# Interior Points Method Implementation for Solving the Optimal Revenue Problem of Hydro Plant Generator in Brazilian Market

Guilherme Machado, Felipe Nazaré

***Abstract*—This paper proposes a model of profit maximization under uncertainty for a hydro power plant agent in the Brazilian Market. It is also presents an implementation of Interior Points Method for solving the presented problem and a benchmark of its implementation with the Simplex method as well as with the commercial solver Clp.**

***Index Terms*—Interior Points, MRE, Contract, Optimization, Simplex, Benchmark.**

## I. Introduction

MOST recently, the Brazilian electricity sector has been dealing with a scenario of low reservoirs level and, consequently, high risk for hydro power generators, who are often facing situations where there is not enough water in their reservoirs to comply with their contractual energy obligations. To try to mitigate the generator's risk, in this paper it is proposed a model of decision making under uncertainty. In this model, it is maximized the generator revenue under the consideration of a risk measure for their contracts aiming at lowering the generator risks while maximizing its revenue.

Additionally it is proposed an implementation of Interior Points Method, to solve the hydro generator problem, with validation test results and a stress test, considering several scenarios for the hydro plant generator problem.

The section II briefly introduces the Brazilian Electricity Market, in order to present in section III the hydro plant generator problem and to propose a linear optimization model to solve it. The section IV discuss the implementation of the Interior Points Method and in section V it is detailed a case study with a stress test for the generator problem and validation tests for the Interior Points Method implementation.

July 4, 2018

## II. Brazilian Electricity Market

The Brazilian Electricity Market is divided by three types of market:

- Retail Market (ACL)
- Single Buyer Market (ACR)
- Short Term Market (MCP)

Each one of the markets has its own characteristics, but as the goal of this work is the application of the algorithm of interior points in solving a contract problem in the Brazilian market, we will only briefly describe them, for the better comprehension of the application that will be shown in the next section.

The Retail Market consumers can freely negotiate between buyers and sellers through a non-organized market with the only condition that all contracts must be registered at the Market Operator and all of them must be backed by a physical guarantee.

In the Single Buyer Market, the consumers do not have the right to choose their supplier, instead they are imposed a a price defined by national public auctions.

The Short Term Market is designed to settle the differences between the contracted energy amounts and the generated/-consumed energy. This market is operated by the Brazilian Market Operator (CCEE), who does the accounting of energy generated by each power plant and its contractual obligations.

In the following sections, it will be briefly explained the electricity market concepts that are necessary for the entire comprehension of the presented problem.

### A. Types Of Contracts

The Single Buyer Market has 2 types of contract offered:

- Energy Quantity
- Energy Availability

Energy Quantity contracts are usually used by hydro plants agents where the risk is allocated to the generator. The contract specifies the maximum energy the plant must be able to provide for the system operator.

In Energy Availability contracts, the agent put their plant available capacity at the distributor disposal and receives a constant revenue for been available to dispatch at any time. When the plant is dispatched, the owner also receives a payment for its variable cost.

### B. Physical Guarantee

The Physical Guarantee is the amount of energy that each power plant is allowed to trade. Its value is determined by the Ministry of Mines and Energy and it represents a physical coverage for the sale of energy, which is mandatory for every energy contract in Brazil.

It is calculated individually, by the amount of energy it can produce almost continuously in a several years simulation with thousands of inflows scenarios.

Guilherme Machado, Felipe Nazaré, are with the Department of Electrical Engineering, PUC-Rio, Rio de Janeiro, RJ.

### C. Energy Relocation Mechanism

The Energy Relocation Mechanism (MRE) is a structural hedging mechanism for hydro power plants, managed by CCEE. In this mechanism, all hydro generation is treated as a pool and each hydro plant has a share of the total hydro generation based on its physical guarantee.

This mechanism is necessary to mitigate the some of the exogenous risks associated to the hydro plant operation. With the MRE, hydro plants located at areas affected by drought due to season conditions are less affected.

