Numerical Methods Fortran

Luis Fabián Huecas López

17 de junio de 2025

1 Equation roots

1.1 Open methods

1.1.1 Fixed point

We want to get a root of a function $f(x)$. That means a value r which follows $f(r) = 0$. Now from that equality we make some algebraic operation which leads to $r = g(r)$. For example, from $sin(r) = 0$ we can get $r = sin(r) + r$, that means $g(r) = sin(r) + r$. Then we can use a seed $r^{(1)}$, evaluate $g(r)$ and getting another value of r that we call $r^{(2)}$. Under some conditions this iteration converges to a root. Then the steps are:

1. Being $f(x)$ the function you wish to find a root, from $f(x) = 0$ make some valid operation to get $x = g(x)$.
2. Use your value of x_0 to get a new value x_1 from $g(x)$.
3. Calculate the relative error, if it is smaller than the asked precision the process is stopped and x_1 is returned as the root. If it is greater, select $x_0 = x_1$ and continue the process from step 2.

The condition for convergence is (sufficient condition):

$$|g'(x)| < 1 \tag{1.1}$$

For all the $g(x)$ evaluated. Of course you don't know it a priori so it's not a very useful condition.

Notice the function you use in the program is not $f(x)$, it is $g(x)$.

1.1.2 Newton-Raphson

In this case we select a seed x_0 for our possible root to the function $f(x)$ then we calculate the root of the slope of the function in that point. That's the new possible root that we expect it to be closer to the real root. The slope in x_0 is:

$$f'(x_0) = \frac{f(x_0) - 0}{x_0 - x_1} \tag{1.2}$$

then we have:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \tag{1.3}$$

Procedure:

1. We select a seed x_0.
2. We calculate the next values for x using eq.[1.3].
3. We evaluate the relative error and number of iterations as the condition to stop the program. If the conditions are not met we change $x_0 = x_1$ and continue from step 2.

The relative error is:

$$\epsilon = \left| \frac{x_1 - x_0}{x_1} \right| \tag{1.4}$$

Notice the derivative of the function $f(x)$ is needed for the calculations

1.1.3 Secant

In order to avoid using the derivative function in eq.[1.3] we can approximate that value by:

$$f'(x_i) \approx \frac{f(x_{i-1}) - f(x_i)}{x_{i-1} - x_i} \tag{1.5}$$

Inserting this in eq.[1.3] we have:

$$x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)} \tag{1.6}$$

The process is the same but here we need two seeds for x_0 and x_1.

1.1.4 Modified secant

Instead of using two seeds for the secant method we could use only one and a small variation δ:

$$f'(x_i) \approx \frac{f(x_i + \delta x_i) - f(x_i)}{\delta x_i} \tag{1.7}$$

Inserting this in eq.[1.3] we have:

$$x_{i+1} = x_i - \frac{\delta x_i f(x_i)}{f(x_i + \delta x_i) - f(x_i)} \tag{1.8}$$

From here the process is the same.

Consider δ is must be a small variation for the validity of eq.[1.7].

1.1.5 Modified Newton-Raphson

In order to improve the methods for multiple roots and new method was developed. Instead of aplying the Newton-Raphson method to the function $f(x)$ we do that for the function $u(x) = \frac{f(x)}{f'(x)}$, which can be demonstrated it has the same roots as $f(x)$. We also have:

$$u'(x) = \frac{f'(x)f'(x) - f(x''}{f}(x)[f(x)]^2 \tag{1.9}$$

Aplying Newton-Raphson method (eq. [1.3]) for $u(x)$ means that we have:

$$x_{i+1} = x_i - \frac{u(x_i)}{u'(x_i)} \tag{1.10}$$

Sustituing and leaving it in terms of $f(x)$ we get:

$$x_{i+1} = x_i - \frac{f(x_i)f'(x_i)}{f'(x_i)^2 - f(x_i)f''(x_i)} \tag{1.11}$$

Notice that the disvantage is that now we need the first and second derivative.

1.2 Closed methods

1.2.1 Bisection

When we have a continuos function inside an interval $[a, b]$ then we have the following obvious property:

$$\text{If} \quad f(a)f(b) < 0 \quad \text{then} \quad \exists r \in [a, b] \ / \ f(r) = 0 \tag{1.12}$$

The algorithom of the bisection method is created taken this simple property as its basis. The idea is to shorten the range till we find a valid value as the root. The validity is specified by the user through the absolute error.

1. We compute $f(a) \cdot f(b)$. If it is positive there's no proof there's a root in the given interval $[a, b]$. If it is zero, a, b or both are roots. If it is negative we know there's a root in the interval a we continue with the next step.

2. We propose $r = \frac{a+b}{2}$ as the root. Then we compute $f(a) \cdot f(r)$. If it is positive there's no root in $[a, r]$ then we take $a = r$. If it is negative we take $b = r$. If it is zero, then r is an exact root.

3. We calculate the current absolute error, if it is less than that specified than the user the procedure stops and r is returned as the root. If we didn't reach that precision we continue from step 1 with the new range.

The current absolute error is initially $b - a$ then we can calculate it in the next iterations as:

$$\Delta E^j = \frac{\Delta E^{j-1}}{2} \tag{1.13}$$

where j is the iteration.

