Preface

In the field of mathematical modeling, many real life problems, in different fields of Science, can be contracted in the form of algebraic or differential equations. Some of these equations have linear or nonlinear behavior that makes obtaining the exact solution of its governing equation difficult or without exact solution. In that case the alternative way is to use the numerical methods to solve these equations such as Newton-Raphson, which can be used to find the approximate solutions of nonlinear equations, and finite difference methods, which can be used to solve different type of differential equations. In fact, the systems of nonlinear algebraic or differential equations are more complicated and have more properties than linear equations, these properties are related to important features of the real world phenomena, on the other hand, these properties are connected with the difficulties of the mathematical treatment. Therefore, since the last century, many authors have studied the numerical solutions for different types of linear and nonlinear systems.

In the last decades, with the development of computers and software programming, numerical analysis has become one of the most active areas of Mathematics research because it has helped Mathematicians to find answers and explanations for many phenomena of the nonlinear world.

In this book, we have considered different mathematical problems with their numerical methods. Moreover, we have discussed the iterative and absolute errors which arise from using these methods. In addition, the numerical codes for these methods, written in MATLAB, are addressed at the end of this book.

I believe that this book will be an excellent guide for undergraduate students at colleges of science and engineering, and it can help them to understand the course of numerical analysis easily. In addition, they can learn basically how to write computer codes for the studied numerical methods.

Organization of the Book

This book is divided into six chapters: In Chapter one, we introduce basic meanings to the numerical analysis, its importance, and the main types of errors. In Chapter two, we consider some methods, which can be used to find the numerical solution of nonlinear equations, such as Bisection and Newton-Raphson methods. Chapter three is devoted to study how to find the numerical solutions of linear algebraic systems, using direct and indirect methods. Chapter four is concerned with interpolation problems and numerical differentiations, different methods are considered, such as Lagrange and Finite difference methods. The numerical integration is addressed in chapter five; different methods are used such as Trapezoidal and Simpson methods. In Chapter six, we study some methods, used to compute the numerical solutions of ordinary differential equation of first order, such as Euler and Runge-Kutta methods.

Finally, in the Appendix chapter, we state the programming codes associated with the numerical methods, studied in this book.

The Author

Maan A. Rasheed
September - 2022
Baghdad

Chapter One

Introduction

Numerical Analysis is a field of mathematics that concerned with the study of approximate solutions of mathematical problems, where it is difficult or impossible to find the exact solutions for these problems.

For instance, we can't find the exact value of the following integral, by using known integration methods,

$$\int_0^1 e^{x^2}\, dx$$

While we can find an approximate value for this integral, using numerical methods.

The other example, if we aim to find the solution of a linear system $Ax = b$, where $A \in R^{n\times n}$, when n is too large , it is so difficult to find the exact solutions for this system by hand using known methods, so in this case, it is easier to think about how to find the approximate solution using a suitable algorithm and computer programs.

The importance of Numerical Analysis

To interpret any real phenomena, we need to formulate it in a mathematical form. To give a realistic meaning for these phenomena we have to choose complicated mathematical models, but the problem is; it is so difficult to find explicit formulas describe the exact solutions for these complicated models.

Therefore, it might be better to find the numerical solutions for complicated form rather than finding the exact solutions for easier forms that can't describe these phenomena in realistic way.

The nature of Numerical analysis

Since for any numerical Algorithm (the steps of the numerical method), we have lots of mathematical calculations, we need to choose a suitable computer language such as Matlab or Mable and write the algorithm processes in programing steps.

In fact, the accuracy of numerical solutions, for any problem, is controlled by three criteria:

1- The type of algorithm,
2- The type of computer language and programs,
3- The advancement of the computers which are used.

Types and sources of Errors

When we compute the numerical solutions of mathematical model that we use to describe a real phenomenon, we get some errors; therefore we should study the types and sources of these errors.

We can point out the most important types and sources of these errors as follows:

1- Rounding errors: this type of errors can be got, because of the rounding of numbers in computer programming languages.

Example 1.1: 5.99…9 round to 6, and 3.0001 round to 3.

2- *Truncation Errors:* Since most of numerical algorithms depend on writing the functions as infinite series, and since it's impossible to take more than few terms of these series when we formulate the algorithm, therefore, we get errors, called truncation errors.

Example 1.2: $f(x) = e^{x^2} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$

If we compute $f(5)$, with taking 4 terms, we get more errors than with taking 10 terms.

3- *Total errors*: Since any numerical algorithm is about iterative presses, the solution in step n depends on the solution in step $(n-1)$. Therefore, for larger number of steps we get more errors, and those errors are the total of all previous types of errors.

Let $\bar{x}$ is the approximate value of x, there are two methods can be used to measure the errors:-

1- Absolute Error: $E_x = |x - \bar{x}|$,

2- Relative error: $R_x = \frac{E_x}{|x|} = \frac{|x-\bar{x}|}{|x|}$

Example 1.3: Let $\bar{x} = 3.14,\ x = 3.141592.$ Find the Absolute and Relative errors

Solution:

$E_x = |x - \bar{x}| = 0.001592$,

$R_x = \frac{0.001592}{3.141592} = 0.000507$

Remark: Clearly, $R_x < E_x$

Exercises

Q1:

a) What is numerical analysis concerned with? And what is the importance of studying this subject?
b) What are the most important types of errors that arise from using a numerical method to compute the numerical solution of a mathematical problem?
c) What are the criteria that control the accuracy of numerical solutions?

Q2: Let E_x, R_x be the absolute and relative errors, respectively, in an approximate value of x. Show that $R_x \leq E_x$, if $|x| \geq 1$.

Chapter Two

Numerical Solutions for Nonlinear Equations

There are lots of real problems those can be solved by mathematical forms, and these forms has nonlinear equations. Mostly, it is difficult to calculate the exact solutions for these equations; therefore, we study some numerical methods in order to be able to find the approximate solutions for these equations.

Examples 2.1:

$f(x) = e^{x^2}\ cosx,\ f(x) = x^2 + x - 1$ nonlinear equations for one variable

$f(x,y) = (x+y)^2, f(x,y) = 1/x + y + lnxy$ nonlinear equations for two variables.

Roots of nonlinear equations of one variable

Finding the solution for a nonlinear equation of one variable, $f(x) = 0$ means, finding a value α, such that $f(\alpha) = 0$, where α is called the root of $f(x) = 0$.

Remark: Some equations have more than one root.

Example 2.2:

1- $f(x) = x^2 + 3x + 2 = (x + 2)(x + 1) = 0$

Since $f(-1) = f(-2) = 0$, it follows that both of $-1, -2$ are roots for the nonlinear equation above.

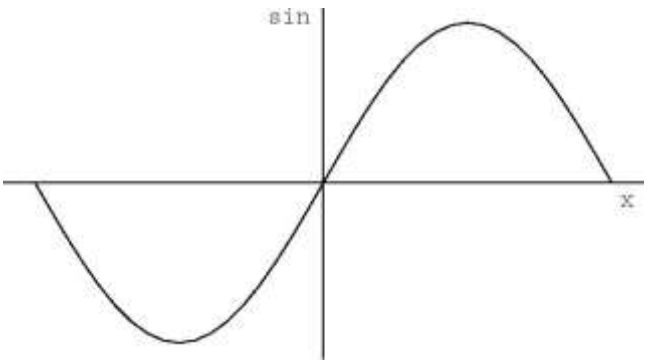
2- $f(x) = sin\, x$, We see that $x = n\pi, n = 0,1,2, ...$, are roots for f

3- $f(x) = (x - 1)^2$, we see that $x = 1$, is the double root for f.

The approximate roots for non-linear equations

In order to ensure that, there exist a root α, for the equation $f(x) = 0$ on the interval $[a, b]$, we have to make sure that $f(a)f(b) < 0$, see the following figure:

$f(b)$



$f(a)$

Remark: Let α be the exact root of the equation $f(x) = 0$, and α_n is a subsequence of approximate roots, that can be got from using a numerical method, for large n the following condition has to satisfy: $|f(\alpha_n)| < \in$, or $|\alpha - \alpha_n| < \in$

Next, we study some known numerical algorithms those can be used to find the approximate solutions (roots) for non-linear equations, which are Bisection algorithm, Newton–Raphson algorithm and fixed point algorithm.

Bisection Algorithm

Let $f \in C[a,b]$,

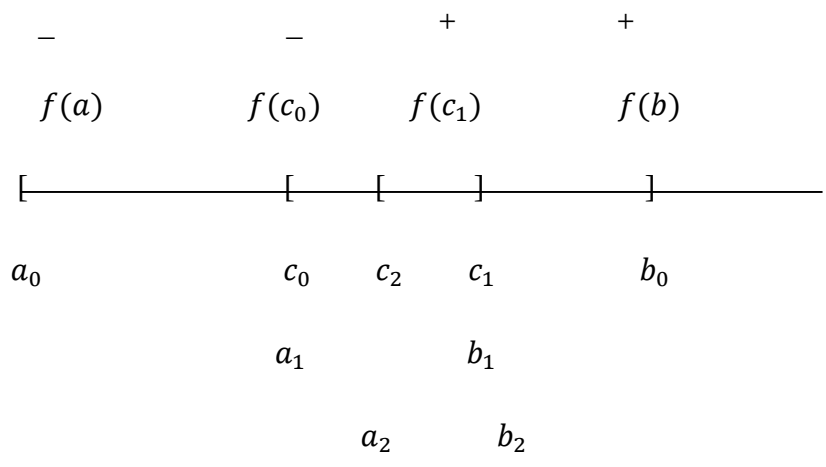
i.e f Is continuous function on the closed interval $[a,b]$.

Assume that the condition $f(a)f(b) < 0$ is satisfied, we follow the following steps:

1- We bisect the space as follows: $c = \frac{a+b}{2}$ or $c_n = \frac{a_n+b_n}{2}$

2- If $f(c) = 0$, it follows that c is the exact root, otherwise check the sign of $f(c)$, if $f(c) < 0, f(b) > 0$, then the exact root belong to $[c,b]$, and we set $a = c, b = b$, and then we repeat the same steps. While, if $f(c) > 0, f(a) < 0$, then the exact root belong to $[a,c]$, And we set $a = a, b = c$, and then we repeat the same steps.

3- We continue iteratively, until we get the following condition is satisfied $|f(c_n)| < \in$, *or* $|c_{n+1} - c_n| < \in$, *or* $|b_n - a_n| < \in$



Remarks:

1- From the iterative processes of Bisection algorithm, we get a sequence of closed intervals$I_i = [a_i, b_i]$, and for i<j, the length of $I_j = [a_j, b_j]$ is shorter than the length of $[a_i,b_i]$ and $I_i \supseteq I_j$. Therefore, according to known real analysis theorems, the intersection of all these intervals contains only one point, which is the exact root of $f(x) = 0$.

2- From the iterative processes of Bisection algorithm, we get a sequence of approximate roots, $\{c_n\}$ for the nonlinear equation $f(x) = 0$, which is convergent to the exact root α i.e. $\{c_n\} \xrightarrow{n\to\infty} \alpha$.

Example 2.3: consider that, we have the following equation

$$f(x) = x^2 - 2 = 0, x \in [1, 2],$$

1- Compute the approximate roots of this equation, with using Bisection algorithm, for three iterative steps.

2- Find the iterative errors as each step.

3- Since the exact solution is known for this equation, find also the absolute errors at each step.

Solution:

Clearly, $\alpha = \sqrt{2} \approx 1.414$, is the exact root of f on the interval [1, 2]

$f(2) = 2,\ ,\ f(1) = -1$, thus there exists a root in this interval

$a_0 = 1,\ \ b_0 = 2,\ c_0 = \frac{a_0+b_0}{2} = 1.5,\ \ \mathrm{f}(c_1) = 0.25 > 0$

So, $a_1 = 1,\ \ b_1 = 1.5,\ c_1 = \frac{a_1+b_1}{2} = 1.25$

The iterative errors can be found as follows:

$E_n = |c_{n+1} - c_n|, \quad E_1 = |1.25 - 1.5| = 0.25$

$c_2 = \frac{a_2 + b_2}{2} = \frac{1.25 + 1.5}{2} = 1.375$

Absolute error $= |c_n - \alpha|$

n	a_n	b_n	c_n	f(c_n)	E_n	Absolute Errors
0	1	2	1.5	0.25	0.25	0. 086
1	1	1.5	1.25	-0.4375	0.125	0.164
2	1.25	1.5	1.375	-0.1094		0.039

The iterative error, $E_2 = |c_3\text{-}c_2|$=1.25, clearly, it is, still too large, so we have to continue in the iterative processes until we get the convergent condition, $|c_{n+1} - c_n| < \in$, is satisfied . or $|b_n - a_n| < \in$

False position Algorithm

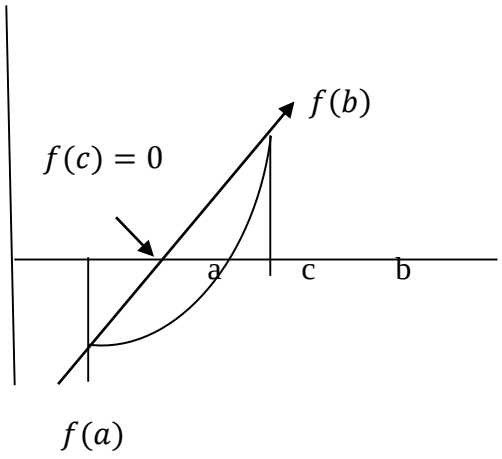
This algorithm is faster than Bisection algorithm in convergence, but they almost smaller in steps. In fact, it has been called by this name, because in each iterative step we find an approximate position for the exact root, and we assume that this place is the real position of the exact root, which means we assume that the equation equal zero at this position.

Before starting in the steps of this algorithm, we have to make sure that:

$f \in C[a,b]$, and $f(a)f(b) < 0$

The main idea of this method is to assume that f is linear on $[a,b]$, which means the point $(c,0)$ belongs to the line AB

Where $A = (a, f(a)),\ B = (b, f(b))$.



Since the two points $(c,0), (b, f(b))$, belong to the same line, we have

$$\frac{\mathrm{f(b)} - \mathrm{f(a)}}{\mathrm{b} - \mathrm{a}} = \frac{\mathrm{f(b)} - 0}{\mathrm{b} - \mathrm{c}}$$

Thus $c = b - f(b)\ (\frac{b-a}{f(b)-f(a)}) = \frac{af(b)-bf(a)}{f(b)-f(a)}$

Algorithm steps

1- Find c by using the relation c= $\frac{af(b)-bf(a)}{f(b)-f(a)}$

2- As in Bisection algorithm, we check whether, $f(c) = 0$, we know that c is the exact root, otherwise check the sign of $f(c)$, if $f(c) < 0, f(b) > 0$, then the exact root belong to [c,b].And we set $a = c, b = b$, and then we repeat the same steps.
While, if $f(c) > 0, f(a) < 0$, then the exact root belong to $[a, c]$,
And we set $a = a, b = c$, and then we repeat the same steps.

3- We continue until to get the following condition is satisfied

$$|c_{n+1} - c_n| < \in$$

Remark: From the iterative processes, as in the Bisection algorithm we get a sequence of approximate roots, which is convergent to the exact root

i.e. $\{c_n\} \xrightarrow{n\to\infty} \alpha$

Example 2.4: Find the approximate roots of the following equation

$f(x) = x^2 + x - 1 = 0$, on the interval [0, 1], by using False position algorithm, for two iterative steps (find only c_1, c_2).

Solution:

N	a_n	$f(an)$	b_n	$f(bn)$	c_n	$f(c_n)$
0	0	-1	1	1	0.50	-0.25
1	0.50	-0.25	1	1	0.60	0.040-

Newton-Raphson Method

This algorithm can be used to find the approximate toots for the equation $f(x) = 0$, when it easy to find the derivative, f'.

Deriving the Newton-Raphson's formula:

Let $f \in C^2[a, b]$,

i.e. f and f' are continuos functions on [a, b],

Let $\{x_n\}$, is a sequence of approximate roots for f, such that:

$$f'(x_n) \neq 0, \qquad \text{and} \quad |x_n - \alpha| < \epsilon, \qquad \forall\, n$$

Where, α is the exact root for $f(x) = 0$.

Use Taylor expansion for f, around x_n , we get

$$f(x) = f(x_n) + (x - x_n)f'(x_n) + \frac{(x-x_n)^2}{2!}f''(x_n) \dots\dots$$

Substitute $x = \alpha$

$$0 = f(\alpha) = f(xn) + (\alpha - x_n)f'(x_n) + (\, small\ terms)$$

$$\alpha - x_n = -\frac{f(x_n)}{f'(x_n)} \quad \Longrightarrow \quad \alpha = x_n - \frac{f(x_n)}{f'(x_n)}$$

Set $\alpha = x_{n+1}$

We get the **Newton-Raphson's equation**

$$x_{n+1} = x_n - \frac{f(xn)}{f'(x_n)}, \quad n = 0,1,2, \dots\dots.$$

Remark: In order to guarantee that, the iterative process is convergent, the initial root, x_0 , should be chosen close to the exact root α, which means:

$$|x_0 - \alpha| < \epsilon$$

Steps of Newton-Raphson algorithm

1- Choose appropriate initial root $x_0 \in [a, b]$

2- Set $n = 0$

3- Calculate $, x_{n+1} = x_n - \frac{f(xn)}{f'(x_n)}$

4- Set $n = n + 1$ and continue in the iterative processes, until we get the stop condition is satisfied: $|x_{n+1} - x_n| < \epsilon$

Example 2.5: Use Newton-Raphson algorithm to find the approximate root of the following equation $f(x) = \sin x - \frac{(x+1)}{(x-1)}$, with $x_0 = -0.2$ for two iterative steps (only find x_1, x_2). Also, find the iterative error at each step, where $E_n = |x_{n+1} - x_n|$

Solution: $n = 0$

$$f'(\mathrm{x}) = \cos \mathrm{x} - \frac{(x-1)-(x+1)}{(x-1)^2} = cos\ x + \frac{2}{(x-1)^2}$$

$$f(x_0) = sin\ (-0.2) - \frac{(-0.2+1)}{(-0.2-1)} = (0.42) = 0.4680$$

$$f'(x_0) = cos\ (-0.2) + \frac{2}{(-0.2-1)^2} = 2.3690$$

Thus

$$x_1 = x_0 - \frac{f(x0)}{f'(x_0)} = (-0.3976)$$

$n = 1$

$$f(x_1) = 0.0439$$

$$f'(x_1) = 1.9460$$

$$x_2 = x_1 - \frac{f(x1)}{f'(x_1)} = (-0.4201)$$

By the same way, we can find $x_3 = -0.4204$, $x_4 = -0.4203$

We continue, iteratively, until we get the stop condition is satisfied:

$$|x_{n+1} - x_n| < \epsilon$$

n	x_n	$f(xn)$	$f'(x_n)$	E_n
0	-0.2000	0.4680	2.3690	0.1976
1	-0.3976	0.0438	1.9459	0.0225
2	-0.4201	0.0005	1.9048	0.0003
3	-0.4204	-0.0001	1.9042	0.0001
4	-0.4203	0.0001	1.9044	

Secant Algorithm

This algorithm is faster than Bisection and False position algorithms in convergence. We use this algorithm, when it is difficult to find the derivative f', where $f(x) = 0$.