## III. Contract Dilemma

Nowadays with the overall low hydro reservoirs, the MRE has not been successful in mitigating the risk of the hydro plants generators, as they are constantly been failing to meet its contractual obligations.

Therefore hydro plants are been constantly exposed to a high level of risk in their commercial operations.

Considering that the goal of all contracted agent is to minimize its risk, in this work it is proposed an optimization model to solve the problem of optimal use of the two mechanisms regulated by the market that can be used to lower the agents exposure. These mechanisms are:

- Reduction of contracted capacity
- Buy new option of buy contracts in the Short Term Market

These two options should be optimized for a certain risk measure in order to change the agent current risk to its desired target.

### A. Optimization Problem

In this work we propose a methodology to optimize the agent profit securing the risk exposure to a desired target.

The objective function will be the maximization of the profit, under a chosen risk measure, subject to the constraints of the market regulation.

The agent must comply with the physical guarantee norm, which specifies that he can not sell more energy contracts than he has of physical guarantee plus the amount of physical guarantee he has in his contracts.

$$Q_{cont} \leq GF + \sum_{i=1}^{I} a_{op_{i,t}} \cdot Qop_{i,t} \forall t \tag{1}$$

$$0 \leq a_{op_{i,t}} \leq 0.005 \ \forall t \ \forall i \tag{2}$$

$$0 \leq \sum_{i=1}^{I} a_{op_{i,t}} \leq 0.1 \tag{3}$$

Where $t$ indicates the stage, $i$ indicates the scenario, $a_{op_{i,t}}$ is the percentage of the availability contract bought. $GF$ is his physical guarantee, $Q_{cont}$ is the amount of energy he is allowed to sell in his contract and $Qop_{i,t}$ is the amount of energy in every option contract he has.

Another option for the agent is to reduce its contractual obligation. This way it can be negotiated a reduction of at most 5% of his contract.

$$Q_{cont} = Q_{original} - Q_{reduced} \tag{4}$$

$$0 \leq Q_{reduced} \leq 0.05 \cdot Q_{original} \tag{5}$$

The generator total owned energy after the computation of the contracts negotiation is:

$$G_{tot_t} = GSF_t \cdot GF + \sum_{i=1}^{I} a_{op_{i,t}} \cdot G_{op_{i,t}} \forall t \tag{6}$$

The generator energy liquidation done by MRE can be calculated after the generator exposure is changed:

$$\begin{aligned} Profit = P_{cont} \cdot Q_{cont} + (G_{tot_t} - Q_{cont}) \cdot PLD_t \\ - \sum_{i=1}^{I} a_{op,i,t} \cdot G_{op,i,t} - \sum_{i=1}^{I} CVU_i \cdot G_{op_{i,t}} \end{aligned} \tag{7}$$

In the above equation, $Q_{cont}$ is the contract quantity and $P_{cont}$ is the contract price. $PLD$ is the liquidation price, which is defined by the Market Operator.

If we consider a risk neutral agent, the risk measure function is the expected value and the whole optimization problem model is the following:

$$\begin{aligned} \max_{a_{op}, Q_{reduced}, G_{tot_t}} \quad & \mathbb{E}\{P_{cont} \cdot Q_{cont} + (G_{tot_t} - Q_{cont}) \cdot PLD_t \\ & - \sum_{i=1}^{I} P_{op_i}^{fix} \cdot a_{op_{i,t}} \cdot Q_{op_{i,t}} - \sum_{i=1}^{I} CVU_i \cdot a_{op_{i,t}} \cdot G_{op_{i,t}}\} \\ \text{s.t.} \quad & G_{tot_t} = GSF_t \cdot GF + \sum_{i=1}^{I} a_{op_{i,t}} \cdot G_{op_{i,t}} \forall t, \forall i, \\ & Q_{cont} \leq GF + \sum_{i=1}^{I} a_{op,i,t} \cdot Q_{op_{i,t}} \quad \forall t, \\ & 0 \leq a_{op_{i,t}} \leq 0.1 \quad \forall t \forall i, \\ & 0 \leq \sum_{i=1}^{I} a_{op_{i,t}} \leq 0.15 \quad \forall t \forall i, \\ & Q_{cont} = Q_{original} - Q_{reduced}, \\ & 0 \leq Q_{reduced} \leq 0.05 \cdot Q_{original}, \\ & a_{op}, Q_{reduced}, G_{tot_t} \geq 0 \end{aligned} \tag{8}$$

