Mobile App For Numerical Computing Method

Final Year Project(FYP)

Project Report

Group Members

S. Abu Owais Bin Nasar B-10101086

M. Umair Khan B-10101067

Osama Ahmed B-10101082

Section B

Class

VIII Semester BSCS-(Morning)

Course Instructor

Dr Jamal Hussain

Department

Computer Science

UNIVERSITY OF KARACHI

UNIVERSITY OF KARACHI

CERTIFICATE

Certified that this project report **"…….Mobile App For Numerical Computing Method……..."**is the bonafide work of "**…….** who carried out the project work under my supervision.

<< **HEAD OF THE DEPARTMENT** >>

_____

Name

_____

Signature

<<**COURSE SUPERVISOR**>>

_____

Name

_____

Signature

Department of Computer Science

ACKNOWLEDGMENT

"We are thankful to Almighty ALLAH for granting us the nerve, determination and intelligence to prepare this assignment successfully.

We would like to express our special thanks of gratitude to our Project Supervisor – ***Dr Jamal Hussain*** *as he has been very cooperative and supportive throughout the semester and to complete this Final Year Project on*

*"****Mobile App For Numerical Computing Method****"*

Secondly, we would also like to thank our sincere thanks and heartfelt appreciations also go to our parents and friends who helped us a lot in finishing this project within the limited time"

S. Abu Owais Bin Nasar
M. Umair Khan
Osama Ahmed

ABSTRACT

Positivity is one of the most common and most important characteristics of mathematical models. In this final project, four problems involving the numerical methods for: Root of Equation, Linear Algebra, Interpolation, and differentials equation are assigned.

In particular, the Root of Equation is Bisection Method, Newton-Rapson Method, Secant Method, and False Position Method. The Linear Algebraic Equation are Cramer's Rule, Elimination of Unknowns Y Combining, and Gauss-Seidal Method. The Interpolation is Newton's Divided differences Interpolation Polynomial, Lagrange Polynomials, Least-Square Fit and Linear Interpolation. Ordinary Differential Equation is solved by Euler Method and Runge–Kutta Methods. The problem statement, the steps to obtain the solution, the results, and discussion related to the solution were presented where necessary.

In 21st century, significant changes in the workplace are the result of new and advanced technology. For the past generation, technological inventions and improvements seem to be introduced every week. Software apps are reaching tens of millions of users within weeks. Underlying these developments: the extraordinary speed at which mobile computers are spreading. This project seeks to improve the existing theory positivity and strong stability preserving discretizations, and to develop robust positivity preserving methods for realistic applications. So we need this android application to run with modern world to save time and technology. Solutions of complex problems are given by this application one click away from the android device.

Table Of Contents

0 NUMERICAL COMPUTING

0.1 INTRODUCTION

Numerical Computing is the study of algorithms that use numerical approximation for the problems of mathematical analysis. The field of numerical computing predates the invention of modern computers by many centuries. Many great mathematicians of the past were preoccupied by numerical computing, as is obvious from the names of important algorithms like Newton-Rapson Method, Bisection Method, Secant Method, Gaussian elimination, or Euler's method.

0.2 BACKGROUND

We are doing Bachelors Program in Computer Science at the Department of Computer Science (UBIT) at the University of Karachi. The field of Computer Science studies the design, analysis, implementation and application of computation and computer technology. Computing plays an important role in virtually all fields, including science and medicine, music and art, business, law, and human communication; hence the study of Computer Science and Engineering can be interdisciplinary in nature. Our university prepare us to understood the general areas of *software systems*, *hardware*, *theory* and *applications*.

0.3 AIMS AND OBJECTIVES

The core objectives which have been designated as fundamental to the project are:

- Identify, understand and describe a range of Numerical Computing method to solve the different problems.
- Numerical Computing application calculate different problem in short time as possible.
- As this project is based on android phone so everyone can calculate different algorithm of numerical Computing at any time or any place.

0.4 SUMMARY

The goal of this project “Mobile App For Numerical Computing Method” is that to solve the different method by using Numerical Computing algorithm that can solve and analysis of techniques to give approximate but accurate solutions to quadratic equation by using the numerical analysis algorithms.

1 PROJECT MODELING AND MANAGEMENT

1.1 PROJECT OVERVIEW

Title of Project

Mobile App For Numerical Computing Method

Problem Statement

Using a mobile phone has become a primary necessity for many people This application would completely Allow the people to solved the Numerical Computing Method with the mobile phone.

Team of Project

S. Abu Owais Bin Nasar

M. Umair Khan

Osama Ahmed

Team Formation / Structure

Our team formation is centralized. We follow the collaborative approach, for better coordination and inter-communication throughout each phase of the project.

