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

Groupe 1 - Equipe 2 - 12401

Responsible: ndo001
Secretary: fboitel
Coders: beboudeau, mpeyrat003, adsilva008

Abstract : The goal of this project was to program algorithms dedicated to the non-linear equation system root research. The promoted method here was the Newton-Raphson algorithm, and the goal was to evaluate the assets and liabilities of such a solution. This was done by testing the method in different settings.

1 Newton-Raphson method

This section deals with the programming of the Newton-Raphson method, in any dimension.

1.1 Newton-Raphson algorithm (Author: Nicolas DO)

Given an arbitrary point U_0, a function $f : \mathbb{R}^n \to \mathbb{R}^m$ that is differentiable along all dimensions and its Jacobian matrix H which are defined as follows:

$$f : \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \to \begin{bmatrix} f_1(x_1, \cdots, x_n) \\ \vdots f_m(x_1, \cdots, x_n) \end{bmatrix}. \quad (1) \qquad H : \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \to \begin{bmatrix} \frac{\partial f_1(x_1,\cdots,x_n)}{\partial x_1} & \cdots & \frac{\partial f_1(x_1,\cdots,x_n)}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m(x_1,\cdots,x_n)}{\partial x_1} & \cdots & \frac{\partial f_m(x_1,\cdots,x_n)}{\partial x_n} \end{bmatrix}. \quad (2)$$

The Newton-Raphson method provides an approximate root of f. To do so, the algorithm consists in following the direction given by the slope of f at a point U_n in order to move closer to a root of f. Therefore, from a point U_n, the next point U_{n+1} is chosen such that $U_{n+1} = U_n + V$, verifying the condition $f(U_{n+1}) = f(U_n + V) = 0$ where $f(U_n + V)$ is approximated by $f(U_n) + H(U_n) \times V$. From this condition, the vector V to add to U_n is determined such that $H(U_n) \times V = -f(U_n)$. This operation is repeated until either U_n converges, that is to say $||f(U_n)|| \leq \epsilon$ where ϵ measures the convergence of the algorithm or a maximal of N iterations is reached.

Thus, the Newton-Raphson function prototype may be defined as $Newton\_Raphson(f, H, U_0, N, \epsilon)$. The function returns a tuple with the latest value of U and a boolean corresponding to whether a root has been found or not. The complete algorithm can be seen in the code.

Tests were carried out for different initial points and various functions $f : \mathbb{R} \to \mathbb{R}$. Therefore, the Newton-Raphson function parameters were easy to compute and the outcomes of the latter were verifiable easily. In fact, in the case of a function $f : \mathbb{R} \to \mathbb{R}$, the Jacobian corresponds to the derivative f'.

Results have shown that a real root of a function f can be found by the Newton-Raphson method as expected. However, they have also shown that the Newton-Raphson method does not work in all cases. In fact, if the point initially given or if the algorithm processes a point which is a stationary point, that is to say, a point where the function's derivative is zero, then the next point will not be able to be computed because of the fact that $U_{n+1} = U_n + V$, with $V = \frac{-f(U_n)}{f'(U_n)}$. In addition, the Newton-Raphson method diverges with some specific functions such as $x \to x^{\frac{1}{3}}$ no matter the point initially given or $x \to x^3 - 5x$ with an initial point $x_0 = 1$ or $x_0 = -1$.

Some of Newton-Raphson method's weaknesses can be solved by including backtracking as described in the section below.

1.2 Newton-Raphson with backtracking (Author: Aloïs DA SILVA)

The divergence issues mentioned above can be corrected by adding backtracking. This process consists of checking whether the points get closer to the desired root or not each time the point is moved. Despite the efficacy of the Newton-Raphson algorithm, if the point obtained is further from the root desired the distance between a point and the next one is divided by two. This action is repeated until the point finally gets closer to the root or until the algorithm hits N occurrences. If the algorithm stops because of the latter, it indicates it is not possible to find the root with this method.