We can derive the formula of this method as follows:

Let $\{f_n\}$ is a sequence of approximate roots for f

Form the definition of derivative, we get

$$f'(p_{n-1}) = \lim_{x \to p_{n_1}} \frac{\mathrm{f(x)} - \mathrm{f}(p_{n-1})}{\mathrm{x} - p_{n-1}}$$

Set $x = p_{n-1},\ we\ get$

$$f'(p_{n-1}) = \lim_{x \to p_{n_1}} \frac{\mathrm{f}(p_{n-2}) - \mathrm{f}(p_{n-1})}{p_{n-2} - p_{n-1}}$$

$f'(p_{n-1}) \cong \frac{\mathrm{f}(p_{n-2}) - \mathrm{f}(p_{n-1})}{p_{n-2} - p_{n-1}}$…….. (1)

Recall N.R. equation

$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}$…… (2)

Substitute (1) in (2), we get the secant equation

$$p_n = p_{n-1} - \frac{f(p_{n-1})(p_{n-2} - p_{n-1})}{\mathrm{f}(p_{n-2}) - \mathrm{f}(p_{n-1})}$$

Remark: Unlike to N.R. algorithm, In order to use the last equation to find the approximate roots, we need to choose two initial roots, p_0, p_1 , and that can be considered one of the disadvantages to using secant algorithm; however we do not need to find the derivative as in N.R. algorithm.

The steps of secant algorithm

1- input p_0, p_1

2- Find p, using secant equation $p \quad = p_1 - \frac{f(p_1)(p_0 - p_1)}{\mathrm{f}(p_0) - \mathrm{f}(p_1)}$

3- set $p_0 = p_1$, $p_1 = p$

4- we repeat the iterative presses until we get $|p - p_n| < \in$

5- print the last approximate root and the number of iterative processes until we get the convergence

Remark: This algorithm has been called secant algorithm, because, the line which contains the two points , $(p_0, f(p_0))$, $(p_1, f(p_1))$ intersect with the x-axis at the point $(p, 0)$.

Example 2.6: Use the secant Algorithm, to find the approximate solution of the equation

$f(x) = x^2 - 4 = 0$, assume that $p_1 = 2.5$, for two iterative steps, $p_0 = 3$

Solution:

$$p \quad = p_1 - \frac{f(p_1)(p_0 - p_1)}{\mathrm{f}(p_0) - \mathrm{f}(p_1)} = 2.0909$$

$p_0 = 2.5$, $p_1 = p = 2.0909$

$$p \quad = p_1 - \frac{f(p_1)(p_0 - p_1)}{f(p_0) - f(p_1)} = 2.0099$$

N	p_0	$f(p_0)$	p_1	$f(p_1)$	p	*f(p)*
0	3	5	2.5	2.2500	2.0909	0.3719
1	2.5	2.2500	2.0909	0.3719	2.0099	0.0397

Fixed Point Algorithm

This method depends on the concept of fixed points for one variable functions

Definition 2.1: The point x_0 which belongs to the domain of the function g is called a **fixed point** for, iff $g(x_0) = x_0$

We can give the idea of this algorithm as follows:

We write the function f, where $f(x) = 0$ as follows:

$$f(x) = x - g(x)$$

Such that α is a root for f, thus $f(\alpha) = 0$,

Which means that, α is a fixed point for g, that is

$$\alpha - g(\alpha) = 0 \rightarrow g(\alpha) = \alpha$$

Therefore, the problem becomes, we have to look for the fixed point of g rather than, looking for the root of f.

Fixed point Theorem:

Let $g \in c[a,b]$, $g(x) \in [a,b]$, $\forall x \in [a,b]$. *Then g has a fixed point on* $[a,b]$. *Moreover, if* g' *exists on* (a,b), *such that* $|g'(x)| \leq k \leq 1$, *Then g has a unique fixed point,* $p \in [a,b]$.

Remark: when we choose a certain form for *g*, we have to make sure that

$$|g'(x)| \leq 1, \forall\, x \in (a,b)$$

This condition can guarantee the convergence for the algorithm.

Fixed point algorithm steps

1- Choose the initial root x_0
2- Choose a form for the function *g*, *such that* $|g'(x)| \leq 1, \forall\, x \in (a,b)$,
3- Set $x_1 = g(x_0)$, $x_{n+1} = g(x_n)$, $n = 1,2,3 \ldots\ldots$
4- We continue in the iterative process until we get: $|x_{n+1} - x_n| < \in$

Example 2.7: Find the approximate root of the following equation $f(x) = x^2 + 4x - 10 = 0$, on the interval [1, 3.5], with considering that$x_0 = \frac{3}{2}$, for two iterative steps (find only x_1, x_2).

Solution:

Let us choose two forms for g as follows:

$g_1(x) = x = \dfrac{10}{x+4} \ldots\ldots (1)$

$g_2(x) = x = \frac{10-x^2}{4} \ldots\ldots\ldots (2)$

$g'_1(x) = \dfrac{-10}{(x+4)^2}$

$g'_2(x) = -\dfrac{x}{2}$

It is clear that, $|g'_1(x)| \leq 1, \ \forall x \in (1,3.5)$,

While $|g'_2(3)| = \frac{3}{2} > 1$

Therefore, we ignore g_2 and choose the convergent form $g = g_1 = \frac{10}{x+4}$

Next, we find x_1, x_2 …

$$x_1 = g(x_0) = \frac{10}{\left(\frac{3}{2}\right) + 4} = \frac{10}{\frac{3+8}{2}} = \frac{20}{11} = 1.8182$$

$$x_2 = g(x_1) = \frac{10}{\left(\frac{20}{11}\right) + 4} = \frac{10}{\frac{64}{11}} = \frac{110}{64} = \frac{55}{32} = 1.7188$$

We continue, with n=3, 4...until 29.

Where $x_{29} = 1.7416$ and $|x_{29} - x_{28}| < \in$

Error Analysis for Iterative Methods

Definition 2.2:- Suppose $\{x_n\}$ is a sequence that can be get from using any iterative method (such as Bisection, N.R., Fixed point) and assume that it converges to the exact root α, and let $e_n = |x_n - \alpha|$ is the absolute error at the iterative step n.

If there exists λ and k such that

$$\lim_{n\to\infty} \frac{e_{n+1}}{{e_n}^k} = \lambda$$

Then $\{\alpha_n\}$ is said to convergence to α of order k with constant error λ

In general, a sequence with a high order of convergence will converge more rapidly than a sequence with low order.

In this lectures notes, two cases of order will be given special attention

1- if $k = 1, the\ method\ is\ called\ linear$

2- if $k = 2, the\ method\ is\ quadratic$

Theorem 2.1: Fixed point algorithm exhibits linear convergence.

In order to prove the last theorem, we need to recall mean value theorem:

Mean value theorem:

Let $h \in C[a,b]$, $(h\ and\ h'\ are\ continous\ functions\ on\ [a,b])$. then exists $c \in (a,b)\ \ such\ that\ \ h(b) - h(a) = h'(c)(b-a)$.

To prove **Theorem 2.1**, suppose we would like to find the approximate solution to $f(x) = 0$, on $[a,b]$ using fixed point algorithm with $x_0 \in [a,b]$,

$$x = g(x), \qquad \text{such that,} \qquad 0 < |g'(x)| \leq M < 1 \quad \forall\, x \in [a, b],$$

This means we have the iterative scheme $x_{n+1} = g(x_n), \ \forall n \geq 0$

Thus, according to fixed point Theorem, g has a unique fixed point α (the exact root of $f(x) = 0$), where $\alpha \in [a, b]$, & $\{x_n\} \xrightarrow{n\to\infty} \alpha$.

By mean value theorem, we get

$e_{n+1} = |x_{n+1} - \alpha| = |g(x_n) - g(\alpha)| = |g'(z_n)||x_n - \alpha| = \lambda_n e_n \ldots..$ (1)

Where $x_n \leq z_n \leq \alpha, \ \lambda_n = |g'(z_n)|, \ \forall\, n$

Since $\{x_n\} \xrightarrow{n\to\infty} \alpha,$ we get $\{z_n\} \xrightarrow{n\to\infty} \alpha$

And since g' is continues**,** by the definition of continuity, we get

$$\lambda_n = |g'(z_n)| \xrightarrow{n\to\infty} |g'(\alpha)| = \lambda < 1$$

Thus, $\lim_{n\to\infty} \frac{e_{n+1}}{e_n} = \lim_{n\to\infty} \lambda_n = \lambda \ldots\ldots\ldots\ldots.$ (2)

Therefore, from equations (1) & (2) and by definition 2.2, we conclude that, fixed point iteration algorithm exhibits linear convergence.

Theorem 2.2: Newton Raphson algorithm converges quadratically

In order to prove Theorem 2.2, we need to state first the following theorem without giving its proof.

Theorem 2.3: Let α is a solution of $x = g(x)$ such that $g'(\alpha) = 0$ and g'' is continuous on an open interval contains α. Then there exists $\delta > 0$ such that for $x_0 \in [\alpha - \delta, \alpha + \delta]$, the sequence defined by $x_{n+1} = g(x_n)$, $\forall n \geq 0$ is quadratically convergent.

Proof of Theorem 2.2: From using N.R. algorithm to find the approximate root of the equation $f(x) = 0$, where $f \in C^2[a, b]$ and $f'(x) \neq 0, \forall\, x \in [a, b]$, we get the iterative scheme:

$$x_{n+1} = g(x_n) = x_n - \frac{f(x_n)}{f'(x_n)}, \quad \forall n \geq 0,$$

i.e. $$g(x) = x - \frac{f(x)}{f'(x)}$$

Thus $g'(x) = 1 - \frac{f'^2(x) - f(x)f''(x)}{f'^2(x)} = 1 - 1 + \frac{f(x)f''(x)}{f'^2(x)} = \frac{f(x)f''(x)}{f'^2(x)}$

This leads to $g'(\alpha) = \frac{f(0)f''(0)}{f'^2(0)} = 0$

Therefore, by last Theorem, if we choose $x_0 \in [\alpha - \delta, \alpha + \delta]$, then N.R. scheme is quadratically convergent

Exercises

Q1: Find the approximate roots of the following equation: $f(x) = x^3 - 8 = 0$, On the closed interval [0, 3].

Q2. For last **example 2.7** find the iterative errors for three steps.

Q3. Find the approximate root of the following equation $f(x) = cos\ x - xe^x = 0$, on the interval [0.25, 0.75], consider $x_0 = 0.5$

Q4: Find the approximate roots of the following equation $f(x) = 4x - x^2 - 2 = 0$ On [0, 1], by using fixed point method (for three iterative steps), and find the iterative errors at each step. What is the stop condition?

Q5: Find the approximate value of $\sqrt[3]{25}$ by using Bisection Algorithm (for three iterative steps) and find the absolute errors at each step.

Hint: consider that $f(x) = x^3 - 25 = 0, \quad 2 < x < 3, \quad \text{and the exact value } \sqrt[3]{25} = 2.9240$

Q6: Use Newton-Raphson algorithm to find the approximate roots of the following equation $f(x) = \frac{x}{\pi} + \cos(x) = 0, \quad$ with cosidering $x_0 = 3$, for three iterative steps, and find the absolute errors at each step.

Q7: Show that secant algorithm is quadratically convergent**,** while Bisection and False position algorithm are linearly convergent.

Chapter Three

The Numerical Solutions of Linear Systems

It is well known from the linear algebra that, that there are many methods used to find the exact solutions of linear systems, $Ax = b$, where $A \in R^{n\times n}$, $b \in R^{n\times 1}$, such as Gauss elimination or, Gauss-Gordan, or Kramer's method. But using these methods becomes so difficult when the dimension n, of the matrix A, is large. Therefore, we need to compute the solutions numerically by using computers.

In general, there are two types of numerical methods, which can be used to find the numerical solutions of linear systems: **direct methods** and **indirect methods**.

Before starting to study these methods, let us revise some equivalent algebraic facts of the linear system: $Ax = b$, $A \in R^{n\times n}$, $b \in R^{n\times 1}$

1- The homogenous system $Ax = 0$, has only zero solution, iff $|A| \neq 0$, which means A is nonsingular matrix.
2- The linear system $Ax = b$, has a unique solution, iff $|A| \neq 0$, which means A is nonsingular matrix.

From above, before to think about finding a numerical solution of a linear system, we need to make sure that the exact solution exists, which means we should check whether $|A| \neq 0$, or A is nonsingular.

Stability of Linear systems

In some linear systems, $Ax = b$, small change on the elements of A or b, leads to big changes on the solution , x, of this system. We call these types of systems ill condition systems, while if any small change on A or b does not make big change on the solution , x, in this case the system is called well condition.

Example 3.1:

$$x_1 + 2x_2 = 3$$

$$1.0001X_1 + 2X_2 = 3.0001$$

It is easy to show that the exact solution of this system is

$x=\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}=\begin{pmatrix} 1 \\ 1 \end{pmatrix}$

While, if we make small change on the matrix A as follows

$$x_1 + 2x_2 = 3$$

$$0.99999x_1 + 2x_2 = 3.0001$$

We can show that, the new exact solution becomes

$$\mathrm{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} -1 \\ 2 \end{pmatrix}$$

Therefore, we conclude that this system is not stable (ill condition), because small changes on **A** lead to big changes on the solution **X**.

Since when we compute the approximate solutions by using computers, we get some changes on the matrix ***A*** and the vector ***b,*** throughout iterative

steps. And that causes big errors on solutions for ill condition systems. Therefore, it is important to check whether the system is stable or not, before starting to compute the numerical solutions.

The condition number

Definition 3.1: let A^TA is a square matrix and let $\lambda_1, \lambda_2, \dots \dots, \lambda_n$ are the eigenvalues of *A*, then the norm of A and A^{-1} can be defined as follows,

$||A||_2 = \sqrt{|\lambda_L|}$,

$$||A^{-1}||_2 = \frac{1}{\sqrt{|\lambda_S|}}$$

Where λ_L and λ_S are the smallest and the biggest eigenvalues of A, respectively

And the condition number $of\ A\ is\ K(A) = \left||A^{-1}|\right|_2 \left||A|\right|_2 = \sqrt{\frac{|\lambda_L|}{|\lambda_S|}}$,

If K (A) is big then the system $Ax = b$ is stable, while, if $K(A)$ is small, the system is stable.

Example 3.2: Study the stability of the following linear system**.**

$$2x_1 + 6x_2 = 2$$

$$5x_1 + 3x_2 = 3$$

Solution: A=$\begin{bmatrix} 2 & 6 \\ 5 & 3 \end{bmatrix}$, $A^T = \begin{bmatrix} 2 & 5 \\ 6 & 3 \end{bmatrix}$

A$A^T = \begin{bmatrix} 40 & 28 \\ 28 & 34 \end{bmatrix}$

The eigenvalues of AA^T are 8.8397, 65.1603

=2. $=\sqrt{\frac{65.1603}{8.8397}}$ K(A) $=\sqrt{\frac{|\lambda_L|}{|\lambda_S|}}$

Since $K(A)$ is large, the system above is unstable.

Direct methods

These methods can be convergent very fast, but when the dimension of A is large, it is not recommended to use these methods because, we need to compute lots of mathematical operations, which means, the errors become bigger.

Next, we will study some of direct methods.

Gauss Elimination Algorithm

The idea of this method is to convert A in the linear system $Ax = b$, to upper U or lower matrix L. Thus, we get a lower or upper triangular system:

$$Ux = b_1 \text{ or } Lx = b_2$$

It is clear that, for solving lower triangular system we use **Forward substitutions** and for solving upper triangular system we use **backward substitutions**.

In fact, solving lower (upper) triangular system is easier than solving the original system.

Steps of Gauss elimination algorithm:

1- Write the system $Ax = b$, in the matrix form [A: b].
2- Convert [A: b] to appear triangular form $[L: b_1]$ or lower triangular from $[U: b_2]$
3- Solve the lower (upper) triangular system, $Lx = b_1$ ($Ux = b_2$), by using forward (backward) substitutions.

Example 3.3: Solve the following linear system by using Gauss algorithm

$$5x_1 + 2x_2 = 3$$
$$4x_1 + 3x_2 = 1$$

Solution: Firstly, we write the system in matrix form$[A: b]$, as follows

$$\begin{bmatrix} 5 & 2 & : & 3 \\ 4 & 3 & : & 1 \end{bmatrix}$$

$(L_1)/5$, $-4(L_1/5) + L_2$

$$\begin{bmatrix} 1 & 2/5 & : & 3/5 \\ 0 & 7/5 & : & -7/5 \end{bmatrix}$$

Thus, we get the following **upper triangular system**

$$x_1 + (2/5)x_2 = 3/5$$

$$(7/5)x_2 = -7/5$$

Finally, we solve the last system by using the backward substitutions, to get

$$x_2 = (-7/5)/(7/5) = -1$$
$$x_1 = 3/5 + 2/5 = 5/5 = 1$$

Thus, the solution is x= $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$

Gauss - Gordan algorithm

This algorithm is considered, as a modified to the Gauss elimination algorithm. We convert the linear system, $Ax = b$ to $I_n x = b_1$, where I_n is the identity matrix of order n. Thus to solve the last system we use the direct substitutions.

$$Ax=b \longrightarrow I_n x = b_1$$

Steps of Gauss - Gordan algorithm

1- Write the system $Ax = b$, in the matrix form [A: b].
2- Do some mathematical operations to convert [A: b] to the diagonal form $[I_n : b_1]$.
3- Find x by solving the system $I_n x = b_1$ using direct substitutions.

