbered unknowns x_i should be 1;
- the sum of the first half of the unknowns xi should be equal to the sum of the second half.

$$c_i = log(i) \cdot \begin{cases} a & \text{if } i \text{ is odd} \\ 1 & otherwise \end{cases}$$

a: positive parameter

We want to evaluate our algorithm in terms of number of iterations needed, time spent to reach the solution, observe the convergence of the duality gap and so the centrality measure with a focus on the difference in convergence between the primal and the dual problem. We repeat our analysis for $n = 10^4, n = 10^6$ a, equal to 2, 20, 200 on a "feasible" starting point (x_0, λ_0, s_0) and on an "unfeasible" one.

Method implementation

In this section we talk about the practice implementation of the method with MATLAB with a focus on the differences between the theoretical algorithm and the practical implementation.

First of all, we construct the matrix A, which has to be full rank according to the theory, and the vector c in the required form. The first constraint "the sum of even-numbered unknowns x_i should be 1" means that $\sum_{i=1}^{n}(x_{1i}) = 1$, so in the first row of our matrix we have (0,1,0,1,…) and the same reasoning can be done for the second constraint and so we have (1,0,1,0,..). In the third row we have the first $n/2$ cells equal to one and in the remaining half we have all minus ones. The vector b has the first two components equal to one and the third one equal to zero. Here we report the expected form for A, b and c:

$$A = \begin{bmatrix} 0 & 1 & 0 & \dots & 1 & 0 & 1 \\ 1 & 0 & 1 & \dots & 0 & 1 & 0 \\ 1 & 1 & 1 & \dots & -1 & -1 & -1 \end{bmatrix} \qquad b = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \qquad c = \begin{bmatrix} 0 \\ log(2) \\ log(3) \cdot a \\ log(4) \\ log(5) \cdot a \\ \dots \end{bmatrix}$$

The matrix A is a full rank matrix with a small conditioning number.

Concerning the algorithm for the IPM, for the reproducibility of the results, we use a fixed random infeasible starting point with always the same seed, $\eta_k = 0.95$, and $\varepsilon = 10^{-6}$ for the stopping criterium. Concerning the implementation of the IPM, now we describe the main steps of the method and their implementation.

We notice that both the linear system related to the Predictor step and the one related to the Corrector step have the same general structure:

$$\begin{bmatrix} 0_{\mathrm{n\times m}} & A^T & I_{\mathrm{n}} \\ A & 0_{\mathrm{m\times n}} & 0_{\mathrm{m\times n}} \\ S & 0_{\mathrm{n\times n}} & X \end{bmatrix} \begin{bmatrix} \Delta x_k \\ \Delta \lambda_k \\ \Delta s_k \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix}$$

In the *predictor* step the vector of known terms $[r_1, r_2, r_3]$ have the following form and the direction computed is called *affine***:**

$$r_1 = c - A^T\lambda - s, \qquad \Delta x_k = {\Delta x_k}^{aff}$$

$$r_2 = Ax - b, \qquad \Delta \lambda_k = {\Delta \lambda_k}^{aff}$$

$$r_3 = XSe \qquad \Delta s_k = {\Delta s_k}^{aff}$$

Since S is a diagonal matrix it is also invertible, we can define $D := S^{-1}X$. We observe that $\Delta s = X^{-1}r_3 - D^{-1}\Delta x$ then to get $\Delta\lambda$ if we replace Δs int the first equation we have:

$$A^T\Delta\lambda - \mathrm{D}^{-1}\Delta x = r_1 - X^{-1}r_3$$

Multiplying on the left by AD and adding the result to the second equation of the linear system:

$$ADA^T\Delta\lambda = ADr_1 - AS^{-1}r_3 + r_2$$

Since A is full rank and D is a diagonal matrix, the matrix ADA^T is invertible so we can write:

$$\Delta\lambda = (ADA^T)^{-1}(ADr_1 - AS^{-1}r_3 + r_2)$$

From the first equation of the linear system:

$$\Delta s = r_1 - A^T\Delta\lambda$$

In the end, from the third equation of the linear system:

$$\Delta x = S^{-1}(r_3 - X\Delta s)$$

In the *correction* step, the left-hand side of the system to compute the new direction has the same form of the predictor, while instead the known terms vector is different:

$$r_1 = c - A^T\lambda - s, \qquad r_2 = Ax - b, \qquad r_3 = XSe - \Delta X_k{}^{aff}\Delta S_k{}^{aff}e + \sigma_k\mu_k e$$

This algorithm is written in our function *ipm*_lp. We can execute it starting from two different initial points (x_0, λ_0, s_0), a random one that just satisfies the constraints $x_0, s_0 > 0$ and a "feasible" one which also satisfies quiet precisely the equations $Ax = b,\ c - s - A^T\lambda = 0$. This method implemented in the function *lp_pdfeasible.m* is obtained by implementing the following heuristic algorithm:

We compute the vectors $(\tilde{x}, \tilde{\lambda}, \tilde{s})$ which satisfy the two previous equations with a small norm:

$$\tilde{x} = A^T(AA^T)^{-1}b \qquad \tilde{\lambda} = (AA^T)^{-1}Ac \qquad \tilde{s} = c - A^T\tilde{\lambda}$$

Then, we add two quantities $\tilde{\partial}_x\ \tilde{\partial}_s$ to obtain the positivity defined as:

$$\tilde{\partial}_x = max\left\{0, -\frac{3}{2}\min(\tilde{x})\right\},\ \tilde{\partial}_s = max\left\{0, -\frac{3}{2}\min(\tilde{s})\right\}$$

Compute:

$$\hat{x} = \tilde{x} + \tilde{\partial}_x e \qquad\qquad \hat{s} = \tilde{s} + \tilde{\partial}_s e$$

In the last step we find (x_0, λ_0, s_0) such that:

$$x_0 = \hat{x} + \hat{\partial}_x e, \qquad \lambda_0 = \tilde{\lambda}, \qquad s_0 = \hat{s} + \hat{\partial}_s e \text{ where } \hat{\partial}_x = \frac{1}{2}\frac{\hat{x}^T\hat{s}}{e^T\hat{s}} \text{ and } \hat{\partial}_s = \frac{1}{2}\frac{\hat{x}^T\hat{s}}{e^T\hat{x}}$$

Results

Expected outcome

We expect to see better performances when we compute our algorithm on a feasible initial point with respect to the case of an infeasible one. The choice of the starting point greatly influences the number of iterations computed, but on the other hand we know that this is not mandatory. Indeed, infeasible methods that start with an infeasible point are often used. So, we will see if in our implementation the feasible starting point is an useful choice or not.

Concerning the number of unknowns n , it should affect mostly the execution time because the size of the two linear systems (predictor and corrector) increases. While regarding a, we are not able to do previsions in terms of performances, but we know that greater values of a increase the components of the coefficient vector c, thus, the optimal value of the objective function $c^T x$ will be greater as well, but nothing more can be said.

Obtained results

In this section we report and comment the results obtained in the several test cases. Here we report a table with the value of the objective function at the convergence and a figure that shows the trend of centrality measure μ in all the test cases.

n	a	f_k primal
10^4	2	9.90
	20	22.38
	200	147.15
10^6	2	14.53
	20	27.07
	200	151.93