1.2.2 False position

In the bisection method we take the new possible root as the middle interval point. This choice can be improved if we trace a straight line between $f(a)$ and $f(b)$ and select the new root as the root of that straight line. This makes sense because it is expected that value is closer in general to the root than just the middle point. The root of that straight line is:

$$r = b - \frac{f(b)(a-b)}{f(a)-f(b)} \tag{1.14}$$

This time we use the relative error as the condition to stop the process.

1. We compute $f(a) \cdot f(b)$. If it is positive there's no proof there's a root in the given interval $[a, b]$. If it is zero, a, b or both are roots. If it is negative we know there's a root in the interval a we continue with the next step.

2. We propose the root as the root of the straight line specified by the points $f(a)$ and $f(b)$ (eq.[1.14]). Then we compute $f(a) \cdot f(r)$. If it is positive there's no root in $[a, r]$ then we take $a = r$. If it is negative we take $b = r$. If it is zero, then r is an exact root.

3. We calculate the current relative error, if it is less than that specified than the user the procedure stops and r is returned as the root. If we didn't reach that precision we continue from step 1 with the new range.

The relative error is:

$$\epsilon = \left| \frac{r^{(j)} - r^{(j-1)}}{r^{(j)}} \right| \tag{1.15}$$

where $r^{(j)}$ is the root after iteration j.

2 Linear equations

2.1 Gauss elimination

The system we aim to solve is constituted by n linear equations and n unknowns:

$$a_{11}x_1 + a_{12}x_2 + ... + a_{1j}x_j + ... + a_{1n}x_n = a_{1n+1} \tag{2.1}$$

$$... \tag{2.2}$$

$$a_{i1}x_1 + a_{i2}x_2 + ... + a_{ij}x_j + ... + a_{in}x_n = a_{in+1} \tag{2.3}$$

$$... \tag{2.4}$$

$$a_{n1}x_1 + a_{n2}x_2 + ... + a_{nj}x_j + ... + a_{nn}x_n = a_{nn+1} \tag{2.5}$$

Then, if we build up a matrice of the coefficients a_{ij} we have a matrice of n rows and $n+1$ columns. This is the matrice that is used in the module programs and that is ask for in the main programs.

Our goal is to have a scalonated matrice. And by that we mean the following form:

$$a_{11}x_1 + a_{12}x_2 + ... + a_{1j}x_j + ... + a_{1n}x_n = a_{1n+1} \tag{2.6}$$

$$... \tag{2.7}$$

$$0 + 0 + ... + 0 + a_{ij}x_j + ... + a_{in}x_n = a_{2n+1} \tag{2.8}$$

$$... \tag{2.9}$$

$$0 + 0 + ... + 0 + a_{nn}x_n = a_{nn+1} \tag{2.10}$$

Then we have a system that can be directly solved. From the last equation we can calculated the value of x_n. Inserting that solution in the next equation we get x_{n-1} and so on, till we have all the system solved.

Then the goal is to apply operations that transfor the system into a equivalent one (same solutions) which leaves a system with the following condition:

$$a_{ij} = 0 \quad \text{if} \quad i > j \tag{2.11}$$

In order to do so we apply what we can call as the main operation. This a operation apply to a row which leaves the system equivalent to the previous one. We have to fixed the column where we want to create all the zeros as in eq.[2.11], let's called this column col.

$$a'_{ij} = a_{ij} - a_{col,j}\frac{\omega}{a_{col,col}} \tag{2.12}$$

where $\omega = a_{i,col}$. So the operation is to substract from the i row the col row multiply by the factor $\frac{\omega}{a_{col,col}}$. It is a row operation so it must be done for all the elements in the row, that's it from $j = 1$ to $j = n + 1$. In the program this can be done from $j = col$ to $j = n + 1$. Becuase if it is well done those values for $j < col$ are already zeros and the main operation leads them to zero again because ω turns out to be zero. This has to be done for all the rows after the reference row which is $row = col$, therefore i goes from $col + 1$ till n.

In order to clarify let's do the process from the begining. Let's call a^1_{ij} the coefficients from the initial matrice and a^{k+1}_{ij} after having applied the proccess k times. We call applying the process to the action of the main operation over all rows. Then, after the first action we have:

$$a^1_{11}x_1 + a^1_{12}x_2 + ... + a^1_{1j}x_j + ... + a^1_{1n}x_n = a^1_{1n+1} \tag{2.13}$$

$$0 + a^2_{22}x_2 + ... + a^2_{2j}x_j + ... + a^2_{2n}x_n = a^2_{2n+1} \tag{2.14}$$

$$... \tag{2.15}$$

$$0 + a^2_{i2}x_2 + ... + a^2_{ij}x_j + ... + a^2_{in}x_n = a^2_{in+1} \tag{2.16}$$

$$... \tag{2.17}$$

$$0 + a^2_{n2}x_2 + ... + a^2_{nj}x_j + ... + a^2_{nn}x_n = a^2_{nn+1} \tag{2.18}$$