2 Lagrangian point (Author: Benjamin BOUDEAU)

The aim of the work described in this section was to find the Lagrangian points. The system is composed of two objects (here called the two masses) applying a gravitational force on a third object itself subject to a centripetal force. The Lagrangian points are the positions of the third object where the forces experienced by it are so that this object follows the orbital movement of the two masses. In other words, this third object is stationary in the reference frame of the two masses during their orbital movement. [1]

2.1 Force functions (Author: Benjamin BOUDEAU)

Mathematically, Lagrangian points are equilibrium positions where the sum of the two gravitational forces and the centripetal force is null. To evaluate these points the Newton-Raphson function implemented in the previous part was used. It requires two functions:

- the function f which Newton-Raphson evaluates the points U where $f(U) = 0$.
- the function Jf representing the Jacobian of f.

Here are the formulas of the gravitational and the centripetal function applied to the point (x,y):

$$f_g = \begin{bmatrix} x \\ y \end{bmatrix} \to \begin{bmatrix} -k.\frac{x-x_0}{((x-x_0)^2+(y-y_0)^2)^{3/2}} \\ -k.\frac{y-y_0}{((x-x_0)^2+(y-y_0)^2)^{3/2}} \end{bmatrix}. \quad (3) \qquad f_c = \begin{bmatrix} x \\ y \end{bmatrix} \to \begin{bmatrix} k(x-x_0) \\ k(y-y_0) \end{bmatrix}. \quad (4)$$

Equation (3) is the formula of the gravitational force and (4) is the formula of the centripetal force. In both, k, x_0 and y_0 are parameters. In Python these functions was implemented with two separated functions:

- The first takes the parameters, a point $U = (x, y$ and evaluates the force at this point.
- The second only takes the parameters and returns a function that requires a point $U = (x, y)$ and returns the result of the first function with specified parameters.

This method is close to a specialization in functional programming.

The Jacobian of these two functions was calculated:

$$Jf_g = \begin{bmatrix} x \\ y \end{bmatrix} \to \begin{bmatrix} J1 & J2 \\ J2 & J3 \end{bmatrix}. \quad (5) \qquad J_c = \begin{bmatrix} x \\ y \end{bmatrix} \to \begin{bmatrix} k & 0 \\ 0 & k \end{bmatrix}. \quad (6)$$

with,

$$J1 = -k \times [((x-x_0)^2 + (y-y_0)^2)^{-3/2} - 3 \times (x-x_0)^2((x-x_0)^2 + (y-y_0)^2)^{-5/2}],$$
$$J2 = 3k \times (x-x_0)(y-y_0)((x-x_0)^2 + (y-y_0)^2)^{-5/2},$$
$$J3 = -k \times [((x-x_0)^2 + (y-y_0)^2)^{-3/2} - 3 \times (y-y_0)^2((x-x_0)^2 + (y-y_0)^2)^{-5/2}].$$

To represent these Jacobian functions, the same functional programming method was used as the previous functions, namely, the specialization for the parameters k, x_0 and y_0.

2.2 Functions representing the problem (Author: Benjamin BOUDEAU)

As stated in the introduction of this part the third object is subject to three forces, and not just one of them. To represent this addition of forces two other functions was implemented. The first was the function that returns the sum of the two gravitational forces and the centripetal force at the point $U = (x, y)$ in parameters. It was just the sum of the functions previously implemented. The second was the Jacobian of this function's sum. It was the sum of the Jacobians representing the three forces.

To verify, the subject gives an example of result for the function sum and its Jacobian at the point (1.5, 0). The implemented functions found the same result as the example so it seems to be working correctly.

