Numerical Methods

Davide Marchesi

March 9, 2022

Contents

Purpose

This Document will be a theoretical explanation of the numerical methods that could be used to solve complex mathematical problems using a virtual machine.

The purpose is to give, for everyone who has a sufficient knowledge of both mathematics and *MATLAB* language, the means to make numerical analysis of practical physical problems.

All the informations in this document are both learned following the "*Engineering' Numerical and Analitical Methods*" course at *Politecnico di Milano*, and studied in deep by myself.

For all the practical code examples please refer to my personal account on *GitHub*:
`https://github.com/davidemarchesi/NumericalMethods`

0 Introduction

The Numerical analysis is a modern approach to all that Physical/Mathematical problems which don't have an analytic resolution, or at least it results to be too complex to be solved in that way.
Take as example a simply temperature distribution problem in a complex space, the equation to resolve will be:

$$\frac{\partial T}{\partial t} - \mu \Delta T = f + \text{B.C.} + \text{I.C.}$$

This equation in fact doesn't have generally a resolution, and so we could use these numerical analysis to solve it.

More in general when facing a Physical problem, first we have to find a mathematical model to describe it, and this will be characterized by an error of the model e_M, and then to solve this mathematical problem.
Here comes into play the Numerical Analysis method, which has the aim to solve in the best possible way, with the lower possible values of the error given by the analysis itself e_N , these models.

1 Finite Arithmetic Fundamentals

A basical as important concept to understand before starting using the Numerical Methods on calculators is that Mathematics as we know it, doesn't exist in the computer world.
While in a theoretical way is possible for us to use and represent irrational numbers ($\sqrt{2}$ or π , for example), this is not possible for computers.
In fact every virtual language will be characterized by a finite number of relevant digits, and so by a minimum and a maximum number which is representable with them (for ex. in MATLAB they rispectively are $x_{min} = 2.23 \cdot 10^{-308}$ and $x_{max} = 1.80 \cdot 10^{308}$).
So in this virtual world we have abandoned the classical set of *Real Numbers* and we are now working with what is called the *Floating Numbers* set:

Theoretical/Physical problems $\rightarrow x \in \mathbb{R}$
Numerical models $\rightarrow x \in \mathbb{F}$

Now lets take a look on the properties and on the representation that the floating numbers have:

Df. Floating Point Rappresentation

$$\begin{aligned} \text{if } y \in (F) \rightarrow y &= (-1)^S \cdot (0.a_1 a_2 ... a_t) \cdot \beta^e \\ &= (-1)^S \cdot (a_1 a_2 ... a_t) \cdot \beta^{e-t} \end{aligned}$$

Where:

$$\begin{aligned} &s = \text{sign} \rightarrow s = 0, 1 \\ &\beta = \text{base} \rightarrow \beta \geq 2 \\ &m = a_1 a_2 ... a_t \rightarrow m = \text{mantissa} \end{aligned}$$

and:

$$0 \leq a_1 \leq \beta - i \qquad\qquad a_i \neq 0$$

Here an example of this rappresentation:
if $x = 0.01235$, then its floating will be $\text{fl}(x) = (-1)^0(1235) \cdot 10^{-6}$
$\rightarrow s = 0, \beta = 10, e = -2, t = 4$

So more in general a Floating Space $\mathbb{F}$ it's charachterized by:
β, which is the base on which we are working;
t, which is the number of relevant digits;
L, the minimum exponent;
U, the maximum exponent;

(in *MATLAB* case, $\mathbb{F}_{MATLAB}(2, 53, -1024, 1024)$)

Df. Array Error
The array error is the error given by a trasposion of a number from its real representation to its floating representation. It's defined as:

$$\frac{|x - \text{fl}(x)|}{|x|} \leq \frac{1}{2}\varepsilon_M \rightarrow \varepsilon_M = \beta^{1-t}$$

It's immediate to understand in this way that the lower the approximation is, the lower the error is (as i take more relevant digits in my calculator).

NB. in $\mathbb{F}$ are _not valid all the properties of $\mathbb{R}$
The commutative property is still valid:

$$\text{fl}(x + y) = \text{fl}(y + x)$$
$$\text{fl}(x \cdot y) = \text{fl}(y \cdot x)$$

But are no more valid: **associative** and **distributive** properties (in fact the results changes in base of how the formula is written, this due to the fact that i have different errors).

Now imagine that facing a Physical problem, we have yet built the Mathematical model which describes it, and we need a Numerical Method to have a quantitative solution;
the question is: how can we understand that our Numerical Method is a good one, or a bad one?
To individuate a good numerical method, we have to control that it has the following properties:

1. Convergence
   Given a mathematical problem $F(x, d)$, (where x is the analitical solution, and d the set of data), and its approximation $F_h(x_h, d_h)$, if $\lim_{h\to 0} x_h = x$, it has the property the property of convergence.

2. Consistency
   If the limit of the approximate function, for $h \to 0$, used on analitical data, tends to 0:

$$\lim_{h \to 0} F_h(x, d) = 0$$

   This, is called consistency.