## IV. Interior Points Method

As described in [1], Interior Points method offers a different approach than the Simplex method for solving linear optimization problems. While the Simplex explores the geometry of the feasible set, going through the vertex searching for the optimal one, the Interior Points method explores a insight on duality to solve numerically a set of non-linear equations. In a geometrical interpretation, the Interior Points method starts from one initial point and goes straight to the solution, instead of walking at the borders like the Simplex.

The first insight needed for the Interior Points method to work is that the original problem must be reformulated, relaxing all constraints. In order to achieve this, one must use Lagrangian relaxation for the the feasibility constraints and Barrier Methods for the variables positive constraints. Barrier Method is a relaxation technique for the constraint $x \geq 0$, which uses the logarithmic function $-\mu \sum_{i=1}^{n} ln\ x_i$. Therefore, the problem in the standard form stated as a minimization can be modeled as an unconstrained one.

$$\begin{aligned} \min_{x} \quad & c^T x \\ \text{s.a.} \quad & Ax = b \\ & x \geq 0 \\ & \equiv \end{aligned} \tag{9}$$

$$\min_{x} \quad c^T x - \mu \sum_{i=1}^{n} ln\ x_i - \lambda^T (Ax - b) \tag{10}$$

As presented in [2], one can define a set of conditions to ensure optimality of the linear programming solution. These conditions are called KKT conditions, or *Karush-Kuhn-Tucker* conditions. They are stated as:

1) $A^T\lambda + s = c$
2) $Ax = b$
3) $x \geq 0$
4) $s \geq 0$
5) $x_i s_i = 0,\ 1 \leq i \leq n$

These conditions can be resumed in the above problem as:

1) $A^T\lambda + \mu X^{-1} e = c$
2) $Ax = b$

Setting $s = \mu X^{-1} e$, the original KKT conditions can be recovered:

$$A^T\lambda + s = c \tag{11}$$

$$Ax = b \tag{12}$$

$$X^{-1} e = \mu e \tag{13}$$

Observe that these conditions for the optimality of a solution to this new barrier subproblem is identical to the conditions for the original problem, except for the complementary condition 5), which is relaxed by the barrier constant $\mu$. This implies that by choosing $\mu = 0$, the optimal solution of the original problem is recovered.

By considering these set of equations, we can convert our original problem with linear constraints into a problem which consists of solving a set of nonlinear equations. The method of solution choosed is the Newton's method for finding roots of nonlinear equations and it is presented next.

### A. Newton Method

The Newton Method is a iterative method for solving non-linear equations which uses a first order multivariable Taylor series expansion. Consider the function $F(z)$ one wishes to find $z^*$ such that $F(z^*) = 0$. From an initial guess $z^0$, one can improve the solution using the first order Taylor expansion:

$$F(z^{k+1}) \approx F(z^k) + J(z^k)d \tag{14}$$

Where $J$ is the Jacobian of $F$ and $d$ is the step, also called *Newton direction*.

