Project 4
Non-linear systems of equations / Newton-Raphson method

Group 4 - Team 8631

Manager : ymougou
Secretary : ilmourid
Programmers : hpierre001, sabidi002, cpreaut

Abstract : This project is based on the development of an algorithm which can search the roots of a non-linear equation system. The method used is the Newton-Raphson algorithm and two examples were done to test this algorithm, the computation of the Lagrangian points and the search of electrostatics equilibrium. Another algorithm was implemented to factorize polynoms with the Bairstow method. (*Part written by Hugo PIERRE.*)

1 Newton-Raphson algorithm

The Newton-Raphson algorithm requires the Jacobian matrix of the function so there were two options. The first one is to calculate the Jacobian matrix manually for each function processed by the Newton-Raphson algorithm and the second one is to write an algorithm that compute this matrix for a function given in parameter and a point where it is applied. *Written by Hugo PIERRE.*

1.1 Jacobian matrix

Part written by Hugo PIERRE.

This part deals with the algorithm that computes the Jacobian matrix. It uses the Taylor-Young formula to calculate partial derivatives discreetly. This algorithm was tested on a non-linear function:

$$\begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} 3xy \\ x^2 \\ 4x(x+y) \end{pmatrix} \tag{1}$$

The Figure 1 below represents the relative gap between the theoretical Jacobian matrix and the computed one for the function given by (1). The relative gap increases when the norm of $\begin{pmatrix} x \\ y \end{pmatrix}$ decreases, this is due to the fact that the parameter h is fixed in the algorithm but theoretically it has to be negligible compared to the norm of the vector in the Taylor-Young formula. Yet, as h is fixed, it is not negligible when x and y are closed to 0. However, because that way it could become negligible for the floating point calculation and thus lead the algorithm to be mistaken.

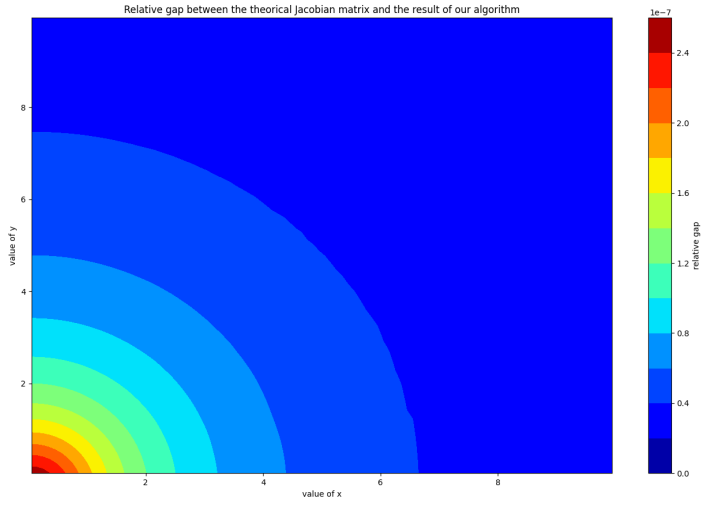

Figure 1: Relative gap between theoretical and computed Jacobian matrix

1.2 Newton-Raphson

Part wrote by Clément PREAUT.