After having applied it i times we will have:

$$a_{11}^1 x_1 + a_{12}^1 x_2 + ... + a_{1j}^1 x_j + ... + a_{1n}^1 x_n = a_{1n+1}^1 \tag{2.19}$$
$$... \tag{2.20}$$
$$0 + 0 + ... + 0 + a_{ii}^i x_i + ... + a_{i+1n}^i x_n = a_{in+1}^i \tag{2.21}$$
$$0 + 0 + ... + 0 + a_{i+1i+1}^{i+1} x_{i+1} + ... + a_{i+1n}^{i+1} x_n = a_{i+1n+1}^{i+1} \tag{2.22}$$
$$... \tag{2.23}$$
$$0 + ... + 0 + a_{ni+1}^{i+1} x_{i+1} + ... + a_{nn}^{i+1} x_n = a_{nn+1}^{i+1} \tag{2.24}$$

After aplying it $n-1$ times we will have:

$$a_{11}^1 x_1 + a_{12}^1 x_2 + ... + a_{1j}^1 x_j + ... + a_{1n}^1 x_n = a_{1n+1}^1 \tag{2.25}$$
$$... \tag{2.26}$$
$$0 + 0 + ... + 0 + a_{ii}^i x_i + ... + a_{in}^i x_n = a_{in+1}^i \tag{2.27}$$
$$... \tag{2.28}$$
$$0 + ... + 0 + a_{nn}^{n-1} x_n = a_{nn+1}^{n-1} \tag{2.29}$$

2.1.1 Pivoting

From eq.[2.12] we can see the main problem in the proccess. If $a_{col,col} = 0$ the proccess just fail. For this reason we use the technique of pivoting. This consist of interchanging two rows, the one with the greatest pivot with the one with the lowest pivot. This way if $a_{col,col}$ is a zero then we interchange the col row with the row with the highest value in that column which is locating after the col row.

Let's see the way we apply pivoting. Let's suppose we have the following system:

$$a_{11}^1 x_1 + a_{12}^1 x_2 + ... + a_{1j}^1 x_j + ... + a_{1n}^1 x_n = a_{1n+1}^1 \tag{2.30}$$
$$... \tag{2.31}$$
$$0 + 0 + ... + 0 + a_{ii}^i x_i + ... + a_{i+1n}^i x_n = a_{in+1}^i \tag{2.32}$$
$$0 + 0 + ... + 0 + a_{i+1i+1}^{i+1} x_{i+1} + ... + a_{i+1n}^{i+1} x_n = a_{i+1n+1}^{i+1} \tag{2.33}$$
$$... \tag{2.34}$$
$$0 + ... + 0 + a_{ni+1}^{i+1} x_{i+1} + ... + a_{nn}^{i+1} x_n = a_{nn+1}^{i+1} \tag{2.35}$$

We take all the values we want to convert to zero. That is, from $a_{i+1,i+1}^{i+1}$ to $a_{i+1,i+1}^{i+1}$ and we find the maximum of the those values. Then we interchange rows: the $i+1$ row with the row where the maximum is. Of course it could that the maximum is $a_{i+1,i+1}^{i+1}$, then we don't need to interchange rows.

2.1.2 Row sorting vector

Instead of interchanging rows we use a vector which take into account where it is every row. So in the matrice we never interchange rows. We use a vector of n-elements called *order*. And *order*[*row*] means: *row* is where the row is supposed to be now after the interchanges, and *order*[*row*] is where the row actually is in the matrice (where as we said we never make interchanges of rows).

2.1.3 Solutions

When the Gauss elimination proccess is done, which means we have a system in the form like in eq.[2.25]. We just have to claer every unknown:

$$x_i = \frac{1}{a_{ii}} \left(a_{in+1} - \sum_{j=i+1}^{j=n} a_{ij} x_j \right) \tag{2.36}$$

As we stated before we will use the row ordering. Instead of using $a[i][j]$ we use $a[order[i]][j]$.

2.1.4 Checking the solutions

Finally we can check the solutions. Using the original coeffcients we calculate the left part for each row and compare it with the indpeendent coefficient.

2.2 LU descomposition

For LU descomposition we will call the free coefficients as b_i instead of a_{in+1}. And system of equations is expressed as:

$$AX = B \tag{2.37}$$

Let's suppose we apply Gauss elimination to A getting the following matrice that we called U:

$$\begin{pmatrix} u_{11} & ... & ... & ... & ... & u_{1n} \\ 0 & ... & 0 & u_{ii} & ... & u_{in} \\ 0 & ... & ... & ... & 0 & u_{nn} \end{pmatrix}$$

And let's define a matrice that we call L as:

$$\begin{pmatrix} 1 & 0 & ... & ... & ... & ... & 0 \\ f_{21} & 1 & 0 & ... & ... & ... & 0 \\ f_{i1} & ... & f_{ii-1} & 1 & 0 & ... & 0 \\ f_{n1} & ... & ... & ... & ... & f_{nn-1} & 1 \end{pmatrix}$$

Or in other words:

$$l_{ii} = 1 \ \forall i \qquad l_{ij} = 0 \text{ if } j > i \qquad l_{ij} = f_{ij} \text{ if } i > j \tag{2.38}$$