The interior points KKT conditions can be formulated so as one can apply the Newton Method:

$$F(z) = \begin{bmatrix} Ax - b \\ A^T\lambda + s - c \\ X^{-1}e - \mu e \end{bmatrix}$$

The Jacobian of $F$ is:

$$J(z) = \begin{bmatrix} A & 0 & 0 \\ 0 & A^T & 0 \\ S & 0 & X_k \end{bmatrix}$$

Where $X = diagm(x_1, \ldots, x_n)$, $S = diagm(s_1, \ldots, s_n)$. Therefore the system of nonlinear equations is resumed to:

$$F(z^{k+1}) \approx F(z^k) + J(z^k)d \tag{15}$$

$$F(x^{k+1}) = \begin{bmatrix} Ax^k - b \\ A^T\lambda + s^k - c \\ {X_k}^{-1}e - \mu e \end{bmatrix}$$

$$+ \begin{bmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S_k & 0 & X_k \end{bmatrix} \begin{bmatrix} b_x^k \\ b_p^k \\ b_s^k \end{bmatrix}$$

But, as we want the roots:

$$0 = F(z^k) + J(z^k)d$$
$$F(z^k) = -J(z^k)d$$

$$\begin{bmatrix} Ax^k - b \\ A^T\lambda + s^k - c \\ {X_k}^{-1}e - \mu e \end{bmatrix} = \begin{bmatrix} Ab_x^k \\ A^T b_x^k + b_p^k \\ S_k b_x^k + X_k b_s^k \end{bmatrix}$$

But, since $Ax^k = b$ and $A^T\lambda + s^k = c$, the equations resume to:

$$\begin{bmatrix} 0 \\ 0 \\ {X_k}^{-1}e - \mu e \end{bmatrix} = \begin{bmatrix} Ab_x^k \\ A^T b_x^k + b_p^k \\ S_k b_x^k + X_k b_s^k \end{bmatrix} \tag{16}$$

### B. Step Length

After obtaining the *Newton direction*, we should find the real step lengths for the vector $[x,\ p,\ s]^T$. It is important that they are close to **d** but not greater, so that the nonnegativity requirements for $x$ and $s$ are not violated. It is stated in [1], that one possible step would be:

$$\beta_P = min\{1,\ \alpha \min_{i|(d_x^k)_i < 0} \{-\frac{x_i^k}{(d_x^k)_i}\}\} \tag{17}$$

$$\beta_D = min\{1,\ \alpha \min_{i|(d_x^k)_i < 0} \{-\frac{x_i^k}{(d_x^k)_i}\}\} \tag{18}$$

Where $\alpha \in [0, 1]$. By applying the step lengths, the new vector of solution is:

$$x^{k+1} = x^k + \beta_P^k d_x^k \tag{19}$$

$$p^{k+1} = p^k + \beta_D^k d_p^k \tag{20}$$

$$s^{k+1} = s^k + \beta_D^k d_s^k \tag{21}$$

### C. Barrier Parameter

Note that the *Newton direction* depends on the barrier parameter $\mu^k$. In order to get the result $[x^k,\ p^k,\ s^k]^T$ with great precision, one must iterate several times with $\mu$ kept fixed. However since it is only necessary that the algorithm gets the right direction, it suffices if only one iteration with the same $\mu$ is done. This at each iteration the $\mu$ is changed. Following [1], one way of updating $\mu$ is:

$$\mu^{k+1} = \rho\ \frac{{x^k}^T s^k}{n}$$

Where $\rho \in [0, 1]$ is typically set to 1, but can be lower if the algorithm has not progressed.

### D. Algorithm

The procedure of the Interior Points algorithm described above can be summarized into the following pseudo-code:

**Algorithm 1** Interior Point Method

1: Start with feasible $x^0 > 0, s^0 > 0, p^0$
2: **while** $x^T s > \epsilon$ **do**
3: Set $\mu^{k+1} = \rho\ \frac{{x^k}^T s^k}{n}$
4: Solve the system $d = -J(z^k)^{-1}F(z^k)$
5: Find the steps

$$\beta_P = min\{1,\ \alpha \underset{i|(d_x^k)_i<0}{min}\{-\frac{x_i^k}{(d_x^k)_i}\}\}$$

$$\beta_D = min\{1,\ \alpha \underset{i|(d_x^k)_i<0}{min}\{-\frac{x_i^k}{(d_x^k)_i}\}\}$$

6: Update solution

$$x^{k+1} = x^k + \beta_P^k d_x^k$$
$$p^{k+1} = p^k + \beta_D^k d_p^k$$
$$s^{k+1} = s^k + \beta_D^k d_s^k$$

The implementation for this algorithm is presented at the appendix A.