Example 3.4: Find the solution of the following system, by using Gauss-Gordan Method

$$x_1 + 2x_2 = 1$$
$$2x_1 + x_2 = 3$$

Solution: Firstly, we write the system in matrix form $[A: b]$, as follows:

$$\begin{bmatrix} 1 & 2 & : & 1 \\ 2 & 1 & : & 3 \end{bmatrix} \ldots\ldots (1)$$

$$-2(L_1) + L_2 \begin{bmatrix} 1 & 2 & : & 1 \\ 0 & -3 & : & 1 \end{bmatrix} \ldots\ldots.. (2)$$

$$-(L_2/3) \begin{bmatrix} 1 & 2 & : & 1 \\ 0 & 1 & : & -1/3 \end{bmatrix} \ldots\ldots (3)$$

$$-2(L_2) + L_1 \begin{bmatrix} 1 & 0 & : & 5/3 \\ 0 & 1 & : & -1/3 \end{bmatrix} \ldots\ldots (4)$$

Thus, we get

$$I_n x = b_1$$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 5/3 \\ -1/3 \end{bmatrix}$$

LU Algorithm

This method is called by LU, because the matrix A in the linear system $Ax = b$, decomposes into the multiplication of two matrices: lower L and upper U, and this decomposition works for any vector b**,** which means $A = LU$.

Thus, we get lower triangular and upper triangular systems.

In order to get the solution of the system $Ax = b$, we need to solve these two systems.

Steps of LU Method

1- Decompose A, in the form $A = LU$, where U is an upper matrix and L is a lower matrix.

2- Set $Ux = y$, which leads to $Ly = b$

3- Solve first the lower triangular system, Ly=b, using the *forward substitutions* to get y, and then solve the lower triangular system, Ux=y, using the *backward substitutions* to get *x.*

Remarks:

1- This method can be considered better than Gauss and Gauss - Gordan methods and that because the decomposition of the matrix A works for any vector b, while in Gauss and Gauss - Gordan, the mathematical operations, which we have to do on [A:b], should be redone again when we choose another vector b.

2- In fact, not any matrix A can be decomposed to LU, unless A is **diagonally dominant matrix**, which means the following condition is satisfied:

$$|a_{ii}| \geq \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}|, \text{ i=1, 2… n}$$

Example 3.5: Can we decompose the matrix A to U ? where A=$\begin{pmatrix} 3 & 1 & 1 \\ 5 & 6 & 0 \\ 0 & 1 & 7 \end{pmatrix}$

Solution:

$$2 = |a_{12}| + |a_{13}| \leq |a_{11}| = 3$$
$$5 = |a_{21}| + |a_{23}| \leq |a_{22}| = 6$$
$$1 = |a_{31}| + |a_{32}| \leq |a_{33}| = 7$$

Clearly, A is diagonally dominant matrix. Therefore, it follows that A can be decomposed to LU.

The following example shows how to use LU algorithm to solve a linear system.

Example 3.6: Use LU Algorithm to find the solution of the following system.

$$x_1 + 2x_2 = 3$$
$$3x_1 + x_2 = 5$$

Solution:

Set A=$\begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix}$=$\begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix}$ $\begin{bmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{bmatrix}$

L U

$u_{11} = 1, u_{12} = 2,\ 3u_{12} + u_{22} = 1$ $u_{22} = 1 - (3)(2) = -5$

It follows that

L=$\begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix}$,u = $\begin{bmatrix} 1 & 2 \\ 0 & -5 \end{bmatrix}$

Set $Ux = y$, $Ly = b$

We need to solve first the system $[Ly = b]$ by using Forward substitutions

$\begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \end{bmatrix}$, thus $y_1 = 3$, $y_2 = 5 - (3)(3) = -4$

Then y= $\begin{bmatrix} 3 \\ -4 \end{bmatrix}$

Secondly, we solve the system $[Ux = y]$, by using Backward substitutions

$\begin{bmatrix} 1 & 2 \\ 0 & -5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ -4 \end{bmatrix}$ $\quad x_2 = -\frac{4}{-5} = \frac{4}{5}$

$x_1 = 3 - 2(\frac{4}{5}) = 3 - \frac{8}{5} = \frac{7}{5}$

Thus $x = \begin{bmatrix} \frac{7}{5} \\ \frac{4}{5} \end{bmatrix}$

Indirect methods

In these methods, we don't need to do lots of matrix operations as in the direct methods, but it is known that indirect methods are slower than direct methods in convergence. Moreover, the main different between direct and indirect methods that indirect methods needs an initial solution, $x^0 = (x_1^0, x_2^0, \dots x_n^0)$, in order to start and depending on this initial condition, we can get $x^1, x^2, \dots \dots$

Therefore, indirect methods are also called the *iterative methods*.

We will study two algorithms of indirect methods: *Jacobi & Gauss-Sidel*

Jacobi iterative algorithm

Consider that, we have the following the linear system:

$$A_{n\times n}x = b_{n\times 1}$$

This can be written as follows:

$a_{11}x_1$+$a_{12}x_2$+…….+$a_{1n}x_n$=b_1

$a_{21}x_1$+$a_{22}x_2$+……+$a_{21}x_n$=b_2

$a_{n1}x_1$+$a_{n2}x_2$ +……..+$a_{nn}x_{n=b_n}$

Where $a_{\text{ii}} \neq 0, \quad i = 1,2, \dots . n$

Remarks:

1- In case of $a_{\text{ii}} = 0,$ for some i, we replace the equation number i, with another equation in order to get $a_{\text{ii}} \neq 0, \quad i = 1,2, \dots . n$

2- On the other hand, in order to get faster convergence, we should make sure that, A is **diagonally dominant matrix**:

i.e. $|a_{ii}| \geq \sum_{\substack{\text{j=1} \\ \text{j}\neq\text{i}}}^{n} |a_{ij}|$

In case of this condition is not satisfied, we can also switch places between the equations, until we get this condition is satisfied.

Steps of Jacobi algorithm

1- Make sure that the system has a unique solution $|A| \neq 0$.

2- Choose the initial solution $x^0 = (x_1^0, x_2^0, \ldots . x_n^0) \in R^n$

3- Make sure that $a_{ii} \neq 0, and\ |a_{ii}| \geq \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}|$ and in case of one of these conditions is not satisfied, switch places of the equations, until we get the two conditions are satisfied.

4- Write the system in iterative form: $x_i^{k+1} = (b_i - \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij}\ x_j^k)/$ a_{ij}, for $i = 1,2, \ldots . n$

5- Compute the solution iterative iteratively, (for $k = 0,1,2, \ldots .$), until we get the following the stop condition is satisfied:

$\left\|x^{(k+1)} - x^{(k)}\right\| < \in$, Where$E_k = \left\|x^{(k+1)} - x^{(k)}\right\| = \max_{1 \leq i \leq n} \left|x_i^{k+1} - x_i^k\right|$

Example 3.7: Use Jacobi algorithm to solve the following linear system, for two iterative steps and find the iterative error at each step.

Note: The exact solution for this system is x= (1; 2; 3)

$$\begin{aligned} 4x_1 + 2x_2 + x_3 &= 11 \\ 2x_1 + x_2 + 4x_3 &= 16 \\ -x_1 + 2x_2 \quad &= 3 \end{aligned}$$

Solution: Since $a_{33} = 0$, we need to switch the places of equation 2 and 3:

$$\begin{aligned} 4x_1 + 2x_2 + x_3 &= 11 \\ -x_1 + 2x_2 \quad &= 3 \\ 2x_1 + x_2 + 4x_3 &= 16 \end{aligned}$$

It is clear that the Matrix A is diagonally dominant.

New, we can rewrite the last system as follows:

$$x_1 = \frac{11}{4} - \frac{1}{2}x_2 - \frac{1}{4}x_3$$

$$x_2 = \frac{3}{2} + \frac{1}{2}x_1$$

$$x_3 = 4 - \frac{1}{2}x_1 - \frac{1}{2}x_2$$

We write the system in the iterative form as follows:

$$x_1^{k+1} = \frac{11}{4} - \frac{1}{2}x_2^{k} - \frac{1}{4}x_3^{k}$$

$$x_2^{k+1} = \frac{3}{2} + \frac{1}{2}x_1^{k} \quad ; k=0, 1\ldots$$

$$x_3^{k+1} = 4 - \frac{1}{2}x_1^{k} - \frac{1}{4}x_2^{k}$$

Let $x^0 = (x_1^0, x_2^0, x_3^0) = (0.8, 1.8, 2.8)$

Set k=0

$$x_1^{(1)} = \frac{11}{4} - \frac{1}{2}(1.8) - \frac{1}{4}(2.8) = 1.15$$

$$x_2^{(1)} = \frac{3}{2} + \frac{1}{2}(0.8) = 1.9$$

$$x_3^{(1)} = 4 - \frac{1}{2}(0.8) - \frac{1}{4}(1.8) = 3.15$$

$$x^{(1)} = (1.15, 1.9, 3.15)$$

$$E_1 = \max\{|1.15 - 0.8|, |1.9 - 1.8|, |3.15 - 2.8|\} = 0.35$$

Set $k = 1$, by using the same way obtain:

$x_1^{(2)} = \frac{11}{4} - \frac{1}{2}(1.9) - \frac{1}{4}(3.15) = 1.0125$,

$x_2^{(2)} = \frac{3}{2} + \frac{1}{2}(1.15) = 2.0750$,

$x_3^{(2)} = 4 - \frac{1}{2}(1.15) - \frac{1}{4}(1.9) = 2.95$

$x^{(2)} = (1.0125, 2.075, 2.95)$

$$E_2 = \max\{|1.0125 - 1.15|, |2.075 - 1.9|, |2.95 - 3.15|\} = 0.175$$

We continue iteratively, until the convergent condition can be satisfied:

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

Gauss – Sidel algorithm

The main difference between this method and Jacobi method, is for any iterative step ,k, the new approximate values of the x_j, $\mathrm{j} = 1, \ldots \mathrm{i} - 1,$ are used directly, to compute the approximate values of x_i, while in Jacobi we don't use the new approximate values x_j until, we consider the next iterative step, $k + 1$.

Remark: The steps of Gauss-Sidel algorithm are the same as the step of Jacobi method expect for in step (4), we use the Gauss-Sidel iterative system:

$$x_i^{k+1} = (\mathrm{bi} - \sum_{\substack{\mathrm{j}=1 \\ \mathrm{j}\neq\mathrm{i}}}^{i-1} \mathrm{a}_{ij}\ x_j^{k+1} - \sum_{\substack{\mathrm{j}=\mathrm{i} \\ \mathrm{j}\neq\mathrm{i}}}^{n} \mathrm{a}_{ij}\, x_j^k)\ /a_{ii}$$

For $i = 1,2,3, \ldots n, k = 0,1,2, \ldots\ldots$

Example 3.8: Use Gauss-Sidel algorithm to find the approximate solution to the following system for two iterative steps, with x^0=(0.8, 1.8, 2.8), and find the iterative errors at each step.

$$4x_1 + 2x_2 + x_3 = 11$$

$$-x_1 + 2x_2 = 3$$

$$2x_1 + x_2 + 4x_3 = 16$$

It is clear that, $a_{ii} \neq 0$ and A is diagonally dominant matrix.

We can rewrite the last system as follows:

$x_1 = \frac{11}{4} - \frac{1}{2}x_2 - \frac{1}{4}x_3$

$x_2 = \frac{3}{2} + \frac{1}{2}x_1$

$x_3 = 4 - \frac{1}{2}x_1 - \frac{1}{2}x_2$

We write the system in the iterative form as follows

$x_1^{k+1} = \frac{11}{4} - \frac{1}{2}x_2^k - \frac{1}{4}x_3^k$

$x_2^{k+1} = \frac{3}{2} + \frac{1}{2}x_1^{k+1}$; $k = 0,1, \ldots \ldots$

$x_3^{k+1} = 4 - \frac{1}{2}x_1^{k+1} - \frac{1}{4}x_2^{k+1}$

Set k=0, we get

$x_1^{(1)} = \frac{11}{4} - \frac{1}{2}(1.8) - \frac{1}{4}(2.8) = 1.15$

$x_2^{(1)} = \frac{3}{2} + \frac{1}{2}(1.15) = 2.075$

$x_3^{(1)}$ = 4 - $\frac{1}{2}$(1.15) - $\frac{1}{4}$(2.075) = 2.9063

$x^{(1)} = (1.15, 2.075, 2.9063)$

$$E_1 = \max\{\,|1.15 - 0.8|, |2.075 - 1.8|, |2.9063 - 2.8|\} = 0.35$$

Set k=1, we can, in the same way, compute

$x_1^{(2)}$= $\frac{11}{4}$ - $\frac{1}{2}$(2.075) - $\frac{1}{4}$(2.9063) =0.9859

$x_2^{(2)}$= $\frac{3}{2}$ + $\frac{1}{2}$(0.9859) =1.993

= $4 - \frac{1}{2}(0.9859) - \frac{1}{4}(1.9930) = 3.0088$

$x^{(2)} = (0.9859, 0.9859, \ 3.0088)$

$E_2 = \max\{\,|0.9859 - 1.15|, |1.993 - 2.075|, |3.0088 - 2.9063|\} = 0.1641$

We continue iteratively, until the convergent condition can be satisfied:

$$||x^{(k+1)} - x^k|| < \in \quad \text{or} \quad \left\|\mathrm{b} - \mathrm{A} * x^{(\mathrm{k})}\right\| < \in$$

Remark: From last example, we see that, the approximate results that we get by using Gauss-Sidel are more accurate and closer to the exact solution, compared with the results that we got by using Jacobi method. Moreover, for $k \geq 2$, the iterative errors those arise from using Gauss-Sidel method are less than the iterative errors those arise from using Gauss-Sidel. Which means Gauss-Sidel method is faster than Jacobi method in convergence.

Exercises

Q1: In the last **example 3.3**, try to change the system $Ax = b$, to a lower triangular system and find its solution by using the forward substitutions.

Q2: Consider that, we have the following linear system

$$\begin{aligned} x_1 - x_2 + x_3 &= 2 \\ 3x_1 + 3x_3 &= 0 \\ 2x_1 + 5x_2 + x_3 &= 1 \end{aligned}$$

Solve the system by using,

a) Gauss elimination (with Backward or Forward substitutions).

b) LU algorithm.

Q3: Consider that, we have the following linear system 4x4

$$\begin{aligned} 2x_1 + x_2 - x_3 + 5x_4 &= 0 \\ x_1 + 2x_3 - x_4 &= 2 \\ x_1 + 4x_2 + x_3 + x_4 &= 1 \\ 3x_1 + x_2 + x_3 - x_4 &= 3 \end{aligned}$$

a) Make sure that the matrix A is diagonally dominant.

b) Use **Gauss-Sidel** method to find the approximate solution for two iterative step ($x^{(1)}, x^{(2)}$), with considering $x^{(0)} = (\ 0\ ,\ -1\ ,\ \frac{1}{2}\ ,\ 2\ )$.

c) Compute the iterative errors, at each step.

d) What is the stop condition?

Q4: Consider that, we have the following linear system 3x3

$$\begin{aligned} ax_1 + bx_2 + c\,x_3 &= 2 \\ dx_1 \qquad\quad + ex_3 &= 8 \\ +f\,x_2 + g\,x_3 &= 1 \end{aligned}$$

a) Under which condition the above system has a unique solution, in terms of the elements of A?

b) Use **Gauss** Method, with Forward substitution, to find the solution of this system in terms of the elements of A.

Q5: Consider that, we have the following linear system

$$\begin{aligned} x_1 - x_2 + 2x_3 &= 2 \\ 3x_1 \qquad + 3x_3 &= 0 \\ 2x_1 + 5x_2 + x_3 &= 1 \end{aligned}$$

a) Does the system have a unique solution? Why?

b) Make sure that the matrix A is diagonally dominant matrix

c) Solve the system by using **LU** method.

Chapter Four

Interpolation, Extrapolation and Numerical Differentiations

Sometimes, we need to estimate an unknown value depending on known values (Data base), for instance, consider that, the numbers of people who have lived in Iraq is known, for the years 57, 67, 77,87, 97,2007, 2017, if we would like to estimate the number of Iraq's people in the year 75, this operation is called the interpolation, because the number 75 belongs to the interval [37, 87], while , if we would like to estimate the number of Iraqi people in the year 2018, this operation is called the extrapolation, because the number 2018 does not belong to the range of the data base.

Mathematical problems of interpolation and extrapolation

Assume that, we have the following database, where f is a continuous function on $[x_0, x_1]$

$y_i = f(x_i)$, $i = 1,2, \dots \dots . n$

x	x_0	x_1	x_2	.....	x_n
y	y_0	y_1	y_2	.....	y_n

$x_i = x_{i-1} + h$, or $x_i = x_0 + ih$, $i = 0,1,2, \dots . . n$

If the aim is to find$f(x^*)$, where $x^* \neq x_i)_{i=0}^{n}$, $x^* \in [x_0, x_n]$, then this operation is called the interpolation**,**

While, if the aim is to find $f(x^*)$, where $x^* \neq x_i)_{i=0}^{n}$, $x^* \notin [x_0, x_n]$,

Then, the operation is called **Extrapolation**

Next, we will study some numerical techniques that can be used to find the solutions for interpolation problems, when the distances between the points in the database are equal and non-equal.

