Université Paris 1 Panthéon-Sorbonne

UFR027 Mathématiques et informatique

X5I12119 Numerical Methods in Optimization

PROJECT REPORT

Descent Methods & Improvement

Student Name	Student ID
1. Ru Zhang	12018820
2. Ken Lin	12112869

Lecturer in charge:

Dr. Kengy BARTY

Submission Date : 24/12/2021

Contents

1 Introduction

In this project, we study the problem of the resolution of the equations describing the state of equilibrium of a drinking water distribution network. This problem can be transformed into the minimization problem of a objective function (representing the energy of the network) under linear constraints of equality (translating the first Kirchho's law). Once the problem is well modeled, we will apply to it the main optimization algorithms of the course, in the unconstrained framework.

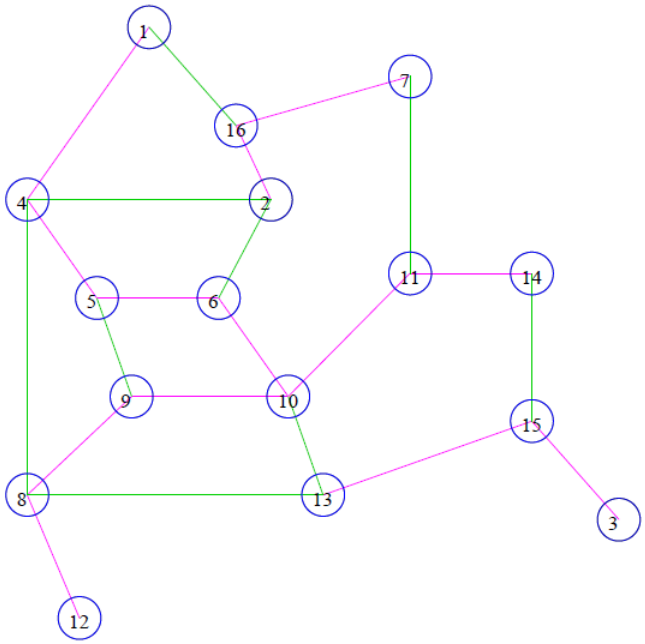

Figure 1: The drinking water distribution network

2 Problem Modeling

2.1 Problem Representation

A drinking water distribution network is in the form of an **directed graph** $\mathcal{G}$ in which the set of arcs, of cardinal n, is denoted by $\mathcal{A}$ and the set of nodes, of cardinal m, is denoted by $\mathcal{N}$ [2]:

$$\mathcal{G} = (\mathcal{A}, \mathcal{N}), \quad \text{card}(\mathcal{A}) = n, \quad \text{card}(\mathcal{N}) = m \tag{1}$$

The set $\mathcal{N}$ is partitioned into a subset $\mathcal{N}_r$, of cardinal m_r, corresponding to the nodes where the reservoirs of the network are located, and a subset $\mathcal{N}_d$, of cardinal m_d, corresponding at the nodes where the network requests (consumption) are located:

$$\mathcal{N} = \mathcal{N}_r \cup \mathcal{N}_d, \quad \text{card}\left(\mathcal{N}_r\right) = m_r, \quad \text{card}\left(\mathcal{N}_d\right) = m_d \tag{2}$$

We suppose that the arcs (resp. nodes) of the graph are numbered from 1 to n (resp. m), and that the nodes where the reservoirs are located are the first in the numbering of $\mathcal{N}$. We also suppose that the graph is connected, which means that there exists at least one path (not oriented) connecting any two nodes of the graph. This connection condition implies the relation:

$$n \geq m - 1 \tag{3}$$

The topology of the network can be described by its **node-arc incidence matrix**, which we denote by A. This matrix has as many rows (resp. Columns) as there are nods (resp. arcs) in the associated graph. Given an arc and a node i of the graph, the element $a_{i,\alpha}$ of the matrix A has the value:

$$a_{i,\alpha} = \begin{cases} -1 & \text{if } i \text{ is the initial node of the arc } \alpha, \\ +1 & \text{if } i \text{ is the final node of the arc } \alpha, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Remark1. Each column of matrix A contains exactly one "-1" and one "$+1$". The sum of the rows of the matrix is therefore identically zero, which proves (at least in the case $n \geq m$) that the matrix A is not of full rank. We can show that the matrix obtained by removing any row from matrix A is of full rank. We then deduce from equation (3) that:

$$\text{rang}(A) = m - 1 \tag{5}$$

Remark2. The i-th row of matrix A describes the connectivity of node i in the graph: the number of non-zero elements of this row is equal to the number of connected arcs at node i, and the sign of an element indicates whether the corresponding arc is entering or exiting at this node.

We then note:

f **the vector of the flows at the nodes of the graph**:

(a) The values of the pressure at the nodes of $\mathcal{N}_r$ are assumed to be known: they are equal to the heights of the water columns contained in the reservoirs (expressed in meters);

(b) The values of the pressures at the nodes of $\mathcal{N}_d$ must be calculated;

p **the vector of the pressures at the nodes of the graph**:

(a) The values of the flow at the nodes of $\mathcal{N}_d$ are assumed to be known: they are equal to the demands (expressed in cubic meters per second) of consumers, and are counted positively in the case of withdrawal and negatively in the case of injection;

(b) The values of the flow at the nodes of $\mathcal{N}_r$ must be calculated, and correspond to debits entering or leaving the reservoirs;

r **the vector of the resistances of the arcs of the graph**:

(a) The values of the resistances are assumed to be known;

(b) They depend on the diameter, length and roughness of the pipes

q **the vector of the flows in the arcs of the graph**;these values must all be calculated.

The equilibrium state of the network results in equations of two types.

1. The first set of equations expresses the fact that there can be no accumulation of water at the nodes of the graph (**Kirchho's first law**). Taking into account remark 2, this law has for matrix expression:

$$Aq - f = 0 \tag{6}$$

2. The second set of equations combines the fact that the pressure drop along an arc drifts from a potential (**Kirchho's second law**) and that this pressure drop is itself a function of the flow of the arc (**nonlinear Ohm's law**): for an arc with initial node i and node j, the pressure drop z_α, equal to the difference $p_i - p_j$ of the pressures at the ends of the arc, is a function 'whose form is, for example, given by the formula of Colebrooks:

$$\varphi_\alpha(q_\alpha) = r_\alpha q_\alpha |q_\alpha| \tag{7}$$

Noting $r \bullet q \bullet |q|$ the vector ($\in \mathbb{R}^n$) of components $r_\alpha q_\alpha |q_\alpha|$, we obtain the relations:

$$A^\top p + r \bullet q \bullet |q| = 0 \tag{8}$$

The partition of all the nodes $\mathcal{N}$ into $\mathcal{N}_r$ and $\mathcal{N}_d$ induces a partition of the matrix A and the vectors f and p into:

$$A = \begin{pmatrix} A_r \\ A_d \end{pmatrix}, f = \begin{pmatrix} f_r \\ f_d \end{pmatrix}, p = \begin{pmatrix} p_r \\ p_d \end{pmatrix} \tag{9}$$

The aim of the study is to calculate the vectors q, f_r and p_d verifying the relations (6) and (7), assuming known the matrix A and the vectors r, f_d and p_r. This is the problem of determining the point of equilibrium of a water network subject to the two laws of Kirchho and to nonlinear Ohm's law.