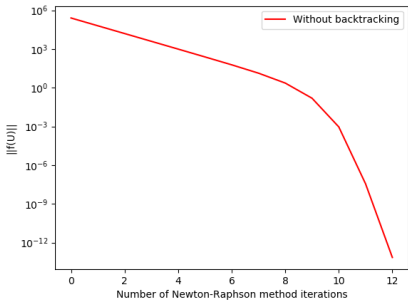

Figure 2: Primary test

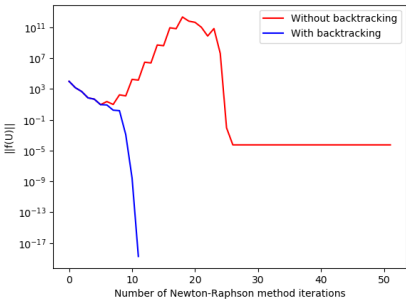

Figure 3: Comparison with and without backtracking

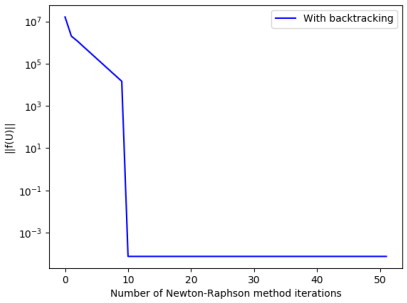

Figure 4: Extremum problem

In order to find a solution for $f(U) = 0$ for a given function f, the iterative Newton-Raphson method was used. For a given vector U , a function f and its associated Jacobian matrix H(U), a vector V was found so that $f(U + V) = 0$, where $f(U) + H(U) \times V$ approximated $f(U + V)$. This operation was reiterated until $||f(U)||$ was close enough to zero.
Therefore, in each iteration of the the *Newton_ Raphson* function's *while* loop, a new V was found by solving $f(U) + H(U) \times V = 0$ thanks to *numpy.linalg.lstsq* . The progress towards the desired result shifted from a linear speed to a quadratic speed below a level.
In *Figure* 2, the *Newton_ Raphson* method was applied on the function :
$f : \mathbf{R} \longrightarrow \mathbf{R}, x \hookrightarrow x^2 + 3 \times x - 4$
The curve began to decrease dramatically after the breaking point of the tenth iteration. The function also returned value $||f(x)|| \simeq 7.10^{-14}$ and 1 for x, which indeed belongs to the solutions.
Nevertheless, problems were encountered. In many cases the algorithm diverges. In consequence, initial conditions were chosen in an area in which a solution could be found. In addition, to avoid ifnite iterations of the algorithm, the implementaion could not execute more than an imposed number. Moreover, in some circumstances, the norm of *H(U)* was very low. It removed the vector V far from the value of the vector U therefore the relation $||f(U+V)|| \geq ||f(U)||$ risked to appear. Consequently *Backtracking* consisted in reducing V until the inequality inverted. In *Figure* 3 above, applications of the *Newton_ Raphson* method with and without backtracking were compared, on the function
$g : \mathbf{R}^2 \longrightarrow \mathbf{R}^2, (x, y) \hookrightarrow (\frac{x+y}{x^2+1}, (x^2 + 2) \times y)$.
First of all, with backtracking, the result constantly approached zero as well as in the first example, while without backtracking, the curve shot up until a peak of more than 10^{11}. Finally, the intended result was obtained after the eleventh iteration thanks to backtracking whereas by applying the primal algorithm, the curve stayed at an unsatisfactory level until passing the limit number of iterations. The main difficulty of the *backtracking* method happened when U was either a local minimum of a positive curve either a local minimum of a negative curve. The derivate oriented such that it made $||f(U)||$ the minimum value available near U. Therefore, no vector V added to U and $||f(U)||$ thus remained steady. *Figure* 1.2 above shows this. Finally, by avoiding useless calls of *numpy.linalg.norm* or f when U had not changed yet,the execution time was reduced.

2 Computation of the Lagrangian points

Part written by Ibrahim LMOURID.
Euler and Lagrange proved the existence of five equilibrium points in the restricted problem of the three bodies. Each point represents a position of space in a 2-body system (the Sun and the Earth for example), where their fields of gravity combine to provide an equilibrium point for a third body of negligible mass (satellite for example). Those points are commonly called: L1, ..., L5.

2.1 Forces

The goal of this section is to write the code of three kinds of forces:

1. An elastic force: **fe :** $\begin{pmatrix} x \\ y \end{pmatrix} \longrightarrow \begin{pmatrix} -k \times x \\ -k \times y \end{pmatrix}$
2. A centrifugal force: **fc :** $\begin{pmatrix} x \\ y \end{pmatrix} \longrightarrow \begin{pmatrix} k \times (x - x_0) \\ k \times (y - y_0) \end{pmatrix}$
3. A gravitational force: **fg :** $\begin{pmatrix} x \\ y \end{pmatrix} \longrightarrow \begin{pmatrix} \frac{-k \times (x - x_0)}{((x-x_0)^2+(y-y_0)^2)^{\frac{3}{2}}} \\ \frac{-k \times (y - y_0)}{((x-x_0)^2+(y-y_0)^2)^{\frac{3}{2}}} \end{pmatrix}$

In order to represent these functions, three main algorithms were implemented:

1. **elastic_force**
2. **centrifugal_force**
3. **gravitational_force**

Each of these cases needs two parameters:

1. k : The intensity of the force
2. (x_0, y_0) : A central point from which the force is issued.

2.2 Equilibrium points

The goal of this section is to use the Newton_Raphson algorithm to obtain the 5 Lagrangian points of an object moving in the plane and subject to these forces:

1. Gravitational force of the Sun, which was subject to: k=1 and x_0=0 and y_0=0.
2. Gravitational force of the Earth, which was subject to: k=0.01 and x_0=1 and y_0=0.
3. Centrifugal force centered on the barycenter of Sun and Earth, which was subject to: k=1 and x_0=0.01/1.01 and y_0=0.