Interpolation by Polynomials forms

This method depends on writing the unknown function f as a polynomial of order m.

$$\text{f(x)} \cong P_m(x) \ , \forall\, x \ \in [x_0, x_n],$$

where $P_m(x) = a_0 + a_1 x + a_2 x^2 \ldots\ldots + a_m x^{\text{m}}$

Such that the following condition is satisfied ***(*interpolation condition*)***

$$f(x_i) = P_m(x_i)_{\text{i=0}}^{\text{n}}$$

This means, $f(x^*) = P_m(x^*)$

$$|\text{f(x)} - P_m(x)| < \in, \qquad \forall x \in [x_0, x_n]$$

Steps of the algorithm

1- Input the values of $x = x_i)_{\text{i=0}}^{\text{n}}$, $\text{y} = y_i)_{\text{i=0}}^{\text{n}}$, and x^*
2- find the coefficients, a_m,……,a_1, a_0 using the relation $f(x_i) = P_m(x_i)_{\text{i=0.}}^{\text{n}}$
3- find the value of $f(x^*) = P_m(x^*)$

Example 4.1: suppose that, we have the following data base, which has different distant space

X	1	2	4	5
Y	0	3	15	24

Find the approximate value for $f(3)$, by using linear interpolation (m=1). Secondly by using quadratic interpolation (m=2).

Solution:

The linear form takes the following form $P_1(X) = a_0 + a_1 x$

Since the last polynomial has two unknown constant coefficients a_0, a_1, we need two value of x_i such that 3 located between them ($3 \in [2,4]$).

$$P_1(2) = f(2) = a_0 + 2a_1 = 3 \dots \dots \dots (1)$$

$$P_1(4) = f(4) = a_0 + 4a_1 = 15 \dots \dots \dots (2)$$

From equations (1) & (2), we get

$$a_0 = -9\,, \qquad a_1 = 6$$

$$P_1(X) = -9 + 6x$$

$$f(3) = P_1(3) = -9 + 6(3) = -8 + 18 = 10$$

Secondly, the quadratic polynomial takes the form

$$P_2(x) = a_0 + a_1 x + a_2 x^2$$

Since the last polynomial has three unknown constant coefficients a_0, a_1, a_2 we need three value of x_i such that 3 located between them. For instance we can choose 1, 2, 4.

It follows that

$$P_2(1) = f(1) = a_0 + a_1 + a_2 = 0$$

$$P_2(2) = f(2) = a_0 + 2a_1 + 4a_2 = 3$$

$$P_2(4) = f(4) = a_0 + 4a_1 + 16a_2 = 15$$

Thus, we get a linear system can be solved by using Gauss algorithm or Gauss-Jordan algorithm.

We get

$$a_0 = -1\,, a_1 = 0\,, a_2 = 1$$

Therefore

$$P_2(x) = -1 + x^2$$

$$f(3) \cong P_2(3) = -1 + (9) = 8$$

Remarks:

1- if we resolved the last example with taking the points 2,4,5, we would get the same result $f(3) = 8$

2- if we consider m=3, $P_3(x) = a_0 + a_1x + a_2x^2 + a_3x^3$ this means, in order to find the approximate value of $f(3)$, we need to find the coefficients, a_0, a_1, a_2, a_3 , therefore, we have to use all the points 1,2,4,5.

Moreover, it is clear that, we cannot consider P(x) is a polynomial of order m where $n < m$, because the numbers of coefficient will be larger than the numbers of the points in the data base.

In fact, the best approximation, when $m = n$ (the number of the points in the data base equal the order of the polynomial P(x)).

3- When the number of points in the data base, n, is large, it is difficult to use the method of polynomial, and that because we need to deal with large order matrices; therefore, in this case we have to think about another method which does not depend on matrices.

4- polynomial method can be considered as an extrapolation method, for instance in the last example we can find the approximate value of f(6), by using the relation

$$f(6) \cong P_2(6) = -1 + 36 = 35$$

Lagrange interpolation polynomial

Let f(x) is a real function defined & continuous on I= [a,b], and assume that we have the following data base $y_i = f(x_i)$, i=1,2,…….n

X	x_0	x_1	x_2	.....	x_n
Y	y_0	y_1	y_2	.....	y_n

Where $\{Xi\}_{i=0}^{n} \in I$.

Our aim is to find the polynomial Pn(x), which satisfies the **interpolation condition**

$$f(x_i) = P_m(x_i)_{i=0.}^{n}$$

It has been proved that, there exists only one polynomial of order less than or equal n, such that the **interpolation condition** is satisfied. **Lagrange** showed that this polynomial can be written as follows:

$$P_n(x) = \sum_{k=0}^{n} f(x_k) L_k(x)\,,$$

$$\mathrm{L_k(x)} = \prod_{\substack{\mathrm{i=0} \\ \mathrm{i \neq k}}}^{\mathrm{n}} \frac{(\mathrm{x - x_i})}{(\mathrm{x_k - x_i})} \qquad \mathrm{k = 0,1,2 \ldots \ldots . n}$$

This is called the Lagrange interpolation formula

Theorem 4.1: $f(x) = P_n(x) + T(x)$, where $T(x)$ is the truncation error formula, which takes the form $T(x) = \frac{f^{(n+1)}(\delta(x))}{(n+1)!} \prod_{i=0}^{n} (x - x_i)$, δ is a number depends on x

Remark: from last theorem, if f is a polynomial of degree less than or equal n, then $T(x) = 0$, which means $\mathrm{f(x) = P_n(x), \forall x \in [a, b]}$.

Steps of Lagrange algorithm

1- input the vectors $\mathrm{X = Xi)_{i=0}^{n}}$, $\mathrm{Y = Yi)_{i=0}^{n}}$

2- input x^*

3- find the approximate value of $\mathrm{f}(x^*)$, using **Lagrange formula**

$$\mathrm{f}(x^*) \cong \; P_n(x^*) = \sum_{k=0}^{n} f(x_k) L_k(x^*)$$

Example 4.2: Let f is a continuous function, find f(4.5), from the following data base, by using Lagrange algorithm

X	2	4	5
y	3	15	24

Solution:

Firstly, we write the general form of Lagrange formula

$$p_2(\mathrm{x}) = \sum_{k=0}^{2} f_{(xk)}\, L_k(\mathrm{x}) = f(x_0)L_0(x) + f(x_1)L_1(x) + f(x_2)L_2(x)$$

$$= \frac{(x-x1)(x-x2)}{(x0-x1)(x0-x2)}\mathrm{f}(x_0) + \frac{(x-x0)(x-x2)}{(x1-x0)(x1-x2)}\mathrm{f}(x_1) + \frac{(x-x0)(x-x1)}{(x2-x0)(x2-x1)}\mathrm{f}(x_2)$$

$$p_2(2) = \frac{(4.5-4)(4.5-5)}{(2-4)(2-5)}3 + \frac{(4.5-2)(4.5-5)}{(4-2)(4-5)}15$$

$$+ \frac{(4.5-2)(4.5-4)}{(5-2)(5-4)}24 = \frac{77}{4} = 19.25$$

Note that, we can get more accurate results if we increase the dimension of the data base.

Inverse Interpolation

$x^* \neq x_i)_{i=0}^{n}$, $x^* \in [x_0, x_n]$ Assume that f(x^*) is known, where f is a continuous function. If we would like to find the value of x^*, this operation is called, the inverse interpolation.

In order to find a solution for this problem, we can use the Lagrange algorithm, with switch places of ***x*** and ***y.***

i.e., we consider **y** is the independent variable and **x** is the dependent variable. Therefore, the inverse Lagrange formula takes the form

$$x^* = p_n(y) = \sum_{k=0}^{n} x_k L_k\,(\mathrm{y})$$

$$L_{k\,(y)} = \prod_{\substack{i=0 \\ i\neq k}}^{n} \frac{(y - yi)}{(y_k - yi)}$$

Example 4.3: Find the value of x^* such that $f(x) = 0$ (the root of f), from the following data base, by using Inverse Lagrange formula.

x	0	2	5
y	-1	3	24

$$p_2(\mathrm{x}) = \sum_{k=0}^{2} f_{(xk)}\, L_k(\mathrm{x}) = f(x_0)L_0(x) + f(x_1)L_1(x) + f(x_2)L_2(x)$$

$$x = \frac{(y-y1)(y-y2)}{(y0-y1)(y0-y2)}x_0 + \frac{(y-y0)(y-y2)}{(y1-y0)(y1-y2)}x_1 + \frac{(y-y0)(y-y1)}{(y2-y0)(y2-y1)}x_2$$

$$x^* = \frac{(0-3)(0-24)}{(-1-3)(-1-24)}(0) + \frac{\big(0-(-1)\big)(0-24)}{\big(3-(-1)\big)(3-24)}(2) + \frac{\big(0-(-1)\big)(0-3)}{\big(24-(-1)\big)(24-3)}(5)$$

$$x^* = 1$$

Remarks:

1- In the last example, it is clear from the data base that the root of f belongs to (0,2), and that because the sign of $f(0)$ is negative and the sign of $f(2)$ is positive .

2- From the last example we conclude that, we can find the approximate root of $f(x) = 0$, when the form of the function f is unknown, and we only have the values of f at some points.

Interpolation by Finite Differences

As we have seen, the Lagrange interpolation method can be used even though, the distances between the points in the data base are not equal, for example $X = 2,5,7,12$.

While the finite difference methods, which we will study, can be used only when the distances are equal.

For example $X = 2,4,6,8,10$.

Therefore, this can be considered as advantage of using Lagrange method. On the other hand, we can point out some disadvantage of using Lagrange method,

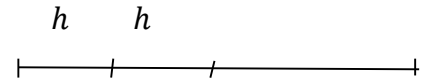
1- Lots of mathematical operations.
2- If we would like to add later other data, to the data base, we have to restart from the beginning, which means we waste time. While in finite difference methods, we can add any other data later.

Finite Difference Operators

The finite difference operators can be classified into three types:

Forward Δ, Backward ∇ and **Central δ.**

In order to understand the difference operators, we divide the interval $[a, b]$, as follows:



$x_{i+1} - x_i = h$

Where h, is the distance between any two points.

Δ Forward difference operator

The Forward difference operator is defined as follows:

$$\Delta y_i = \Delta f(x_i) = f(x_i + h) - f(x_i)$$

$$= f(x_{i+1}) - f(x_i) \qquad i = 0,1,2, \ldots.$$

$$= y_{i+1} - y_i$$

$$\Delta^2 y_i = \Delta^2 f(x_i) = \Delta\big(\Delta f(xi)\big) = \Delta\big(f(x_{i+1}) - f(x_i)\big)$$
$$= \Delta f(x_{i+1}) - \Delta f(x_i)$$
$$= f(x_{i+2}) - f(x_{i+1}) - f(x_{i+1}) + f(x_i) = y_{i+2} - 2y_{i+1} + y_i$$

By the same way, we can find $\Delta^3 y_i$

∇ Backword difference operator

The Backward difference operator is defined as follows:

$\nabla y_i = \nabla f(x_i) = f(x_i) - f(x_{i-1}),\ \ i = 1,2, \ldots\ldots, n$

$= y_i - y_{i-1}$

$$\nabla^2 y_i = \nabla(\nabla y_i) = \nabla(y_i - y_{i-1})$$

$$= \nabla y_i - \nabla y_{i-1}$$

$= y_i - y_{i-1} - y_{i-1} + y_{i-2}$

$= y_i - 2y_{i-1} + y_{i-2}$

δ Central difference operator

The central difference operator is defined as follows:

$$\delta y_i = \delta f(x_i) = \mathrm{f}\left(x_i + \frac{h}{2}\right) - \mathrm{f}\left(x_i - \frac{h}{2}\right) = y_{i+\frac{1}{2}} - y_{i-\frac{1}{2}}$$

$$\delta^2 y_i = \delta\left(y_{i+\frac{1}{2}} - y_{i-\frac{1}{2}}\right) = \delta\, y_{i+\frac{1}{2}} - \delta y_{i-\frac{1}{2}}$$

$= y_{i+1} - y_i - y_i + y_{i-1}$

$= y_{i+1} - 2y_i + y_{i-1}$

Newton's formulas by using (Forward, Backward, Central) Finite Difference Operators

Consider, we have the following data base:

Where $x_i = x_0 + ih, \quad h = \frac{x_n - x_0}{n}$

X	x_0	x_1	.....	x_n
Y	y_0	y_1	.....	y_n

Our aim is to find $f(x_p)$, where $0 < p < 1$, $x_p = x_0 + ph$

If x_p close to the beginning of the database ($x_0 < x_p < x_1$), then we use Newton *Forward formula*, which takes the form:

$$f(x_p) = y_0 + p\Delta y_0 + \frac{p(p-1)}{2!}\Delta^{2y_0} + \cdots$$

While, if x_p close to the end of the data base, ($x_{n-1} < x_p < x_n$), then we use Newton *Backward formula*, which takes the form:

$$f(x_p) = y_n + p\nabla y_n + \frac{p(p+1)}{2!}\nabla^{2y_n} + \cdots$$

Where, $x_p = x_n + ph$

If x_p close to the centre of the database, (x_i), then we use Newton *central formula*, which takes the form:

$$f(x_p) = py_{i+1} + \frac{(p^3 - p)}{6}\delta^2 y_{i+1} + qy_i + \frac{(q^3 - q)}{6}\delta^2 y_i$$

Where $x_p = x_i + ph$, $p + q = 1$

Steps of Newton's formulas:

1- Input the values of $x = x_i)_{i=0}^{n}, y = y_i)_{i=0,}^{n}$

2- Input x_p

3- set $h = x_1 - x_0$

4- If x_p is close to x_0, and set $p = \frac{x_p - X0}{h}$, compute $\Delta^2 y_0, \Delta y_0$ and use Newton's forward formula, to find $f(x_p)$,

If x_p close to x_n , set $p = \frac{x_p - Xn}{h}$, compute $\nabla^2 y_0, \nabla y_0$, and use Newton's Backward formula, to find $f(x_p)$,

If $x_i \leq x_p \leq x_{i+1}$, where $i \neq 1, i+1 \neq n$, set $p = \frac{x_p - x_i}{h}$, compute $\delta^2 y_i, \delta^2 y_{i+1}$

And use Newton's central formula, to find $f(x_p)$.

Example 4.4: consider the following data base

X	4	6	8	10
$f(x)$	1	3	8	20

Find the approximate value for each of the following: 1- $f(4.5)$ 2- $f(6.4)$ 3- $f(9)$

Solution:

1- Since 4.5 close to the beginning of the data base, we use the forward Newton formula

$x_p = x_0 + ph \quad \Longrightarrow \quad P = \frac{x_p - x_0}{h} = \frac{4.5-4}{2} = \frac{0.5}{2} = 0.25$

$$f(x_p) = y_p = y_0 + p\Delta y_0 + \frac{p(p-1)}{2!}\Delta^2 y_0$$

$$\Delta y_0 = y_1 - y_0 = 3 - 1 = 2$$

$$\Delta^2 y_0 = y_2 - 2y_1 + y_0 = 8 - 2(3) + 1 = 3$$

Or for simplicity $\Delta y_0, \Delta^2 y_0$ can be also found from the following figure

x	y	Δ	Δ^2
4	1		
		$\longrightarrow$	$2 = \Delta y_0$
6	3	$\longrightarrow$	$3 = \Delta^2 y_0$
	$\longrightarrow$	5	
8	8	$\longrightarrow$	$7 = \Delta^2 y_1$
	$\longrightarrow$	12	
10	20		

Thus

$f(4.5) = 1 + \frac{1}{4}(2) + \left(\frac{1}{4}\right)\left(-\frac{3}{4}\right)\frac{(3)}{2!} = \frac{39}{32}$= 1.2188

2- Since the point 9 close to the end of the database ($x_3 = 10$), we use the Backward Newton formula,

Set, $x_p = x_n + ph$

$$p = \frac{x_p - x_n}{h} = \frac{9 - 10}{2} = -\frac{1}{2}$$

$$f(x_p) = y_p = y_n + p\nabla y_n + \frac{p(p+1)}{2!}\nabla^2 y_n$$

$$\nabla y_n = y_n - y_{n-1} = 20 - 8 = 12$$
$$\nabla^2 y_n = y_n - 2y_{n-1} + y_{n-2} = 20 - 2(8) + 3 = 7$$

Or for simplicity ∇y_3 , $\nabla^2 y_3$ can be also found from the following figure

x y ∇ ∇^2

4 1

$\longrightarrow$ 2

6 3 $\longrightarrow$ $3 = \nabla^2 y_2$

$\longrightarrow$ 5

8 8 $\longrightarrow$ $7 = \nabla^2 y_3$

$\longrightarrow$ $12 = \nabla y_3$

10 20

Thus $f(9) = 20 + (-\frac{1}{2})(12) + \frac{\left(-\frac{1}{2}\right)\left(\frac{1}{2}\right)}{2}(7) =$

Finally, since 6.4 close to the middle of the database (between x_1 & x_2) , we use the Central Newton formula

$x_p = x_1 + ph$

$p = \frac{x_p - x_1}{h} = \frac{6.4-6}{2} = 0.2$, $q = 1 - p = 0.8$

$$f(x_p) = py_2 + \frac{(p^3 - p)}{6}\delta^2 y_2 + qy_1 + \frac{(q^3 - q)}{6}\delta^2 y_1$$

$$\delta^2 y_1 = y_2 - 2y_1 + y_0 = 8 - 2(3) + 1 = 3$$

$$\delta^2 y_2 = y_3 - 2y_2 + y_1 = 20 - 2(8) + 3 = 7$$

For simplicity, $\delta^2 y_1, \delta^2 y_2$ can be found from the following figure

x y δ δ^2

4 1

$\longrightarrow$ 2

6 3 $\longrightarrow$ $3 = \delta^2 y_1$

$\longrightarrow$ 5

8 8 $\longrightarrow$ $7 = \delta^2 y_2$

$\longrightarrow$ 12

10 20

Thus

$$f(6.4) = (0.2)(8) + \frac{[(0.2)^3 - 0.2]}{6}(7) + (0.8)(3) + \frac{[(0.8)^3 - 0.8]}{6}(3) = 3.6320$$

Interpolation by using divided finite differences

Newton's finite difference methods, which we have studied, cannot be used unless the spaces between any two points are the same. Therefore, we will study the divided finite difference, which can be used for any data base.

Consider the following data base:

x	x_0	x_1	.....	x_n
y	y_0	y_1	.....	y_n

Our aim is to find f (x_m), where $x_m \neq x_i)_{i=0}^{n}$, and whether x_m close or far from the beginning or the end of the data base, that does not make difference.