2.2 Optimization Problem Formalization

We can show that the equilibrium problem described at **section 2.1** is equivalent to a constrained minimization problem. Noting the primitive of the function defined in (7) [3]:

$$\Phi_\alpha(q_\alpha) = \frac{1}{3} r_\alpha q_\alpha^2 |q_\alpha| \tag{10}$$

we define the function J (which can be interpreted as the energy of the network) by:

$$\bar{J}(q, f_r) = \sum_{\alpha \in \mathcal{A}} \Phi_\alpha(q_\alpha) + \sum_{i \in \mathcal{N}_r} p_i f_i \tag{11}$$

Noting $\langle \cdot, \cdot \rangle$ the usual scalar product (in $\mathbb{R}^n$ or $\mathbb{R}^{m_r}$), this function takes the compact form:

$$\bar{J}(q, f_r) = \frac{1}{3} \langle q, r \bullet q \bullet |q| \rangle + \langle p_r, f_r \rangle \tag{12}$$

The determination of the equilibrium of the network then amounts to minimizing the energy of the network while respecting Kirchho's first law, which amounts to solving the problem of following optimization:

$$\begin{aligned} \min_{(q \in \mathbb{R}^n, f_r \in \mathbb{R}^{mr})} & \frac{1}{3} \langle q, r \bullet q \bullet |q| \rangle + \langle p_r, f_r \rangle \\ s.t. & \\ & Aq - f = 0 \end{aligned} \tag{13}$$

In this form, it is (a little) tricky to prove the existence and uniqueness of the solution, mainly because of the presence of the linear term in fr in the script. ere, which means that the function J is neither coercive nor strictly convex. To overcome this difficulty, we use the partition (9) of the matrix A to express the flows fr as a function of the flows q. The problem (13) is then put in the equivalent form:

$$\begin{aligned} \min_{(q \in \mathbb{R}^n)} & \frac{1}{3} \langle q, r \bullet q \bullet |q| \rangle + \langle p_r, A_r q \rangle \\ s.t. & \\ & A_d q - f_d = 0 \end{aligned} \tag{14}$$

This constrained problem can itself be put in the form of a free minimization: according to **Remark1**, the matrix A_d is of full rank m_d, and we can therefore extract a sub-invertible square matrix of dimension m_d. Assuming that this sub-matrix is made up of the first m_d columns of the matrix A_d, the latter is partitioned into:

$$A_d = \begin{pmatrix} A_{d,T} & A_{d,C} \end{pmatrix} \tag{15}$$

This partition of the columns of A_d corresponds to a partition of the set of arcs of the graph according to which the vector of flows is written:

$$q = \begin{pmatrix} q_T \\ q_C \end{pmatrix} \tag{16}$$

Then, the constraint of problem (14) allows to express q_T in terms of q_C:

$$q_T = A_{d,T}^{-1} (f_d - A_{d,C} q_C) \tag{17}$$

from which we deduce the expression of the vector q itself as a function of q_C:

$$q = q^{(0)} + Bq_C \quad with \quad q^{(0)} = \begin{pmatrix} A_{d,T}^{-1} f_d \\ 0_{n-m_d} \end{pmatrix}, \quad B = \begin{pmatrix} -A_{d,T}^{-1} A_{d,C} \\ \mathrm{I}_{n-m_d} \end{pmatrix} \tag{18}$$

Problem (14) is rewritten in the unconstrained form:

$$\min_{\left(q_C \in \mathbb{R}^{n-m_d}\right)} \frac{1}{3} \left\langle q^{(0)} + Bq_C, r \bullet \left(q^{(0)} + Bq_C\right) \bullet \left|q^{(0)} + Bq_C\right| \right\rangle + \left\langle p_r, A_r \left(q^{(0)} + Bq_C\right) \right\rangle \tag{19}$$

Remark3. The matrix B has an interesting interpretation in terms of graph because each of its columns represents a cycle of the network, that is to say a path connecting two reservoirs of the network, i.e. a path whose initial node and final node are identical. The set of columns of B even forms a base of cycles, in the sense that any cycle of the network is generated by a combination of the columns of B. The matrix B is called the arcs incidence matrix-network cycles.

3 Methods

In our project [2], to solve the problem (19), we need to compute the oracles associated with this function F:

$$\min_{q_C \in \mathbb{R}^{n-m_d}} \underbrace{\frac{1}{3} \left\langle q^{(0)} + Bq_C, r \bullet \left(q^{(0)} + Bq_C\right) \bullet \left|q^{(0)} + Bq_c\right| \right\rangle + \left\langle p_r, A_r \left(q^{(0)} + Bq_c\right) \right\rangle}_{F(q_C)} \tag{20}$$

We must firstly calculate the gradient and the Hessian. For that, we appeal to the rules of the multivariate differential calculus. In general, for a function $\varphi : \mathbb{R}^\alpha \to \mathbb{R}^\beta$, we will note: We write the function (20), a composite function f, g, with:

$$g : \mathbb{R}^{n-m_d} \to \mathbb{R}^n, \quad g\left(q_c\right) = q^{(0)} + Bq_c \tag{21}$$

$$f : \mathbb{R}^n \to \mathbb{R}, \quad f(q) = \underbrace{\frac{1}{3} \langle q, r \bullet q \bullet |q| \rangle}_{f_1(q)} + \underbrace{\langle p_r, A_r q \rangle}_{f_2(q)} \tag{22}$$

Applying the chain rule and we have:

$$(f \circ g)'\left(q_c\right) = f'\left(g\left(q_c\right)\right) \cdot g'\left(q_c\right) \tag{23}$$

Therefore,

$$\nabla F\left(q_c\right) = \nabla g\left(q_c\right) \cdot \nabla f_1\left(g\left(q_c\right)\right) + \nabla g\left(q_c\right) \cdot \nabla f_2\left(g\left(q_c\right)\right) \tag{24}$$

where,

- The function g is affine, then $\nabla g\left(q_C\right) = B^\top$;
- The function f_2 is linear, then $\nabla f_2\left(q_C\right) = A_r^\top p_r$;
- The function f_1 is the scalar product of two vectors: its gradient can be obtained by the product rule:

$$\nabla(\langle q, r \bullet q \bullet |q| \rangle) = \nabla(q) \cdot (r \bullet q \bullet |q|) + \nabla(|r \bullet q \bullet |q\,|) \cdot q$$

- $\nabla(q)$ is the gradient of the function: $q \mapsto q$: it is therefore the identity matrix of $\mathcal{R}^n$;
- $\nabla(r \bullet q \bullet |q|)$ is the gradient of the function: $q \mapsto r \bullet q \bullet |q|$: it is a diagonal matrix (to be calculated) that we denote by $R(q)$.