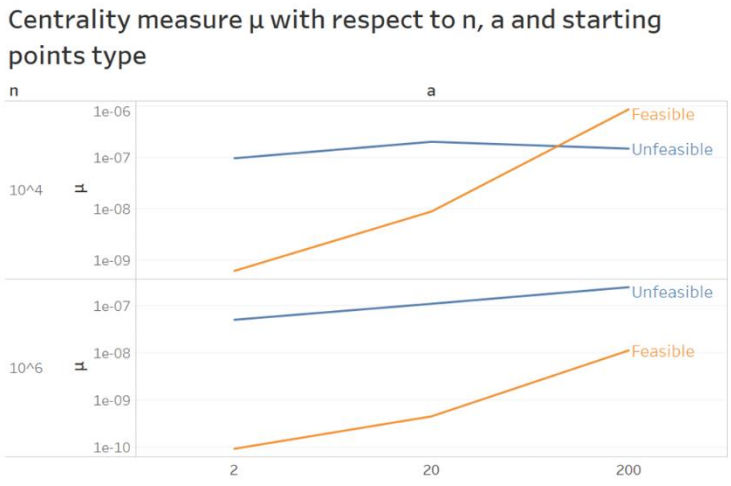

Figure 1

The table shows the value of the objective function f_k starting from an infeasible point. We ensure that the values of f_k obtained with a feasible starting point are very close to the ones reported in the table, indeed the difference is not visible in the first two or three decimal digits. Moreover, we can observe that with greater values of a, f_k increases, as we as we expected.

The gap in the value of the objective function in the case of feasible and infeasible starting point can be easily observed by looking at the centrality parameter μ. The centrality parameter is defined as the ratio of the duality gap over the number of unknowns, and we recall that we want to have the smallest possible value of μ. As we can see, in this graph, we have better performances starting from a feasible point rather than a random one, indeed in the (feasible case) μ_F is always smaller than μ_U (unfeasible case) for both $n = 10^4$ and $n = 10^6$. This means that the solution found with the feasible starting point is more accurate i.e., the dual solution and the primal solution are closer. There is only an exception with $a = 200$ where μ_F is greater than μ_U. Probably, in this case the function that computes the "feasible" starting point fails and so the arbitrary point turns out to be a better choice. Furthermore, the increase of a brings to a lower performance in almost all the test cases, specially in the case of a feasible starting point. Another unexpected result that should be underlined is that, in the case of $n = 10^6$, the precision of our algorithm is greater than the case of $n = 10^4$ both for feasible starting point and infeasible one.

Let us discuss now about the number of iteration and the time of execution.