The general form of the Newton's equation with using divided finite difference, takes the following form:

$$f(x_m) = f(x_0) + (x_m - x_0)\mathrm{D}y_0 + (x_m - x_0)\,(x_m - x_1)\,\mathrm{D}^2y_0 + (x_m - x_0)\,(x_m - x_1)\,(x_m - x_2)\,\mathrm{D}^3y_0$$

Where $\mathrm{D}y_0, \mathrm{D}^2y_0, \mathrm{D}^3y_0$ can be found as follows:

$\mathrm{D}y_0 = (\frac{y_1 - y_0}{x_1 - x_0})$, $\mathrm{D}y_1 = (\frac{y_2 - y_1}{x_2 - x_1})$

$\mathrm{D}^2y_0 = (\frac{\mathrm{D}y_1 - \mathrm{D}y_0}{x_2 - x_0})$, $\mathrm{D}^2y_1 = (\frac{\mathrm{D}y_2 - \mathrm{D}y_1}{x_3 - x_1})$

$$\mathrm{D}^3y_0 = (\frac{\mathrm{D}^2y_1 - \mathrm{D}^2y_0}{x_3 - x_0})$$

Steps of Newton's algorithm of divided finite difference.

1- input $x = x_i)_{i=0}^{n}$, $y = y_i)_{i=0,}^{n}$

2- input x_m

3- find Dy_0, D^2y_0, D^3y_0

4- use Newton's formula of divided finite difference

$$f(x_m) = f(x_0) + (x_m - x_0)Dy_0 + (x_m - x_0)(x_m - x_1)D^2y_0 + (x_m - x_0)(x_m - x_1)(x_m - x_2)D^3y_0$$

Example 4.5: Find the approximate value of f (2), from the following data base**.**

X	0	1	4	6
Y	-10	20	14	30

X Y D D^2 D^3

x_0 y_0 $\longrightarrow Dy_0 = \left(\frac{y_1 - y_0}{x_1 - x_0}\right) = 30$

x_1 y_1 $\longrightarrow D^2y_0 = \left(\frac{Dy_1 - Dy_0}{x_2 - x_0}\right) = -8$

$\longrightarrow Dy_1 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right) = -2$ $\longrightarrow D^3y_0 = \left(\frac{D^2y_1 - D^2y_0}{x_3 - x_0}\right) = \frac{10}{6}$

x_2 y_2 $\longrightarrow D^2y_1 = \left(\frac{Dy_2 - Dy_1}{x_3 - x_1}\right) = 2$

$\longrightarrow Dy_2 = \left(\frac{y_3 - y_2}{x_3 - x_2}\right) = 8$

x_3 y_3

$$f(x_m) = f(x_0) + (x_m - x_0)\mathrm{D}y_0 + (x_m - x_0)\,(x_m - x_1)\,\mathrm{D}^2y_0 + (x_m - x_0)\,(x_m - x_1)\,(x_m - x_2)\,\mathrm{D}^3y_0$$

$$f(2) = -10 + (2-0)(30) + (2-0)(2-1)(-8) + (2-0)(2-1)(2-4)(\frac{10}{6}) = 27.33$$

Numerical Differentiations

Let f, be a differentiable function on $[a, b]$, and let $x_i)_0^n \in [a, b]$, and $f(x_i)\ ]_0^n$ are known. So, we have the following database

x	x_0	x_1 …… x_n
y	y_0	y_1 …… …… y_n

Let $x^* \in [a, b]$

Our aim is to find the derivative of f at x^*, which is $f'(x^*)$

Here, we have to study two cases, first when the distances between the points in the data base are not equal. In this case we can use Lagrange formula to find f(x) as follows

$$\mathrm{f(x)} \cong P_n(x) = \sum_{k=0}^{n} f(x_k)L_k(x)$$

$$f'(\mathrm{x}) \cong p_n'(x) = \sum_{k=0}^{n} f(x_k)L'_k(x)$$

Example 4.6: Find the approximate value of $f'(2.3)$, from the following data base.

X	1.1	1.7	3
Y	10.6	5.2	20.3

Solution: Since the spaces between the points are not equal, we use Lagrange algorithm

$$f(x) \cong p_2(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x2)}f(x_0) + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}f(x_1) + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}f(x_2)$$

Thus

$$f'(x) \cong p_2'(x) = \frac{(x-x_1)+(x-x_2)}{(x_0-x_1)(x_0-x2)}f(x_0) + \frac{(x-x_0)+(x-x_2)}{(x_1-x_0)(x_1-x_2)}f(x_1) + \frac{(x-x_0)+(x-x_1)}{(x_2-x_0)(x_2-x_1)}f(x_2)$$

Thus $f'(2.3) = (\quad)$

Remark: as we have mentioned before, Lagrange method needs lots of calculations, moreover, in case of, we add more data to the data base, we need to start from the beginning, therefore, we have to think about another method to find, $f'(x^*)$ which is Newton's method with divided finite differences**,** as follows

$$f(x) = f(x_0) + (x - x_0)\mathrm{D}y_0 + (x - x_0)\,(x - x_1)\,\mathrm{D}^2y_0 + (x - x_0)\,(x - x_1)\,(x - x_2)\,\mathrm{D}^3y_0$$

$$f'(x) = \mathrm{D}y_0 + [(x - x_0) + (x - x_1)]\,\mathrm{D}^2y_0 + [(x - x_0)(x - x_1) + (x - x_2)\big((x - x_0) + (x - x_1)\big)]\,\mathrm{D}^3y_0$$

Example 4.7: Find $f'(2)$, from the following data base

x	0	1	4	6
y	-10	20	14	30

Going back to previous example, we have

$$\mathrm{f(x)} \cong pn(x) = -10 + 30x - 8x(x-1) + \frac{10}{6}x(x-1)(x-4)$$

$$\mathrm{f(x)} \cong pn(x) = -10 + 30x - 8x^2 + 8x + \frac{10}{6}(x^3 - 5x^2 + 4x)$$

$$f'(x) \cong p'n(x) = 30 - 16x + 8 + \frac{10}{6}(3x^2 - 10x + 4)$$

$$f'(2) = 30 - 32 + 8 + \frac{10}{6}(12 - 20 + 4) = -0.6667$$

Using Finite difference operators to find derivatives

In case of the distances between the points, $x_i)_0^n$, are equal, $x_{i+1} - x_i = h$, we can also use Forward, Backward, and Central Newton's formulas, to find $f'(x^*)$, we will study two cases :

Case1: Interpolation Problems

If $x^* \neq x_i)_{i=0}^n$, $x^* \in [x_0, x_n]$, We use Newton's finite difference formulas as follows:

Set $x^* = x_p = x_j + ph$, where

$$j = \begin{Bmatrix} 0 & in\ Forward\ \ formula, \\ n & in\ Backward\ \ formula \\ 1 \leq j \leq n-1 & in\ Central\ \ formula \end{Bmatrix}$$

So,

$$\frac{dp}{dx_p} = \frac{1}{h} \quad \Longrightarrow \quad p = \frac{x_p - x_j}{h}$$

$$f'(x_p) = \frac{df}{dx_p} = \frac{df}{dp}.\frac{dp}{dx_p} = \frac{df}{dp}\left(\frac{1}{h}\right)$$

$$f'(x_p) = \left(\frac{1}{h}\right)\frac{df(x_p)}{dp}$$

Therefore, Forward, Backward and Central Newton formulas for differentiation take the forms, respectively:

$f'(x_p) = \left(\frac{1}{h}\right)\frac{df(x_p)}{dp} = \frac{1}{h}(\Delta y_0 + \frac{(2p-1)}{2}\Delta^2 y_0)$,

$f'(x_p) = \left(\frac{1}{h}\right)\frac{df(x_p)}{dp} = \frac{1}{h}(\nabla y_n + \frac{(2p+1)}{2}\nabla^2 y_n)$,

$$f'(x_p) = \left(\frac{1}{h}\right)\frac{df(x_p)}{dp}$$

$$= y_{i+1} + \frac{(3p^2-1)}{6}\delta^2 y_{i+1} - y_i + \frac{[-3(1-p)^2+1]}{6}\delta^2 y_i$$

Example 4.8: Going back the example that we have considered before, find $f'(4.5), f'(9), f'(6.4)$ by using (forward, backward, central) Newton formulas.

Solution: Since 4.5 close to the beginning of the data base, we use forward formula to find $f'(4.5)$

$$p = \frac{x_p - x_0}{h} = \frac{1}{4}, \quad h = 2$$

$$f'(x_p) = \frac{1}{h}(\Delta y_0 + \frac{(2p-1)}{2!}\Delta^2 y_0)\,,$$

Where $\Delta y_0 = 2, \quad \Delta^2 y_0 = 3$

$$\therefore \quad f'(4.5) = \frac{1}{2}\left(2 + \left(\frac{-0.5}{2}\right)3\right) = 0.6250$$

Case2: Computing the Derivatives at the Points of Database

Consider that, we have the following data base:

x	x_0	x_1 …… x_n
y	y_0	y_1 …… …… y_n

Where $x_{i+1} - x_i = h, \quad \forall\, i = 0,1, \ldots n-1$

The problem is: how to find the approximate values for $f'(x_i), f''(x_i)$, for $i = 0,1, \ldots . n$

It is well known that,

$$f'(x_i) = \lim_{h\to 0} \frac{f(x_i + h) - f(x_i)}{h} = \frac{\Delta f(x_i)}{h}$$

Thus, for small values of h

$$f'(x_i) \cong \frac{\Delta f(x_i)}{h} \quad , \quad i = 0,1, \dots n-1 \quad \dots\dots\dots (1)$$

Or $$f'(x_i) = \lim_{-h\to 0} \frac{f(x_i - h) - f(x_i)}{-h} = \frac{f(x_i) - f(x_i - h)}{h} = \frac{\nabla f(x_i)}{h}$$

For small values of h, we have

$$f'(x_i) \cong \frac{\nabla f(x_i)}{h}, \quad i = 1,2, \dots n \quad \dots\dots\dots (2)$$

From (1) & (2), for small values of h, we get

$$f'(x_i) \cong \frac{f(x_{i+1}) - f(x_{i-1})}{2h}, \qquad i = 1,2, \dots n-1 \quad \dots\dots (3)$$

Remarks:

1- Equation (3) is more accurate than (1) & (2), therefore, we will use it to find $f'(x_i)$, for $1 \le i \le n-1$, while, we can only use equation (1) & (2) to find $f'(x_0)$ & $f'(x_n)$, respectively.

2- The formula (3) can be used to find $f'(x_i)$ even when $f(x_i)$ is unknown, therefore, it can be considered an interpolation formula.

Next, our aim is to derive a formula, which can be used to find the second derivatives, $f''(x_i)$, $1 \le i \le n-1$

From Taylor series, we have

$$f(x_i + h) = f(x_i) + hf'(x_i) + \frac{h^2}{2} f''(x_i) + \cdots \dots\dots\dots (1)$$

And

$$f(x_i - h) = f(x_i) - hf'(x_i) + \frac{h^2}{2}f''(x_i) + \cdots \ldots \ldots \ldots (1)$$

From (1) & (2) , we get

$f''(x_i) = \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{h^2}$, for $1 \le i \le n-1 \ldots \ldots \ldots \ldots (4)$

Example 4.9: Consider that, we have the following data base

X	0	0.2	0.4	0.6
$f(x)$	1	1.04	1.16	1.36

Find the approximate values for:

1- $f'(x_i),\ 0 \le i \le n$, & $f''(x_i),\ 1 \le i \le n-1$

2- $f'(0.1),\ f'(0.3)$

Solution:

$$f'(x_0) \cong \frac{\Delta f(x_0)}{h} = \frac{f(x_1) - f(x_0)}{h} = \frac{1.04 - 1}{0.2} = 0.2$$

$$f'(x_1) \cong \frac{f(x_2) - f(x_0)}{2h} = \frac{1.16 - 1}{0.4} = 0.4$$

$$f'(x_2) \cong \frac{f(x_3) - f(x_1)}{2h} = \frac{1.36 - 1.04}{0.4} = 0.8$$

$$f'(x_3) \cong \frac{\nabla f(x_3)}{h} = \frac{f(x_3) - f(x_2)}{h} = \frac{1.36 - 1.16}{0.2} = 1$$

$$f''(x_1) = \frac{f(x_2) - 2f(x_1) + f(x_0)}{h^2} = \frac{1.16 - 2(1.04) + 1}{0.04} = 2$$

$$f''(x_2) = \frac{f(x_3) - 2f(x_2) + f(x_1)}{h^2} = \frac{1.36 - 2(1.16) + 1.04}{0.04} = 2$$

To compute, $f'(0.1), f'(0.3)$ assume now, $h = 0.1$

$$f'(0.1) \cong \frac{f(0.2) - f(0)}{2(0.1)} = \frac{1.04 - 1}{0.2} = 0.2$$

$$f'(0.3) \cong \frac{f(0.4) - f(0.2)}{2(0.1)} = \frac{1.16 - 1.04}{0.2} = 0.6$$

Exercises

Q1: Consider that, we have the following data base

x	0.2	0.4	0.6	0.8	1
y	1.4	1.8	2.2	2.6	3

a) Find a, b , where $y = ax + b$,

b) Use Newton's formulas to find the approximate values of $y(0.25)$, $y(\ 0.7)$, $y(0.5)$, and $y'(\ 0.9)$.

c) Compute the absolute errors at each point.

Q2: in the same way **(for example 4.8)** we can find $f'(9)$ and $f'(6.4)$, by using the backward and central formulas, respectively.

Q3:- Consider that, we have the following data base

x	0.25	0.5	0.75	1	1.25
y	1	1.4	1.6	2	2.5

Find the approximate values for $y'(0.5)$, $y''(0.75)$, $y'(0.625)$

Chapter Five

Numerical integration

In this chapter, we study some methods; used to find the approximate value for the definite following integral

$$\int_a^b f(x)dx \ \ (\forall x \in [a,b] \ \ f \in C[a,b]) \ ,$$

when it's difficult to find the exact value by using known methods (integration methods), such as:

$$\int_a^b e^{x^2} dx$$

The general idea of the integration methods is to divide the interval $[a,b]$, into n of subintervals:

$$[a,b] = [x_0,x_1] \ \cup [x_1,x_2] \ldots\ldots \cup [x_{n-1},x_n].$$

It is not needed to be the distances between the points $\{x_i\}$ are equal.

Next, we consider the polynomial $p_n(\mathrm{x})$ as an approximate form for *f*.

$f(x) \cong p_n(x), \ \ \forall\, x \in [a,b]$

which means the problem becomes

$$\int_a^b f(x)dx \cong \int_a^b p_n(x)\, dx$$

From the last form, we note that, the formula of numerical integration depends on the way of choosing the polynomial p_n.

The general formula of integration methods takes the form:

$$\int_a^b f(x)dx \cong \int_a^b p_n(x)\,dx = \sum_{i=0}^{n} a_i\, f(xi) \dots\dots\dots (*)$$

Where are called the coefficients $\{a_i\}_{i=0}^n$

$\{x_i\}_{i=0}^n$ are called the nodes.

If we used Lagrange polynomial, we could very easy get $\{a_i\}_{i=0}^n$, $\{x_i\}_{i=0}^n$ as follows:

We divide the interval $[a, b]$, to the n of subintervals

$$[a, b] = [x_0, x_1] \cup [x_1, x_2] \dots\dots \cup [x_{n-1}, x_n],$$

Where the distances between the nodes $\{\text{xi}\}_{i=0}^n$ are equal

i.e. $x_{i+1} - x_i = h$

Thus $f(x) = p_n(x) + T(x),\ \forall\, x \in [a, b]$,

Where p_n is the Lagrange polynomial

$\int_a^b f(x)dx = \int_{x_0}^{x_n} p_n(x)\,dx = \int_{x_0}^{x_n} \sum_{i=0}^{n} L_i(x) f(x_i)dx +$

$\int_{x_0}^{x_n} \frac{f^{(n+1)}(\delta(x))}{(n+1)!} \prod_{i=0}^{n}(x - x_i)dx,$

Where $\int_{x_0}^{x_n} \frac{f^{(n+1)}(\delta(x))}{(n+1)!} \prod_{i=0}^{n}(x - x_i)dx,$ is the truncation error formula.

So, $\int_a^b f(x)dx \cong \sum_{i=0}^{n}(\int_{x_0}^{x_n} L_i(x)dx) f(x_i))$ …… (1),

Which means

$$a_i = (\int_{x_0}^{x_n} L_i(x)dx)$$

In the last formula,

For n=1, method is called (**Trapezoidal method**),

For n=2, method is called (**Simpson method**).