The f_{ij} are the factors used in the Gauss elimination process in eq.[2.12] that we write as $\omega / a_{col,col}$:

$$f_{ij} = \frac{a^j_{ij}}{a^j_{ii}} \tag{2.39}$$

It can be demonstrated that if we multiply L and U we get the original matrice A:

$$LU = A \tag{2.40}$$

On the other hand when multiplying the matrice U and the solutions we get a vector of different coefficients than B, let's call it D:

$$UX = D \tag{2.41}$$

Using eq.[2.37] and eq.[2.41] we can put:

$$AX - B = UX - D \tag{2.42}$$

Multiplying on the right side by L we get:

$$LD = B \tag{2.43}$$

Taking into account the form of L we can solved D directly. We have the first unknown as:

$$d_1 = b_1 \tag{2.44}$$

Then we can use this solution to get d_2 and so on. The general formula comes from:

$$f_{i1}d_1 + ... + f_{ij}d_j + ... + f_{ii-1}d_{i-1} + d_i = b_i \tag{2.45}$$

Clearing d_i we have:

$$d_i = b_i - \sum_{j=1}^{j=i-1} f_{ij}d_j \tag{2.46}$$

Then we can use eq.[2.41] to get the system solutions X. Taking into account the form of U which is a reduced matrice, we can have the solutions as in the elimination Gauss section in eq.[2.36]. Which here means:

$$x_i = \frac{1}{u_{ii}} \left(b_i - \sum_{j=i+1}^{j=n} u_{ij}x_j \right) \tag{2.47}$$

2.2.1 Usefulness of LU descomposition

We can see from the proccess we made that in the LU descomposition we make the Gauss elimination process and by sustitution we get an auxiliar vector D and then the solutions X. So this involved the same calculations for U and X and some extra more than we make for simple Gauss elimination method. Then, where does it the advantage of this development? Why is it interesting to calculate matrice L and D? The advantage happens when you need to solve the solutions for the same system A but for different coefficients B, having different solutions X each time. If you use simple Gauss elimination method you need to make the Gauss elimination again and again for the different sets of B, while using LU descomposition you only make the Gauss elimination one time and then you can use the same matrice U and L and the vector D to calculate the solutions X for different sets of B.

That is, for only one ste of B there's no point in using LU descomporition but for several sets of B you are using much less operations.

2.2.2 Trigonal systems

In the case we have a trigonal system the algorithom becomes much more simple, which is called Thomas algorithom. First of all let's define a trigonal system in the following way:

$$f_1 x_1 + g_1 x_2 + 0 + ... + 0 = r_1 \tag{2.48}$$
$$e_2 x_1 + f_2 x_2 + g_2 x_3 + 0 + ... + 0 = r_2 \tag{2.49}$$
$$0 + e_3 x_2 + f_3 x_3 + g_3 x_4 + 0' + ... + 0 = r_3 \tag{2.50}$$
$$... \tag{2.51}$$
$$0 + ... + 0 + e_i x_{i-1} + f_i x_i + g_i x_{i+1} + 0 + ... + 0 = r_i \tag{2.52}$$
$$... \tag{2.53}$$
$$0 + ... + 0 + e_n x_{n-1} + f_n x_n = r_n \tag{2.54}$$

In this kind of system the elimination and subtitution process is much more simple. First of all we need to take the values from the matrix for the new vectors e, f, g and r:

$$e[i] = M[i][i-1] \qquad i = 2, ..., n \tag{2.55}$$
$$f[i] = M[i][i] \qquad i = 1, ..., n \tag{2.56}$$
$$g[i] = M[i][i+1] \qquad i = 1, ..., n-1 \tag{2.57}$$
$$r[i] = M[i][n+1] \qquad i = 1, ..., n \tag{2.58}$$

From here we have to follow three steps and we get the solutions, the method is the LU descomposition but we do not demostrate it here. First, it is the descomposition:

$$e[k] = \frac{e[k]}{f_{[k-1]}} \qquad f[k] = f[k] - e[k]g[k-1] \qquad k = 2, ..., n \tag{2.59}$$

$$r[k] = r[k] - e[k]r[k-1] \qquad k = 2, ..., n \tag{2.60}$$

$$x[n] = r[n]/f[n] \qquad x[k] = (r[k] - g[k]x[k+1])/f[k] \qquad k = n-1, ..., 1 \tag{2.61}$$

2.3 Gauss-Seidel

We have a system of equations expressed as:

$$a_{11} x_1 + a_{12} x_2 + ... + a_{1j} x_j + ... + a_{1n} x_n = b_1 \tag{2.62}$$
$$... \tag{2.63}$$
$$a_{i1} x_1 + a_{i2} x_2 + ... + a_{ij} x_j + ... + a_{in} x_n = b_i \tag{2.64}$$
$$... \tag{2.65}$$
$$a_{n1} x_1 + a_{n2} x_2 + ... + a_{nj} x_j + ... + a_{nn} x_n = b_n \tag{2.66}$$

Clearing a generic solution x_i from the row i we have:

$$x_i = \frac{1}{a_{ii}} \left[b_i - \sum_{j=1, j\neq i}^{j=n} a_{ij} x_j \right] \tag{2.67}$$

We select a seed set for the solutions called x_i^1. We introduce this set in eq.[2.67] for $i = 1$ to get x_1^2. Then use the set $(x_1^2, x_2^1, ..., x_n^1)$ to get x_2^2 and so on, till we have the complete set x_i^2. Then you can use this set to get x_1^3 and continue the process. We call x_i^k to the set calculated after $k-1$ iterations.