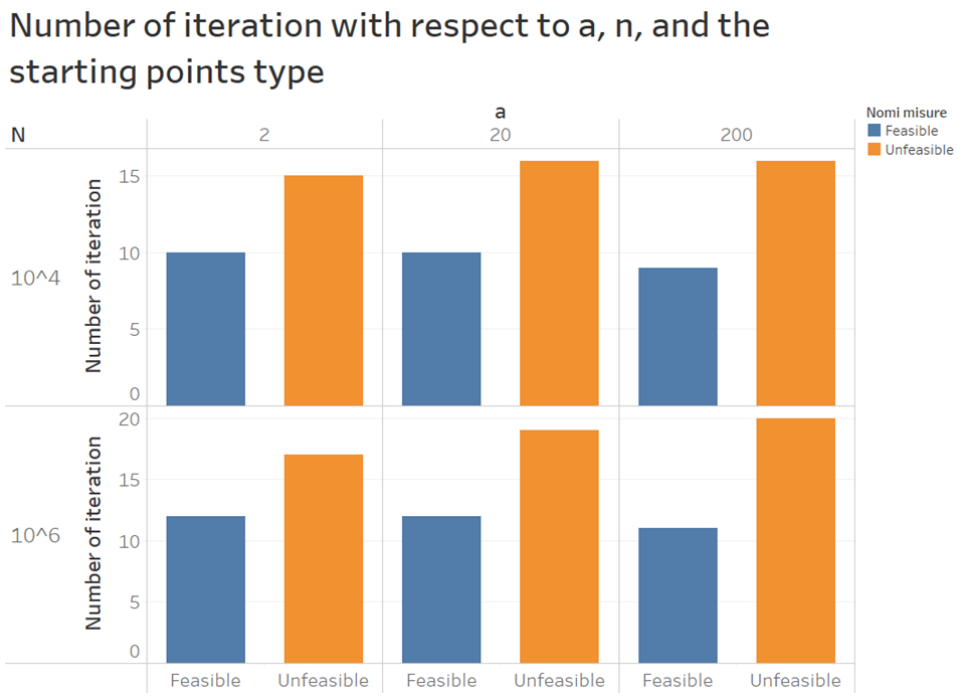

Figure 2

As we expected, in terms of number of iterations, the performances of our algorithm are better if we start from the feasible point instead of the random one. Indeed, both for $n = 10^4$ and $n = 10^6$ the number of iterations needed to reach the solution remains close to 10 even if we change the value of a, and it has a small decrease as long as we increase a. On the other hand, if we look the number of iterations needed to converge to the solution starting from the random point, we can see that it is higher as long as we increase the value a. Moreover, even if we increase the dimension of our problem, we are still able to solve it in a reasonable number of iterations. With $n = 10^6$ we compute just few iterations more than the ones needed for $n = 10^4$.

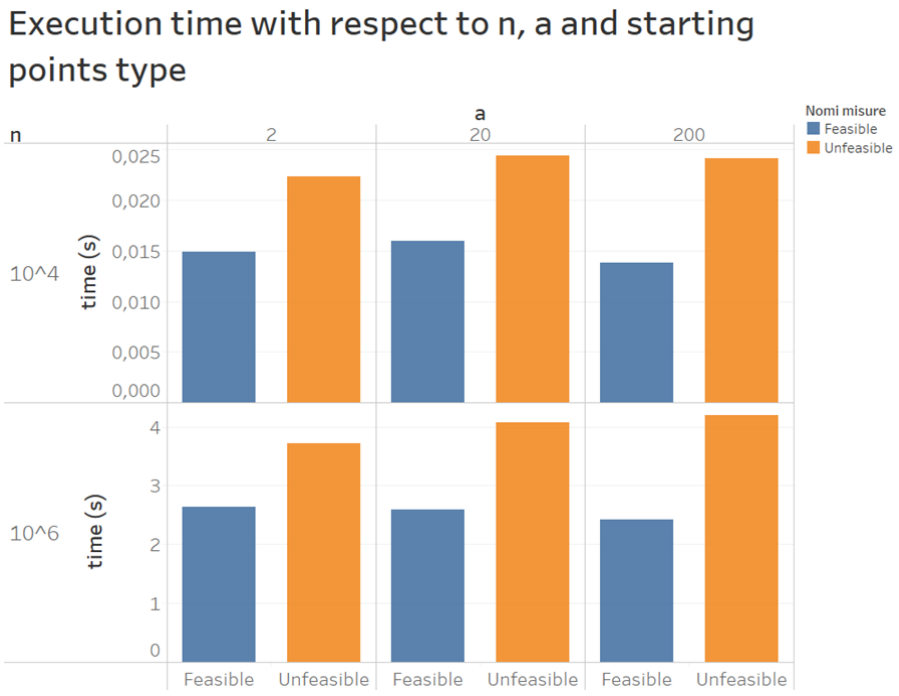

Figure 3

The bar chart of the time of execution is very similar to the one of the number of iterations and this shows how closely the two measures are related with respect to the type of starting point and a. In contrast, we can observe that in the *Figure 2* the number of iterations does not seem affected by the problem's size. Instead, here we have a dramatic rise in terms of time cost, which probably depends on the greater computational cost needed to solve the linear systems whose matrices are bigger. More in details, we can see that our algorithm behaves better starting from the feasible point, in fact, with n fixed the time spent is always around $0.015\ s$ for $n = 10^4$ and $2.5\ s$ for $n = 10^6$ for the feasible point. Instead, for the unfeasible one, it increases as long as a gets larger until it almost doubles the time needed in the feasible case. Indeed for $a = 200$, $T.U. = 0.024\ s$ against $T.F. = 0.014\ s$ with $n = 10^4$ and $T.U. = 4.1$ against $T.F. = 2.3\ s$.