3. Stability
   If the data are changed a little, also the results are changed only a little, which means:

$$F_h(\overline{x_h}, d_h + \varepsilon) = 0$$
$$F_h(x_h, d_h) = 0$$
$$\text{if } \varepsilon \to 0 \text{ so } (\overline{x_h} - x_h) \to 0$$

(**NB.** in the numerical calculus if consistency and stability are verified, so it's convergence.)
In the end of all, after solving our problem in a numerical way, we will have a final solution which will have intrinsically an error e_N given by two different types of sub-errors:
e_T= the truncation error, given from a theoretical point of view (like for example the Taylor approximation);
e_A= the approximation error, given intrinsically by the calculator, for all the motivations that are explained at the beginning of this chapter (acceptable **only** if there is stability).

2 Non Linear Equations

The general solving process for *non linear equations* can be summarized in finding, if it exists, the 'zero' of a generic function $f(x)$.
In order to do this, some generic methods were developed, and i will discuss two of them: the *Bisection Method* and *Newton's Method*.

2.1 Bisection Method

The *Bisection Method* exploits the **zeros theorem**, so the algorithm will be build starting by simply it:

$$\text{"If } f \in \mathbb{C}^0([a,b]) \text{ and } f(a) \cdot f(b) < 0 \text{ so } \exists \alpha \text{ s.t. } f(\alpha) = 0\text{"}$$

So if this hypothesis are verified, an algorithm can be build in this way:
0.I set a desired tolerance for the resolution: tol $< \epsilon$ (or either a maximum number of iterations M which I am disposed to do);

1. I take a value $x^{(k)}$ in the interval $[a,b]$;

$\rightarrow$ is $f(x^{(k)})$ <tol ?

2.**yes** If the answer is **yes**, the resolution stops here, and $\alpha = x^{(k)}$;

2.**no** If the answer is **no**, I'll take a value $x^{(k+1)} \in [a, x^{(k)}]$, or either $x^{(k+1)} \in [x^{(k)}, b]$, and restart the algorithm from the point 1 until I don't have a value that fits my initial conditions.

Watching the algorithm, how the intervals in point 2 are selected?
Simply applying the zeros theorem: the value of $x^{(k)}$ is substituted to b for example, and then if the expression $f(a) \cdot f(x^{(k)}) < 0$ is verified, the zero will be into the interval $[a, x^{(k)}]$, otherwise it will be into the other one $[x^{(k)}, b]$.
Moreover, as from the name itself of the method suggest, to find every $x^{(k)}$ will be taken using the *bisection formula*:

$$x^{(k)} = \frac{a^{(k)} + b^{(k)}}{2}$$

(where $a^{(k)}$ and $b^{(k)}$ surely represent the extrems of the interval in the k-esim interaction)
At the end of all I'll have: $\lim_{k \to \infty} x^{(k)} = \alpha$

Weaknesses and Highlights of the model

One of the Weaknesses of the problem is surely that it individuates only a zero, the one included in the interval set, and not all the zeros of a function.
Than there is also the fact that the method converges to a solution only with zeros with *odd* multiplicity, and not to the one with *even* multiplicity, where the *zeros theorem* is not applicable ($f(a) \cdot f(b) > 0$).
Last weakness is the fact that the convergence is _not monotonous. Observe that the distance between the solution and the $x^{(k)}$ value is expressible by: $e^{(k)} = |x^{(k)} - \alpha|$.
So we do not have any certainty that $e^{(k+1)} < e^{(k)}$.
To make an example of this, let's take a function with $\alpha = 4$, if the starting interval taken is $[0, 10]$, after one iteration we will have $x^{(0)} = 5$ and $e^{(0)} = 1$;
But then with a second iteration we will have the values $x^{(1)} = 2.5$ and $e^{(1)} = 1.5$.
Surely at the end of all the method will be convergent, but this will lead us to do a not negligible higher value of iterations if compared with the next method that will be presented.

On the other hand we have a big highlight in these method: without having informations on the function, i know before starting the method the error i will committ at the k-esim iteration, so having set a tolerance i know already the number of iterations necessary. That's why:

$$I^{(k)} = [a^{(k)}, b^{(k)}]; \text{ with } x^{(k)} = \frac{a^{(k)} + b^{(k)}}{2}$$
$$|x^{(k)} - \alpha| < \frac{1}{2}|I^{(k)}| = \frac{1}{2}\frac{1}{2}|I^{(k+1)}| = \frac{1}{2^{k+1}}|I^{(0)}| = \frac{1}{2^{k+1}}(b-a)$$

A priori i can set the tolerance s.t. : $\frac{(b-a)}{\text{tol}} < 2^{k+1}$
And the related number of iterations will be: $k > \log_2(\frac{b-a}{tol}) - 1$.