2.3.1 Stopping the process

A criterium to stop the process can be the maximum number of iterations. Another criterium can be the relative error. It is defined for the i element in percentage as:

$$|\epsilon_i| = \left| \frac{x_i^j - x_i^{j-1}}{x_i^j} \right| \cdot 100 \tag{2.68}$$

We can choose a particular value ϵ_s and when the following condition is reached the process stops:

$$|\epsilon_i| < \epsilon_s \qquad \forall i \tag{2.69}$$

2.3.2 Relaxation

Whenever we calculate a new x_i using eq.[2.67] we can make an addiotional step modifying that solution with the following formula:

$$x_i^j = \lambda x_i^j + (1-\lambda) x_i^{j-1} \tag{2.70}$$

1. If $\lambda = 1$ then the formula does not change anything at all.
2. If $0 < \lambda < 1$ then the new value is a ponderation between the current value and the previous one.
3. If $1 < \lambda < 2$ then we have an extra ponderation for the current value.

2.3.3 Posibility condition

Gauss-Seidel method can not be applied if any of the diagonal elements a_{ii} is a zero. As you can see from the eq.[2.67].

2.3.4 Sufficient condition

If the following condition is met the convergence of the process is guaranteed:

$$|a_{ii}| > \sum_{j=1, j\neq i}^{j=n} |a_{ij}| \tag{2.71}$$

3 Fitting curve

3.1 Least sqaures regression

Let's suppose we have a set of points x_i, y_i that we wish to fit to a straight line. Let's call that function $y(x)$:

$$y(x) = a_1 x + a_0 \tag{3.1}$$

Whichever criterium we use to fit the points to this function, we have an associted error e_i for each point x_i, y_i:

$$e_i = y(x_i) - a_1 x_i - a_0 \tag{3.2}$$

Our goal is to find the values of a_1 and a_0 that minimize this error. But for that porpouse we need to take into account all the points. One strategy is to minimize the quantity $\sum_i e_i$ but due to the signs of e for each error, some high values for e can cancel each other, having as a result a line that fit some points very poorly. Then another strategy that eliminate that problem is to use the sum of the absolute values $|\sum_i e_i|$. But this way leads sometimes to lines that pass through some points (zero error) and leave high errors for others, while the real goal is to fit all the points regularly except those that really don't follow the average trend.

It has been proven that the best strategy is to minimize the sum of the error squares, we call this quantity S_r:

$$S_r = \sum_i e_i^2 = \sum_i (y(x_i) - a_1 x_i - a_0)^2 \tag{3.3}$$

Impossing the minimum conditions:

$$\frac{\partial S_r(a_0, a_1)}{\partial a_0} = 0 \qquad \frac{\partial S_r(a_0, a_1)}{\partial a_1} = 0 \tag{3.4}$$

we have two equations with two variables: a_0 and a_1. One can get the following results:

$$a_1 = \frac{n \sum_i x_i y_i - \sum_i x_i \sum_i y_i}{n \sum_i x_i^2 - (\sum_i x_i)^2} \tag{3.5}$$

$$a_0 = \frac{\sum_i y_i}{n} - a_1 \frac{\sum_i x_i}{n} \tag{3.6}$$

3.1.1 Generalizing to m rank

Now our fitting curve has the following form:

$$y(x) = a_0 + a_1 x + a_2 x^2 + ... + a_m x^m = \sum_{j=0}^{m} a_m x^m \tag{3.7}$$

And now the sum of squares is:

$$S_r = \sum_i e_i^2 = \sum_i \left(y_i - \sum_{j=0}^{m} a_j x_i^j \right)^2 \tag{3.8}$$

To minimize this quantity through the coefficients we have:

$$\frac{\partial S_r}{\partial a_k} = -2 \sum_i x_i^k \left(y_i - \sum_{j=0}^{m} a_j x_i^j \right) = 0 \qquad k = 0, ..., m \tag{3.9}$$

We can express these set of equations as:

$$\sum_i x_i^k y_i - \sum_{j=0}^{m} a_j \sum_i x_i^{k+j} = 0 \qquad k = 0, ..., m \tag{3.10}$$

This constitutes a system of $m+1$ equations and $m+1$ unknowns. That can be solve using the methods for lineal systems. The coefficients of this system of equations are:

$$a_{kj} = \sum_i x_i^{k+j} \qquad a_{km+1} \equiv b_k = \sum_i x_i^k y_i \qquad k = 0, ..., m \quad j = 0, ..., m \tag{3.11}$$

The indexes start from zero, so let's change the expression to start from one like it is done in Fortran. The suitable expression is:

$$a_{kj} = \sum_i x_i^{k+j-2} \qquad a_{km+2} \equiv b_k = \sum_i x_i^{k-1} y_i \qquad k = 1, ..., m+1 \quad j = 1, ..., m+1 \tag{3.12}$$