Technology used

We have used **Eclipse 7.1 and Android SDK** as a basic tool, and for documentation and designing we have used MS word, MS Project, Adobe Photoshop CS5, and MS Visio.

Distribution of work

We focused on utilizing each members expertise, so the whole group worked together and compiled the

results for a better and feasible outcome.

1.2 PROBLEM STATEMENT

We have seen some android app with respect to Numerical Computing like Bisection Method or Secant Method but these are not complete. So we decided to do something like this but should be complete Numerical Computing method, even though making an android application is not an easy task. So we need this android application to run with modern world to save time and technology. Solutions of complex problems are given by this application one click away from the android device.

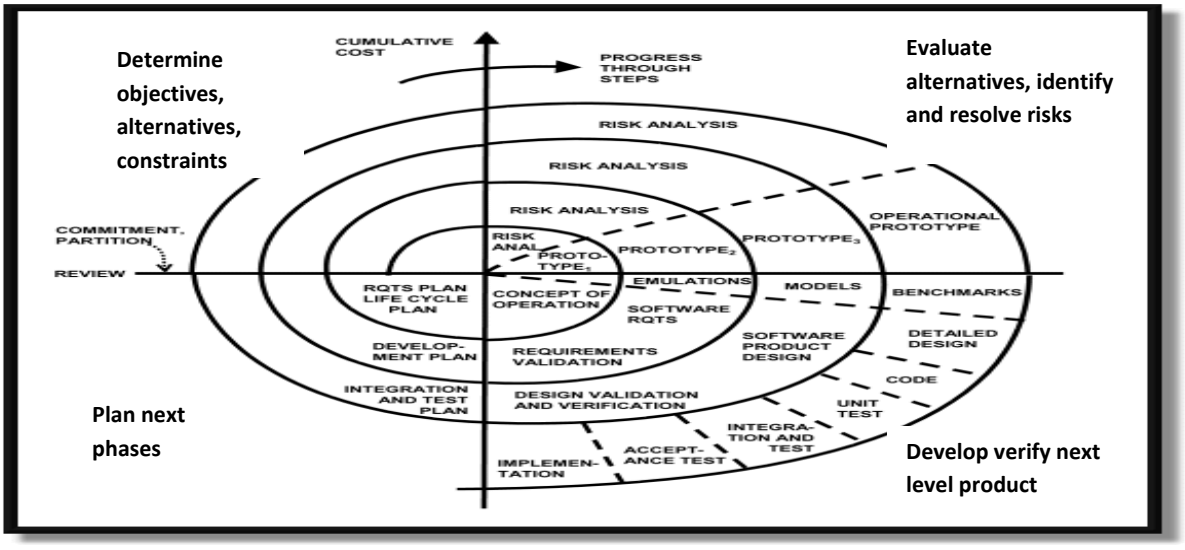
1.3 SUMMARY OF METHODOLOGY

Our approach is to build and provide such a supporting hand for people Andriod phone users which would help them to calculate the Numerical Computing method by using their .

1.4 WHAT GENERAL DEVELOPMENT APPROACH WILL BE USED?

By looking into the suitable software process models for such a project undertaking, we came to the conclusion of following the Spiral Process Model. The benefit of this approach is that it is extensive and led us to accommodate and obtain the flexibility of the project easily and achieve the required result in a proper manner.

SPIRAL MODEL



1.5 RISK ASSESSMENT

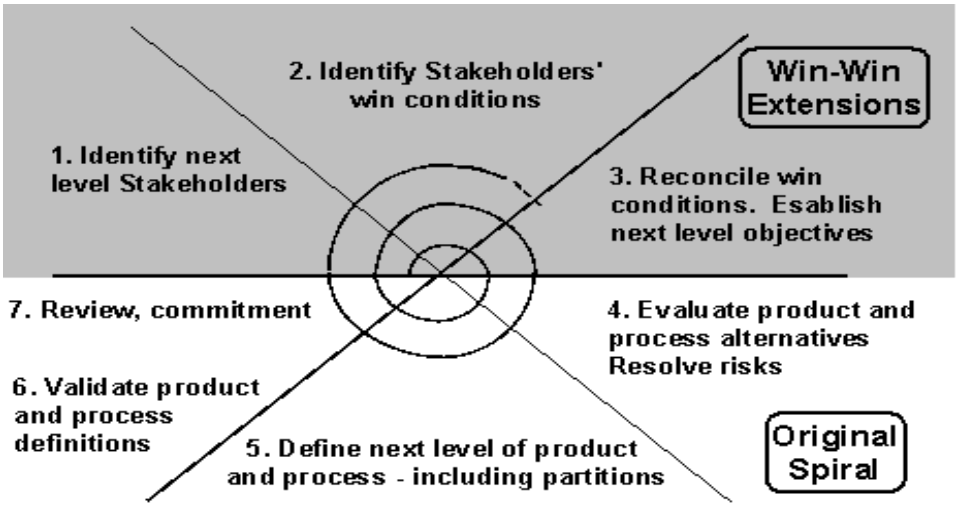
- Spiral Model – risk driven rather than document driven.
- The "risk" inherent in an activity is a measure of the uncertainty of the outcome of that activity.
- High-risk activities cause schedule and cost overruns.
- Risk is related to the amount and quality of available information. The less information, the higher the risk.