## V. Case Study

In this section some test results for the Interior Points are presented, as well as one stress test for problem proposed in section II. For the stress test, a benchmark will be made comparing the Interior Points algorithm performance to that of a commercial solver *Clp* and a Simplex implementation.

### A. Interior Points Algorithm Unitary Tests

This subsection presents 4 unitary tests, which proves the implemented method works for unbounded, unfeasible and regular cases.

*1) Regular Case:* The regular case test presents a simple production problem modeled as follows:

$$\begin{aligned} \max_{x_1,x_2} \quad & 4x_1 + 3x_2 \\ \text{s.a.} \quad & 2x_1 + x_2 \leq 4 \\ & x_1 + 2x_2 \leq 4 \\ & x_1, x_2 \geq 0 \end{aligned} \tag{22}$$

The result of the Interior Points implementation was:

| | |
|---|---|
| x1 | 1.33334 |
| x2 | 1.33326 |
| s1 | 5.48651e-5 |
| s2 | 1.371791e-4 |
| z | 9.3331 |
| iterations | 13 |

TABLE I
Results for regular problem case

*2) Regular Case Extended:* The regular case extended is similar to the regular case production problem, but with one more constraint added:

$$\begin{aligned} \max_{x_1,x_2} \quad & 4x_1 + 3x_2 \\ \text{s.a.} \quad & 2x_1 + x_2 \leq 4 \\ & x_1 + 2x_2 \leq 4 \\ & x_1 + x_2 \geq 1 \\ & x_1, x_2 \geq 0 \end{aligned} \tag{23}$$

The results for this problem were:

| | |
|---|---|
| x1 | 1.33334 |
| x2 | 1.33327 |
| s1 | 4.81635e-5 |
| s2 | 1.20431e-4 |
| s3 | 1.66661 |
| z | 9.3331 |
| iterations | 13 |

TABLE II
Results for regular problem extended case

*3) Unbounded Case:* The unbounded problem tested was formulated as follows:

$$\begin{aligned} \max_{x_1,x_2} \quad & x_1 + x_2 \\ \text{s.a.} \quad & 0.5x_1 - x_2 \leq 0.5 \\ & -4x_1 + x_2 \leq 1 \\ & x_1, x_2 \geq 0 \end{aligned} \tag{24}$$

The algorithm did 6 iterations after which it indicated that the result was unbounded. At that point the partial results were:

| | |
|---|---|
| x1 | 3.24144e10 |
| x2 | 1.29658e11 |
| s1 | 1.1345e11 |
| s2 | 1497.18 |
| z | 1.620719544426749e11 |
| iterations | 6 |

TABLE III
Results for unbounded problem case

*4) Unfeasible Case:* The unfeasible problem tested was formulated as follows:

$$\begin{aligned} \max_{x_1,x_2} \quad & x_1 + x_2 \\ \text{s.a.} \quad & 1x_1 + 3x_2 \leq 8 \\ & 3x_1 + 2x_2 \leq 12 \\ & -x_1 - 2x_2 \leq -13 \\ & x_1, x_2 \geq 0 \end{aligned} \tag{25}$$

The algorithm did 5 iterations after which it stoped. At that point the partial results were:

| | |
|---|---|
| x1 | 1.18589 |
| x2 | 1.55707 |
| s1 | 1.98543e-5 |
| s2 | 1.04244 |
| s3 | 1.97908e-5 |
| z | 2.742960689180599 |
| iterations | 5 |

TABLE IV
RESULTS FOR UNFEASIBLE PROBLEM CASE

### B. Stress Test

com o teste de carga com a tabela de instancias comparando os metodos programados (pontos interiores e simplex) e o solver comercial em termos de tempo e fobj @@@