Combining all the terms, we obtain the gradient of F:

$$\nabla F\left(q_c\right) = \frac{1}{3} B^\top (r \bullet q \bullet |q| + R(q)q) + B^\top A_r^\top p_r \tag{25}$$

It is easily shown that the diagonal matrix $R(q)$ obtained during the calculation of the gradient of the function F is such that:

$$R(q)q = 2r \bullet q \bullet |q| \tag{26}$$

The complete expression of the gradient of the function F is then:

$$\nabla F\left(q_c\right) = B^\top (r \bullet q \bullet |q|) + B^\top A_r^\top p_r \tag{27}$$

The Hessian expression is given by the gradient of the gradient. The term $B^\top A_r^\top p_r$ being constant, it is therefore sufficient to differentiate:

$$q_c \mapsto q = q^{(0)} + Bq_c \mapsto z = r \bullet q \bullet |q| \mapsto B^\top z \tag{28}$$

The gradient of the function $q \mapsto z$ is $R(q)$(obtained when calculating the gradient), and the other 2 functions in the above expression are completed. The rule in the chain then provides the desired result.

3.1 Descent Gradient Method

From the previous part, we have computed the gradient of the objective function (20) of the optimization problem (19). Then we can solve this problem with the numerical method: descent gradient method to get an approximation to the solution. A natural choice for the search direction is the negative gradient $\Delta x = -\nabla f(x)$. The resulting algorithm is called the gradient algorithm or gradient descent method[1].

Algorithm 1 Gradient Descent Method with Fixed Step Size

1: Initialization: a starting point $q_c^0 \in \operatorname{dom} F$
2: **repeat**
3: $\quad \Delta q_c := -\nabla F(q_c)$
4: $\quad$ *Line search.* Choose fixed step size t.
5: $\quad$ Update $q_c := q_c + t\Delta q_c$
6: **until** stopping criterion is satisfied

Here we develop other two types of descent gradient algorithms with different optimal step size t from exact line search and Wolfe's conditions: One line search method sometimes used in practice is exact line search, in which t is chosen to minimize $F(q_c)$ along the ray $\{x + t\Delta x \mid t \geq 0\}$:

$$t = \operatorname{argmin}_{s \geq 0} f(x + s\Delta x) \tag{29}$$

The introduction to Wolfe's conditions would be at **Section 5.3**

Algorithm 2 Gradient Descent Method with Exact Line Search

1: Initialization: a starting point $q_c^0 \in \operatorname{dom} F$
2: **repeat**
3: $\quad \Delta q_c := -\nabla F(q_c)$
4: $\quad$ *Line search.* Choose step size t via exact line search.
5: $\quad$ Update $q_c := q_c + t\Delta q_c$
6: **until** stopping criterion is satisfied

Algorithm 3 Gradient Descent Method with Wolfe's Conditions

1: Initialization: a starting point $q_c^0 \in \operatorname{dom} F$
2: **repeat**
3: $\quad \Delta q_c := -\nabla F(q_c)$
4: $\quad$ *Line search.* Choose step size t via Wolfe's conditions.
5: $\quad$ Update $q_c := q_c + t\Delta q_c$
6: **until** stopping criterion is satisfied

3.2 Newton's Method

From the previous part, we have computed the gradient of the objective function (20) of the optimization problem (19). Then we can solve this problem with the numerical method: Newton's method to get an approximation to the solution. Newton's method, as outlined below, is sometimes called the damped Newton method or guarded Newton method, to distinguish it from the pure Newton method [1], which uses a fixed step size $t = 1$.

Algorithm 4 Newton's method with Fixed Step Size

1: Initialization: a starting point $q_c^0 \in \operatorname{dom} F$, tolerance $\epsilon > 0$.
2: **repeat**
3: Compute the Newton step and decrement:

$$\Delta q_c nt := -\nabla^2 F(q_c)^{-1} \nabla F(q_c); \quad \lambda^2 := \nabla F(q_c)^T \nabla^2 F(q_c)^{-1} \nabla F(q_c)$$

4: Stopping criterion. **quit if** $\lambda^2/2 \leq \epsilon$
5: *Line search.* Choose fixed step size t.
6: Update $q_c := q_c + t\Delta q_c nt$
7: **until** stopping criterion is satisfied

Here we develop a new Newton's algorithms with optimal step size t from Wolfe's conditions:

Algorithm 5 Newton's method with Wolfe's Conditions

1: Initialization: a starting point $q_c^0 \in \operatorname{dom} F$, tolerance $\epsilon > 0$.
2: **repeat**
3: Compute the Newton step and decrement:

$$\Delta q_c nt := -\nabla^2 F(q_c)^{-1} \nabla F(q_c); \quad \lambda^2 := \nabla F(q_c)^T \nabla^2 F(q_c)^{-1} \nabla F(q_c)$$

4: Stopping criterion. **quit if** $\lambda^2/2 \leq \epsilon$
5: *Line search.* Choose step size t via Wolfe's conditions.
6: Update $q_c := q_c + t\Delta q_c nt$
7: **until** stopping criterion is satisfied

3.3 Linear Search with Wolfe's Conditions

3.3.1 General Principles

Let F be a function to be minimized. We notice:

- $x^{(k)}$ the current point;
- $g^{(k)}$ the gradient of f in $x^{(k)}$;
- $d^{(k)}$ the descent direction;
- $\alpha^{(k)}$ the step size.

The purpose of Wolfe's conditions is to determine a step size $\alpha^{(k)}$ satisfying the following two conditions:

1. the function f must decrease in a significant way:
$$f\left(x^{(k)} + \alpha^{(k)} d^{(k)}\right) \leq f\left(x^{(k)}\right) + \omega_1 \alpha^{(k)} g^{(k)^\top} d^{(k)} \tag{30}$$
2. the step size $\alpha^{(k)}$ must be large enough:
$$\left(\nabla f\left(x^{(k)} + \alpha^{(k)} d^{(k)}\right)\right)^\top d^{(k)} \geq \omega_2 g^{(k)^\top} d^{(k)} \tag{31}$$

with $0 < \omega_1 < \omega_2 < 1$ (typically, $\omega_1 = 0.1$ and $\omega_2 = 0.9$).

3.3.2 Line Search Algorithm Implementation with Wolfe's Conditions

The following iterative procedure, inspired by an implementation due to Fletcher Lemarechal, allows in a particularly simple way to compute a step size $\alpha^{(k)}$ verifying the two Wolfe's conditions [4].