3.2 Interpolation

3.2.1 Newton Interpolation

We have a set of $n+1$ points $f(x_i), x_i$ that we wish to interpolate through a function. This function is a polynomial of n rank that joints all the points. The formula for this polynomial is:

$$\begin{aligned} f_n(x) = f(x_1) + (x - x_1) f[x_2, x_1] + (x - x_1)(x - x_2) f[x_3, x_2, x_1] + ... \\ ... + (x - x_1)(x - x_2)...(x - x_n) f[x_{n+1}, ..., x_1] \end{aligned} \tag{3.13}$$

where the quantities $f[]$ are defined as:

$$f[x_j, ..., x_i] = \frac{f[x_j, ..., x_{i+1}] - f[x_{j-1}, ..., x_i]}{x_j - x_i} \tag{3.14}$$

and to complete the definition $f[x_j, x_i]$ is defined as:

$$f[x_j, x_i] = \frac{f(x_j) - f(x_i)}{x_j - x_i} \tag{3.15}$$

In order to define the polynomial in eq. [3.13] we need to find the following set of n divided differences:

$$(f[x_2, x_1], f[x_3, x_2, x_1], f[x_4, x_3, x_2, x_1], ..., f[x_{n+1}, ..., x_1]) \tag{3.16}$$

So the method is to calculate these quantities in first place, then we can calculate the polynomial for every x in its domain. In order to calculate them in the program we define the quantity *totdiv* as:

$$totdiv[j, i] = f[x_j, ..., x_i] \tag{3.17}$$

where we define $totdiv[i, i] = f(x_i)$ and what we want as output, that's it, the set in the previous equation we define it as:

$$div[j] = f[x_j, x_1] = totdiv[j, 1] \tag{3.18}$$

The recursive formula for the divided differences in eq.[3.14] uding the matrix in eq.[3.17]:

$$totdiv[x_j, x_i] = \frac{totdiv[x_j, x_{i+1}] - totdiv[x_{j-1}, x_i]}{x_j - x_i} \tag{3.19}$$

And the polynomial in eq. [3.13] is:

$$\begin{split} f_n(x) = div(1) + (x - x_1)div(2) + (x - x_1)(x - x_2)div(3) + ... \\ ... + (x - x_1)(x - x_2)...(x - x_n)div(n + 1) \end{split} \tag{3.20}$$

3.2.2 Lagrange polynomials

It is a reformulation of Newton polynomial. It's formulated as:

$$f_n(x) = \sum_{i=1}^{n+1} L_i(x) f(x_i) \tag{3.21}$$

where $L_i(x)$ are defined as:

$$L_i(x) = \prod_{j=1, j\neq i}^{n+1} \frac{x - x_j}{x_i - x_j} \tag{3.22}$$

3.2.3 Cubic splines

Our goal is interpolate each two consecutive points by a third rank polynomial using some continuity conditions to fix all the coefficientes. We have $n + 1$ points and n intervals, then we have the following set of third polynomials:

$$f_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i \qquad i = 1, ..., n \tag{3.23}$$

Then, we have four unknowns for each polynomials, which means $4n$ unknowns so we need $4n$ conditions to solve the whole system.

Inner nodes value condition:

$$f(x_i) = a_{i-1}x_i^2 + b_{i-1}x_i + c_{i-1} \qquad f(x_i) = a_i x_i^2 + b_i x_i + c_i \qquad i = 2, ..., n \tag{3.24}$$

here we have $2n - 2$ conditions. We have the condition for the edge values:

$$f(x_1) = a_1 x_1^2 + b_1 x_1 + c_1 \qquad f(x_{n+1}) = a_n x_{n+1}^2 + b_n x_{n+1} + c_n \tag{3.25}$$

Inner nodes continuity in the derivative and in the second derivative conditions:

$$f'_{i-1}(x_i) = f'_i(x_i) \qquad f''_{i-1}(x_i) = f''_i(x_i) \qquad i = 2, ..., n \tag{3.26}$$

which adds $2n - 2$ conditions In total we have $4n - 2$ conditions, we need to add arbitrarialy two more in order to get the neccesary conditions to solve the system. One usual way to acomplish this is to set the second derivatives at the edges as null:

$$f''_1(x_1) = 0 \qquad f''_n(x_{n+1}) = 0 \tag{3.27}$$

This is known as the natural spline. We have the result that the n polynomials are:

$$\begin{aligned} f_i(x) = \frac{f''_i(x_i)}{6(x_{i+1} - x_i)}(x_{i+1} - x)^3 + \frac{f''_i(x_{i+1})}{6(x_{i+1} - x_i)}(x - x_i)^3 + \\ + \left[\frac{f(x_i)}{x_{i+1} - x_i} - \frac{f''(x_i(x_{i+1} - x_i))}{6} \right](x_{i+1} - x) + \\ + \left[\frac{f(x_{i+1})}{x_{i+1} - x_i} - \frac{f''(x_{i+1}(x_{i+1} - x_i))}{6} \right](x - x_i) \end{aligned} \tag{3.28}$$

from $i = 1$ to $i = n$. Where $f''_i(x_i)$ and $f''_i(x_{i+1})$ are the unknowns that are obtained from the following equations:

$$\begin{aligned} (x_{i+1} - x_i)f''(x_i) + 2(x_{i+2} - x_i)f''(x_{i+1}) + (x_{i+2} - x_{i+1})f''(x_{i+2}) = \\ = \frac{6}{x_{i+2} - x_{i+1}}[f(x_{i+2}) - f(x_{i+1})] + \frac{6}{x_{i+1} - x_i}[f(x_i) - f(x_{i+1})] \end{aligned} \tag{3.29}$$