The application of the Newton_Raphson algorithm on a vector U_0(x,y) yielded a single equilibrium point. Thus, it was necessary to change the vector U_0 and to apply again the Newton_Raphson algorithm in order to obtain another equilibrium point, with the new vector U_0. However, it was possible to find the same equilibrium point, with the new vector U_0. Consequently, it was very important to apply the Newton_Raphson algorithm on a very large number of vectors. This approach made it possible to find all Langrangian points.
It was decided to apply Newton_Raphson algorithm on 1600 vectors, belonging to $[-5, 5] \times [-5, 5]$.
The graph in *Figure 5* below shows the results given by the algorithm **Lagrangian_points()**:

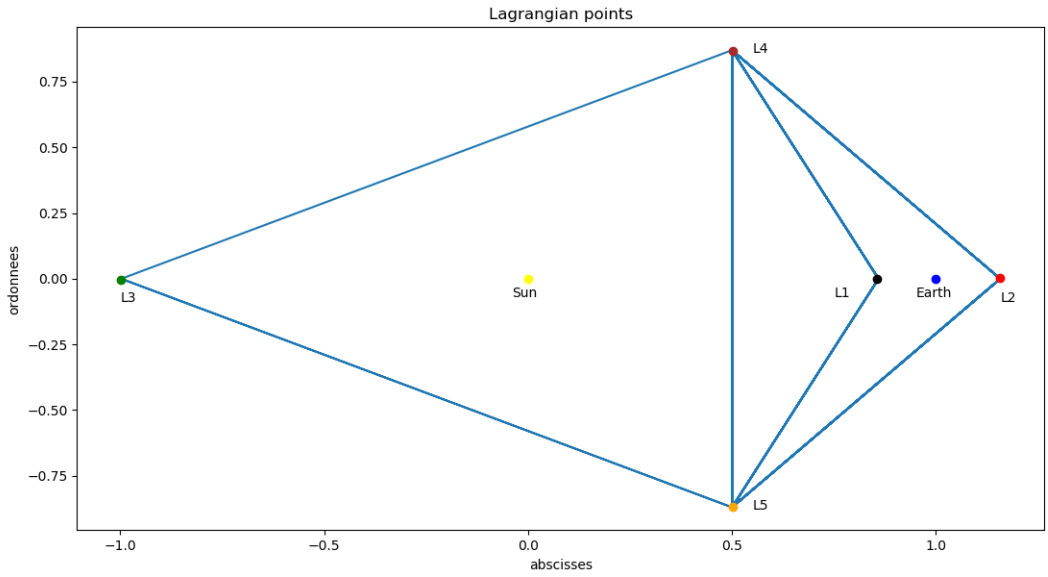

Figure 5: Lagrangian points

3 Electrostatic equilibrium

Part written by Yassine MOUGOU.

This section deals with the calculation of the total energy of a N-charge system, provided that each charge is in circulation over the interval $[-1, 1]$, and to find the equilibrium position for this system.

* The total energy of the system is given by (2):

$$E(x_1, ..., x_N) = \sum_{i=1}^{N} (log|x_i + 1| + log|x_i - 1| + \frac{1}{2} \sum_{j=1, j\neq i}^{N} log|x_i - x_j|) \tag{2}$$

where:

$$x_i : position\ of\ the\ charge\ i\ \forall i \in 1, 2, ..., N$$

$$E : Total\ energy$$

* Now let's move to our main aim which is the equilibrium position(maximum or minimum). Our method consists in calculating the gradient of the system's total energy and looking for a vector $U = (x_1, x_2, ..., x_N)$ that makes that gradient null.

- In order to calculate $\nabla E(x_1, x_2, ...x_N)$, the function ***gradient_elec_energy*** has been used. It takes a vector x as a parameter and returns a vector that represents $\nabla E(x)$ such as $x = (x_1, x_2, ...x_N)$:

$$\nabla E(x) = \sum_{j=1, j\neq i}^{N} \frac{1}{x_i - x_j} + \frac{1}{x_i - 1} + \frac{1}{x_i + 1} \tag{3}$$

After that, the function ***gradient_jacob*** was used and returns the Jacobian matrix:

$$\begin{cases} jacob_{i,j} & = & \sum_{j=1, j\neq i}^{N} \frac{-1}{(x_i - x_j)^2} + \frac{-1}{(x_i + 1)^2} + \frac{-1}{(x_i - 1)^2} & \forall i = j \\ jacob_{i,j} & = & \frac{1}{(x_i + x_j)^2} & \forall i \neq j \end{cases} \tag{4}$$

- Unfortunately, our goal has not yet been reached. The existence of an equilibrium position had to be proved. This is why the function ***Newton_ Raphson_ back_ modif*** was used. It takes the function f as a parameter, its Jacobian matrix J, the starting point $U0$, max iterations count N and finally the precision eps, and provides us with two lists ***iteration*** and ***norme_f***.

- This function is an ideal application for our problem. `f` and `J` need to be chosen by this way: f = ***gradient_ elec_ energy*** and J = ***gradient_ jacob*** and as it can be seen in Figure 6 below.