Algorithm 6 Line Search Algorithm with Wolfe's Conditions

Initialization: $\alpha = 0, \quad \bar{\alpha} = +\infty, \quad \alpha^{(k,1)} \in]\underline{\alpha}, \bar{\alpha}[$.
repeat
  if $\alpha^{(k,\ell)}$ does not satisfy the Wolfe's condition (30) **then**
    Decrease the upper bound: $\bar{\alpha} = \alpha^{(k,\ell)}$
    Choose a new step: $\alpha^{(k,\ell+1)} = \frac{1}{2}(\underline{\alpha} + \bar{\alpha})$
  else
    if $\alpha^{(k,\ell)}$ does not satisfy the Wolfe's condition (31) **then**
      Increase the lower bound: $\underline{\alpha} = \alpha^{(k,\ell)}$
      Choose a new step: $\alpha^{(k,l)} = \begin{cases} 2\underline{\alpha} \text{ if } \bar{\alpha} = +\infty \\ \frac{1}{2}(\underline{\alpha} + \bar{\alpha}) \text{ otherwise} \end{cases}$
    else
      the step size $\alpha^{(k,l)}$ satisfies Wolfe's conditions and will be used to move in the direction
    end if
  end if
until stopping criterion is satisfied

Note that conditions (30) and (31) are not treated symmetrically in this procedure. Indeed, the second condition of Wolfe is tested only if the first condition is satisfied. On the other hand, the modification of the lower bound in the case where the second condition is not true obliges to redo the test of the first condition.

3.4 Polak-Ribiere Algorithm

[4]Let f be a function to be minimized. We notice:

- $x^{(k)}$ the current point;

- $g^{(k)}$ the gradient of f in $x^{(k)}$.

We look for a direction $d^{(k)}$ in which the function f decreases from the point $x^{(k)}$

$$\exists \alpha^{(k)} > 0, \quad f\left(x^{(k)} + \alpha^{(k)} d^{(k)}\right) < f\left(x^{(k)}\right) \tag{32}$$

The above inequality is induced by the relation: $g^{(k)\top} d^{(k)} < 0$. The step size $\alpha^{(k)}$ is generally determined by linear search. In conjugate gradient-type methods, the direction of descent d (k) at point x (k) is of the form:

$$d^{(k)} = \begin{cases} -g^{(1)} & \text{if } k = 1 \\ -g^{(k)} + \beta^{(k)} d^{(k-1)} & \text{otherwise} \end{cases} \tag{33}$$

In the implementation of Polak-Ribiere algorithm, the coefficient $g^{(k)}$ is given by the formula:

$$\beta^{(k)} = \frac{g^{(k)^\top}\left(g^{(k)} - g^{(k-1)}\right)}{\left\|g^{(k-1)}\right\|^2} \tag{34}$$

Then we develop the two types of Conjugate gradient algorithms with different optimal step size t from fixed-step size and Wolfe's conditions:

Algorithm 7 Polak-Ribiere Algorithm with Fixed Step

1: Initialization: a starting point $q_c^0 \in \operatorname{dom} F$
2: **repeat**
3: Get Δq_c from Polak-Ribiere Algorithm
4: *Line search.* Choose fixed step size t
5: Update $q_c := q_c + t\Delta q_c$
6: **until** stopping criterion is satisfied

Algorithm 8 Polak-Ribiere Algorithm with Wolfe's Conditions

1: Initialization: a starting point $q_c^0 \in \operatorname{dom} F$
2: **repeat**
3: Get Δq_c from Polak-Ribiere Algorithm
4: *Line search.* Choose step size t via Wolfe's conditions
5: Update $q_c := q_c + t\Delta q_c$
6: **until** stopping criterion is satisfied

3.5 Quasi-Newton's algorithm with BFGS

In quasi-Newton's methods[4], the descent direction is of the form:

$$d^{(k)} = -W^{(k)} g^{(k)} \tag{35}$$

where $W^{(k)}$ represents an approximation of the inverse of the Hessian matrix $H^{(k)}$ of f at point $x^{(k)}$. This approximation is updated at each iteration of the algorithm.
With the following notations:

- $\delta_x^{(k)} = x^{(k)} - x^{(k-1)}$
- $\delta_g^{(n)} = g^{(k)} - g^{(k-1)}$

The BFGS (Broyden-Fletcher-Goldfarb-Shanno) discount formula is given by:

$$W^{(k)} = \left(I - \frac{\delta_x^{(k)} \delta_g^{(k)\top}}{\delta_g^{(k)\top} \delta_x^{(k)}} \right) W^{(k-1)} \left(I - \frac{\delta_g^{(k)} \delta_x^{(k)\top}}{\delta_g^{(k)\top} \delta_x^{(k)}} \right) + \frac{\delta_x^{(k)} \delta_x^{(k)^\top}}{\delta_g^{(k)^\top} \delta_x^{(k)}} \tag{36}$$

In the absence of a best idea, the matrix $W^{(1)}$initializing this formula is taken as the identity. Then we can give the Quasi-Newton's algorithm with BFGS under the Wolfe's conditions

Algorithm 9 Quasi-Newton's method with Wolfe's Conditions

1: Initialization: a starting point $q_c^0 \in \operatorname{dom} F$, tolerance $\epsilon > 0$.
2: **repeat**
3: Compute the approximation W via $BFGS$ formular
4: Compute the Newton step and decrement:

$$\Delta q_c nt := -W \nabla F(q_c); \quad \lambda^2 := \nabla F(q_c)^T W \nabla F(q_c)$$

5: Stopping criterion. **quit if** $\lambda^2/2 \leq \epsilon$
6: *Line search.* Choose step size t via Wolfe's conditions.
7: Update $q_c := q_c + t\Delta q_c nt$
8: **until** stopping criterion is satisfied

4 Experiment Design

4.1 Equipments

4.2 Programming Language

This lab will be done in Python. However, the data set and most of the functions allowing to carry out the practical work are also available in Jupyter.

For the Python, the prototype notebook is provided, which allows to guide the user in the realization of the practical work and to simplify the execution of the codes to be developed. It is also possible to insert calculations, answers to theoretical questions and comments associated with the practical work. In the absence of a notebook, a flow monitor is available for all languages, which also allows you to be guided in carrying out the practical work.

4.3 Variables

As reminded, we give below the correspondence table between the mathematical symbols used in the description of the problem and the data-processing variables used in the codes. The last part of the table (q, q_C, z, f and p) corresponds to the variables that need to be calculated when solving the problem, while the variables in the first parts of the table correspond to data supplied to users (see the Problem R and Structures N files).

We will pay attention to the fact that the names of the variables appearing in the first part of the table above are reserved: they should therefore not be redefined under penalty of losing, in some languages the access to the corresponding data.

4.4 Algorithm Implementation

1. We are interested in the primal problem of unconstrained optimization (19):

   - from the data and notations of the Problem R and Structures N, we can obtain the oracle function associated with the problem which provides at a point q_C the values of the criterion and of the gradient of the problem (19):

     $$[F, G] = OraclePG\,(q_C) \tag{37}$$

     where q_C is the vector of reduced debits, F is the value of the criterion evaluated at the point q_C; C is the vector of the derivatives of the criterion with respect to q_C.