Trapezoidal method

From the general form of integration, with choosing Lagrange polynomial, and n=1, we get

$h = x_1 - x_0$,$b = x_1$, $a = x_0$

$$\int_a^b f(x)dx = \sum_{i=0}^{n} (\int_{x_0}^{x_n} L_i(x)dx) f(x_i)dx + \int_{x_0}^{x_1} \frac{f''(\delta(x))}{2}(x - x_0)(x - x_1)dx$$

Set, $a_0 = \int_{x_0}^{x_1} L_0(x)dx = \int_{x_0}^{x_1} \frac{(x-x_1)}{(x_0-x_1)}dx = \frac{(x-x_1)^2}{2(x_0-x_1)} = |_{x_0}^{x_1} = \frac{h}{2}$

$$a_1 = \int_{x_0}^{x_1} L_1(x)dx = \int_{x_0}^{x_1} \frac{(x - x_0)}{(x_1 - x_0)}dx = \frac{(x - x_0)^2}{2(x_1 - x_0)} = |_{x_0}^{x_1} = \frac{h}{2}$$

Substitute a_0, a_1 in equation (1), we get

$$\int_a^b f(x)dx \cong \sum_{i=0}^{1} a_i\, f(x_i) = \frac{h}{2}f(x_0) + \frac{h}{2}f(x_1),$$

Which is the **Trapezoidal formula.**

Where the truncation error formula takes the form

$$T(h) = \int_{x_0}^{x_1} \frac{f''(\delta(x))}{2}(x - x_0)(x - x_1)\mathrm{dx} = -\frac{h^3}{12}f''(\delta),$$

Remark: We note that, if f is polynomial of order less than or equal one, then the truncation error equal zero, which means:

$$\int_a^b f(x)dx = \sum_{i=0}^{1} a_i\, f(x_i) = \frac{h}{2}f(x_0) + \frac{h}{2}f(x_1),$$

Example 5.1: Use the Trapezoidal method to find the approximate value of the following integral

$$\int_a^b (x^3 + 1)\, dx$$

Solution: $n = 1$, $x_0 = 0$, $x_1 = 1$, $h = \frac{x_1 - x_0}{1} = 1$

$$\int_a^b f(x)dx \cong \frac{1}{2}\big(f(x_0) + f(x_1)\big)$$

$$I_n = \int_a^b (x^3 + 1)\, dx \cong \frac{1}{2}\big((x^3 + 1)|_0 + (x^3 + 1)|_1\big) = \frac{1}{2}(1 + 2) = \frac{3}{2} = 1.5$$

While, we can find the exact value as follows:

$$I_e = \int_a^b (x^3 + 1)\, dx = \left(\frac{x^4}{4} + x\right)|_0^1 = \frac{1}{4} + 1 = 1.25$$

Thus, the absolute error is

$$E = |I_n - I_e| = |1.5 - 1.25| = 0.25$$

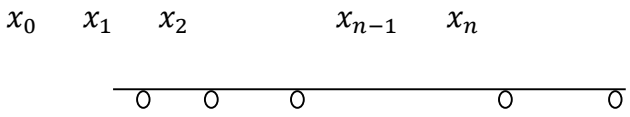
In order to get more accurate value to the integration, we use the composite Trapezoidal methods.

Composite Trapezoidal methods

The idea is, we divide the interval [a, b] into *n* of subintervals

$$[a, b] = [x_0, x_1] \cup [x_1, x_2] \dots \dots \cup [x_{n-1}, x_n],$$

Since the summation of all the integrals on the subintervals is equal the integral on the whole interval $[a, b]$, we can apply the Trapezoidal formula, on each of the integrals as follows:



$$I_N = I_1 + I_2 + \cdots I_n$$

$$\int_a^b f(x)dx = \int_{x_0}^{x_1} f(x)dx + \int_{x_1}^{x_2} f(x)dx + \cdots + \int_{x_{n-1}}^{x_n} f(x)dx$$

$$= [\frac{h}{2}\big(f(x_0) + f(x_1)\big)] + [\frac{h}{2}\big(f(x_1) + f(x_2)\big)] + \cdots$$

$$+ [\frac{h}{2}\big(f(x_{n-1}) + f(x_n)\big) - \sum_{i=1}^{n} \frac{h^3}{12} f''(\delta_i),$$

where $x_{i-1} \leq \delta_i \leq x_i$

$$\therefore \int_a^b f(x)dx \cong \left[\frac{h}{2}\left(f(x_0)+f(x_n)\right)\right] + h\sum_{i=2}^{n} f(x_{i-1})$$

The last equation is called, the **Composite Trapezoidal Formula.**

Steps of composite Trapezoidal algorithm

1- Input a, b

2- Input the number of partitions *n*

3- Define the function $f(x)$

4- Find $h = \frac{b-a}{n}$

- Use composite Trapezoidal formula

$$\int_a^b f(x)dx \cong \left[\frac{h}{2}\left(f(x_0)+f(x_n)\right)\right] + h\sum_{i=2}^{n} f(x_{i-1})$$

Example 5.2: For the last example, find the approximate value of the integral, using composite Trapezoidal methods with taking $n = 2$

$$\int_0^1 (x^3+1)\mathrm{dx}$$

Solution:

$$h = \frac{b-a}{n} = \frac{1}{2}$$

$$[0,1] = [0,0.5] \cup [0.5,1]$$

$$\int_0^1 (x^3+1)\mathrm{dx} = \int_0^{1/2} (x^3+1)\mathrm{dx} + \int_{1/2}^1 (x^3+1)\mathrm{dx}$$

$$I_n \cong \frac{h}{2}[(x^3+1)dx|_0 + (x^3+1)dx|_{0.5}] + \frac{h}{2}[(x^3+1)dx|_{0.5} + [(x^3+1)dx|_1]$$

$$= \frac{1}{4}(1 + \frac{1}{8} + 1 + \frac{1}{8} + 1 + 2) = \frac{1}{4}[\frac{21}{4}] = [\frac{21}{16}] = 1.3125$$

$$E = |I_n - I_e| = |1.31 - 1.25| = 0.06$$

Remark:

In last example, it is clear that, from the absolute errors, the result of the composite Trapezoidal method is more accurate than the result that we get by using normal Trapezoidal method**.**

Simpson method

Here, we set $n = 2$**,** which means

$$[a,b] = [x_0, x_1] \cup [x_1, x_2], \qquad h = \frac{x_2 - x_0}{2}$$

We use Lagrange polynomial, of order 2, to find an approximate form for the function f.

$$\mathrm{f(x)} \cong P_2(x), \qquad \forall x \in [a,b]$$

$$\int_a^b f(x)dx = \sum_{i=0}^{n} (\int_{x_0}^{x_n} L_i(x)dx) f(x_i) + \int_{x_0}^{x_1} \frac{f'''(\delta(x))}{6}(x - x_0)(x - x_1)(x - x_2)dx$$

We can show that:

$$a_0 = \int_{x_0}^{x_2} \mathrm{L}_0(\mathrm{x})\mathrm{dx} = \frac{\mathrm{h}}{3}$$

$$a_1 = \int_{x_0}^{x_2} \mathrm{L}_1(\mathrm{x})\mathrm{dx} = \frac{4\mathrm{h}}{3}$$

$$a_2 = \int_{x_0}^{x_2} \mathrm{L}_2(\mathrm{x})\mathrm{dx} = \frac{\mathrm{h}}{3}$$

$$T(h) = \int_{x_0}^{x_1} \frac{f'''(\delta(x))}{6}(x - x_0)(x - x_1)(x - x_2)dx$$

$$= -\frac{h^5}{90} f^{(4)}(\delta) \ldots\ldots (2)$$

Thus, we get

$$\therefore \int_a^b f(x)dx \cong \frac{h}{3}\mathrm{f}(\mathrm{x}_0) + \frac{4h}{3}\mathrm{f}(\mathrm{x}_1) + \frac{h}{3}\mathrm{f}(\mathrm{x}_2)$$

which is the **Simpson integral formula**

Remark: We note that, if f is polynomial of order less than or equal 3, then the truncation error equal zero, which means:

$$\int_a^b f(x)dx = \sum_{i=0}^{2} a_i\, f(x_i) = \frac{h}{3} f(x_0) + \frac{4h}{3} f(x_1) + \frac{h}{3} f(x_2),$$

Example 5.3: find the approximate value of the following integral, using Simpson method

$$\int_1^2 e^{x^2}\,dx$$

Solution:

$$h = \frac{b-a}{n} = \frac{2-1}{2} = \frac{1}{2}$$

$$[1,2] = [1,1.5]U[1.5,2]$$

$$I_n = \int_1^2 e^{x^2}\,dx = \frac{1}{6}[e^{x^2}\Big|_1 + 4e^{x^2}\Big|_{1.5} + e^{x^2}|_2]$$

$$=\frac{1}{6}[2.71 + 4(9.48) + 54.5] = 15.855$$

Composite Simpson method

We divide, the interval $[a, b]$, into k, pairs of subintervals, where $n = 2k$ (n should be odd), as follows:

$$[a,b] = ([x_0, x_1] \cup [x_1, x_2]) \cup ([x_2, x_3] \cup [x_3, x_4]) \dots\dots \cup ([x_{n-2}, x_{n-1}] \cup [x_{n-1}, x_n])$$

Apply Simpson formula for each of the pairs of subintervals

$$\int_a^b f(x)dx = \int_{x_0}^{x_2} f(x)dx + \int_{x_2}^{x_4} f(x)dx + \dots\dots \int_{x_{n-2}}^{x_n} f(x)dx$$

$$= \frac{h}{3}[f(x_0) + 4f(x_1) + f(x_2)]$$

$$+ \frac{h}{3}[f(x_2) + 4f(x_3) + f(x_4)] \dots\dots + \frac{h}{3}[f(x_{n-2})$$

$$+ 4f(x_{n-1}) + f(x_n)] - \sum_{i=1}^{k} \frac{h^5}{90} f^{(4)}(\delta_i)$$

where $x_{i-1} \leq \delta_i \leq x_i$

$$\int_a^b f(x)dx \cong \frac{h}{3}[f(x_0) + f(x_n)] + \frac{4h}{3}\sum_{i=1}^{k} f(x_{2i-1}) + \frac{2h}{3}\sum_{i=1}^{k-1} f(x_{2i})$$

which is the **'Composite Simpson Integral formula '**

Steps of composite Simpson algorithm

1- Input a,b

2- Input the numbers of the pairs of subintervals, k, and set $n = 2k$

3- Define the function f

4- $h = \frac{b-a}{n}$

5- Use the composite Simpson integral formula,

$$\int_a^b f(x)dx \cong \frac{h}{3}[f(x_0) + f(x_n)] + \frac{4h}{3}\sum_{i=1}^{k} f(x_{2i-1}) + \frac{2h}{3}\sum_{i=1}^{k-1} f(x_{2i})$$

Example 5.4: Use Composite Simpson integral formula to find the value of the following integral, consider n=4

$$\int_1^2 e^{x^2} dx$$

Solution:

$h = \frac{2-1}{4} = \frac{1}{4}$

$[1,2] = ([1,1.25] \cup [1.25,1.5]) \cup ([1.5,1.75] \cup [1.75,2])$

$$\int_1^2 e^{x^2}\, dx = \int_1^{1.5} e^{x^2} dx + \int_{1.5}^2 e^{x^2}\, dx$$

Apply Simpson formula for each integral, we get that

$$\int_1^2 e^{x^2} dx \cong \frac{h}{3}(e^{x^2}|_1 + 4e^{x^2}|_{1.25} + e^{x^2}|_{1.5}) + \frac{h}{3}(e^{x^2}|_{1.5} + 4e^{x^2}|_{1.75} + e^{x^2}|_2)$$

$= \frac{1}{12}(2.71 + 4(4.77) + 9.48) + \frac{1}{12}(9.48 + 4(21.3) + 54.5) = 15.0375$

Remark: If we increase *n*, $(n = 6, or\ n = 8)$, we can get more accurate results to the integration.

Note that $I_N = I_e = 1.25$, and that because f is a polynomial of order three.

Newton-Cotes formulas

Newton-Cotes rules are a group of formulas for numerical integration, also called quadrature. These methods were first suggested by **Isaac Newton & Roger Cotes.**

As we have mentioned before the general formula of integration methods takes the form:

$$\int_a^b f(x)dx \cong \int_a^b p_n(x)\,dx = \sum_{i=0}^{n} a_i\, f(xi)$$

Where p_n can be considered as Lagrange polynomial

For n=1, we have showed that, the general form above is the **Trapezoidal formula**

$$\int_a^b f(x)dx \cong \sum_{i=0}^{1} a_i\, f(x_i) = \frac{h}{2}f(x_0) + \frac{h}{2}f(x_1),$$

While for n=2, we get the **Simpson** formula:

$$\int_a^b f(x)dx \cong \sum_{i=0}^{2} a_i\, f(x_i) = \frac{h}{3}\mathrm{f(x_0)} + \frac{4h}{3}\mathrm{f(x_1)} + \frac{h}{3}\mathrm{f(x_2)},$$

For n=3, we can show using a similar way that the integral formula takes the following form (which called **Simpson's 3/8** rule):

$$\int_a^b f(x)dx \cong \frac{3h}{8}(f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3))$$

And so on... We can find a Newton-cotes formula for any n.

All these methods above are called **closed Newton-Cotes formulas,** because it use the function value at all points. While, there are another type of Newton-Cotes **methods**, which does not use the function values at the endpoints (x_0 or x_n) , these methods are called **open Newton-Cotes formulas**

Next, we state some of these methods that can be derived in a manner similar to the closed formulas**:**

for $n = 2$, $\int_a^b f(x)dx \cong 2hf(x_1)$,

for $n = 3$, $\int_a^b f(x)dx \cong \frac{3h}{2}(f(x_1) + f(x_2))$

for $n = 4$, $\int_a^b f(x)dx \cong \frac{4h}{3}(2f(x_1) - f(x_2) + 2f(x_3))$

It is well known that, any Newton-Cotes method of order n, gives exact results where f is polynomial of order less than or equal n, and that because the truncation error formula equal zero, where

$$E_n = \int_{x_0}^{x_n} \frac{f^{(n+1)}(\delta(x))}{(n+1)!} \prod_{i=0}^{n} (x - x_i)dx,$$

This fact can be used to derive closed Newton-Cotes formulas, without using Lagrange polynomial as follows:

For $n = 1$,

$$\int_a^b f(x) = a_0 f(x_0) + a_1 f(x_1) + E$$

Where E is the truncation error, which equal zero for $f = \{1, x\}$, $0 \leq x \leq h$

$$\int_0^h 1dx = h = a_0 + a_1 \ldots \ldots (1)$$

$$\int_0^h xdx = \frac{h^2}{2} = a_0 x_0 + a_1 x_1 = a_0(0) + a_1(h) \ldots \ldots (2)$$

From equation (2), we get $a_1 = \frac{h}{2}$,

If we substitute a_1 in equation (1), we get $a_0 = \frac{h}{2}$

Thus for any function f,

$$\int_a^b f(x) \cong \frac{h}{2} f(x_0) + \frac{h}{2} f(x_1),$$

Which is the Trapezoidal formula

For $n = 2$, since it is known that E=0, where, $f = \{1, x, x^2\}$,

$$0 \leq x \leq 2h$$

$$\int_a^b f(x) = a_0 f(x_0) + a_1 f(x_1) + a_3 f(x_2) + E$$

Let $x_0 = 0,\ x_1 = h,\ x_2 = 2h$

By using similar way to that above, we have

$$\int_0^{2h} 1dx = 2h = a_0 + a_1 + a_3,$$

$$\int_0^{2h} xdx = 2h^2 = a_0(0) + a_1(h) + a_2(2h) = a_1 h + 2a_2 h$$

$$\int_0^{2h} x^2 dx = \frac{8h^3}{3} = a_0(0) + a_1(h^2) + a_2(4h^2) = a_1 h^2 + 4a_2 h^2$$

From last three equations, we get

$$a_0 = \frac{h}{3}\ , \quad a_1 = \frac{4h}{3}\ , \quad a_2 = \frac{h}{3}$$

Thus for any function f, we have

$$\int_a^b f(x) \cong \frac{h}{3} f(x_0) + \frac{4h}{3} f(x_1) + \frac{h}{3} f(x_2),$$

Which is the Simpson formula

By using similar way we can get a closed Newton-Cotes formula for any n.

Remark: For Newton-cotes methods, to be accurate, the step size h needs to be small, which means that the interval of integration must be small itself. For this reason, composite Newton-Cotes rules give more accurate results with small h, such as composite Trapezoidal or composite Simpson formula.

Gaussian Quadrature Methods

Since Newton-Cotes formulas of order n give exact results, where f is of order less than or equal n, we need to study another type of methods such as Gauss Quadrature method that give the exact results where f is of degree less than or equal $2n-1$.

The quadrature form for a function f can be given as follows

$$Q(f)=\sum_{i=1}^{n} w_i f(x_i),$$

Where $\{w_i\}_{i=1}^{n}$ are called the weights,

$\{x_i\}_{i=1}^{n}$ Are the roots of a certain polynomial.

Gauss-Legendre Method

Suppose that, we need to find the integral

$\int_{-1}^{1} f(x)dx,$ And let $\{x_i\}_{i=1}^{n}$ are the roots of Gauss-Legendre polynomial, which takes the general form

$$P_n(x)=(\frac{1}{2^n n!})\frac{d^n}{dx^n}(x^2-1)^n, \quad -1\le x\le 1.$$

So, $P_1(x)=x,\ P_2(x)=\frac{1}{2}(3x^2-1),\ P_3(x)=\frac{1}{2}(5x^3-3x)$ and so on

The problem becomes

$$\int_{-1}^{1} f(x)dx \cong Q(f)=\sum_{i=1}^{n} w_i f(x_i),$$

With error of order $2n$th derivative of the function f

Therefore, our aim is to find the weights $\{w_i\}_{i=1}^{n}$

For n=2, the problem is

$$\int_{-1}^{1} f(x)dx \cong \sum_{i=1}^{2} w_i f(x_i) = w_1 f(x_1) + w_2 f(x_2),$$

Where x_1, x_2 are the roots of $P_2(x) = \frac{1}{2}(3x^2 - 1) = 0$.

Thus $x_1 = -\frac{1}{\sqrt{3}}$, $x_2 = \frac{1}{\sqrt{3}}$

So,

$$\int_{-1}^{1} f(x)dx \cong w_1 f\left(-\frac{1}{\sqrt{3}}\right) + w_2 f(\frac{1}{\sqrt{3}})$$

Since the truncation error equal zero where f is of order $2n - 1 = 3$,

For $f = \{1, x, x^2, x^3\}$,

$\int_{-1}^{1} 1dx = 2 = w_1 + w_2$,

$\int_{-1}^{1} x\, dx = 0 = w_1 x_1 + w_2 x_2 = -\frac{w_1}{\sqrt{3}} + \frac{w_2}{\sqrt{3}}$,

$\int_{-1}^{1} x^2\, dx = \frac{2}{3} = w_1 {x_1}^2 + w_2 {x_2}^2 = \frac{w_1}{3} + \frac{w_2}{3}$,

$\int_{-1}^{1} x^3\, dx = 0 = w_1 {x_1}^3 + w_2 {x_2}^3 = -\frac{w_1}{(\sqrt{3})^3} + \frac{w_2}{(\sqrt{3})^3}$,

From these equations, we get $w_1 = w_2 = 1$

Thus, the problem becomes

$$\int_{-1}^{1} f(x)dx \cong f\left(-\frac{1}{\sqrt{3}}\right) + f\left(\frac{1}{\sqrt{3}}\right) \dots\dots (1)$$

For n=3, by using similar way, we can show that

$$\int_{-1}^{1} f(x)dx \cong \frac{5}{9} f\left(-\sqrt{\frac{3}{5}}\right) + \frac{8}{9} f(0) + \frac{5}{9} f\left(\sqrt{\frac{3}{5}}\right) \dots\dots (2)$$

Where $-\sqrt{\frac{3}{5}}$, 0, $\sqrt{\frac{3}{5}}$ are the roots of Legendre polynomial of order 3.