## VI. Conclusion

As showed by the stress test, the Interior Points algorithm is much more robust to the size of the problem than the Simplex algorithm. This could be explained by the Simplex approach, which is combinatorial and has the step limited to one adjacent vertex, while the Interior Points goes straight to the solution. Although Its worth to notice that as the Interior Points solve a nonlinear set of equations, therefore it can be slower than the Simplex when its initial solution is close to the optimal one, or in cases where the problem is small. @@@ adicionar algo sobre o problema

## Appendix A
## Interior Points Method Implementation

The Interior Points Method implementation was made at Julia Language.

```
#
# Interior Points
#

# Guilherme Pereira Freire Machado

function interior_points(A::Array{Float64}, b::
    Array{Float64}, c::Array{Float64}, debug=true)

    open_log_i(A, b, c)
    stream = get_log_i()

    % initial guess
    m, n = size(A)

    x0 = ones(n)
    p0 = zeros(m)
    s0 = ones(n)

    x = ones(n)
    p = zeros(m)
    s = ones(n)

    x, s, p, status, it = interior_algorithm(A, b,
         c, x0, s0, p0, stream, debug)

    return x, p, s, status, it
end

function interior_bigM(A, b, c, debug)
    stream = get_log_i()

    m, n = size(A)

    U = maximum(c)
    M = U*1e5

    % add bigM variable if necessary
    if any((b - A*ones(n)) .!= 0)
        c_1 = [c ; M]
        A_1 = [A (b - A*ones(n))]
        b_1 = b
        % initial feasible solution
        x0 = ones(n+1)
    else
        c_1 = c
        A_1 = A
        b_1 = b
        % initial feasible solution
        x0 = ones(n)
    end
    p = (c'[1:(m)]*A[:,1:(m)])' % ones(m)%
    s0 = (c_1' - p'A_1)'

    A=A_1
    c=c_1
    x=x0
    s=s0
    pwrite(stream, "Usando big M para comecar no
        ponto viavel x = e", debug)
    pwrite(stream, "Problema:", debug)
    pwrite(stream, "A = $A_1", debug)
    pwrite(stream, "b = $b_1", debug)
    pwrite(stream, "x0 = $x0", debug)
    pwrite(stream, "s0 = $s0", debug)
    pwrite(stream, "p = $p", debug)
    pwrite(stream, "")

    x, s, p, status, it = interior_algorithm(A_1,
        b_1, c_1, x0, s0, p, stream, debug)

    return x, p, s, status, it
end

function interior_algorithm(A, b, c, x, s, p,
    stream, debug=true)
    err = 1e-3
    alpha= 0.9
    rho = 0.5
    n = size(A)[2]
    mu = 0.0
    dx = 0.0
```

```
maxit = 200
it=0
for i in 1:maxit
    pwrite(stream, "It: $i - epsilon = $(
        convergence_error(s, x, mu)) - x = $x",
         debug)

    % 2) 1st test for convergence
    if s'*x < err % i > 40 && optimality_test(s
        , x, mu, err) && maximum(abs.(dx)) <
        err
        % convergiu
        status = 1

        if norm(x) < norm(p) % check for
            unbounded problem
            status = -1
            pwrite(stream, "The problem is
                unbounded!")
        else

            pwrite(stream, "Interior Points
                algorithm converged! ( s*x
                criteria)")
        end


        % if check_unbounded(A, b, c, x)
        % status = -1
        % end

        % if check_infeasible(x)
        % status = -2
        % end

        result_log_i(i, x, (c'*x), status,
            stream)
        return x, s, p, status, it
    end

    % 3) computation of newton directions
    mu = rho * x' * s / n
    dx, ds, dp = compute_directions_old(x, s, p
        , mu, A, b, c)
    rho = update_rho(rho, x + dx, x)

    % 4) and 5) update variables