   - The second oracle incorporating the Hessian calculus:

     $$[F, G, H] = OraclePH\,(q_C) \tag{38}$$

     where q_C is the vector of reduced debits, F is the value of the criterion evaluated at the point q_C; C is the vector of the derivatives of the criterion with respect to q_C; H is the matrix of second derivatives of the criterion with respect to q_C.

Physical Description	Notation	Programming Variable	Matrix Form
Total number of arcs	n	n	$\mathcal{N}$
Total number of nodes	m	m	$\mathcal{N}$
Number of demand nodes	m_d	md	$\mathcal{N}$
Number of reservoir nodes	m_r	mr	$\mathcal{N}$
Demands to demand nodes	f_d	fd	$\mathcal{M}(m_d, 1)$
Reservoir node pressures	p_r	pr	$\mathcal{M}(m_r, 1)$
Arc resistances	r	r	$\mathcal{M}(n, 1)$
Initial flow vector	q^0	q0	$\mathcal{M}(n, 1)$
Node-arc incidence matrix	A	A	$\mathcal{M}(m, n)$
"Demand" sub-matrix of A_d	A_d	Ad	$\mathcal{M}(m_d, n)$
"Reservoir" sub-matrix of A_r	Ar	Ar	$\mathcal{M}(m_r, n)$
"tree" sub-matrix of A_d	$A_{d,T}$	AdT	$\mathcal{M}(m_d, m_d)$
"Co-tree" sub-matrix of A_d	$A_{d,C}$	AdC	$\mathcal{M}(m_d, n - m_d)$
Inverse matrix of $A_{d,I}$	$A_{d,I}$	AdI	$\mathcal{M}(m_d, m_d)$
Arc-cycle incidence matrix	B	B	$\mathcal{M}(n, n - m_d)$
Arc-flow rates	q	q	$\mathcal{M}(n, 1)$
Reduced arc-flow rates	q_C	qC	$\mathcal{M}(n - m_d, 1)$
Arc pressure losses	z	z	$\mathcal{M}(n, 1)$
Node flow	f	f	$\mathcal{M}(m, 1)$
Node pressures	p	p	$\mathcal{M}(m, 1)$

Table 1: Correspondence between notation and programming variables

2. We test these two oracles by interfacing them respectively with the fixed-step gradient descent algorithm (Gradient F function) and Newton's algorithm with fixed-step (Newton F function) both provided as part of the TP. We will note in these algorithms the memorization of the values of the criterion, of the norm of the gradient and of the length of the gradient step (in fact the log_{10} of these last two quantities), making it possible to visualize the behavior of the algorithm during iterations (Visualg function).

3. we test the first oracle by interfacing with an optimization function available in the language used: Minimize function of PYTHON

4.5 Comparison Experiment

The goal of these sessions is to solve problem (19) by 4 different optimization algorithms.

1. We implement a linear search verifying Wolfe's conditions and the descent gradient algorithm with optimal step from this linear search. Linear research will implement the Fletcher algorithm (see the **Section 5 Methods**). We will take as a basis the Wolfe function to implement this algorithm.

2. Implementing the conjugate gradient and quasi-Newton methods with Wolfe search by:

   - Implementing the Polak-Ribiere algorithm (non-linear conjugate gradient algorithm)

- Implementing the BFGS algorithm (using an approximation of the inverse of Hessian;
- Solving problem (19) by these two methods using oracle $OraclePG$

3. To implement Newton's method; we will have to:
   - Implement the Newton's algorithm with the optimal step from Wlofe's conditions;
   - Solve problem (19) using oracle $OraclePH$.
4. We compare the results (Optimal values, number of iterations, CPU time, speed of convergence) of these four optimization methods and the initial descent gradient method with fixed-step.

4.6 Remark on Practical Programming

1. The implementation of the linear search method and the optimization algorithms must be done independently of the problem under consideration. We recall (see the previous session) that we encapsulate each optimization method in a function of the form:

$$\left[F^\sharp, x^\sharp, G^\sharp\right] = Minimise\,(\,Oracle\,, x_0, \alpha_0) \tag{39}$$

   where the following sub-functions would be applied in implementation:
   - $GradientV$: the descent gradient algorithm with the optimal step from the Wolfe's conditions;
   - $PolakRibiere$: the Polak Ribiere algorithm;
   - $BFGS$: the quasi-Newton's algorithm with BFGS formular;
   - $NewtonV$: the Newton's algorithm with the optimal step from the Wolfe's conditions;
2. In all the optimization methods, we will verify that the direction in which the linear search is carried out is indeed a direction of descent. In Newton's method, we will check that the Hessian matrix is indeed invertible.
3. To compare the convergence speeds, we will plot for each method the variation of the base 10 logarithm of the gradient norm as a function of the iteration index; we will be inspired by what is done in the Gradient F function (see Gradient F).
4. Note that the linear search uses the oracle only for criteria and gradient calculations. The syntax of the two oracles written in the previous session is such that they can be used interchangeably in the linear search algorithm.