In conclusion we show a picture of the trend of the objective functions primal and dual in the case of a = 20, n = 10^4 for both feasible and infeasible starting point.

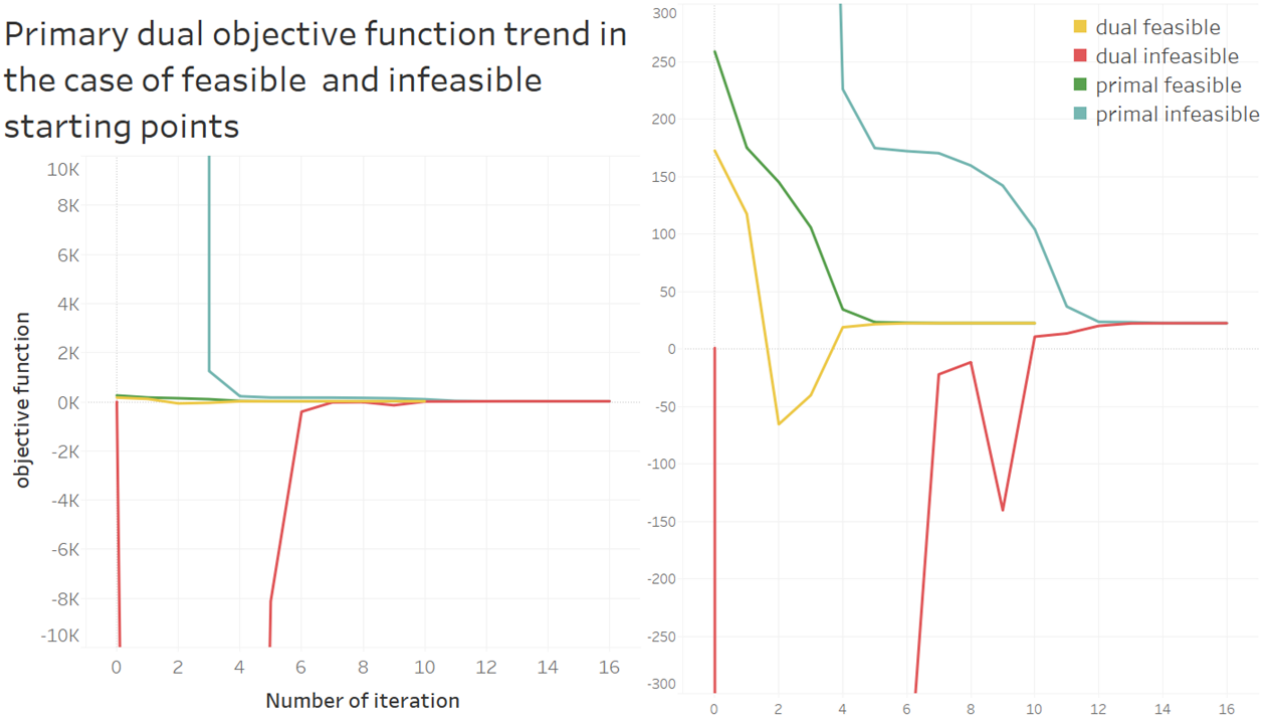

Figure 4: the trend of the objective functions for a = 20 and n = 10^4, on the right is shown a zoomed version.

Concerning the method with infeasible starting point, we can see how the primal objective function ($c^T x$) in the first iteration is very large and this is justified by the large values of the vector c. Instead, regarding the dual, we know that the vector b has small values, and since the random choice for the starting point generates vectors with values between zero and one, the objective function of the dual ($b^T \lambda$) starts, as expected, with a value close to zero. In the first iteration the value of $q(\lambda, s)$ drops down until -100 000, (*Figure 5)*, but then from the second iteration it starts to be maximized until the convergence and in the ninth one has an anomalous minimum that we have not been able to justify.