NB. The *Bisection Method* can also be used to find the intersection between two functions (if it exists) creating a function that comprehends both, and putting it equals to zero. For example:

$$\text{given } f(x) \text{ and } g(x) \rightarrow F(x) = f(x) + g(x) = 0$$

2.2 Newton's Method

This particular method needs, if compared with the *Bisection Method*, needs more informations about the function we want to analize. It is based on the *Taylor's approximation theorem*, and the algorithm developed will be:

$$f(x^{(k+1)}) - f(x^{(k)}) = f'(x^{(k)})(x^{(k+1)}) - x^{(k)})$$

So at the end of the algorithm, we will have that $f(x^{(k+1)}) = 0$, and following to this:

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}$$

So basically what we are doing is simply moving to the α point using the intercepts of the derivates of the function step by step (please for a practical example refer to my Github page cited at the beginning of the paper).
It's interesting to watch closer how this method converges to the numerical solution:
Convergence Theorem
If $f \in \mathbb{C}^{\not\in}([a,b])$ s.t $f(\alpha) = 0$ with $\alpha \in [a,b]$, and that $f'(\alpha) \neq 0$, we have that:

1. $\forall k \geq 1 \ |x^{(k)} - \alpha| < \eta$;

2. (*convergence*) $\lim_{k\to 0} x^{(k)} = \alpha$;

3. $\lim_{k\to\infty} \frac{x^{(k+1)}-\alpha}{(x^{(k)}-\alpha)^2} = \frac{f''(\alpha)}{(f'(\alpha))^2}$
   This last condition leads to the *Quadratic Convergence Property*:

   $$\frac{e^{(k+1)}}{(e^{(k)})^2} = c$$

   (Otherwise if this last third condition is not verified, but only the two above, we have a simple convergence).

Lemma: Newton's method converges also for zeros with multiplicity ¿ 1, but in this case only in a linear way (we cannot have the quadratic convergence);
To regain the quadratic convergence the *modified Newton's method* can be used, which algorithm will be:

$$x^{(k+1)} = x^{(k)} - m\frac{f(x^{(k)})}{f'(x^k)}$$

Now lets watch to the **stop criteria**:
The first criteria could be one based on the *increment*, which watches to the x in the k-esim interaction

$$|x^{(k+1)} - x^{(k)}| < \text{toll}$$

And this one become problematic if applied to functions which are particularly pending.

Another one is a criteria based on the *residual* of the function

$$|f(x^{(k+1)})| < \text{toll}$$

And this one, on the contrary with the other, has problems with flat functions. It's clear that to adjust this problem, both criteria could be used.

2.2.1 Extension of the Newton's Method to Non-Linear Equations Systems

Consider a system of N equations with N unknowns:

$$f_1(x_1, x_2, ..., x_N) = 0;$$
$$f_2(x_1, x_2, ..., x_N) = 0;$$
$$f_3(x_1, x_2, ..., x_N) = 0;$$

The Newton method can find the zero to which this system converges (starting in a defined area).
(**N.b.** the method doesn't give us all the solutions of the system, but only the one which is strictly connected with the area in which we are starting the analysis).

The system will be written in this way:

$$\overrightarrow{f} = \begin{pmatrix} f_1 \\ f_2 \\ ... \\ f_N \end{pmatrix}; \qquad \overrightarrow{x} = \begin{pmatrix} x_1 \\ x_2 \\ ... \\ x_N \end{pmatrix};$$

So the system can be written like:

$$\overrightarrow{f}(\overrightarrow{x}) = 0$$

Implementing the algorithm it's clear that we cannot refer to a simple derivative for the system, we have to use the **Jacobian** of $\overrightarrow{f}(\overrightarrow{x})$:

$$J_f(\overrightarrow{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & ... & \frac{\partial f_1}{\partial x_N} \\ \frac{\partial f_2}{\partial x_1} & ... & ... & ... \\ ... & & & \\ \frac{\partial f_N}{\partial x_1} & ... & ... & \frac{\partial f_N}{\partial x_N} \end{bmatrix}$$

So the algorithm becomes:

$$\text{given } \overrightarrow{x}^{(0)}, \text{ for } k = 0, 1, ...$$
$$\delta\overrightarrow{x} = -J_f^{-1}(\overrightarrow{x}^{(k)})\overrightarrow{f}(\overrightarrow{x}^{(k)})$$
$$\overrightarrow{x}^{(k+1)} = \overrightarrow{x}^{(k)} + \delta\overrightarrow{x}$$

In the practice due to the fact that the calculus of the inverse matrix is very heavy from the computational point of view, i can transform the algorithm in order that it will be a linear system that has to be resolved at every step:

$$J_f(\overrightarrow{x}^{(k)})\delta\overrightarrow{x} = -\overrightarrow{f}(\overrightarrow{x}^{(k)})$$

observe that for this type of problem, the convergence conditions are the same, but in a vector definition:
$\rightarrow$ The function must be of $\mathbb{C}^2$ class;
$\rightarrow$ The Jacobian must be Invertible (that is the request related to the fact that the derivate must be $\neq 0$);
$\rightarrow$ I must be in the neighbourhood of the solution.

A last observation that is possible to do in this paragraph is about how to use both methods together;
In fact is possible to implement a code in order to use in a first moment the *bisection method* in order to restrict, with a pair of iterations, the interval in which we could have the solution, and then to use the *Newton's method* to have a quickly convergence to it.

3 Numerical Methods for Linear Systems