5 Result

5.1 Descent Gradient Algorithm

5.1.1 Descent Gradient Algorithm with Fixed-Step

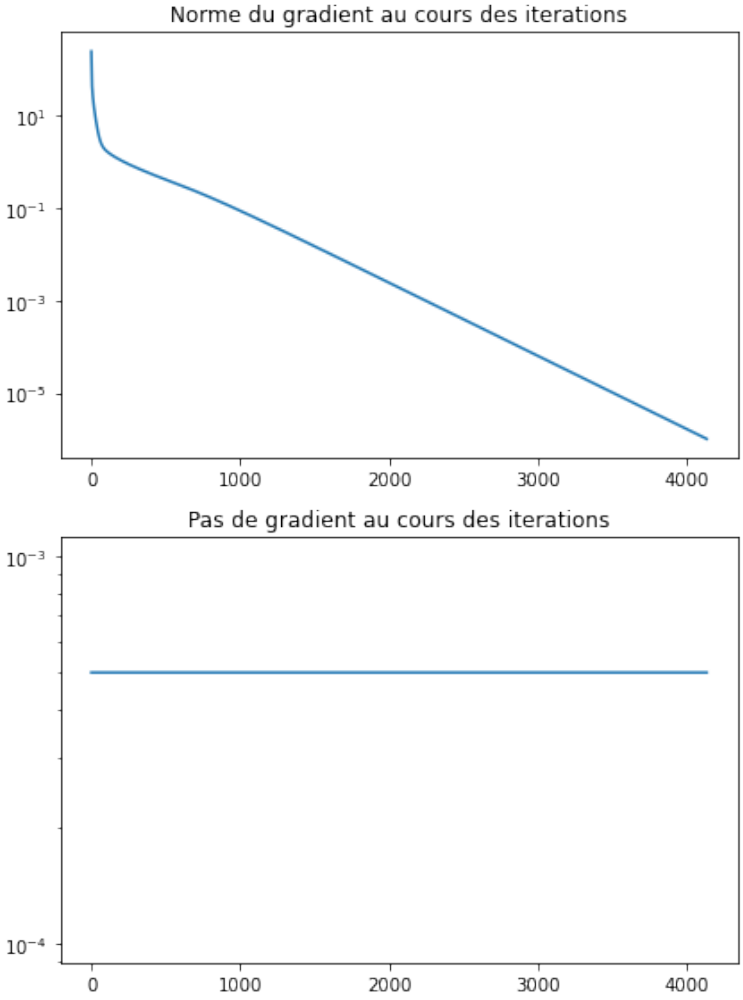

Figure 2: Gradient descending algorithm with fixed step size

5.1.2 Descent Gradient Algorithm with Optimal Step from the Exact Line Search

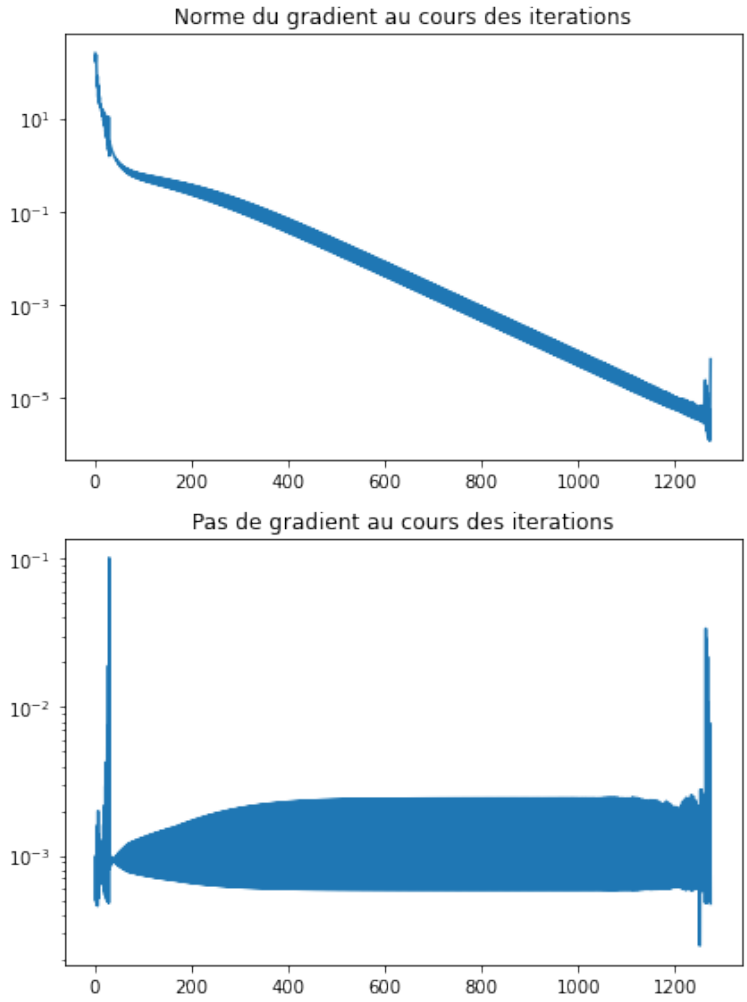

Figure 3: Gradient descending algorithm with optimal step size by exact line search