These are $n - 1$ equations for $n - 1$ unknowns, from $i = 1$ to $i = n - 1$. And it is a trigonal system. So we just have to construct our matrix, let's call its elements as a_{ij}, and solve the system as in section [2.2.2].

$$a_{jj-1} = (x_{j+1} - x_j) \qquad j = 2, ..., n - 1 \tag{3.30}$$

$$a_{jj} = 2(x_{j+2} - x_j) \qquad j = 1, ..., n - 1 \tag{3.31}$$

$$a_{jj+1} = (x_{j+2} - x_{j+1}) \qquad j = 1, ..., n - 2 \tag{3.32}$$

$$a_{jn+1} = \frac{6}{x_{j+2} - x_{j+1}}[f(x_{j+2}) - f(x_{j+1})] + \frac{6}{x_{j+1} - x_j}[f(x_j) - f(x_{j+1})] \qquad j = 1, ..., n - 1 \tag{3.33}$$

4 Numerical integration

4.1 Newton-Cotes integration

Newton-Cotes integration is based on using an approximation for the function or set of points which one wishes to integrate with a polynomial. Calling $f(x)$ the function to integrate and $f_k(x)$ the approximation, we have:

$$I = \int_a^b f(x)dx \approx \int_a^b f_k(x)dx \tag{4.1}$$

being $f_k(x)$:

$$f_k(x) = \sum_{i=1}^{k} a_i x^i \tag{4.2}$$

But this way of handling the integration is quite unsatisfactory because, using a polynomial of low n as the approximation for the function $f(x)$ can be very poor. Then we make different approximation for different segments. We divide the total range $[a, b]$ in n segments, and for each segment we have a different function f_k^i to be integrated. We call the increment or length of each segment as h:

$$h = \frac{b-a}{n} \tag{4.3}$$

Then we have our total segement divided as:

$$[a, b] = [a, a+h] + [a+h, a+2h] + ... + [a-h+ih, a+ih] + ... + [a-h+nh, a+nh = b] \tag{4.4}$$

Then the integral becomes:

$$I = \int_a^b f(x)dx \approx \int_a^{a+h} f_k^1(x)dx + ... + \int_{a-h+ih}^{a+ih} f_k^i(x)dx + ... + \int_{a-h+nh}^{b} f_k^n(x)dx$$
$$= \sum_{i=1}^{n} \int_{a-h+ih}^{a+ih} f_k^i(x)dx \tag{4.5}$$

For polynomial of rank k you need $k+1$ points in total to approximate the function, or, in other words, that's the number of points including in each segment. Then the total number of points n_p given a total number of segments n for some rank k is the sum from segment 1 to segment n of the number of points per segment $2k+1$ less the total points which are shared by segments which is $n-1$. Then:

$$n_p = \left[\sum_{s=1}^{n}(k+1)\right] - (n-1) = nk + 1 \tag{4.6}$$

Then the increment between points q is:

$$q = x_{i+1} - x_i = \frac{b-a}{n_p - 1} = \frac{b-a}{nk} = \frac{h}{k} \tag{4.7}$$

and we define the indexes for x_i as:

$$x_i = a + q(i-1) = a + \frac{h(i-1)}{k} \tag{4.8}$$

4.1.1 Analysing the error

Let's define a new quantity called S_t that takes into account the sqaures of the differences between the points y_i and the average value $\bar{y}$:

$$S_t = \sum_i (y_i - \bar{y})^2 \tag{4.9}$$

this a quantity similar to S_r because we sum the squares of the differences between the points and a central value (in our lineal fitting $y(x)$ tends to be a central value). For S_t is like our fitting curve were just the average value $y(x) = \bar{y}$. Then we define s_y as the standard deviation:

$$s_y = \sqrt{\frac{S_t}{n-1}} \tag{4.10}$$

which is a measurement of the dispersion from the average. Similarly we can define the fitting error $s_{x/y}$ as:

$$s_{x/y} = \sqrt{\frac{S_r}{n-2}} \tag{4.11}$$

which measures the dispersion from the lineal regression straight.

Then the difference $S_t - S_r$ quantifies how our regression fitting improves the average value to fit the points. And this is normalized using the value S_t (otherwise it would be meaningful because this depends on the scale) getting what we call the correlation coefficient:

$$r^2 = \frac{S_t - S_r}{S_t} \tag{4.12}$$

If S_r is very low then the fitting is in general close to the points. But it depends on the scale. If r^2 is close to 1 that means there is a great improvement with respect using just the average value.

Then, in order to have a good analysis of how good the fitting you must graph it. These quantities are not enough by their own. Usually r^2 is used to claim the quality of the fitting but here we can see that it's meaning is not that depth by its own.