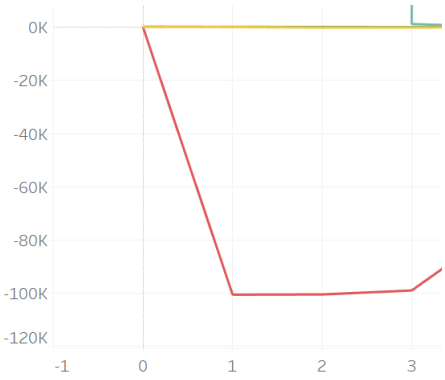

Figure 5

Regarding the results of the method with feasible starting point, we can clearly see how this choice helps the performances. The primal objective function has from the beginning a value much smaller than the preceding case and this is probably also the reason why the dual does not have here the initial negative peak. In this case the dual starts from a value bigger than the final value and so the curve first goes down minimizing $q(\lambda, s)$ and from the second iteration it grows until reaching the convergence by maximization. Moreover, the smoother start ensures to reach the convergence much earlier than with infeasible starting points.

We can clearly see how in the latest iterations, from all the four curves, the convergence is reached in the dual problem by maximization and instead in the primal by minimization. Moreover, all of them converge to the

same solution (22.38, as shown in the previous table), after 10 iterations in the feasible case and after 16 iterations in the infeasible one.

Code

Main:

```
%% Test of the IMP method on given A,b and c
clear
clc
format long

% IMP paramenters loading
load test_iplp.mat

%
% eta =
% eps =
% kmax =

% Setting of n and a and generation of A, b and c
%n = [10^4, 10^6]
%a = [2,20,200]
n = ;
a = ;
[A,b,c] = Abc_setup(n,a);
%% IMP infeasible

% set of x0, lambda0, s0 to random values
% we set the random generator seed for repeatability
rng(5);
x0 = rand(n,1);
rng(10);
lambda0 = rand(3,1);
rng(20);
s0 = rand(n,1);

disp('**** IMP infeasible: OPTIONS *****')
disp(['eta: ', eta])
disp(['eps: ', eps])

disp('**** IMP : START *****')
tic
[xk, lambdak, sk, fk, muk, k_inf, mu_inf, x_inf, lambda_inf, s_inf] = ...
ipm_lp(A, b, c, eta, eps, kmax, x0, lambda0, s0);
t = toc;
disp('**** IMP : FINISHED *****')
disp('**** IMP : RESULTS *****')
disp('************************************')
disp(['muk: ', num2str(muk), ';'])
disp(['fk: ', num2str(fk), ';'])
disp(['N. of Iterations: ', num2str(k_inf),'/',num2str(kmax), ';'])
disp(['time of execution: ', num2str(t), ';'])

disp('************************************')
```

```
%% IMP feasible

% set of x0, lambda0, s0 to feasible points using the function
% lp_pdfeasible()
[x0, lambda0, s0] = lp_pdfeasible(A, b, c);

% disp('*** new x0, s0 and lambda0')
% disp(['x0: ', vec2str(x0), ';'])
% disp(['s0: ', vec2str(s0), ';'])
% disp(['lambda0: ', num2str(lambda0), ';'])

disp('**** IMP feasible: OPTIONS *****')
disp(['eta: ', eta])
disp(['eps: ', eps])

disp('**** IMP : START *****')
tic
[xk, lambdak, sk, fk, muk, k_f, mu_f, x_f, lambda_f, s_f] = ...
ipm_lp(A, b, c, eta, eps, kmax, x0, lambda0, s0);
t = toc;
disp('**** IMP : FINISHED *****')
disp('**** IMP : RESULTS *****')
disp('************************************')
% disp(['xk: ', vec2str(xk), ';'])
% disp(['sk: ', vec2str(sk), ';'])
% disp(['lambdak: ', vec2str(lambdak), ';'])
disp(['muk: ', num2str(muk), ';'])
disp(['fk: ', num2str(fk), ';'])
disp(['N. of Iterations: ', num2str(k_f),'/',num2str(kmax), ';'])
disp(['time of execution: ', num2str(t), ';'])

disp('************************************')

%% Computation of the series of value of dual function and primal
function

pri_inf = zeros(1,k_inf);
dual_inf = zeros(1,k_inf);
pri_f = zeros(1,k_inf);
dual_f = zeros(1,k_inf);

for i = 1:k_inf
    pri_inf(i) = c'*x_inf(:,i);
    dual_inf(i) = b'*lambda_inf(:,i);
end

for i = 1:k_f
    pri_f(i) = c'*x_f(:,i);
    dual_f(i) = b'*lambda_f(:,i);
end

M = zeros(k_inf,4);
M(:,1) = pri_inf';
M(:,2) = dual_inf';
M(:,3) = pri_f';
M(:,4) = dual_f';

writematrix(M,'M_tab.csv','Delimiter',',')
type 'M_tab.csv'
```