    x, s, p, unbounded = update_variables(x, s,
         p, dx, ds, dp, alpha)

    if convergence_error(s, x, mu) > 1e5 && all
        (x .> 0)
        status = -1
        result_log_i(i, x, (c'*x), status,
            stream, debug)
        return x, s, p, status, it
    end

    it += 1
end

pwrite(stream, "Maximum number of iterations(
    $maxit) exceeded!")
result_log_i(maxit, x, (c'*x), 0, stream)
    return x, s, p, 0, it
end

function optimality_test(s::Array{Float64}, x::
    Array{Float64}, mu::Float64, err::Float64)
    nx = length(x)
    op = convergence_error(s, x, mu)
    return op < err
end

function check_infeasible(x)
    if round(x[end],2) > 0
        println(x)
        return true
    end

    return false
end

function convergence_error(s::Array{Float64}, x::
    Array{Float64}, mu::Float64)
    nx = length(x)
    op = x'*s - (mu*ones(nx)')*ones(nx)
    return op
end

function compute_directions(x::Array{Float64}, s::
    Array{Float64}, p::Array{Float64}, mu::Float64
    , A::Array{Float64}, b::Array{Float64}, c::
    Array{Float64})
    % solve linear system:
    nx = length(x)
    np = length(p)
    ns = length(s)
    e = ones(nx)

    inv_s = diagm(s) \ I
    D2 = diagm(x) * inv_s
    D = D2^0.5

    parcial_1 = (A * D2 * A') \ (A * D)
    P = D * A' * parcial_1
    vmu = (diagm(x) \ D) * (mu * ones(nx) - diagm(
        x) * diagm(s) * ones(nx))


    dx = D * (I - P) * vmu
    dp = -((A * D2 * A') \ (A*D)) * vmu
    ds = (D \ P) * vmu

     return dx, ds, dp
end

function compute_directions_old(x::Array{Float64},
     s::Array{Float64}, p::Array{Float64}, mu::
    Float64, A::Array{Float64}, b::Array{Float64},
     c::Array{Float64})
    % solve linear system:
    nx = length(x)
    np = length(p)
    ns = length(s)
    e = ones(nx)

    matrix = [A zeros(size(A)[1], size(A')[2])
        zeros(size(A)[1], nx) ;
```

```
            zeros(size(A')[1], size(A)[2]) A' eye(
                size(A')[1]) ;
            diagm(s) zeros(length(x), size(A')[2])
                diagm(x)]

    d = matrix \ - [A * x - b ;
              A' * p + s - c;
              diagm(x)*diagm(s)*e - mu * e]

    dx = d[1:nx]
    dp = d[(nx + 1) : (nx + np)]
    ds = d[(nx + np + 1) : end]

     return dx, ds, dp
end

function update_variables(x::Array{Float64}, s::
    Array{Float64}, p::Array{Float64}, dx::Array{
    Float64}, ds::Array{Float64}, dp::Array{
    Float64}, alpha::Float64)
    % compute step beta for primal
    ratio_x = -x ./ dx
    ratio_x[ dx .>= 0] = Inf
    % ratio_x = [v for (i,v) in enumerate(ratio_x)
         if dx[i] < 0]
    betap = minimum([1, alpha*minimum(ratio_x)])

    % compute step beta for dual slack
    ratio_s = -s ./ ds
    ratio_s[ds .>= 0] = Inf
    % ratio_s = [v for (i,v) in enumerate(ratio_s)
         if ds[i] < 0]
    betad = minimum([1, alpha*minimum(ratio_s)])

    % update variables
    x = x + betap * dx
    s = s + betad * ds
    p = p + betad * dp

    return x, s, p, false
end

function update_rho(rho, x_new, x_old)
    if maximum(abs.(x_new - x_old)) < 1
        rho *= 0.9

    else
        rho = 0.8
    end
    return rho
end

function open_log_i(A::Array{Float64,2}, b::Array{
    Float64,1}, c::Array{Float64,1}, debug=true)
    fname = "Interior_Points.log"
    if isfile(fname)
        stream = open(fname, "a")
        pwrite(stream, "======================",
            debug)
        pwrite(stream, "Comeco da Solucao do PL",
            debug)
        pwrite(stream, "======================",
            debug)