4.1.2 Trapezoid rule

For the trapezoid method the approximation is made using a polynomial of grade 1. Each approximation function can be expressed as:

$$f_1^i = f(a-h+ih) + \frac{f(a+ih) - f(a-h+ih)}{h}(x - a + h - ih) \tag{4.13}$$

Then each integral in the multiple process is:

$$\int_{a-h+ih}^{a+ih} f_1^i(x)dx = h\frac{f(a-h+ih)+f(a+ih)}{2} \tag{4.14}$$

Inserting this in eq.[4.5] we have:

$$I = \frac{h}{2}\sum_{i=1}^{n} f(a-h+ih)+f(a+ih) = \frac{h}{2}\left[f(a)+f(b)+2\sum_{i=1}^{n-1} f(x_{i+1})\right] \tag{4.15}$$

In this case $h = q$ and expressing this integral in terms of number of points we have(we make the index change $i = i+1$ in the sum):

$$I = \frac{(b-a)}{2(n_p-1)}\left[f(a)+f(b)+2\sum_{i=1}^{n_p-1} f(x_i)\right] \tag{4.16}$$

4.1.3 1/3 Simpson rule

For the trapezoid method the approximation is made using a polynomial of grade 2. This time the grade of the polynomial is takes as 2. Each approximation function can be expressed as:

$$f_2^i = \frac{(x-x_{i+1})(x-x_{i+2})}{(x_i-x_{i+1})(x_i-x_{i+2})}f(x_i) + \frac{(x-x_i)(x-x_{i+2})}{(x_{i+1}-x_i)(x_{i+1}-x_{i+2})}f(x_{i+1}) + \frac{(x-x_i)(x-x_{i+1})}{(x_{i+2}-x_i)(x_{i+2}-x_{i+1})}f(x_{i+2}) \tag{4.17}$$

Integrating from $x = x_i$ to $x = x_{i+2}$ we have:

$$\frac{h}{3}[f(x_i)+4f(x_{i+1})+f(x_{i+2})] \tag{4.18}$$

But the segment don't go from x_i to x_{i+1}, they go from x_i to x_{i+2}. So if we want to insert the previous expression in eq.[4.5] we have to make the following index change:

$$2j = i+1 \tag{4.19}$$

And we have:

$$I = \frac{q}{3}\sum_{j=1}^{n}[f(x_i)+4f(x_{i+1})+f(x_{i+2})] = \frac{q}{3}\sum_{j=1}^{n}[f(x_{2j-1})+4f(x_{2j})+f(x_{2j+1})] \tag{4.20}$$

Here the $4f(x_{2j})$ terms appear in the sum only one time while the $f(x_{2j+1})$ terms appear two times except $f(a)$ and $f(b)$. Then the final expression is:

$$I = \frac{q}{3}\left[f(a)+f(b)+4\sum_{j=1}^{n} f(x_{2j})+2\sum_{j=1}^{n-1} f(x_{2j+1})\right] \tag{4.21}$$

The integral here is expressed in terms of segments n, expressing in terms of number of points we have (we make the index change $2j = i$ in the first sum and $2j+1 = i$ in the second):

$$I = \frac{(b-a)}{3(n_p-1)}\left[f(a)+f(b)+4\sum_{i=2,\ i\neq odd}^{n_p-1} f(x_{i+1}) + 2\sum_{i=3,\ i\neq even}^{n_p-2} f(x_{i+2})\right] \tag{4.22}$$

4.2 Romberg integration

When one uses a numerical method to calculate an integral using some number of points n_p, then having a particular value of q we can express the real value of the integral as:

$$I = I(q) + O(q^{2k}) \tag{4.23}$$

where $I(q)$ is the value obtained using the numerical method and $O(q^{2k})$ is the error associated to that particular method which we suppose is proportional to q^{2k}:

$$O(q^{2k}) = \alpha q^{2k} \tag{4.24}$$

actually this is true if the segments between points are small enough to consider the derivatives as constants inside.

Now let's suppose we use a particular method for two different values of q: q_1 and q_2. Using the previous formula we can write:

$$I(q_1) + O(q_1^{2k}) = I(q_2) + O(q_2^{2k}) \tag{4.25}$$

Using eq.[] we have:

$$O(q_1^{2k}) = O(q_2^{2k})\left(\frac{q_1}{q_2}\right)^{2k} \tag{4.26}$$

Inserting in eq.[] we have:

$$O(q_2^{2k}) = \frac{I(q_1)-I(q_2)}{1-\left(\frac{q_1}{q_2}\right)^{2k}} \tag{4.27}$$

Then we can write

$$I = I(q_2) + \frac{I(q_2)-I(q_1)}{\left(\frac{q_1}{q_2}\right)^{2k}-1} \tag{4.28}$$

Let's apply this equation to the special case $q_2 = q_1/2$:

$$I = I(q_2) + \frac{I(q_2)-I(q_1)}{(4)^k-1} = \frac{4^k I(q_2)-I(q_1)}{4^k-1} \tag{4.29}$$

Let's identify now the integrals by two indexes $I_{j,k}$. j stablishes the segments used and k the integration level (magnitude of the associated error) and we suppose I is an integral that can be thought as having used q_1 as the segment increment, then:

$$I_{j,k} = \frac{4^k I_{j+1,k-1} - I_{j,k-1}}{4^k-1} \tag{4.30}$$

If we denote q as the first increment, then:

$$q_j = \frac{q}{2^{(j-1)}} \tag{4.31}$$