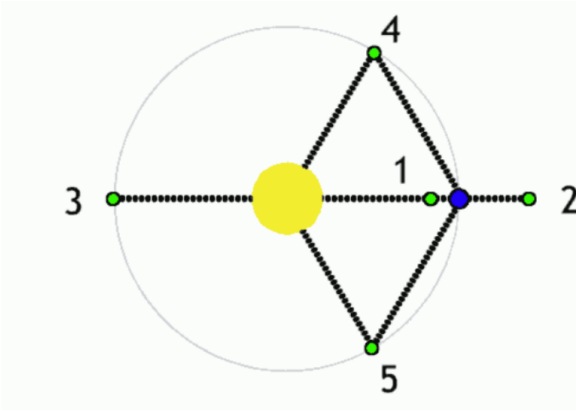
2.3 Obtaining Lagrangian points (Author: Faustin BOITEL)

All the necessary tools are now gathered to find the Lagrangian points. As explained in the previous paragraphs, at a Lagrangian point L, the different forces are governed by equation (7),

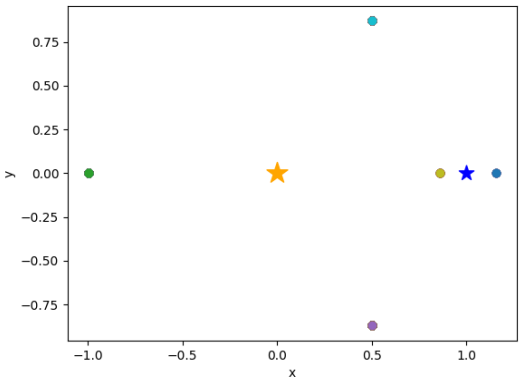
$$F_{grav1}(L) + F_{grav2}(L) + F_{centripetal}(L) = 0, \quad (7)$$

considering that the gravitational force exercised by body 1 (resp. body 2) at point L is represented by $F_{grav1}(L)$ (resp. $F_{grav2}(L)$) and the centripetal force at point L represented by $F_{centripetal}(L)$

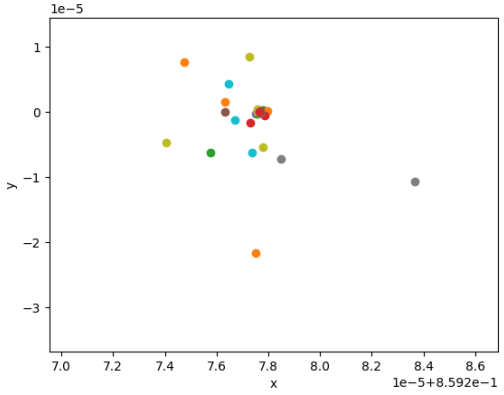
The Newton-Raphson algorithm is ideal to find the Lagrangian points, in fact, the equation (7) is a non-linear system of equations in 2-dimension, and the expressions of the forces and their jacobian are perfectly known. However, for some initial conditions, the algorithm returns one single solution, but there are 5 here, as shown on the reference Figure 1a (The Figure 1a is extracted from Wikipedia [1], it show the Lagrangian points for two bodies. We used such a figure to help us in the visualization of the points, and therefore save time to make our experiments, as explained below). Therefore, considering that the algorithm returned the closest solution to the initial point, we chose to use the algorithm repeatedly, sweeping the whole area for initial conditions, thus we used the algorithm on (x_i, y_i) for $i \in [\![0\ ,\ N]\!]$ such that $x_i = -\frac{h}{2} + \frac{(h*i)}{N}$ (where h is the width of the chosen area and N the number of points, we will explain later how we chose these), and similarly for y_i . After computing the Newton-Raphson algorithm for all of these points, 5 points of convergence emerged, as shown in



(a) Reference for the Lagrangian points (source: Wikipedia[1])



(b) Highlighting of the Lagrangian points



(c) A zoom in a convergence area

Figure 1: Determination of Lagrangian points

Figure 1b. The Figure 1c is a zoom of one of the 5 points: it shows the good precision of the experiment, because for one area of convergence, all the points are included is a $(1e-5)$x$(1e-5)$ square.