5.1.3 Descent Gradient Algorithm with Optimal Step from the Line Search with the Wolfe's Conditions

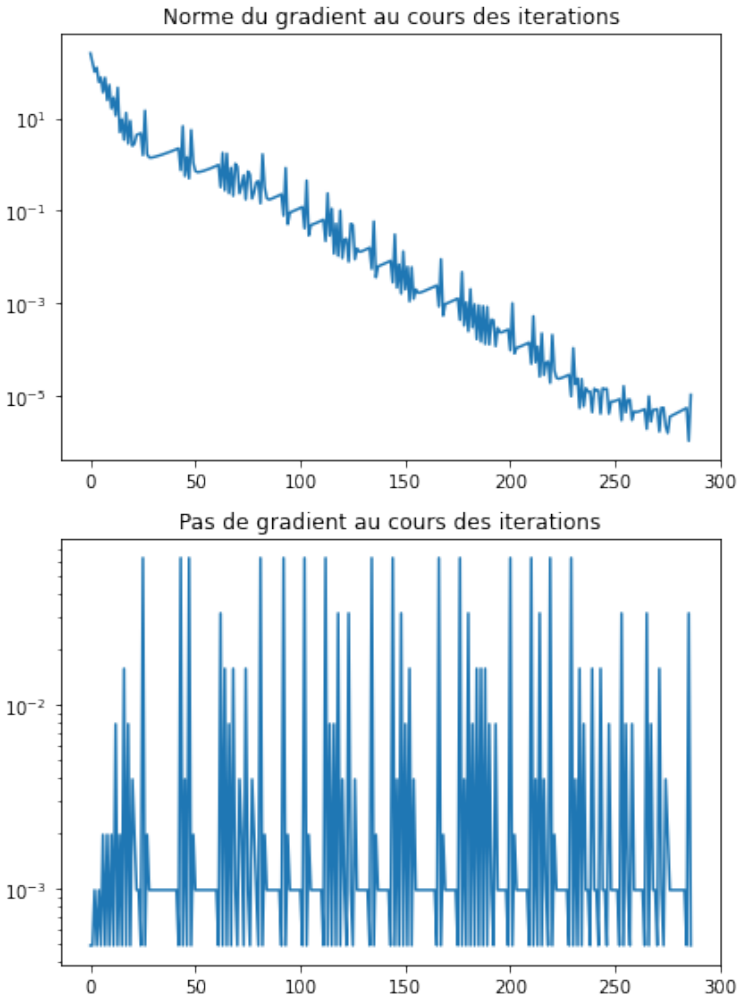

Figure 4: Gradient descending algorithm with optimal step size by Wolf algorithm

5.1.4 Polak-Ribiere Algorithm with Fixed-Step

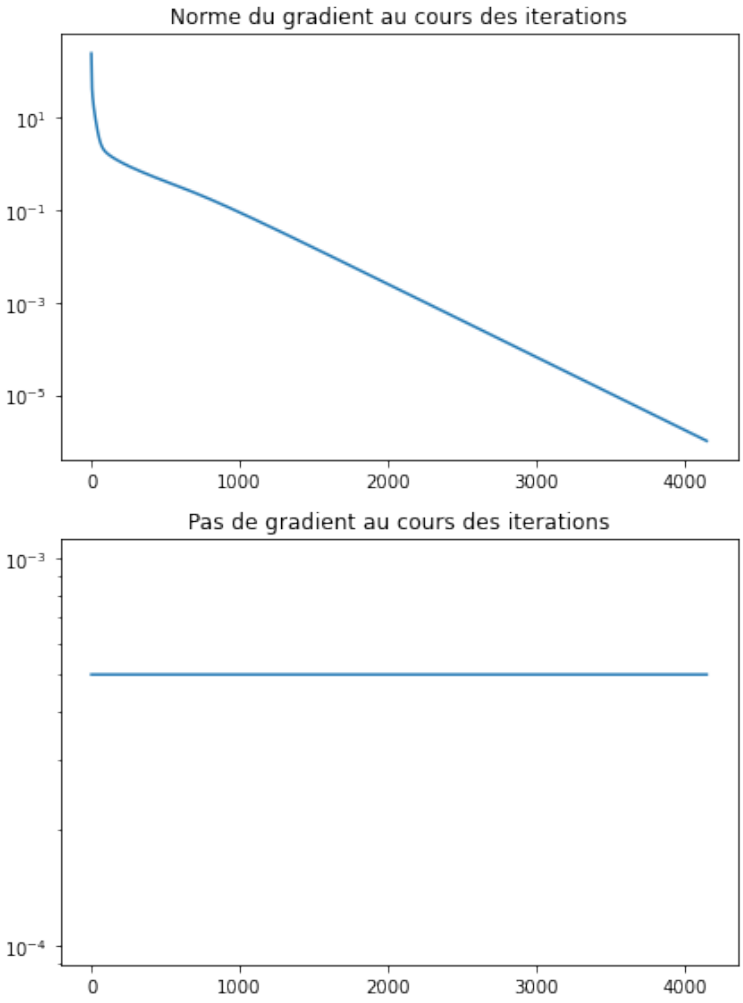

Figure 5: Polak-Ribiere algorithm with Fixed-Step

5.1.5 Polak-Ribiere algorithm with Optimal Step from the Line Search with the Wolfe's Conditions

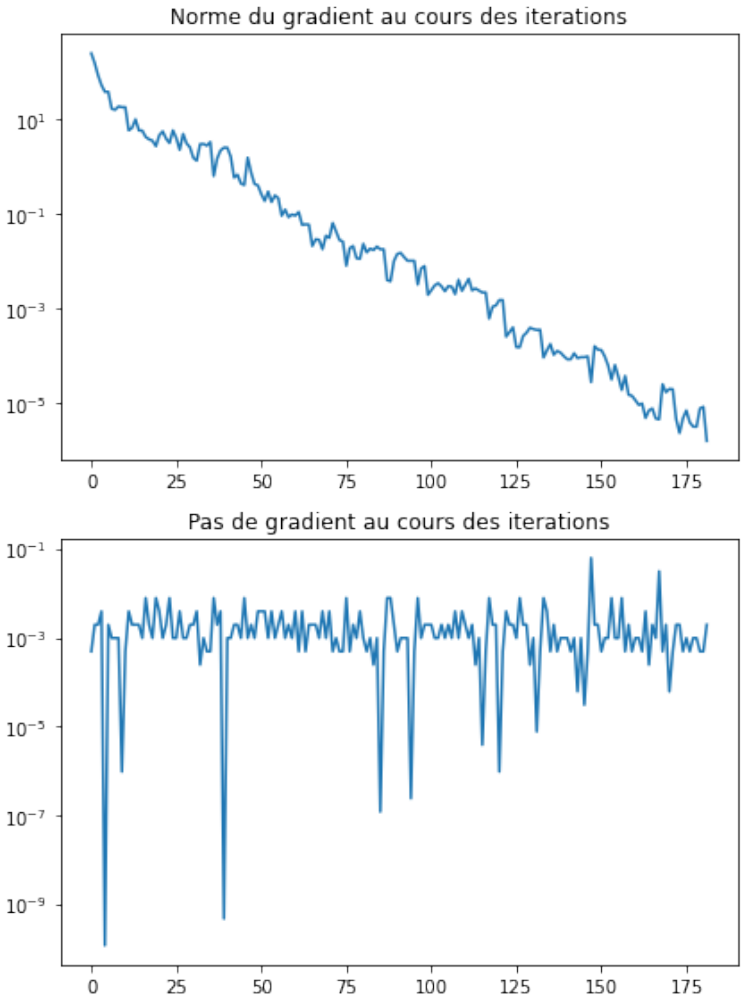

Figure 6: Polak-Ribiere Algorithm with Optimal Step from the Line Search with the Wolfe's Conditions