Coordinate transformation from [a,b] to [-1,1]

Set $t = \frac{b-a}{2}x + \frac{b+a}{2}$, $dt = \frac{b-a}{2}\, dx$

$x = -1 \;\rightarrow\; t = a,$

$x = 1 \;\rightarrow\; t = b$

Thus

$$\int_{a}^{b} f(t)dt = \int_{-1}^{1} f\left(\frac{b-a}{2}x + \frac{b+a}{2}\right)\left(\frac{b-a}{2}\right) dx$$

Example 5.5: Find the following integral, using the 2 Points Gauss-Legendre formulas

$$\text{I} = \int_{0}^{4} te^{2t}dt = 5216.9$$

$$t = \frac{b-a}{2}x + \frac{b+a}{2} = 2x + 2, \quad dt = 2dx$$

$$\mathrm{I} = \int_0^4 te^{2t}dt = \int_{-1}^{1} (4x+4)\, e^{4x+4}dx = \int_{-1}^{1} f(x)$$

Use equation (1), we get

$$\mathrm{I} = \int_{-1}^{1} f(x) \cong f\left(-\frac{1}{\sqrt{3}}\right) + f\left(\frac{1}{\sqrt{3}}\right)$$

$$= \left(4 - \frac{4}{\sqrt{3}}\right) e^{4-\frac{4}{\sqrt{3}}} + \left(4 + \frac{4}{\sqrt{3}}\right) e^{4+\frac{4}{\sqrt{3}}} = 3477.5$$

If we use 3 point Gauss- Legendre formula (2), we get much more accurate results

$I = 5197.5$

Exercises

Q1: Find the following integral, by using Simpson method.

$$\int_0^1 (x^3 + 1)\mathrm{dx}$$

Q2: Compare between the two absolute errors, those can arise from using *Simpson* method with n=2 and 4, respectively, to find the approximate value for the following integral:

$$\int_0^1 (x^4 + 1)dx$$

Q3: Use both of Trapezoidal and composite Simpson formulas to derive a new composite formula, with n=5.

a) What is the truncation error's form of this new formula?
b) What should be the degree of f to guarantee that there is no absolute error.
c) Use the new formula to find the approximate value of the following integral, and find the absolute error.

$$\int_0^1 e^{2x} dx$$

Q4: Use **Trapezoidal** method, with **n=1**, and **n=3** to find the approximate value of the following integral, and find the absolute error in each case.

$$\int_1^2 (x^2 + 2x + 1)dx$$

Q5: Write a Matlab program which can be used to find the approximate value to the integral in the last example by using Gauss-Legendre 2-points method.

Q6: Prove that Gauss-Legendre 2-points method gives the exact results for the following integral

$$\int_0^1 (x^3 + 1)dx$$

Q7: Derive a newton cotes formula with $n = 3$.

Chapter Six

Numerical Solutions of First Order Ordinary Differential Equations

It is well known that, an ordinary differential equation (O.D.E.) is an equation has unknown function *y*, of one variable *x*, and some of its derivatives. For instance

$$\frac{dy}{dx} = y \sin x$$

While, partial differential equation, has unknown function of two or more variables and some of its partial derivatives. For instance

$$\frac{\partial y}{\partial t} = \frac{\partial^2 y}{\partial x^2}$$

The order of the differential equation is the highest derivative appears in the differential equation.

In this chapter, we will study, the numerical solutions for first order ordinary differential equations, which takes the general form:

$$y' = f(x, y)$$

The solution of the differential equation above, is an differentiable function defined on an interval $[a, b]$, and satisfies the differential equation. Each

solution has a constant *c*, which can be determined, if *y* is known for at least one point $x_0 \in [a, b]$. ($y(x_0) = y_0$ *initial condition*)

Remark: The problem of the differential equation with an initial condition is called **Initial Value Problem:** $y' = f(x, y), \quad y(x_0) = y_0$

Example 6.1:

$$y = -sinx$$
$$y(0) = 1$$

While, if y are given at more than one point, then the problem is called **Boundary values problem.**

Example 6.2:

$$y = xy$$
$$\mathrm{y}(0) = 1\,, \qquad \mathrm{y}(1) = 2$$

In this chapter, we will only study the numerical solutions of first order initial vale problems.

Our aim is find the approximate solution, for this problem at certain points: $\{x_i\}_{i=1}^{n} \in [a, b]$, which means, we only need to find $\{y_i\}_{i=1}^{n}$

Next, we will study some important methods that can be used to find the numerical solutions of initial value problems

Euler Algorithm

The general idea of this method is to divide the interval $[a, b]$, into n of subintervals, as follows:

$$x_0 = a\ , x_1 = x_0 + h \ldots \ldots \quad x_n = x_{n-1} + h = b$$

$$h = \frac{(b-a)}{n}$$

In order to find the approximate solution of the initial value problem at the point $x_{i+1}, i = 0,1,2, \ldots n-1$, we consider the definition of $y'(x_i)$, as follows:

$$y'(x_i) = \lim_{x \to x_i} \left(\frac{y(x_{i+1}) - y(x_i)}{x_{i+1} - x_i} \right) \cong \frac{y(x_{i+1}) - y(x_i)}{x_{i+1} - x_i}$$

$y' = f(x, y)$, we obtain since,

$$\frac{y(x_{i+1}) - y(x_i)}{x_{i+1} - x_i} \cong f(x_i, y_i), \quad i = 0,1,2, \ldots \ldots n$$

$$\therefore \ y(x_{i+1}) \cong y(x_i) + hf\big(x_i, y(x_i)\big)$$

Assume that, y_i is the approximate value of $y(x_i)$, we get

$$y_{i+1} = y_i + hf(x_i, y_i) \qquad i = 0,1,2, \ldots \ldots \ldots, n-1$$

The last equation is called **Euler formula.**

Remark: Euler formula, can only be used only for finding the numerical solutions at the points $x_i)_0^{\mathrm{n}}$, while if we would like to find the approximate value of $y(x^*)$, where $x^* \neq x_i)_0^{\mathrm{n}}$, $x^* \in [a, b]$, then in this case, we can use an interpolation method.

Steps of Euler algorithm

1- Input a, b

2- Input the points, x_1^*, x_2^*, ...

3- Define $f(x, y)$

4- Input, n, $h = \frac{(b-a)}{n}$

5- Set $x_{i+1} = x_i + h, \quad i = 0,1,2, \ldots . n-1$

6- Compute the approximate values of $y(x_i), i = 1,2, \ldots n$, using Euler formula:

$$y_{i+1} = y_i + hf(x_i, y_i) \qquad i = 0,1,2, \ldots \ldots, n-1$$

7- If $x_k^* \in [a, b], \quad x_k^* \neq x_i, \forall i$, then use a Newton's finite difference formula, to find the approximate value of the initial value problem at this point.

Example 6.3: Consider that, we have the following initial value problem

$$y' = -xy, \qquad y(0) = 0.5$$
$$x \in [0,0.2],$$

Use Euler method to find the approximate values of y (0.1), y (0.15), y (0.2) and compare these values with the exact solution (find the absolute errors)

Solution: $h=\frac{b-a}{2} = \frac{0.2-0}{2} = 0.1, \quad x_0 = 0\,, x_1 = 0.1\,, x_2 = 0.2$

By using Euler formula

$$y_1 = y_0 + h\, f(x_0, y_0) \; = 0.5 + (0.1)(-(0)(0.5)) = 0.5$$

$$y_2 = y_1 + h\, f(x_1, y_1) = 0.5 + (0.1)(-(0.1)(0.5)) = 0.495$$

Thus, we get the following database

x	0	0.1	0.2
y	0.5	0.5	0.495

We can show that, by using separation of variables, the exact solution of this problem takes the form: $y = \frac{1}{2} e^{\frac{-x^2}{2}}$

Which leads to $y(0.1) = 0.4975,\ \ y(0.2) = 0.4901$

Thus, the absolute errors can be computed as follows:

$$E_1 = |y(0.1) - y_1| = |0.4975 - 0.5| = 0.0025$$

$$E_2 = |y(0.2) - y_1| = |0.4901 - 0.495| = 0.0049$$

From the above database, we can compute the approximate value of $y(0.15)$, by using Backward Newton's formula studied in Chapter 4. (**Home Work**)

Modified Euler method

In order to get more accurate results than Euler method, we define the modified Euler method, which has the following general formula

$$y_{i+1} = y_i + \frac{h}{2}\left(f(x_i, y_i) + f(x_{i+1}, y^*_{i+1})\right), \qquad \textbf{Modified Euler}$$

Where

$$y^*_{i+1} = y_i + h\ f(x_i, y_i) \qquad \textbf{Normal Euler}$$

$$i = 0,1,2, \ldots \ldots . n - 1$$

In fact, the first equation depends on the second equation, and this way is called (**Estimation–Correction**), because from the first equation, we get estimate value for $y(x^*)$, and then in the second equation, we get a correction for this value.

The steps of Modified Euler algorithm are the same as the steps of normal Euler algorithm and we only have to add one step more after step 5, which is:
6- Correct the estimated value y_{i+1}^{*}, that we get from using normal Euler formula, by using the modified Euler formula.

Example 6.4: Use Modified Euler method for the last example.

Solution: n=2 $h = \frac{b-a}{2} = \frac{0.2-0}{2} = 0.1$,

Estimation: $y_1^* = y_0 + h\ f(x_0, y_0) = 0.5 + (0.1)(0) = 0.5$

$y_1 = y_0 + \frac{h}{2}(f(x_0, y_0) + f(x_1, y_1^*))$ Correction:

$$= 0.5 + \frac{0.1}{2}\big(-(0.1)(0.5)\big) = 0.4975$$

Estimation: $y_2^* = y_1 + h\ f(x_1, y_1)$

$$= 0.4975 + (0.1)\big(-(0.1)(0.4975)\big) = 0.4925$$

$y_2 = y_1 + \frac{h}{2}(f(x_1, y_1) + f(x_2, y_2^*))$ Correction:

$$= 0.4975 + \frac{(0.1)}{2}(-(0.1)(0.4975) - (0.2)(0.4925)) = 0.4901$$

Thus, we get the following data base

X	0	0.1	0.2
Y	0.5	0.4975	0.4901

Next, we compute the absolute errors:

$$E_1 = |y(0.1) - y_1| = |0.4975 - 0.4975| = 0$$

$$E_2 = |y(0.2) - y_1| = |0.4901 - 0.4901| = 0$$

It is clear that, this database is much different from that we have got from using normal Euler method. Moreover, it is more accurate.

Iteration modified Euler method

In order to increase the speed of convergence with much more accurate results, we can use the Iteration modified Euler method**,** which takes the following form:

$$y_{i+1}^{0} = y_i + h\ f(x_i, y_i)$$

$$y_{i+1}^{(k+1)} = y_i + \frac{h}{2}(f(x_i, y_i) + f(x_{i+1}, y_{i+1}^{(k)}) \quad k = 0,1,2, \ldots.\ i = 0,1,2, \ldots\ldots, n-1$$

Remark: The steps of Iteration modified Euler method, are the same as those of modified Euler method, but the only different is after computing the estimated value y_i^0, we compute $y_i^{(1)}, y_i^{(2)}, \ldots.$

Until, we get the convergence condition is satisfied:

$$|y_i^{(k+1)} - y_i^{(k)}| < \in$$

And then, we continue to compute the other points $y_{i+1}, y_{i+2}, \ldots\ldots$, using the same way.

Example 6.5: Use Iteration modified Euler methods for the last example

$$\text{h}=\frac{b-a}{2} = \frac{0.2-0}{2} = 0.1, \quad x_0 = 0\ , x_1 = 0.1\ , x_2 = 0.2$$

$$y_1^0 = y_0 + h\ f(x_0, y_0)\ = 0.5 + (0.1)(0) = 0.5$$

$$y_1^{(1)} = y_0 + \frac{h}{2}\ (\text{f}(x_0, y_0) + \text{f}(x_1, {y_1}^0))$$

$$= 0.5 + \left(\frac{0.1}{2}\right)\Big(0 + \big(-(0.1)(0.5)\big)\Big) = 0.4975$$

$$y_1^{(2)} = y_0 + \frac{h}{2}\,(\mathrm{f}(x_0, y_0) + \mathrm{f}(x_1, {y_1}^{(1)}))$$

$$= 0.5 + \frac{(0.1)}{2}\big(-(0.1)(0.4975)\big) = 0.4975$$

Since the convergence condition is satisfied, which is

$|{y_1}^{(2)} - {y_1}^{(1)}| < \in$, we move on computing y_2

$y_2^0 = y_1 + h\, f(x_1, y_1) = 0.4975 + (0.1)\big(-(0.1)(0.4975)\big) = 0.4925$

$$y_2^{(1)} = y_1 + \frac{h}{2}\,(\mathrm{f}(x_1, y_1) + \mathrm{f}(x_2, {y_2}^0))$$

$$= 0.4975 + \frac{(0.1)}{2}\big(-(0.1)(0.4975) - (0.2)(0.4925)\big) = 0.4901$$

$$y_2^{(2)} = y_1 + \frac{h}{2}\,(\mathrm{f}(x_1, y_1) + \mathrm{f}(x_2, {y_2}^{(1)}))$$

$$= 0.4975 + \frac{(0.1)}{2}\big(-(0.1)(0.4975) - (0.2)(0.4901)\big) = 0.4901$$

Since $|\,{y_2}^{(2)} - {y_2}^{(1)}| < \in$, we stop here.

Explicit and Implicit Methods

Normal Euler method, belongs to group of methods called **explicit methods**, and that because of, computing y_{i+1} depends only on x_i ,while modified Euler methods belongs to **implicit methods**, and that because of , computing y_{i+1} depends on both of x_i & x_{i+1}

Runge – Kutta Methods

Since modified Euler method needs two steps to get the solutions, it is considered a two-steps method. Moreover, Euler methods need to approximate the derivatives by using special forms. Thus, we will use *Runge-Kutta* method, which is a one-step method and it can be used to avoid determining higher order derivatives.

Runge - Kutta Method of order 2:

Set $k_1 = hf(x_i, y_i)$

$$k_2 = hf(x_i + h, y_i + k_1)$$

Recall modified Euler formula

$$y_{i+1} = y_i + \frac{1}{2}\left(hf(x_i, y_i) + hf(x_{i+1}, y_{i+1}^*)\right),$$

Where $y_{i+1}^* = y_i + h\ f(x_i, y_i)$

This equation can rewrite as follows:

$$y_{i+1} = y_i + \frac{1}{2}(k_1 + k_2) \ldots\ldots \textbf{Ruga} - \textbf{kutta formula of order2}$$

Runge - Kutta Method of order 4:

Set $k_1 = hf(x_i, y_i),$

$$k_2 = hf\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1\right),$$

$$k_3 = hf\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2\right),$$

$$k_4 = hf(x_i + h, y_i + k_3)$$

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

which is **Ruga-Kutta** formula of order4.

Example 6.6: Consider that, we have the following intial value problem:

$$y' = x + y, \qquad y(0) = 1$$

Use Runga-Kutta method of order 2 and order 4 to compute the approximate values of $y(0.25), y(0.5)$

Solution:

$$h = 0.25, \qquad x_0 = 0, \qquad x_1 = 0.25, \;\; x_2 = 0.5$$

$$i = 0, \qquad k_1 = hf(x_0, y_0) = 0.25(0 + 1) = 0.25$$

$$k_2 = hf(x_0 + h, y_0 + k_1) = 0.25(0.25 + 1 + 0.25) = 0.3750$$

$$y_1 = y_0 + \frac{1}{2}(k_1 + k_2) = 1 + 0.5(0.25 + 0.3750) = 1.3125$$

$$i = 1, \qquad k_1 = hf(x_1, y_1) = 0.25(0.25 + 1.3125) = 0.3906$$

$$k_2 = hf(x_1 + h, y_1 + k_1) = 0.25(0.25 + 0.25 + 1.3125 + 0.3906) = 0.5508$$

$$y_2 = y_1 + \frac{1}{2}(k_1 + k_2) = 1.3125 + 0.5(0.3906 + 0.5508) = 1.7832$$

Next, we compute these values by using Euler method of order4

$$i = 0, \qquad k_1 = hf(x_0, y_0) = 0.25(0 + 1) = 0.25$$

$$k_2 = hf\left(x_0 + \frac{1}{2}h, y_0 + \frac{1}{2}k_1\right) = 0.25\left(\frac{1}{8} + 1 + \frac{1}{8}\right) = 0.3125$$

$$k_3 = hf\left(x_0 + \frac{1}{2}h, y_0 + \frac{1}{2}k_2\right) = 0.25\left(\frac{1}{8} + 1 + \frac{0.3125}{2}\right) = 0.3203$$

$$k_4 = hf(x_0 + h, y_0 + k_3) = 0.25(0.25 + 1 + 0.3203) = 0.3926$$

$$y_1 = y_0 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$= 1 + \frac{1}{6}(0.25 + 2(0.3125) + 2(0.3203) + 0.3926)$$

$$= 1.3180$$

$$i = 1, \qquad k_1 = hf(x_1, y_1) = 0.25(0.25 + 1.3180) = 0.3920$$

$$k_2 = hf\left(x_1 + \frac{1}{2}h, y_1 + \frac{1}{2}k_1\right) = 0.25\left(\frac{1}{4} + \frac{1}{8} + 1.3180 + \frac{0.3920}{2}\right)$$

$$= 0.4723$$

$$k_3 = hf\left(x_1 + \frac{1}{2}h, y_1 + \frac{1}{2}k_2\right) = 0.25\left(\frac{1}{4} + \frac{1}{8} + 1.3180 + \frac{0.4723}{2}\right)$$

$$= 0.4823$$

$$k_4 = hf(x_1 + h, y_1 + k_3) = 0.25(0.25 + 0.25 + 1.3180 + 0.4823)$$

$$= 0.5751$$

$$y_2 = y_1 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$= 1.3180 + \frac{1}{6}(0.3920 + 2(0.4723) + 2(0.4823) + 0.5751)$$

$$= 1.7974$$

Since $y' = x + y$ is linear equation, we can show that, by using integration factor, the exact solution of problem in the last example takes the form:

$$y = 2e^x - x - 1$$

Thus $y(0.25) = 1.3181,\ y(0.5) = 1.7974$

Thus, the absolute errors, obtained by Ruge-Kutta of order 2 are as follows:

$$E_1 = |y(0.25) - y_1| = |1.3181 - 1.3125| = 0.0056$$

$$E_2 = |y(0.5) - y_1| = |1.7974 - 1.7832| = 0.0142$$

while, the absolute errors, obtained by Ruge-Kutta of order 4 are as follows:

$$E_1 = |y(0.25) - y_1| = |1.3181 - 1.3180| = 0.0001$$

$$E_2 = |y(0.5) - y_1| = |1.7974 - 1.7974| = 0$$

It is clear that, the results of Ruga-Kutta method of order 4 are much accurate than the results of Ruga-Kutta method of order 2.