For this approach, the following parameters had to be chosen:

- The swept surface upon which the algorithm was applied. In order to decide upon this, we based ourselves on the reference Figure 1a given by Wikipedia and determined a large zone within which the Lagrangian points would be for sure. Here, for a distance of 1 unity between the two bodies, we were sure that the Lagrangian points would have not been out of a 3x3 sized square centered on the most massive body.
- The sampling step. After consulting again the reference Figure 1a, we noticed that two Lagrangian points were never closer than 0.5 unity, for these reasons, a step of 0.1 was chosen.

- The *epsilon* and the number of iterations which set the Newton-Raphson algorithm. An *epsilon* of 0.0001 was chosen for this experiment, the characteristic distance of the figure being 1 (distance between the two bodies) . In fact, it was 4 magnitudes behind the characteristic distance, and with an even smaller *epsilon*, the execution became too time-consuming. The number of iteration was set at 1000, in fact experimentally, with even more iterations, the result was not enhanced.

Finally, to obtain the coordinate of each Lagrangian points, the average of each experimental point in the different convergence zones should be done. In order to have an even more precise result, we could simply increase the value of epsilon and reduce the step.

2.4 Optimization (Author: Benjamin BOUDEAU)

Finding Lagrangian points with the previous method can be very expensive in terms of complexity due to the number of tested points. In order to reduce this complexity a more optimized function was implemented. Tested points can be seen in Figure 2. It was verified with the previous example and the optimized function found the same Lagrangian points. However, if the number of tested points is lower, the precision can decrease according to Subsection 2.3.

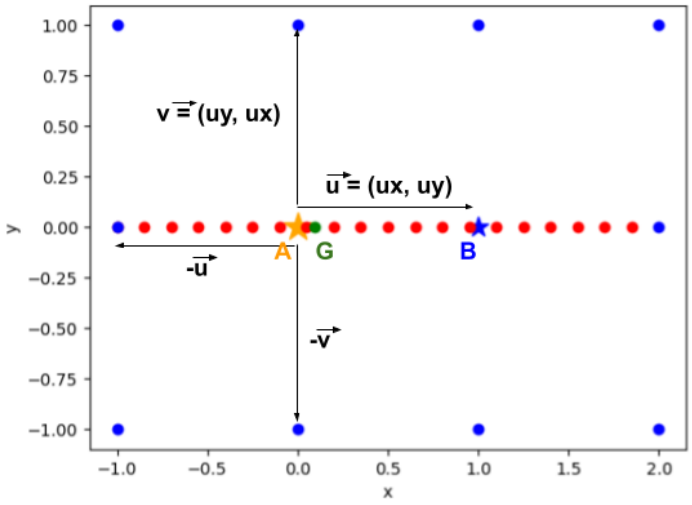


Figure 2: Tested points in the optimized function

Caption:

- Blue points are the always tested points.
- Green point is the barycenter of the two masses.
- Red points are points obtained by dividing the segment from $B - \vec{u}$ to $B + 2\vec{u}$ by n. In other words, n is a parameter of the function which makes the user choose the number of points he wants to test on the segment from $B - \vec{u}$ to $B + 2\vec{u}$.

3 Bairstow's method for polynomial factorization

Introduction (Author: Mélanie PEYRAT)

The Bairstow method specification is to factorize a polynomial into a multitude of polynomials of second degree in order to find the roots of said polynomial easily. Bairstow method utilizes an iterative method in order to get closer to a real polynomial. In this part, Bairstow's algorithm was explained first, the results on the different tests were discussed second, and finally we concluded.

Algorithm construction (Author: Mélanie PEYRAT)

The website [2] indicates how to apply the algorithm of Bairstow. A particularity of Bairstow's method is the domain of definition of this polynomial. It is possible to use $\mathbb{R}$ for polynomial's domain of definition in order to find a real root but also to find a complex root.