5.2 Newton's Algorithm

5.2.1 Newton's Algorithm with Fixed-Step

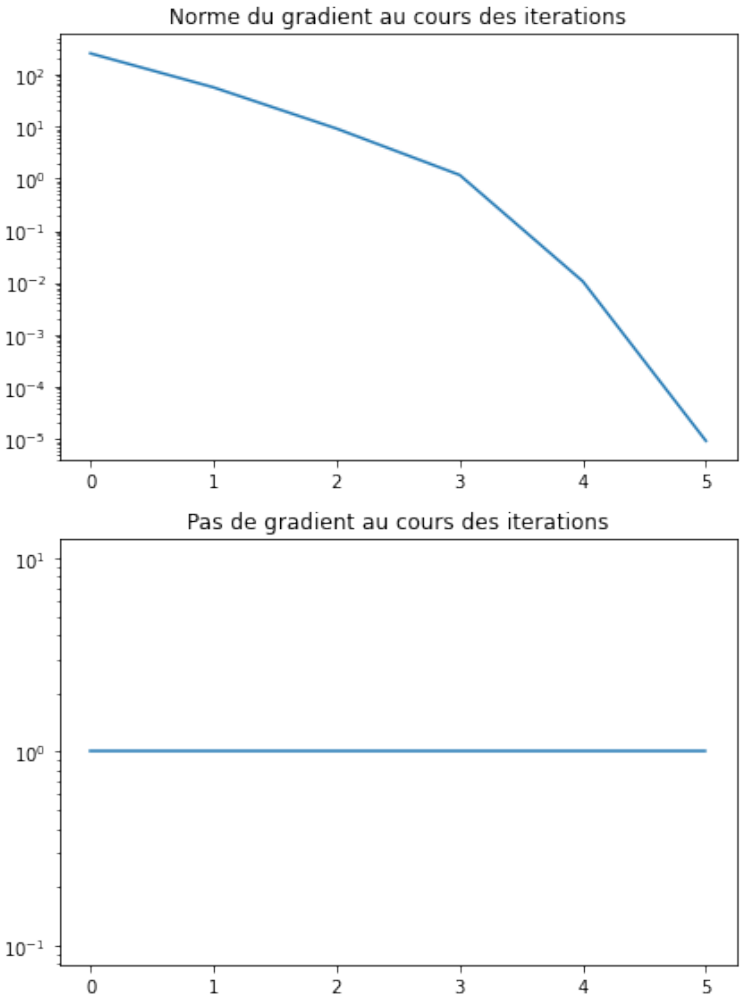

Figure 7: Newton's Algorithm with Fixed-Step

5.2.2 Newton's Algorithm with Optimal Step from the Line Search with the Wolfe's Conditions

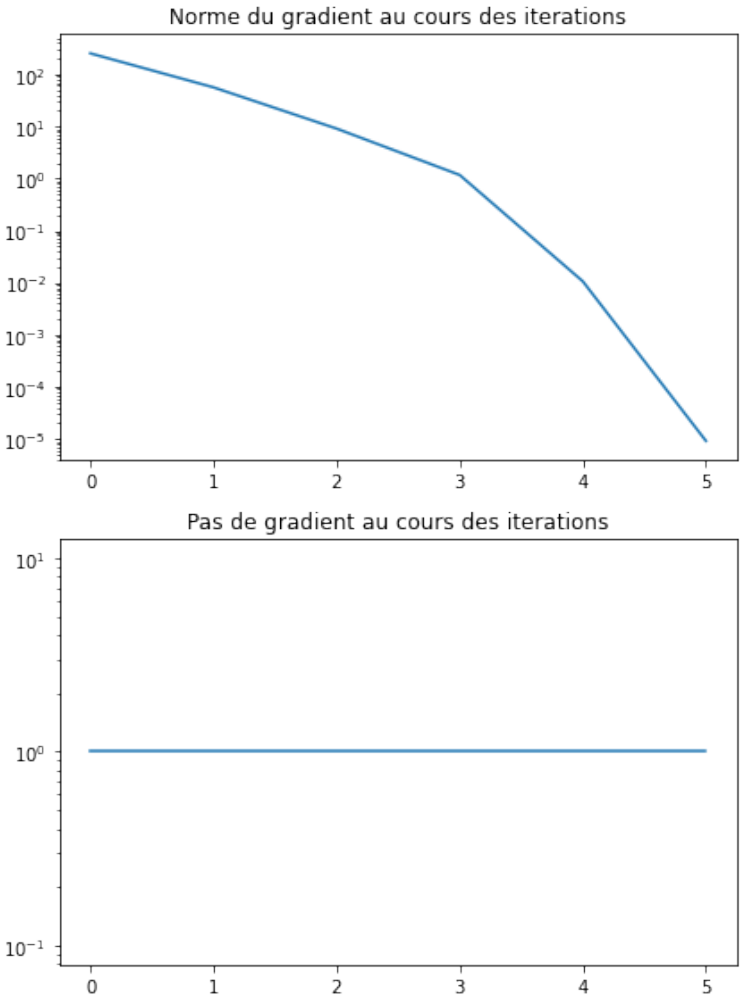

Figure 8: Newton's Algorithm with Optimal Step from the Line Search with the Wolfe's Conditions

5.2.3 Quasi - Newton's Algorithm with Optimal Step from the Line Search with the Wolfe's Conditions

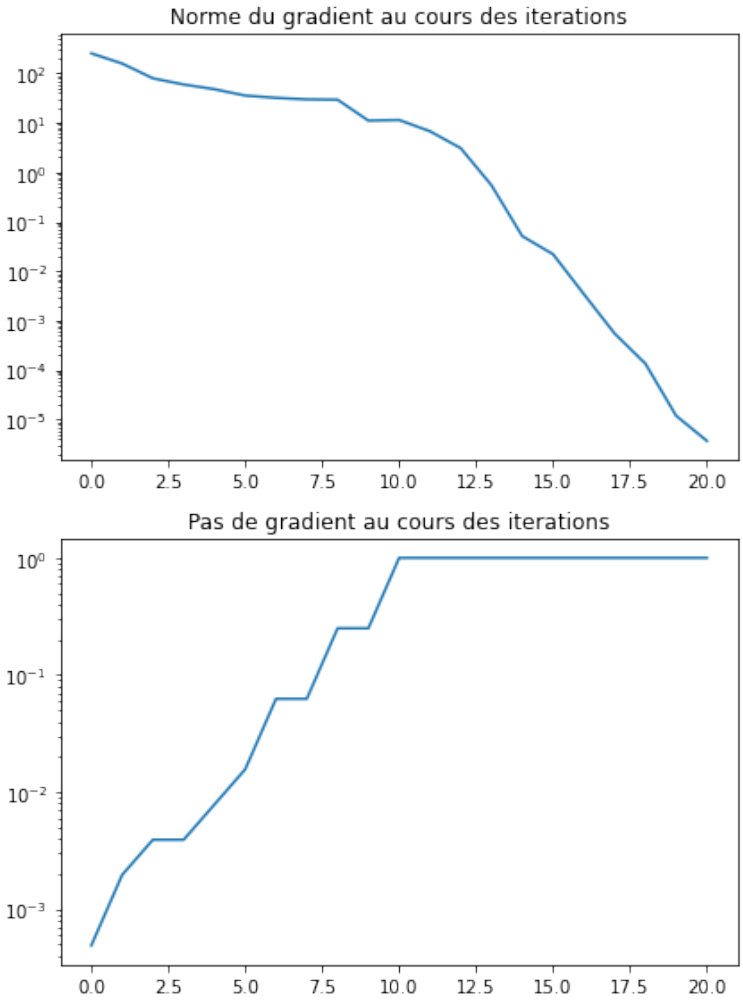

Figure 9: Quasi Newton method with optimal step size by Wolf algorithm

5.3 Experimental Result

Algorithms	*Parameter*	*Iterations*
Gradient Algorithm (Fixed-Step)	$t = 0.0005$	4132
Gradient Algorithm (Exact-Step)	$t = 1$	1289
Gradient Algorithm (Wolfe's)	$t = 0.0005$	1273
Polak-Ribiere Algorithm (Fixed-Step)	$t = 0.0005$	4149
Polak-Ribiere Algorithm (Wolfe's)	$t = 0.0005$	204
Newton's Algorithm(Fixed-Step)	$t = 1$	6
Newton's Algorithm(Wolfe's)	$t = 1$	6
Quasi-Newton's Algorithm(Wolfe's)	$t = 1$	21

Table 2: Experimental result for algorithms

6 Conclusion

In this project, we solve the problem of the resolution of the equations describing the state of equilibrium of a drinking water distribution network. This problem can be transformed into the minimization problem of a objective function (representing the energy of the network) under linear constraints of equality(translating the first Kirchho's law). We will apply to it the main optimization algorithms , in the unconstrained framework

7 References

[1] Boyd, Stephen, Stephen P. Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[2] Description du Projet https://perso.ensta-paris.fr/~pcarpent/TP_Reseau/Textes/TP_Reseau_Document.pdf.

[3] Slides de Présentation https://perso.ensta-paris.fr/~pcarpent/TP_Reseau/Textes/TP_Reseau_Slides.pdf.

[4] Note sur la Recherche Linéaire https://perso.ensta-paris.fr/~pcarpent/TP_Reseau/Textes/Methodes.pdf.