Numerical optimization for large scale problems, constrained optimization

Assignment 1: Interior Point Method applied to linear programming problems

Giammarinaro Silvia, Gullotto Marco

July 14, 2020

1 Introduction

Let's start with the definition of an Interior point method: an Interior point method is an algorithm that is used in solving both linear and nonlinear convex optimization problems that contain inequalities as constraints. The aim of this research is to implement the algorithm for a linear programming problem which it'll be discussed in the next section, the programming language used for this assignment is MATLAB.

We consider the linear programming problem in standard form:

$$min\ f(x)$$
$$s.t.\ h(x) = 0,\ g(x) \leq 0$$

where $f(x) : \mathbb{R}^n \to \mathbb{R},\ h(x) : \mathbb{R}^n \to \mathbb{R}^m,\ g(x) : \mathbb{R}^n \to \mathbb{R}^r$

In order to solve the problem we start computing its optimality conditions. Optimality conditions can be derived from the lagrangian function theory. By introducing the lagrangian function for the equality constraint $h_i(x)$

$$\mathcal{L}(x, \lambda) = f(x) + \lambda_i h_i(x)$$

and noting that $\nabla_x \mathcal{L}(x, \lambda) = \nabla f(x) + \lambda \nabla h(x)$, we observe that at the solution x^*, there exists a scalar λ_i^* such that

$$\nabla_x \mathcal{L}(x^*, \lambda_i^*) = 0.$$

This observation suggests that we can search for solutions of the equality-constrained problem by looking for stationary points of the Lagrangian function. The scalar λ_i is called a Lagrange multiplier vector for the constraint $h_i(x)$.

Considering all the constraints, the Lagrangian function is the following:

$$\mathcal{L}(x, \lambda, \mu) = f(x) + \sum_{i=1}^{n} \lambda_i h_i(x) + \sum_{j=1}^{r} \mu_j g_j(x)$$

The necessary conditions defined below are called first-order conditions because they are concerned with properties of the gradients (first-derivative vectors) of the objective and constraint functions. These conditions are the foundation for many algorithms concerning numerical optimization, also Interior point method.

Before defining the solution of the system, we introduce the concept of the active set. The active set is the set of indices where the inequality constraints are active.

$$\mathcal{A}(x) = \{j \in 1, \ldots, r : g_j(x) = 0\}$$

Suppose that x^* is a local solution of the general optimization problem, that the functions $f(x),\ h(x),\ g(x)$ are continuously differentiable. Then there exist λ^*, μ^*, such that:

$$\begin{gathered}
\nabla_x \mathcal{L}(x^*, \lambda^*, \mu^*) = 0 \\
\nabla_\lambda \mathcal{L}(x^*, \lambda^*, \mu^*) = 0 \\
\mu_j \geq 0,\ \forall j \in 1, \ldots, r \\
\mu_j = 0,\ \forall j \notin \mathcal{A}(x^*)
\end{gathered}$$

These are called first-order conditions or Karush–Kuhn–Tucker (KKT) conditions. Convexity of the problem ensures that these conditions are sufficient for a global minimum.

2 Problem analysis

From now on, we show how we evaluate the assignment. In the next paragraphs we report all calculations in order to solve the problem followed by the MATLAB implementation.
We consider the following function:

$$min\ c^T x\ s.t.\ \sum_{i=1}^{n} x_i = 1,\ x_i \geq 0\ \forall i$$
$$with\ x, c\ \in \mathbb{R}^n,$$
$$c_i = \begin{cases} a & \text{if i is odd} \\ 1 & \text{otherwise} \end{cases} \tag{1}$$

with a being a positive parameter. We have to solve the problem with n = 10^4, n= 10^6 and a = 2, 20, 200, 2000.
In order to handle inequality constraints we introduce a logarithmic barrier function $\mathcal{B}$, its value tends to infinity as the point approaches the boundary of the feasible region. In this case the barrier function is the following

$$\mathcal{B} = -\mu \sum_{i=1}^{n} ln(x_i)$$

where μ is a positive parameter.
Now we can rewrite the problem

$$min\ c^T x\ \ - \mu \sum_{i=1}^{n} ln(x_i)\ s.t.\ \sum_{i=1}^{n} x_i = 1$$

We define the Lagrangian function

$$\mathcal{L}(x, \lambda, \mu) = c^T x\ \ - \mu \sum_{i=1}^{n} ln(x_i)\ \ - \lambda(rx - 1)$$
$$with\ r = [1, \ldots, 1]$$