G's polynomial expression was

$$\sum_{i=0}^{n} a_{n-i}x^i. \tag{8}$$

The polynomial G could be written as $G(x) = \alpha(x) * Q(x) + Rx + S$ with $\alpha(x) = x^2 + Bx + C$. $Q(x)$(respectively $Rx + S$) was the Euclidean division's quotient (respectively remainder) of $G(x)$ by $\alpha(x)$.
$Q(x)$'s expression was

$$Q(x) = \sum_{i=0}^{n-2} b_{n-i-2}x^i. \tag{9}$$

The problem was solved by finding B and C such that

$$R(B, C) = 0, \qquad S(B, C) = 0. \tag{10}$$

A solution could be obtained by finding corrections ΔB and ΔC on a random number B and C. Nevertheless, in our program B and C was fixed with B = 1 and C = 2.
With the corrections ΔB and ΔC on a random number B and C, Equation 10 could be written like Equation 11 and 12.

$$R(B + \Delta B, C + \Delta C) = 0, \tag{11}$$

$$S(B + \Delta B, C + \Delta C) = 0. \tag{12}$$

With Taylor's expansion the problem on R became (and it is the same for S)

$$0 = R(B, C) + \frac{\partial R}{\partial B}\Delta B + \frac{\partial R}{\partial C}\Delta C. \tag{13}$$

The value of R and S could be obtained with the link between G and Q

$$a_0x^n + a_1x^{n-1} + ... + a_n = (x^2 + Bx + C)(b_0x^{n-2} + b_1x^{n-3} + ... + b_{n-2}) + Rx + S. \tag{14}$$

The value of b was obtained by identification

$$a_0 = b_0, \qquad a_1 = b_1 + pb_0, \qquad a_k = b_k + Bb_{k-1} + Cb_{k-2}. \tag{15}$$

By adding four variables, b's terms could be identified with the same method

$$b_{n-1} = R, \qquad b_n + Bb_{n-1} = S, \qquad b_{-2} = b_{-1} = 0, \tag{16}$$

$$b_k = a_k - Bb_{k-1} - Cb_{k-2}, \forall i \in [|0; n|] \tag{17}$$

So, the new expression of Equation 11 and 12 with Equation 13 by replacing R and S with the new expression given by Equation 3 was

$$0 = b_{n-1} + \frac{\partial b_{n-1}}{\partial B}\Delta B + \frac{\partial b_{n-1}}{\partial C}\Delta C, \tag{18}$$

$$0 = b_n + \left(\frac{\partial b_n}{\partial B} + b_{n-1}\right)\Delta B + \frac{\partial b_n}{\partial C}\Delta C. \tag{19}$$

b_k's partial differential was calculated with the third part of Equation 3

$$\frac{\partial b_k}{\partial B} = \frac{\partial a_k}{\partial B} - \frac{\partial B b_{k-1}}{\partial B} - \frac{\partial C b_{k-2}}{\partial B} \implies \frac{\partial b_k}{\partial B} = -b_{k-1} - B\frac{\partial b_{k-1}}{\partial B} - C\frac{\partial b_{k-2}}{\partial B}.$$

Nevertheless ,

$$\frac{\partial b_0}{\partial B} = \frac{\partial b_{-1}}{\partial B} = \frac{\partial b_{-2}}{\partial B} = 0.$$

A new expression of $\frac{\partial b_k}{\partial p}$ and $\frac{\partial b_{k+1}}{\partial q}$ with recurrence relation on b is

$$\frac{\partial b_k}{\partial B} = -b_{k-1} - B\frac{\partial b_{k-1}}{\partial B} - C\frac{\partial b_{k-2}}{\partial B}, \tag{20}$$

$$\frac{\partial b_{k+1}}{\partial C} = -b_{k-1} - B\frac{\partial b_k}{\partial C} - C\frac{\partial b_{k-1}}{\partial C}. \tag{21}$$