Function Implp:

```
function [xk, lambdak, sk, fk, muk, k, mu_series, x_seq,
lambda_seq, s_seq] = ...
    ipm_lp(A, b, c, eta, eps, kmax, x0, lambda0, s0)
%
% Function that performs the Interior Point Method for a LP
%
% INPUTS:
% c = vector of the objective function of the LP;
% A = matrix of the equality constraints in the LP;
% b = vector of the equality constraints in the LP;
% eta = parameter for the correction of the sequence.
% kmax = maximum number of iterations;
% eps = parameter for the stopping criterion;
% x0 = starting element of the xk sequence;
% lambda0 = starting element of the lambdak sequence;
% s0 = starting element of the sk sequence;
%
% OUTPUTS:
% xk = the last x computed by the function;
% lambdak = the last lambda computed by the function;
% sk = the last s computed by the function;
% fxk = objective function computed in xk;
% muk = the mu computed w.r.t. yk and lambdak;
% k = last iteration.
% mu_series = vector with the value of mu at each iteration
% x_seq = matrix with in the columns the value of x at each iteration
% lambda_seq = matrix with in the columns the value of lambda at each
iteration
% s_seq = matrix with in the columns the value of s at each iteration
%


% INITIALIZATION - PREPARING THE ITERATIONS
n = length(x0);
mu0 = (x0' * s0) / n;

mu_series = zeros(1, kmax);
x_seq = zeros(n, kmax);
lambda_seq = zeros(3, kmax);
s_seq = zeros(n, kmax);


xk = x0;
lambdak = lambda0;
sk = s0;

muk = mu0;
k = 0;

% START OF The WHILE CYCLE
while k < kmax && muk > eps * mu0

    % INITIALIZE -F(xk, lambdak, sk)
```

```
    r1k = -A' * lambdak - sk + c;
    r2k = b - A * xk;
    r3k = - xk .* sk;

    % PREDICTION: COMPUTE (dxk_aff, dlambdak_aff, dsk_aff), i.e. solve
the
    % Newton step represented by the lin. syst.
    % JF(xk, lambdak, sk) * (dx, dlambda, ds)' = -F(xk, lambdak, sk)
    AD = A .* (xk ./ sk)';
    ADAt = AD * A';
    dlambdak_aff = ADAt\(AD * r1k - A * (r3k ./ sk) + r2k);
    dsk_aff = r1k - A' * dlambdak_aff;
    dxk_aff = (r3k - dsk_aff .* xk) ./ sk;

    % COMPUTE aP_aff, aD_aff, muk_aff, sigmak
    iidxk_aff = (dxk_aff < 0);
    iidsk_aff = (dsk_aff < 0);

    aP_aff = min([1, ...
        min(-xk(iidxk_aff)./dxk_aff(iidxk_aff))]);
    aD_aff = min([1, ...
        min(-sk(iidsk_aff)./dsk_aff(iidsk_aff))]);

    muk_aff = ((xk + aP_aff * dxk_aff)' * (sk + aD_aff * dsk_aff)) / n;
    sigmak = (muk_aff/muk)^3;

    % CORRECTION: COMPUTE (dxk, dlambdak, dsk)
    % REMEMBER: modify r3k
    r3k = r3k - dxk_aff .* dsk_aff + sigmak * muk;
    % Newton step represented by the lin. syst.
    % JF(xk, lambdak, sk) * (dx, dlambda, ds)' = -F(xk, lambdak,
sk)+Corr.
    dlambdak = ADAt\(AD * r1k - A * (r3k ./ sk) + r2k);
    dsk = r1k - A' * dlambdak;
    dxk = (r3k - dsk .* xk) ./ sk;

    % COMPUTE aP and aD
    iidxk = (dxk < 0);
    iidsk = (dsk < 0);

    aP_hat = min([1, ...
        min(-xk(iidxk)./dxk(iidxk))]);
    aD_hat = min([1, ...
        min(-sk(iidsk)./dsk(iidsk))]);

    aP = min([1, eta * aP_hat]);
    aD = min([1, eta * aD_hat]);

    % UPDATE
    xk = xk + aP * dxk;
    lambdak = lambdak + aD * dlambdak;
    sk = sk + aD * dsk;

    muk = (xk' * sk) / n;
    k = k + 1;

    mu_series(k) = muk;
    x_seq(:,k) = xk;
    lambda_seq(:,k) = lambdak;
```

```
    s_seq(:,k) = sk;
end

fk = c' * xk;
mu_series = mu_series(1:k);
x_seq = x_seq(:,1:k);
lambda_seq = lambda_seq(:,1:k);
s_seq = s_seq(:,1:k);


end
```