In order to solve the problem, we have to compute the optimality conditions, which are the following:

$$\begin{cases} \nabla_x \mathcal{L}(x, \lambda, \mu) = 0 \\ \nabla_\lambda \mathcal{L}(x, \lambda, \mu) = 0 \end{cases} \Rightarrow \begin{cases} c - \mu X^{-1} e - r^T \lambda = 0 \\ 1 - rx = 0 \end{cases}$$
$$with\ e = [1, \ldots, 1]^T$$

We introduce a slack variable s

$$s = \mu X^{-1} e, \;\; s = Se$$
$$S = \begin{bmatrix} s_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & s_n \end{bmatrix}$$

We rewrite the optimality conditions

$$\begin{cases} c - s - r^T \lambda = 0 \\ 1 - rx = 0 \\ XSe = \mu e \end{cases}$$

Here is the code for the main script

```
%variables
n1 = 10^4;
n2 = 10^6;
a1 = 2;
a2 = 20;
a3 = 200;
a4 = 2000;
kmax = 20;
eps1 = 1e-3;
eps2 = 1e-6;
eps3 = 1e-9;

%vectors c generated with different a values
c_n1_a1 = ones(n1,1);
c_n1_a2 = ones(n1,1);
c_n1_a3 = ones(n1,1);
c_n1_a4 = ones(n1,1);
for i = 1:n1
    if rem(i,2) ≠ 0    %i is odd
        c_n1_a1(i) = a1;
        c_n1_a2(i) = a2;
        c_n1_a3(i) = a3;
        c_n1_a4(i) = a4;
    end
end

c_n2_a1 = ones(n2,1);
c_n2_a2 = ones(n2,1);
c_n2_a3 = ones(n2,1);
c_n2_a4 = ones(n2,1);
```

```
for i = 1:n2
    if rem(i,2) ≠ 0    %i is odd
        c_n2_a1(i) = a1;
        c_n2_a2(i) = a2;
        c_n2_a3(i) = a3;
        c_n2_a4(i) = a4;
    end
end

tic
[x_star, lambda_star, s_star, fx_star, k] = ...
    interior_point_method(c_n1_a1, eps1, kmax, n1);
toc
```

2.1 Starting point

The starting point (x_0, λ_0, s_0) has to be choosen wisely in order to satisfy the optimality conditions, picking a random starting point leads often to failure of convergence. We define a feasible set called primal-dual strictly feasible set:

$$\mathcal{F} = \{(x, \lambda, s) \in \mathbb{R}^{2n+m} \mid rx = 1,\ c\ - s - r^T\lambda = 0, x > 0, s > 0\}$$

In order to obtain a starting point that satisfies the equality constraints, we follow the framework shown below:

Step 1: Compute

$$\widetilde{x} = r^T(rr^T)^{-1},\ \widetilde{\lambda} = (rr^T)^{-1}rc,\ \widetilde{s} = c - r^T\widetilde{\lambda}$$

In general, $\widetilde{x}$ and $\widetilde{s}$ will have negative components, so they are not suitable for use as a starting point.

Step 2: We define

$$\delta_x = max\left(\frac{3}{2}\ \underset{i}{min}\ \widetilde{x}_i, 0\right)$$
$$\delta_s = max\left(\frac{3}{2}\ \underset{i}{min}\ \widetilde{s}_i, 0\right)$$

and adjust the x and s vectors as follows:

$$\widehat{x} = \widetilde{x} + \delta_x e,\ \widehat{s} = \widetilde{s} + \delta_s e$$

Note that both $\widehat{x}$ and $\widehat{s}$ are greater than zero, but to ensure they are not too close we introduce another two scalars:

Step 3:

$$\widehat{\delta}_x = \frac{\widehat{x}^T \widehat{s}}{2e^t \widehat{s}}, \ \widehat{\delta}_s = \frac{\widehat{x}^T \widehat{s}}{2e^t \widehat{x}}$$

Step 4:

$$x_0 = \widehat{x} + \widehat{\delta}_x e, \ \lambda_0 = \widetilde{\lambda}, \ s_0 = \widehat{s} + \widehat{\delta}_s e$$