Equation 22 is obtained by replacing b_{k-1} in Equation 21:

$$\frac{\partial b_k}{\partial B} + B\frac{\partial b_{k-1}}{\partial B} + C\frac{\partial b_{k-2}}{\partial B} = \frac{\partial b_{k+1}}{\partial C} + B\frac{\partial b_k}{\partial C} + C\frac{\partial b_{k-1}}{\partial C}. \tag{22}$$

A new relation between $\frac{\partial b_k}{\partial B}$ and $\frac{\partial b_{k+1}}{\partial C}$ with Equation 22 was obtained. A new variable c_k was introduced; it was the coefficient of polynomial $Q_2(x)$. $Q_2(x)$ was the Euclidean division's quotient of $Q(x)$ by $\alpha(x)$.
The relation between c_k and b_k could be obtained with the same method as equation 3, then

$$c_k = b_k - Bc_{k-1} - Cc_{k-2}. \forall k \in [|0; n|] \tag{23}$$

Consequently

$$\frac{\partial b_k}{\partial C} = \frac{\partial b_{k-1}}{\partial B} = c_{k-2}, \tag{24}$$

$$c_{-1} = c_{-2} = 0. \tag{25}$$

The solution was obtained by replacing $\frac{\partial b_k}{\partial B}$ $\frac{\partial b_k}{\partial C}$ in equation 11 and 12

$$\begin{bmatrix} c_{n-2} & c_{n-3} \\ c_{n-1} - b_{n-1} & c_{n-2} \end{bmatrix} \begin{bmatrix} \Delta B \\ \Delta C \end{bmatrix} = \begin{bmatrix} b_{n-1} \\ b_n \end{bmatrix}. \tag{26}$$

Finally, ΔB and ΔC were obtained by solving the linear system in equation 26. The polynomial $\alpha(x)$ changed after each iteration in our algorithm with the new value of B and C. The new values were the result of the addition of B (respectively C) and ΔB (respectively ΔC). The algorithm was stopped when R and S were less than the precision h. Two roots of G were found by finding the roots of polynomial α by calculating the discriminant. The rest of G's roots was found by recursively applying the algorithm on Q until Q's degree was less than 3.

3.1 Tests (Author: Aloïs DA SILVA)

When it came to testing Bairstow's method, proceeding gradually by using polynomials whose roots were known first seemed to be the best option. These tests worked perfectly except when roots with a multiplicity ≥ 2 were faced. As a matter of fact, it might be a consequence of approximations. For instance, a polynomial of degree 3 led to complex roots when real roots were expected. Then, the tests were extended to a more global test by using random libraries and a seed, creating a polynomial of degree *degree* with random integer coefficients. Later, the implemented function was applied to it and the roots were compared to the *numpy.root* function to check whether they were close enough to what should have been obtained. Despite that, the rest obtained with the polynomial division sometimes increased without apparent reasons and the approximations were suspected for being responsible for such a discrepancy between what should have been obtained and what is actually obtained.

4 Analysis and conclusion (Author: Aloïs DA SILVA)

Many issues were raised when it came to implement the functions. First, the issue of representing a function $f : \mathbb{R}^n \to \mathbb{R}^m$ or of representing a Jacobian matrix was faced and was solved by using arrays.
Using the Newton-Raphson's method fostered simplicity as its implementation was easy and it only needed a function and the Jacobian. In addition, finding roots raised issues and the implementation of a global function finding roots avoided the implementation of case by case functions treating each issue in an isolated manner.

References

[1] Wikipedia, "Lagrange point." `https://en.wikipedia.org/wiki/Lagrange_point`. viewed in 04/2021.

[2] unknown, "Bairstow's method." `https://www.cbpbu.ac.in/userfiles/file/2020/STUDY_MAT/PHYSICS/NP%203.pdf`. viewed in 04/2021.