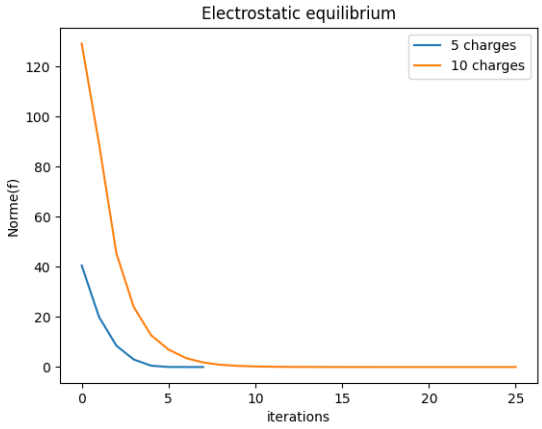

Figure 6: Changes of $||\nabla E(x)||$

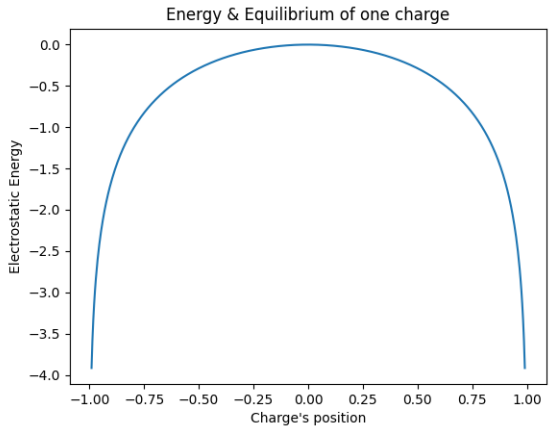

Figure 7: Changes of the energy

NB:

* There is an Electrostatic equilibrium because $||\nabla E(x)||$ converges to 0.

* For Figure 6 above, the Newton_Raphson method with the backtracking was used and even if the Newton_Raphson method was applied it would roughly lead to the same result.

** As it can be seen above, the existence of an equilibrium position was proved. Now it's time to check weather it is a maximum or minimum.

** According to Figure 7 above, the equilibrium position is a ***maximum*** reached when $x = x1 = 0$.

4 Bairstow method for polynomial factorization

Part written by Saad ABIDI.

To find the roots of a polynomial P the Newton & Raphson method is valid, but only if it has real roots, which is not always the case. For this the Bairstow method is called, which searches for quadratic factors. The advantage of looking for quadratic factors is that it avoids any complex arithmetic. The method consists in taking the polynomial in question and dividing it by a quadratic factor:

$$P(X) = (X^2 + BX + C) * Q(X) + RX + S \quad (B, C, R, S)\varepsilon\mathbb{R} \tag{5}$$

Given B and C, R and S can be readily found.Therefore it can be said that R and S are functions of B and C,and they are nil if the quadratic factor is a divisor of P(X). The aim was to find where (B,C) are simultaneously roots of R and S with the Newton & Raphson method.
In other words Newton & Raphson's method have to be applied on the function which takes as its parameter B C and returns R (B, C) and S (B, C). Then the Jacobian matrix had to be determined,it represents the matrix of the partial derivatives of R and S compared to B and C. This matrix is calculated from a second division of Q by the quadratic factor. By finding this ,the Newton&Raphson function could be applied to find where (R,S)=(0,0).
After that the roots of the quadratic factor were found, and the same operation was repeated for the

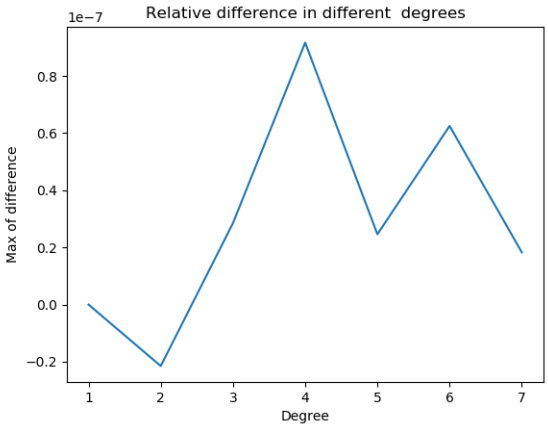

Figure 8: Relative difference in different degrees

polynomial Q until a second order polynomial was obtained.
To test our function the Bairstow method was applied to several polynomial generated randomly in order to see the difference between the roots of this polynomial obtained through our method and those provided by the np.roots function . Based on that a graph was generated to show the maximum relative difference in each degree between the two methods :
The graph 8 above shows that the difference between the two methods fluctuate around 10^{-7} and that's what makes the Bairstow method efficient.

4.1 Conclusion

Part written by Hugo PIERRE.

To conclude, this project leaded us to work in group and share the work. Moreover it make us discover the Newton-Raphson method which is efficient to find roots of non-linear equations system but one starting vector lead to one root so if there are many roots it demands to think the value of this vector for each root in order find them all. However this method is limited for many particular cases.