Function Abc_setup:

```
%% Function that generates the matrix A
%This function creates a matrix A having: the first row with ones in the
%even positions and zeros in the odd positions; the second row with ones
%in the odd positions and zeros in the even positions; the third row with
%ones in the first half and minus 1 in the second half;
%It returns also the vector b as a all one column vector with size (3,1)
%and the vector c

function [A, b, c] = Abc_setup(n, a)

% generation of A
    A = ones(3,n);

    A(1,1:2:n) = 0;

    A(2,2:2:n) = 0;

    A(3,n/2+1:1:n) = -1;

%   generation of b
    b = [1;1;0];

%   generation of c
%generation of c
    c = zeros(n,1);
    for i = 1:n
        l = log(i);
        if(mod(i,2) == 0)
            c(i) = l*a;
        else
            c(i) = l;
        end
    end
end
```

Function lp_pdfeasible: For the computation of the feasible point

```
function [x0, lambda0, s0] = lp_pdfeasible(A, b, c)

% "TILDE"-STEP) (very high) does not allow us to solve the linear system
```

```
y_tilde = (A * A')\(b);
x_tilde = A' * y_tilde;

lambda_tilde = (A * A')\(A * c); % (A)
s_tilde = c - A' * lambda_tilde;

dx_tilde = max([0, ...
    -1.5 * min(x_tilde)]);

ds_tilde = max([0, ...
    -1.5 * min(s_tilde)]);

% "HAT"-STEP
x_hat = x_tilde + dx_tilde;
s_hat = s_tilde + ds_tilde;

dx_hat = 0.5 * (x_hat' * s_hat) / sum(s_hat);
ds_hat = 0.5 * (x_hat' * s_hat) / sum(x_hat);

% FINAL STEP
x0 = x_hat + dx_hat;
s0 = s_hat + ds_hat;
lambda0 = lambda_tilde;

end
```