Note that the computation cost of the starting point is the same as one step of the primal-dual method. The MATLAB implementation is the following

```
function [xk, lambdak, sk, fxk, k ] = interior_point_method(c, ...
    epsilon, kmax, n)

e = ones(n,1);
r = e';
%step 1
x_hat = r'*inv(r*r')*1;
lambda_hat = inv(r*r')*r*c;
s_hat = c - r'*lambda_hat;
%step 2
Δ_x = max(-1.5*min(x_hat), 0);
Δ_s = max(-1.5*min(s_hat), 0);
x_hat = x_hat + Δ_x*e;
s_hat = s_hat + Δ_s*e;
%step 3
Δ_hat_x = 0.5*(x_hat'*s_hat)/(e'*s_hat);
Δ_hat_s = 0.5*(x_hat'*s_hat)/(e'*x_hat);
%step 4
x0 = x_hat + Δ_hat_x*e;
lambda0 = lambda_hat;
s0 = s_hat + Δ_hat_s*e;
```

2.2 Interior point method implementation

We implemented two different different version of the algorithm, the second one is optimized in order to solve the problem with $n = 10^6$.

Standard version

Step 1: We take ϵ, $k_{max} = 20$ and the starting point (x_0, λ_0, s_0) as shown in the previuos paragraph.

Step 2: For $k \geq 0$

a. Compute the affine scaling step $(\Delta x_k^{aff}, \Delta\lambda_k^{aff}, \Delta s_k^{aff})$

$$\begin{pmatrix} r & 0 & 0 \\ 0 & r^T & I \\ S_k & 0 & X_k \end{pmatrix} \begin{pmatrix} \Delta x_k^{aff} \\ \Delta\lambda_k^{aff} \\ \Delta s_k^{aff} \end{pmatrix} = \begin{pmatrix} 1 - r x_k \\ c - s_k - r^T\lambda_k \\ -X_k S_k e \end{pmatrix}$$

$$with\ X_k = \begin{bmatrix} x_{k,1} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & x_{k,n} \end{bmatrix}, S_k = \begin{bmatrix} s_{k,1} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & s_{k,n} \end{bmatrix}$$

b. Compute

$$\alpha_{aff}^{primal} = min\left(1, \min_{1\leq i\leq n|(\Delta x_k^{aff})_i<0} -\frac{(x_k)_i}{(\Delta x_k^{aff})_i}\right)$$
$$\alpha_{aff}^{dual} = min\left(1, \min_{1\leq i\leq n|(\Delta s_k^{aff})_i<0} -\frac{(s_k)_i}{(\Delta s_k^{aff})_i}\right)$$

c. Compute

$$\mu_k^{aff} = \frac{(x_k + \alpha_{aff}^{primal}\Delta x_k^{aff})^T(s_k + \alpha_{aff}^{dual}\Delta s_k^{aff})}{n},\ \sigma_k = \left(\frac{\mu_k^{aff}}{\mu_k}\right)^3$$

d. Compute the corrector step $(\Delta x_k, \Delta\lambda_k, \Delta s_k)$

$$\begin{pmatrix} r & 0 & 0 \\ 0 & r^T & I \\ S_k & 0 & X_k \end{pmatrix} \begin{pmatrix} \Delta x_k \\ \Delta\lambda_k \\ \Delta s_k \end{pmatrix} = \begin{pmatrix} 1 - r x_k \\ c - s_k - r^T\lambda_k \\ -X_k S_k e - \Delta X_k^{aff}\Delta S_k^{aff} e + \sigma_k\mu_k e \end{pmatrix}$$

e. Compute

$$\alpha_{k,max}^{primal} = min\left(1, \min_{1\le i\le n|(\Delta x_k)_i<0} -\frac{(x_k)_i}{(\Delta x_k)_i}\right)$$
$$\alpha_{k,max}^{dual} = min\left(1, \min_{1\le i\le n|(\Delta s_k)_i<0} -\frac{(s_k)_i}{(\Delta s_k)_i}\right)$$
$$\eta_k = max(1-\mu_k, 0.9)$$
$$\alpha_k^{primal} = min(1, \eta_k \alpha_{k,max}^{primal})$$
$$\alpha_k^{dual} = min(1, \eta_k \alpha_{k,max}^{dual})$$

f. Update

$$x_{k+1} = x_k + \alpha_k^{primal}\Delta x_k,\ \lambda_{k+1} = \lambda_k + \alpha_k^{dual}\Delta\lambda_k,\ s_{k+1} = s_k + \alpha_k^{dual}\Delta s_k$$

g. Update

$$\mu_{k+1} = \frac{x_{k+1}^T s_{k+1}}{n}$$
$$if\ \mu_{k+1} \le \epsilon\mu_0,\ STOP$$