Exercises

Q1: Consider the following initial value problem:

$$y' = x/y \ , \quad y(0) = 2 \ .$$

a) Use **Runge-Kutta** method of order 2 to find the approximate values of $y(0.1),\ y(0.2), y(0.3)$ and $y'(0.2)$.

b) Depending on the results of a. , find the approximate value of $y(0.15)$

c) Find the absolute error at each point in a.

Q2: Consider the following initial value problem:

$$y' - 2xy = 0 \ , \quad y(0) = 1$$
$$x \in [0, 0.4]$$

a) Use **Modified Euler** method to find the approximate values for $\{x_i\}_{i=1}^{2}$, with $h = 0.2$

b) Find the absolute error at each point.

Appendix

Matlab Codes

In this index, we state the Matlab codes for most of the numerical methods considered in this book. We note that, the codes are written for specific examples, however, it can be used in general with any others examples.

Matlab Codes for numerical methods of Chapter two

- **Matlab Code for Bisection Method**

Write the Matlab code which can be used to find the approximate root of $f(x) = x^2 + x - 1 = 0,$ on the interval [0,1], with considering $\epsilon = 0.0001$.

```
1- a=input('a=');
2- b=input('b=');
3- x=sym('x');
4- f=x^2+x-1;;
5- fa=subs(f,x,a);
6- fb=subs(f,x,b);
7- k=0;
8- if fa*fb>0
9- fprintf('the function f(x) has no root')
10- break;
11- else
```

```
12- while abs(b-a)>0.0001
13- c=(a+b)/2;
14- fc=subs(f,x,c);
15- if fc==0
16- fprintf('the exact root=%f',c);
17- fprintf('the number of iteration=%d',k);
18- break;
19- end
20- if fa*fc>0
21- a=c; fa=fc;
22- else
23- b=c; fb=fc;
24- mend
25- k=k+1;
26- end
27- fprintf('the approximate root=%f',c);
28- fprintf('the number of iteration=%d',k)
29- end
```

a=0

b=1

Answer: the approximate root=0.617981

The number of iteration=14

Note: The steps of Matlab code for false position algorithm are almost the same as that of

Bisection algorithm except for in step 13 we set c= $\frac{af(b)-bf(a)}{f(b)-f(a)}$

- **Matlab Code for Newton-Raphson algorithm**

We can write a Matlab code to find the approximate root of the last example using N.R. algorithm, as follows:

```
1- x0=input('x0=');
2- x=sym('x');
3- f=sin(x)-((x+1)/(x-1));
4- g=diff(f);
5- fx0=subs(f,x,x0)
6- gx0=subs(g,x,x0)
7- k=0;
8- x1=x0-(fx0/gx0)
9- fx1=subs(f,x,x1);
10- while abs(fx0/gx0)>eps;
11- fx1=subs(f,x,x1);
12- if fx1==0
13- fprintf('The exact root=%f',x1);
14- break;
15- else
16- x0=x1;
17- end
18- k=k+1;
19- fx0=subs(f,x,x0)
20- gx0=subs(g,x,x0)
21- x1=x0-(fx0/gx0)
22- end
23- fprintf('the approximate root=%f',x1);
24- fprintf('the number of iteration=%d',k);
```

$x_0 = -0.2$

A**nswer:** the approximate root= - 0.420362; the number of iteration=4

- ## Matlab Code for Secant algorithm algorithm

Write the Matlab code which can be used to find the approximate root of $f(x) = cos\ \pi x + x = 0$, using secant algorithm, with considering $p_0 = 2$, $p_1 = 1.8$

```
p0=input('p0=');
p1=input('p1=');
x=sym('x');
f=cos(pi*x)+x;
fp0=subs(f,x,p0);
fp1=subs(f,x,p1);
k=0;
p=p1-(fp1*(p1-p0)/(fp1-fp0));
while abs(p1-p)>eps
fp=subs(f,x,p);
if fp==0
fprint ('the exact root=%f',p);
break
end
k=k+1;
p0=p1; fp0=fp1;
p1=p; fp1=fp;
p=p1-(fp1*(p1-p0)/(fp1-fp0));
end
fprintf('the approximate root=%f',p)
```

21- fprintf('the number of iterations=%d',k)

P0=2

P1=1.8

Answer: The exact root=1

The number of iterations=15

- ## Matlab Code for Fixed point algorithm

Example: write the Matlab code that can be used to find the approximate root of the following equation $f(x) = x - e^{-x} = 0$, on the interval [0, 1], consider $x_0 = 0.9$

Before to write the grogram let us study the possible forms for g

$$x = g_1(x) = e^{-x}, \quad so \;\; g_1'(x) = -e^{-x},$$

$$x = g_2(x) = -ln\, x, \quad so \;\; g_2'(x) = -\frac{1}{x}$$

Since $|g_1'(x)| = e^{-x} < 1$, while $|g_2'(x)| = \frac{1}{x} > 1\,, \forall\, x \in (0,1)$,

Therefore, we choose the first form for g

```
1- a=input('a=');
2- b=input('b=');
3- x0=input('x0=');
4- x=sym('x');
5- f=x-exp(-x);
6- g=exp(-x);
7- fa=subs(f,x,a);
8- fb=subs(f,x,b);
```

```
k=0;
if fa*fb>0
fprintf('the function f has no root');
break
end
 if abs(subs(diff(g),x,x0))>1
fprintf('the algorithm is divergent' );
break;
end
 x1=subs(g,x,x0);
while abs(x1-x0)>eps
fx1=subs(f,x,x1);
if fx1==0
fprintf('the exact root=%f',x1);
break;
end
k=k+1;
x0=x1;
x1=subs(g,x,x0);
end
fprintf('the approximate root=%f',x1)
fprintf('the number of iterations=%d',k)
```

a=0

b=1

$x_0 = 0.9$

Answer: The exact root=0.567143

The number of iterations=63

Matlab Codes for numerical methods of Chapter Three

- ## Matlab Code for Gauss Method

Write a Matlab program, which can be used to find the solution of the following system by using Gauss method with back ward substitutions

$$\begin{aligned} 4x_1 - 9x_2 + 2x_3 &= 5 \\ 2x_1 - 4x_2 + 6x_3 &= 3 \\ x_1 - x_2 + 3x_3 &= 4 \end{aligned}$$

```
1- A=[4,-9,2;2,-4,6;1,-1,3];
2- b=[5;3;4];
3- n=3;
4- for k=1:n-1;
5- for i=k+1:n
6- m(i,k)=A(i,k)/A(k,k);
7- for j=k:n
8- A(i,j)=A(i,j)-m(i,k)*A(k,j);
9- End
10- b(i)=b(i)-m(i,k)*b(k);
11- end
12- end
13- x(n)=b(n)/A(n,n);
14- for i=n-1:-1:1
15- s=0;
16- for j=i+1:n
17- s=s+A(i,j)*x(j);
18- end
19- x(i)=(b(i)-s)/A(i,i);
20- end
```

21- disp(x);

6.95002.5000 -0.1500

- **Matlab Code for Jacobi and Gauss-Sidel methods**

Write a matlab program, which can be used to find the approximate solution of the following system by using

1- Jacobi Method
2- Gauss-Sidel

$$\begin{aligned} 9x_1 - 4x_2 + 2x_3 &= 5 \\ 2x_1 - 4x_2 + x_3 &= 3 \\ x_1 - x_2 + 3x_3 &= 4 \end{aligned}$$

With $x^0 = (0,0,0)$

Jacobi

1- A= [9,-4, 2; 2,-4, 1; 1,-1, 3];
2- b=[5;3;4];
3- n=3;
4- x0=[0;0;0];
5- r=norm(b-A*x0);
6- k=0;
7- while r> 0.01
8- k=k+1;
9- for i=1:n
10- s=0;
11- for j=1:i-1
12- s=s+A(i,j)*x0(j);
13- end
14- for j=i+1:n

```
15- s=s+A(i,j)*x0(j);
16- end
17- x(i)=(b(i)-s)/A(i,i);
18- end
19- x0=x' ;
20- r=norm(b-A*x0);
21- end
22- disp(x);
23- disp(k);
```

0.1205 -0.4005 1.1605 k=19

Gass-Sidel

```
1-  A=[9,-4,2;2,-4,1;1,-1,3];
2-  b=[5;3;4];
3-  n=3;
4-  x0=[0;0;0]; x=x0';
5-  r=norm(b-A*x0);
6-  k=0;
7-  while r> 0.01
8-  k=k+1;
9-  for i=1:n
10- s=0;
11- for j=1:i-1
12- s=s+A(i,j)*x(j);
13- end
14- for j=i+1:n
15- s=s+A(i,j)*x0(j);
16- end
```

```
17- x(i)=(b(i)-s)/A(i,i);
18- end
19- x0=x' ;
20- r=norm(b-A*x0);
21- end
22- disp(x);
23- disp(k);
```

0.1188 -0.4004 1.1603 k=5

Matlab Codes for numerical methods of Chapter Four

- ### Matlab Code for Lagrange algorithm

Write a Matlab program that can be used to find f(0.4) from the following data base, using Lagrange algorithm

X| 0.25, 0.5, 1, 1.75

Y| 1.0625, 1.25, 2, 4.0625

```
1- x=[0.25,0.5,1,1.75];
2- y= [1.0625,1.25,2,4.0625];
3- xm=input('xm=')
4- sum=0;n=4
5- for k=1:n
6- prod=1;
7- for i=1:n
8- if i~=k
9- prod=prod*(xm-x(i))/(x(k)-x(i));
10- end
11- end
```

12- sum=sum+prod*y(k);

13- end

14- fprintf('f(%f)=%f',xm,sum);

xm=0.4

f(0.4)=1.16

- **Matlab Codes for Finite Difference Methods**

Example: Consider the following database

X	0	0.5	1	1.5
$f(x)$	1	1.25	2	3.25

Write Matlab codes that can be used to find:

1- f (0.25) (Forward)

2- f (1.25) (Backward)

3- f (0.75) (Centre)

Forward finite difference

```
x= [0, 0.5, 1, 1.5];
y= [1, 1.25, 2, 3.25];
xp=input('xp=');
h=0.5;
p= (xp-x(1))/h;
Dy0=y (2)-y(1);
D2y0=y (3)-2*y(2)+y(1);
yp=y (1)+p*Dy0+(p*(p-1)/2)*D2y0;
```

fprintf('f(%f)=%f',xp,yp);

$xp = 0.25$

$f(0.25) = 1.0625$

Central finite difference

x=[0,0.5,1,1.5];

y= [1,1.25,2,3.25];

xp=input('xp=');

h=0.5;

p=(xp-x(2))/h;

q=1-p;

S2y1=y(3)-2*y(2)+y(1);

S2y2=y(4)-2*y(3)+y(2);

yp=p*y(3)+p*(p+1)*(p-1)*S2y2/factorial(3)+q*y(2)+q*(q+1)*(q-1)*S2y1/factorial(3);

fprintf('f(%f)=%f',xp,yp);

$\boldsymbol{xp = 0.75}$

$\boldsymbol{f(0.75) = 1.5625}$

Backward finite difference

x=[0,0.5,1,1.5];

y= [1,1.25,2,3.25];

xp=input('xp=');

h=0.5;

p=(xp-x(4))/h;

By3=y(4)-y(3);

B2y3=y(4)-2*y(3)+y(2);

yp=y(4)+p*By3+(p*(p+1)/2)*B2y3;

fprintf('f(%f)=%f',xp,yp);

xp=1.25

f(1.25)=2.5625

Matlab Codes for numerical methods of Chapter Five

- **Matlab Code for composite Trapezoidal method**

Next, we write dawn the Matlab Code for last example with $n = 40$

```
1- a=0; b=1; n=40;
2- h=(b-a)/n; g=0;
3- x=sym('x');
4- f=x^3+1;
5- m=subs(f,x,a)+subs(f,x,b);
6- for i=1:n-1
7- d=a+i*h;
8- g=g+2*subs(f,x,d);
9- end
10- T=(h/2)*(m+g);
11- fprintf('I=%f',T);
```

I=1.250156

- **Matlab Code for composite Simpson method**

Next, we write dawn the Matlab Code for which can be used to find the following integer $\int_0^1 (x^3 + 1)\mathrm{dx}$, using composite Simpson method, with $n = 40$.

```
1- a=0; b=1; n=40;
```

```
2-  h=(b-a)/n; g=0;
3-  x=sym('x');
4-  f=x^3+1;
5-  m=subs(f,x,a)+subs(f,x,b);
6-  r=0; g=0;
7-  for i=1:n-1
8-  d=a+i*h;
9-  if rem(i,2)==0
10- r=r+2*subs(f,x,d);
11- else
12- g=g+4*subs(f,x,d);
13- end
14- end
15- T=(h/3)*(m+r+g);
16- fprintf('I=%f',T);
```

I=1.25

Matlab Codes for numerical methods of Chapter Six

- **Matlab Codes of Euler methods**

For the last example, write Matlab codes, that can be used to find $y(0.1), y(0.2)$, by using Euler and modified Euler methods

Normal Euler

```
1- x0=0; y0=0.5;h=0.1; X(1)=x0+h; X(2)=X(1)+h;
2- x=sym('x');
3- y=sym('y');
4- f=-x*y;
5- for i=1:2
6- Y(i)=y0+h*subs(f,{x,y},{x0,y0});
7- x0=X(i); y0=Y(i);
8- end
9- disp(Y)
```

Answer: 0.5 0.4950

Modified Euler

```
1- x0=0; y0=0.5;h=0.1; X(1)=x0+h; X(2)=X(1)+h;
2- x=sym('x');
3- y=sym('y');
4- f=-x*y;
5- for i=1:2
6- Ye(i)=y0+h*subs(f,{x,y},{x0,y0});
7- Yc(i)=y0+(h/2)*(subs(f,{x,y},{x0,y0})+subs(f,{x,y},{X(i),Ye(i)}))
8- x0=X(i); y0=Yc(i);
9- end
```

```
10- disp(Yc)
```

Answer: 0.4975 0.4901

- **Matlab Codes for Runge-Kutta Methods**

We can write matlab codes that can be used to solve the last example by using Ruga-Kutta method of order 2 and 4, as follows:

Order 2

```
1- x0=0; y0=1; h=0.25;
2- X(1)=0.25; X(2)=0.5;
3- x=sym('x');
4- y=sym('y');
5- f=x+y;
6- for i=1:2
7- k1=subs(f,{x,y},{x0,y0});
8- k2=subs(f,{x,y},{x0+h,y0+h*k1});
9- Y(i)=y0+(h/2)*(k1+k2);
10- x0=X(i);
11- y0=Y(i);
12- end
13- disp(Y)
```

1.3125 1.7832

Order 4

```
x0=0; y0=1; h=0.25;
1- X(1)=0.25; X(2)=0.5;
2- x=sym('x');
3- y=sym('y');
```

```
4- f=x+y;
5- for i=1:2
6- k1=subs(f,{x,y},{x0,y0});
7- k2=subs(f,{x,y},{x0+h/2,y0+h/2*k1});
8- k3=subs(f,{x,y},{x0+h/2,y0+h/2*k2});
9- k4=subs(f,{x,y},{x0+h,y0+h*k3});
10- Y(i)=y0+(h/6)*(k1+2*k2+2*k3+k4);
11- x0=X(i);
12- y0=Y(i);
13- end
14- disp(Y)
1.3180    1.7974
```

Bibliography

1. A. Quarteroni and F. Saleri, Scientific Computing with Matlab, Springer, 2003.
2. Begnaud Francis Hildebrand, Introduction to numerical analysis: 2nd edition, Dover Publications, Inc. 31 E. Second St. Mineola, NY United States, 1987.
3. C. Moler: Numerical Computing with MATLAB, SIAM, 2004.
4. E. Suli and D. Mayers: An Introduction to Numerical Analysis, Cambridge, 2003.
5. E. W. Cheney and D. R. Kincaid: Numerical Mathematics and Computing, Brooks Cole, 6 editions, 2007.
6. G. Forsythe and C. Moler, Computer Solution of Linear Algebraic Systems, Prentice Hall, 1967.
7. G. W. Stewart: Afternotes Goes to Graduate School: Lectures on Advanced Numerical Analysis, SIAM, 1998.
8. J. C. Butcher, Numerical Methods for Ordinary Differential Equations, 2nd ed., Wiley, 2003.
9. James F Epperson, An Introduction to Numerical Methods and Analysis, Wiley Publishing, 2013.
10. M. T. Heath, Scientific Computing: An Introductory Survey, 2nd ed., McGraw-Hill, 2002.
11. R. L. Burden, J. D. Faires**,** Applied Numerical Analysis, Brooks/Cole, Cengage Learning, *USA*, 2015.
12. S.R. Otto and J.P. Denier, An Introduction to Programming and Numerical Methods in MATLAB, Springer-Verlag *London* Limited, 2005.

13. U. M. Ascher and L. R. Petzold, Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations, SIAM, 1998.
14. W. Gautschi: Numerical Analysis: an introduction, Birkhauser, 1997.