        pwrite(stream, "Problema:", debug)
        pwrite(stream, "A = $A", debug)
        pwrite(stream, "b = $b", debug)
        pwrite(stream, "c = $c", debug)
        pwrite(stream, "", debug)
        close(stream)
    else
        stream = open(fname, "w", debug)
        pwrite(stream, "======================",
            debug)
        pwrite(stream, "Comeco da Solucao do PL",
            debug)
        pwrite(stream, "======================",
            debug)
        pwrite(stream, "Problema:", debug)
        pwrite(stream, "A = $A", debug)
        pwrite(stream, "b = $b", debug)
        pwrite(stream, "c = $c", debug)
        pwrite(stream, "", debug)
        close(stream)
    end
    nothing
end

function get_log_i()
    fname = "Interior_Points.log"
    stream = open(fname, "a")
    pwrite(stream, "Interior Points")
    pwrite(stream, "--------------")

    return stream
end

function result_log_i(it::Int, x::Array{Float64
    ,1}, z::Float64, status::Int, stream::IOStream
    , debug=true)
    pwrite(stream, "iter $it:", debug)
    pwrite(stream, "x = $x", debug)
    pwrite(stream, "", debug)

    if status == 1
        pwrite(stream, "| Solucao otima obtida:",
            debug)
        pwrite(stream, "| ---------------------",
            debug)
        pwrite(stream, "| x = $x", debug)
        pwrite(stream, "| z = $z", debug)
        pwrite(stream, "| status = $status", debug)
        pwrite(stream, "")
    elseif status == -1
        pwrite(stream, "| Solucao ilimitada obtida:
            ", debug)
        pwrite(stream, "| -------------------------
            ", debug)
        pwrite(stream, "| x = $x", debug)
        pwrite(stream, "| z = $z", debug)
        pwrite(stream, "| status = $status", debug)
        pwrite(stream, "", debug)
    elseif status == 0
        pwrite(stream, "| Solucao subotima obtida:"
            , debug)
        pwrite(stream, "| -------------------------
            ", debug)
        pwrite(stream, "| x = $x", debug)
        pwrite(stream, "| z = $z", debug)
        pwrite(stream, "| status = $status", debug)
        pwrite(stream, "")
    elseif status == -2
```

```
        pwrite(stream, "| Problema e inviavel:",
            debug)
        pwrite(stream, "| ------------------------
            ", debug)
        pwrite(stream, "| status = $status", debug)
        pwrite(stream, "", debug)
    end
end

function pwrite(stream::IOStream, string::
    AbstractString, debug=true)
    if debug
        println(string)
    end
    write(stream, string * "\n")
end
```

## ACKNOWLEDGMENT

The authors would like to thank the University PUC-Rio for the fertile environment for research and learning, as well as the teachers and teacher assistants of the electrical engineering department.

## REFERENCES

[1] D. Bertsimas and J. N. Tsitsiklis, *Introduction to linear optimization*. Athena Scientific Belmont, MA, 1997, vol. 6.

[2] R. Robere, *Interior Point Methods and Linear Programming*, 12 2012.

**Guilherme Machado** received the BA.Sc in Electrical Engineering from, Rio de Janeiro Federal University, Rio de Janeiro, Brazil, in 2017. He also graduated at Automatized Systems Engineering at a double degree program in the French ”Grand Ecole” Supélec in 2016. Currently he is pursuing a Masters degree at PUC-Rio.

**Felipe Nazaré** received the BA.Sc in Electrical Engineering from, Rio de Janeiro Federal University, Rio de Janeiro, Brazil, in 2017. Currently he is pursuing a Masters degree at PUC-Rio.