The MATLAB implementation is shown below

```
% variables
xk = x0;
lambdak = lambda0;
sk = s0;
k = 0;
muk = (xk' * sk) / n;
mu0 = muk;
% matrices
I = speye(n);
F = spalloc(2*n+1, 2*n+1, 5*n);
F(1, 1:n)=r;
F(2:n+1, n+1) = r';
F(2:n+1, n+2:2*n+1) = I;
whos F
whos full(F)

while k<kmax
    %2.a
    Xk = spdiags(xk, 0, n, n);
    Sk = spdiags(sk, 0, n, n);
    F(n+2:2*n+1, 1:n) = Sk;
```

```
F(n+2:2*n+1, n+2:2*n+1) = Xk;
spy(F)
r1 = -r*xk + 1;
r2 = c - r'*lambdak - sk;
r3 = - Xk*Sk*e;
BB_aff = [r1; r2; r3];
XX_aff = F\BB_aff;
Δ_xk_aff = XX_aff(1:n);
Δ_sk_aff = XX_aff(n+2:2*n+1);
%2.b
z1 = -xk./Δ_xk_aff;
alpha_aff_p = min(1, min(z1(Δ_xk_aff<0)));
if isempty(alpha_aff_p)
    alpha_aff_p = 1;
end
z2 = -sk./Δ_sk_aff;
alpha_aff_d = min(1, min(z2(Δ_sk_aff<0)));
if isempty(alpha_aff_d)
   alpha_aff_d = 1;
end
%2.c
muk_aff= (xk + ...
    alpha_aff_p*Δ_xk_aff)'*(sk+alpha_aff_d*Δ_sk_aff)/n;
sigmak = (muk_aff/muk)^3;
%2.d
Δ_Xk_aff = spdiags(Δ_xk_aff, 0, n,n);
Δ_Sk_aff = spdiags(Δ_sk_aff,0,n,n);
BB = [r1; r2; r3 - Δ_Xk_aff*Δ_Sk_aff*e + sigmak*muk*e];
XX = F\BB;
%2.e
Δ_xk= XX(1:n);
Δ_lambdak= XX(n+1);
Δ_sk= XX(n+2:2*n+1);
z1 = -xk./Δ_xk;
alphakmax_p =  min(z1(Δ_xk<0));
z2 = -sk./Δ_sk;
alphakmax_d = min(z2(Δ_sk<0));
etak = max(1-muk, 0.9);
alphak_p = min(1, etak*alphakmax_p);
if isempty(alphak_p)
    alphak_p = 1;
end
alphak_d = min(1, etak*alphakmax_d);
if isempty(alphak_d)
    alphak_d = 1;
end
%2.f
xk = xk + alphak_p*Δ_xk;
```

```
        lambdak = lambdak + alphak_d*Δ_lambdak;
        sk = sk + alphak_d*Δ_sk;
        %2.g
        k = k+1;
        muk = xk'*sk / n;
        if  muk<mu0*epsilon
            fxk = c'*xk;
            return
        end
    end
    end
```

Optimized version

What differs from the standard version is the linear system used to compute the steps, the first one can be rewritten in the following way:

$$\begin{pmatrix} -D^{-2} & r' \\ r & 0 \end{pmatrix} \begin{pmatrix} \Delta x_k \\ \Delta \lambda_k \end{pmatrix} = \begin{pmatrix} -r_c + X_k^{-1} r_x s \\ -r_b \end{pmatrix}$$

