Retour à IS104 – Algorithmique Numérique (http://mfaverge.vvv.enseirb-matmeca.fr/wordpress/?page_id=159)

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

The goal of this project is to program algorithms dedicated to the research of roots of systems of non-linear equations. The method promoted here is the Newton-Raphson algorithm, and the goal is to evaluate the assets and liabilities of such a solution. This shall be done by testing the method in different settings. In the following, you will be proposed a list of applications where this algorithm is necessary. You must program at least 2 of these applications, and then write a summary of your experiments as a conclusion.

This project covers a number of algorithms and programming techniques that are not directly addressed in the course. We advise you to be autonomous by making the most of your own knowledge and the documents available. The teacher will not answer the questions that he feels you can answer on your own.

Newton-Raphson method

This section deals with the programming of the Newton-Raphson method, in any dimension. Given a function $f : \mathbb{R}^n -> \mathbb{R}^m$ that is differentiable along both dimensions, it is possible to write f and its derivatives in the following way :

$$f : \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \rightarrow \begin{bmatrix} f_1(x_1, \ldots, x_n) \\ \vdots \\ f_m(x_1, \ldots, x_n) \end{bmatrix}$$

$$H : \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \rightarrow \begin{bmatrix} \frac{\partial f_i(x_1, \ldots, x_n)}{\partial x_j} \end{bmatrix}$$

The H function is called the Jacobian matrix of f. The idea of the Newton-Raphson algorithm consists in considering that, given a point X, the best direction leading to a root of the function is the direction given by the slope of f at X. In following this direction, the algorithm assumes that it moves closer to a root of f.

Given a vector U representing the current position, the algorithm computes $f(U)$, a vector representing the

value of f at U, as well as $H(U)$ the Jacobian matrix of the derivatives of f at U. Now, the next position U + V is chosen such that :

$$f(U+V)=0 \quad \text{where } f(U+V) \text{ is approximated by } f(U)+\underbrace{H(U)}_{\text{matrix}}\times\underbrace{V}_{\text{vector}}$$

Then, U is replaced by $U+V$, and this operation is repeated until either U converges or a maximal number of iterations is reached.

1. Write the (very simple) equation linking H, U and V.
1. Write the Newton-Raphson method in a generic way :

   ```
   function U = Newton_Raphson(f, J, U0, N, epsilon)
   ```

   … where `f` is the function under study, `J` is its Jacobian matrix, `U0` is the starting position of the algorithm, `N` is the maximal number of steps allowed during the algorithm and `epsilon` measures the convergence of the algorithm.

   Remarks :

   - One requirement is to be able to compute the values taken by the function f and all its derivatives in a priori any point of the plane. Therefore, it is necessary to specify f and its derivatives in the form of functions. Python allows programming using functional parameters. These parameters may be provided by functions defined either with the keyword `lambda`, or with simple `def` definitions.
   - The algorithm requires the resolution of several linear systems with matrices M that are possibly singular (i.e $det(M) \approx 0$. Prefer the function `numpy.linalg.lstsq` in order to avoid such numerical problems.
   - One of the drawbacks of this algorithm is its tendency to diverge in numerous cases. It is therefore imperative to limit the number of steps taken, and to detect to what extent the algorithm has converged.
2. Propose a very simple test protocol for a function $f : \mathbb{R} \longrightarrow \mathbb{R}$.
3. Enhance your implementation to include backtracking inside your Newton-Raphson implementation.

Computation of the Lagrangian points

Consider an object moving in the plane and subject to a set of forces. Our goal in this section consists in determining the equilibrium positions of this object.

1. Write the code to represent the following kinds of forces :
   - an elastic force (http://en.wikipedia.org/wiki/Force#Elastic_force) (with zero natural length)
   - a centrifugal force (http://en.wikipedia.org/wiki/Centrifugal_force)

     $$f_c : \begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} k(x-x_0) \\ k(y-y_0) \end{bmatrix}$$
   - a gravitational force (http://en.wikipedia.org/wiki/Newton%27s_law_of_universal_gravitation)

$$f_g : \begin{bmatrix} x \\ y \end{bmatrix} \to \begin{bmatrix} -k \cdot \dfrac{x - x_0}{((x-x_0)^2 + (y-y_0)^2)^{3/2}} \\ -k \cdot \dfrac{y - y_0}{((x-x_0)^2 + (y-y_0)^2)^{3/2}} \end{bmatrix}$$

Remarks :

- Is is strongly recommended to use functional programming techniques to represent the forces and their Jacobian matrices.
- For each of these cases, it should be possible to parameterize the force by a constant k symbolizing its intensity, as well as a central point (x_0, y_0) from which the force is issued.

1. Use the Newton-Raphson method to obtain the equilibrium points in the following case :
   - Two gravitational forces with respective coefficients 1 (resp. 0.01) originating from $[0,0]$ (resp. $[1,0]$);
   - And a centrifugal force centered on the barycenter of the two masses, with coefficient 1.

   As a means of verification, and with the conditions above, at $U = [1.5, 0]$, one gets the following values for the total force and its Jacobian :

$$f(U) = \begin{bmatrix} 1.00565457 \\ 0 \end{bmatrix} \text{ and}$$

$$df(U) = \begin{bmatrix} 1.75259259 & 0. \\ 0. & 0.6237037 \end{bmatrix}$$

---

1. Is it possible to obtain the Lagrangian points (http://en.wikipedia.org/wiki/Lagrangian_point) ?

Bairstow method for polynomial factorization

Consider the problem of finding the roots of a polynomial P. When the roots of P are real, the Newton-Raphson method can be applied to find the roots. Nevertheless, this method is unable to find the complex roots of P. The Bairstow method is a method computing the irreducible factors of degree 2 of a real polynomial. Incidentally, it avoids the computations with complex numbers. Technically, it simply consists in applying the Newton-Raphson method to a specific function.