1.6 SPIRAL MODEL STRENGTH AND WEAKNESSES

- STRENGTHS
  - Introduces risk management.
  - Prototyping controls costs.
  - Evolutionary development.
  - Release builds for beta testing.
  - Marketing advantage.
- WEAKNESSES
  - Lack of risk management experience.
  - Lack of milestones.
  - Management is dubious of spiral process.
  - Change in management.
  - Prototype Vs Production.

1.7 WIN-WIN SPIRAL MODEL

Win-Win Spiral Process Model is a model of a process based on Theory W, which is a management theory and approach "based on making winners of all of the system's key stakeholders as a necessary and sufficient condition for project success."



- Identifying the system's stakeholders and their win conditions.
- Reconciling win conditions through negotiation to arrive at a mutually satisfactory set of objectives, constraints, and alternatives for the next level.
- Evaluate Product and Process Alternatives. Resolve Risks.
- Define next level of product and process - including partitions.
- Validate Product and Process Definitions.
- Review commitment.

1.8 HOW THE PROJECT TEAM WILL BE ORGANIZED?

The development team consists of four members. Every team member's work is to update their work status weekly. Tasks are distributed among all members and everyone is required to follow the deadlines.

There are different phases of our project like:

- Specification to list down the needs and requirements.
- Designing for how the development strategies will work.
- Coding and implementation.
- Testing.

1.9 WHAT DEVELOPMENT AND COLLABORATION TOOLS WILL BE USED?

We plan to use the following tools extensively throughout the project:

Development:

- Eclipse 7.1
- Andriod phone sdk

Documentation and Designing:

- Adobe Photoshop CS5.
- MS PowerPoint.
- MS Word.

1.10 HOW WILL CHANGES BE CONTROLLED?

- With the complete feature milestone achieved, no new features will be added to this release.
- After the complete code milestone, no entirely new product source code will be added to this release.
- We will also focus on what should be and what should not be included in the project's daily report.

1.11 HOW WILL THIS PLAN BE UPDATED?

This project plan will be updated as needed throughout the project. Every task and change in the requirements and functions is noted carefully, so that the project moves along smoothly. Any change to the plan will be notified to all members of the group and the project in-charge, subject to his approval.

2 INTRODUCTION

2.1 ANDROID PHONE :

Android is a mobile operating system (OS) based on the Linux kernel and currently developed by Google. With a user interface based on direct manipulation, Android is designed primarily for touch screen mobile devices such as smart phones and tablet computers, with specialized user interfaces for televisions (Android TV), cars (Android Auto), and wrist watches (Android Wear). The OS uses touch inputs that loosely correspond to real-world actions, like swiping, tapping, pinching, and reverse pinching to manipulate on-screen objects, and a virtual keyboard. Despite being primarily designed for touch screen input, it also has been used in game consoles, digital cameras, regular PCs (e.g. the HP Slate 21) and other electronics.

Android is the most widely used mobile OS and, as of 2013, the highest selling OS overall. Android devices sell more than Microsoft Windows, iOS, and Mac OS X devices combined with sales in 2012, 2013 and 2014 close to the installed base of all PCs. As of July 2013 the Google Play store has had over one million Android applications ("apps") published, and over 50 billion applications downloaded. A developer survey conducted in April–May 2013 found that 71% of mobile developers develop for Android. At Google I/O 2014, the company revealed that there were over one billion active monthly Android users, up from 538 million in June 2013.

Android's source code is released by Google under open source licenses, although most Android devices ultimately ship with a combination of open source and proprietary software, including proprietary software developed and licensed by Google.Initially developed by Android, Inc., which Google backed financially and later bought in 2005, Android was unveiled in 2007 along with the founding of the Open Handset Alliance—a consortium of hardware, software, and telecommunication companies devoted to advancing open standards for mobile devices

2.2 PRACTICE:

A mobile app, short for mobile application or just app, is application software designed to run on smart phones, tablet computers and other mobile devices. Apps are usually available through application distribution platforms, which began appearing in 2008 and are typically operated by the owner of the mobile operating system, such as the Apple App Store, Google Play, Windows Phone Store, and BlackBerry App World. Some apps are free, while others must be bought. Usually, they are downloaded from the platform to a target device, such as an iPhone, BlackBerry, Android phone or Windows Phone, but sometimes they can be downloaded to laptops or desktop computers.