$$\Delta s_k = -X_k^{-1} - X_k^{-1} S \Delta x_k$$

$$with\ r_c = -c + s_k + r^T \lambda_k,\ r_b = r x_k - 1,\ r_x s = X_k S_k e$$

$$D = S_k^{-1/2} X_k^{1/2}$$

This system is know as augmente system. We can rewrite the system as follows

$$r D^2 r' \Delta \lambda_k = -r_b - r X_k S_k^{-1} rc + A S_k^{-1} r_{xs}$$

$$\Delta s_k = -r_c - r' \Delta \lambda$$

$$\Delta x_k = S_k^{-1} r_x s - X_k S_k^{-1} \Delta s_k$$

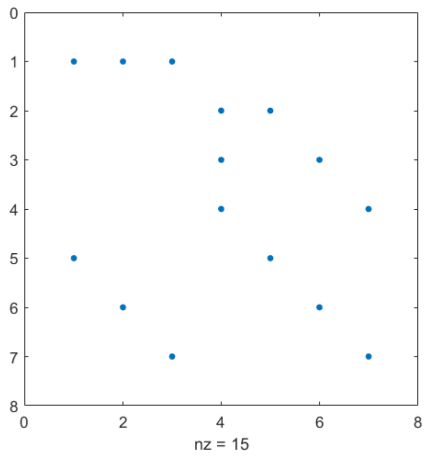
The MATLAB coe is the following

```
    Xk = spdiags(xk, 0, n, n);
    Sk = spdiags(sk, 0, n, n);
    rb = −1+r*xk;
    rc = − c + r'*lambdak + sk;
    rxs =  Xk*Sk*e;
    Dk = spdiags(xk./sk, 0, n, n);
    invs = spdiags(1./sk, 0, n,n);
    Δ_lambda_aff = (r*Dk*r')\(−rb − r*Xk*invs*rc + r*invs*rxs);
    Δ_sk_aff = − rc − r'*Δ_lambda_aff;
    Δ_xk_aff = −invs*rxs − Xk*invs*Δ_sk_aff;
```

We report here some observations and tips:

- The matrix F is used in steps 2.a and 2.d, so we decided to allocate it at the begging with the command `spalloc(2*n+1, 2*n+1, 5*n))`, which is used to store in memory a sparse matrix with dimensions $(2n+1, 2n+1)$ and $5n$ nonzeros elements, which are the ones stored in the disk. To see how many disk space is required, we can run the command `whos F` which return the numbers of bytes needed to store the matrix F. With $n = 10^4$, the sparse matrix F requires 0,96 MB of disk storage and `full(F)`, which is the full version of the matrix F, requires 3,20 GB. This is a huge improvement and we have to take it into account when solving large scale problems. Note that also the command `spdiags` is used for X_k and S_k in order to create a sparse diagonal matrix.

- To visualize which elements of the matrix F are not equal to zero we can use the command `spy(F)` with n=3 and we obtain the picture reported below. Note that we have 15 nonzeros elements and 10 zeros elements, in respect to our declaration using the command `spalloc`.



- We always verify that vectors are not empty in order to avoid the following behaviour

```
min(1, [])
  ans = []
```

3 Results and conclusion

We start analysing the results obtained with n=2 and a=2000 to have a approximated graphic representation of the problem.
The function is $f(x) = 2000x_1 + x_2$ with the constraint $x_1 + x_2 = 1$ and $\mu = 0$. As we can see the solution is $x^* = (0, 1)$ with $f(x^*) = 1$. We report below the plots and the points computed by our implementation of the algorithm which lead to the solution x^* as expected. In this case the number of iterations required is $k = 8$ and the computing time is 0.05 seconds.