The Bairstow algorithm is described on page 377 of the Numerical Recipes available at this location (http://www.nrbook.com/a/bookcpdf/c9-5.pdf) (chap. 9.5 pp. 369–379).

1. Describe (literally) the function whose zero is computed in dimension 2. In particular, define its domain of definition and its range, and explain the role of the variables B, C, R and S. What are the zeroes of this function ?

---

2. Write the code of this function by reusing the function `numpy.polydiv` (which computes simultaneously the quotient and the remainder of the division). In order to test this function, it suffices to construct the adequate polynomial of the form $Q(X) * (X^2 + BX + C) + RX + S$ and to apply the function.

1. In the Numerical Recipes, find the 4 partial derivatives of the previous function. Write the equations defining these derivatives, and program them.
1. Write the Bairstow algorithm, and test it using a method of your choice.

Electrostatic equilibrium

Let us consider the interval $[-1,1]$ and two electrostatic charges fixed at the positions -1 and 1. We assume that there exists N charges positioned at $x_1, x_2, \cdots, x_N$, and that these charges can move freely in the interval $[-1,1]$. The total electrostatic energy of this system is equal to :

$$E(x_1, x_2, ...x_N) = \sum_{i=1}^{N} \log|x_i + 1| + \log|x_i - 1| + \frac{1}{2}\sum_{j=1, j\neq i}^{N} \log|x_i - x_j|$$

The equilibrium positions are found by minimizing or maximizing this energy. In order to do this, it is necessary to solve a non-linear system of equations that is equal to :

$$\nabla E(x_1, x_2, \cdots, x_N) = \left[\frac{\partial E(x_1,...,x_N)}{\partial x_i} \right] = 0$$

Notice that $\nabla E(x_1, x_2, \cdots, x_N)$ is a vector with N coordinates.

1. Compute the Jacobian of $\nabla E(x_1, x_2, \cdots, x_N)$.
1. Use Newton's method in order to solve this equation. Plot the points and the real axis.
   Do these solutions resemble to the roots of the derivative of the Legendre polynomials (http://en.wikipedia.org/wiki/Legendre_polynomials) ? (cf. `numpy.polynomial.legendre`)
1. Does the solution correspond to a maximum or a minimum of the energy ?

The wave equation

When considering the propagation of waves on shallow water surfaces, a simple model is the KdV (Korteweg de Vries) equation :

$$\zeta_t + \zeta_x + \frac{3\varepsilon}{2}\zeta\zeta_x + \frac{\varepsilon}{6}\zeta_{xxx} = 0$$

where ζ_x denotes the derivative of ζ with respect to X. $\zeta(x,t)$ is the « scaled » elevation of the surface of the wave and is a function of the time t and the space X. The true elevation is equal to $h \times (1 + \varepsilon\zeta)$, the parameter ε will be chosen less than 1 (e.g. 0.1). We will reduce the computations on the interval $[-L, L]$ and impose periodic boundary conditions.

Let U be a vector of size N, discretizing ζ on a regular subdivision $x_k = \frac{2k-N+1}{N-1}L$ of the interval $[-L, L]$ ($U_k = \zeta(x_k, t)$). We are going to compute a sequence of vector U^n representing the temporal evolution of the wave. The initial condition will be set to a gaussian function : $\zeta(x, 0) = \exp(-\alpha x^2)$ with $\alpha > 0$. As for the heat equation, a finite difference scheme is used to solve this equation. In the following, we describe an algorithm to compute U^{n+1} as a function of U^n.

Periodic boundary conditions induce the following conditions :

$$\begin{cases} U_{-1} = U_{N-1} \\ U_{-2} = U_{N-2} \\ U_N = U_0 \\ U_{N+1} = U_1 \end{cases}$$

We denote G the application which transforms the vector U into the vector :

$$[G(U)]_i = \frac{U_{i+1}-U_{i-1}}{2\Delta x} + \frac{\varepsilon}{4\Delta x}(U_{i+1} - U_{i-1})(U_{i-1} + U_i + U_{i+1}) + \frac{\varepsilon}{12\Delta x^3}(U_{i+2} - 2U_{i+1} + 2U_{i-1} - U_{i-2})$$

This application is non-linear and can not be represented as a matrix. We have the following evolution system : $\frac{dU}{dt} + G(U) = 0$. In order to solve this evolution equation, we consider the following equation :

$$\frac{U^{n+1}-U^n}{\Delta t} + G\left(\frac{U^n+U^{n+1}}{2}\right) = 0$$

---

1. Compute and draw U^0.
2. Write the function whose roots you must find. Compute the Jacobian matrix of this function.
3. Compute U^{n+1} as a function of U^n with the Newton-Raphson method. Draw the results. As an example, it should be possible to generate a movie like this one (wp-content/cours/IS104/files /kdv_movie.mpg).

Analysis and conclusion

Conclude your experimentations by an analysis of the problems encountered and the benefits of using an algorithm such as the Newton-Raphson.

The comparison shall be based on the results investigated in the course, and possibly the reading of the chapter 9 of the Numerical Recipes, available at this location (http://www.nrbook.com/a/bookcpdf.html) (chap. 9, Root Finding and Nonlinear Sets of Equations).

Dans cette section

Construit avec ♥ par Thèmes Graphene (https://www.graphene-theme.com/).