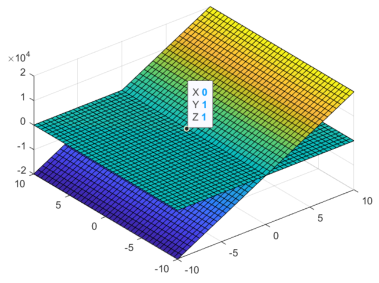


Figure 1: Problem representation

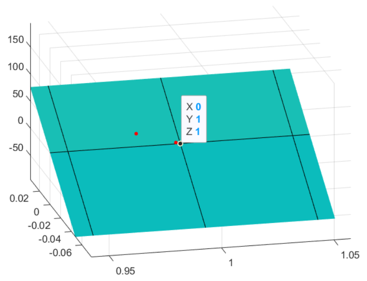
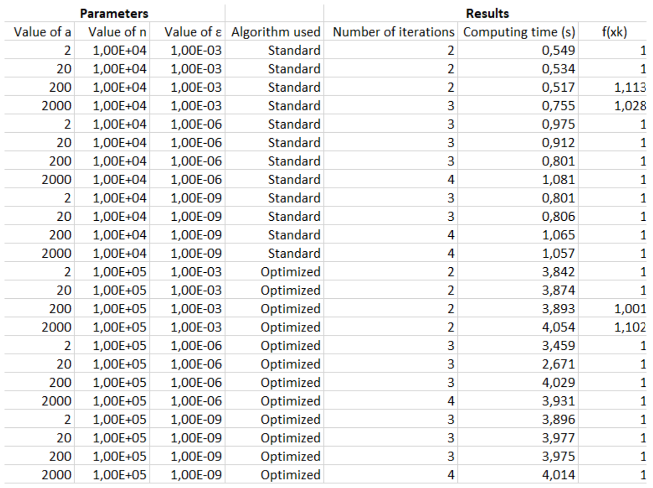


Figure 2: Path towards the solution

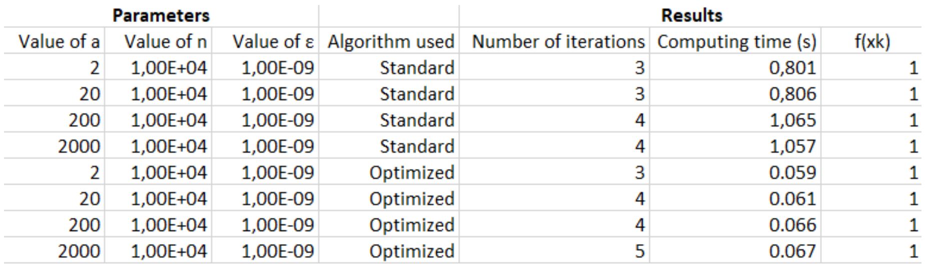
We report below the results obtained using different parameters, we set $k_{max} = 20$.



Parameters				Results		
Value of a	Value of n	Value of ε	Algorithm used	Number of iterations	Computing time (s)	f(xk)
2	1,00E+04	1,00E-03	Standard	2	0,549	1
20	1,00E+04	1,00E-03	Standard	2	0,534	1
200	1,00E+04	1,00E-03	Standard	2	0,517	1,113
2000	1,00E+04	1,00E-03	Standard	3	0,755	1,028
2	1,00E+04	1,00E-06	Standard	3	0,975	1
20	1,00E+04	1,00E-06	Standard	3	0,912	1
200	1,00E+04	1,00E-06	Standard	3	0,801	1
2000	1,00E+04	1,00E-06	Standard	4	1,081	1
2	1,00E+04	1,00E-09	Standard	3	0,801	1
20	1,00E+04	1,00E-09	Standard	3	0,806	1
200	1,00E+04	1,00E-09	Standard	4	1,065	1
2000	1,00E+04	1,00E-09	Standard	4	1,057	1
2	1,00E+05	1,00E-03	Optimized	2	3,842	1
20	1,00E+05	1,00E-03	Optimized	2	3,874	1
200	1,00E+05	1,00E-03	Optimized	2	3,893	1,001
2000	1,00E+05	1,00E-03	Optimized	2	4,054	1,102
2	1,00E+05	1,00E-06	Optimized	3	3,459	1
20	1,00E+05	1,00E-06	Optimized	3	2,671	1
200	1,00E+05	1,00E-06	Optimized	3	4,029	1
2000	1,00E+05	1,00E-06	Optimized	4	3,931	1
2	1,00E+05	1,00E-09	Optimized	3	3,896	1
20	1,00E+05	1,00E-09	Optimized	3	3,977	1
200	1,00E+05	1,00E-09	Optimized	3	3,975	1
2000	1,00E+05	1,00E-09	Optimized	4	4,014	1

Figure 3: Results obtained

Note that we have used the optimized version with $n = 10^5$ because the standard version doesn't lead to a solution in reasonable time, so the optimized version seems to be faster. Let's compare the computing time of the two versions with the same parameters n and ϵ:



	Parameters				**Results**	
Value of a	Value of n	Value of ε	Algorithm used	Number of iterations	Computing time (s)	f(xk)
2	1,00E+04	1,00E-09	Standard	3	0,801	1
20	1,00E+04	1,00E-09	Standard	3	0,806	1
200	1,00E+04	1,00E-09	Standard	4	1,065	1
2000	1,00E+04	1,00E-09	Standard	4	1,057	1
2	1,00E+04	1,00E-09	Optimized	3	0.059	1
20	1,00E+04	1,00E-09	Optimized	4	0.061	1
200	1,00E+04	1,00E-09	Optimized	4	0.066	1
2000	1,00E+04	1,00E-09	Optimized	5	0.067	1

Figure 4: Comparing the standard and the optimized implementation

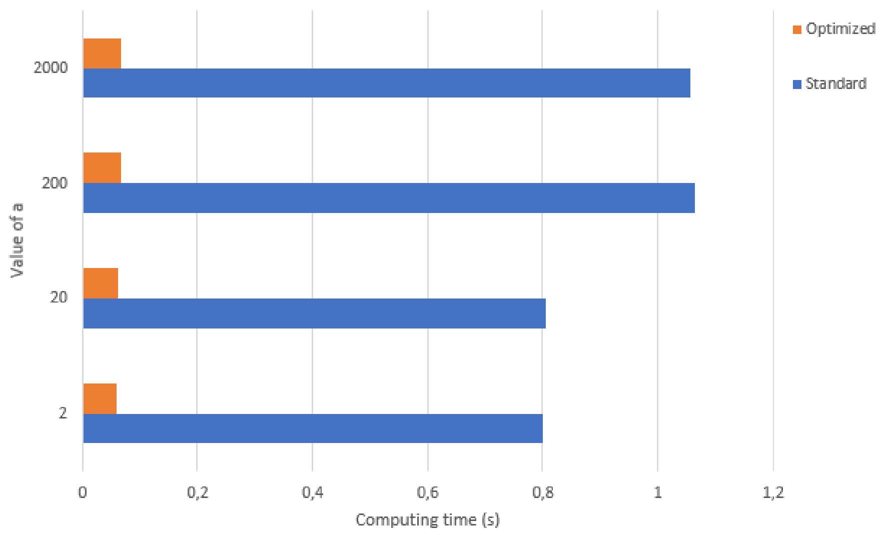


Figure 5: Comparing computing time

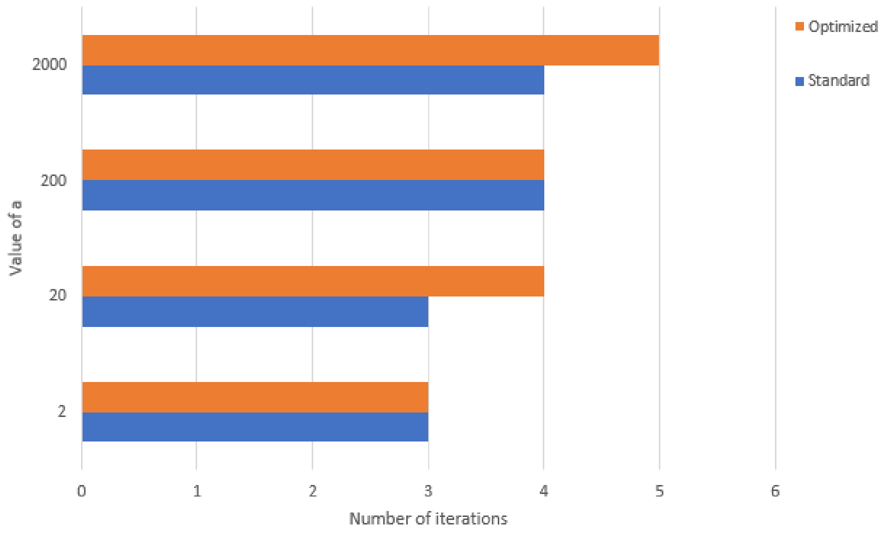


Figure 6: Comparing number of iterations

We can confirm that the optimized version is faster than the standard one.

In the end we note that the algorithm converges with a small number of iterations, so we didn't have to compare different values of k. Changing the value of the parameter a slightly modifies the number of iteration required and the computing time. The solution of the system is $f_{xk} = 1$ the correct one and the constraint $\sum_{i=1}^{n} x_i = 1$ is always verified.