2.3 PURPOSE

This project uses the Google Android platform to build an application numerical computing algorithms. The following are the algorithm:

- Bisection Method
- Newton-Rapson Method
- Secant Method
- False Position Method
- Cramer's Rule
- Elimination of Unknowns Y Combining
- Gauss-Seidal Method
- Newton's Divided differences Interpolation Polynomial
- Lagrange Polynomials
- Least-Square Fit
- Linear Interpolation
- Euler Method
- Runge–Kutta Methods

Numerical Computing app that the solving and analysis of techniques to give approximate but accurate solutions to quadratic equation by using the numerical analysis algorithms.

2.4 PRODUCT SCOPE

Numerical Computing is the area of mathematics and computer science that creates, analyzes, and implements algorithms for solving numerically the problems of continuous mathematics. Such problems originate generally from real-world applications of algebra, geometry, and calculus, and they involve variables which vary continuously. These problems occur throughout the natural sciences, social sciences, medicine, engineering, and business. The formal academic area of numerical analysis varies from highly theoretical mathematical studies to computer science issues involving the effects of computer hardware and software on the implementation of specific algorithms.this application is helpful to people attach with the fields of economics and finance,optimization analyst ,risk management ,operation research ,etc etc . we need this app to run with modern world to save time and technology . Solutions of complex problems are given by this application one click away from the android device.

2.5 SOFTWARE DESIGN DESCRIPTION

2.6 DESIGN OVERVIEW

Our application structure is purely a Android mobile based application. Includes development tools and languages Eclipse 7.1 as a basic tool, and for documentation and designing we have used MS word, MS project and Adobe Photoshop CS4. A flow structures will be generated for developers' use as it is a high level view of how the software will be structured.

2.7 REQUIREMENT TRACEABILITY MATRIX

The purpose of this document is to analyze the requirement, designing and specification phase at a glance.

2.8 SOFTWARE & HARDWARE INTERFACE

Though not necessarily concerned with the hardware interface, the application is purely a Android -based software with no specific hardware requirements.

2.9 USER INTERFACE DESIGN DESCRIPTION

As the application is developed by the Eclipse Android-based development scheme, so we didn't require any additional graphics for the software application. We kept in mind the user's ease, thus developing software with a simple and friendly user interface.

3 ERROR ANALYSIS

3.1 INTRODUCTION

There are several potential sources of errors in a numerical calculation. Two sources are universal in the sense that they occur in any numerical computation. They are round-off and truncation errors. Inaccuracies of numerical computations due to the errors result in a deviation of a numerical solution from the exact solution, no matter whether the latter is known explicitly or not. To examine the effects of finite precision of a numerical solution

3.2 SIMPLE ERROR DEFINITIONS

There are a number of ways to describe errors in measurements and calculations. The simplest is the absolute error, this is the difference between the measured or calculated value and the true value.

A shortcoming of the absolute error is that it doesn't take into account the order of magnitude of the value under consideration. One way to account for the magnitude is to consider instead the relative error.

Many numerical methods are interative, that is they involve repeating the same calculation many times. In terms of error analysis, two types of error emerge, local and global errors. The local error is the error introduced during one operation of the iterative process. The global error is the accumulative error over many iterations. Note that the global error is not simply the sum of the local errors due to the nonlinear nature of many problems although often it is assumed to be so because of the difficulties in measuring the global error.

3.3 SOURCES OF ERRORS IN NUMERICAL CALCULATIONS

There are at least two sources of errors in numerical calculations:

1. Rounding Errors 2. Truncation Errors

3.4 ROUND-OFF ERRORS:

Round-off error occurs because computers use fixed number of bits and hence fixed number of binary digits to represent numbers. In a numerical computation round-off errors are introduced at every stage of computation. Hence though an individual round-off error due to a given number at a given numerical step may be small but the cumulative effect can be significant.

Consider the number pi. It is irrational, i.e. it has infinitely many digits after the period:

pi = 3.14159265358979...

The round-off error of computer representation of the number pi depends on how many digits are left out. Make sure that you understand each line of the following rounding off the number pi:

number of digits	approximation for pi	absolute error	relative error
1	3.100	0.141593	1.3239%
2	3.140	0.001593	0.0507%
3	3.142	0.000407	0.0130%

Round-off errors may accumulate, propagate and even lead to catastrophic cancellations leading to loss of accuracy of numerical calculations.

3.5 TRUNCATION ERRORS

Many mathematical processes are infinite (e.g. the sine series), which in practice have to be terminated at some point. The terms omitted (which are infinite in number) introduce

an error into the result. This error is called the truncation error, since it is caused by the truncation of an infinite process. Many of the algorithms used in processing data and calculating results use infinite series or processes so this error is of great importance.

The decision to be taken is at what point to terminate the process. This can be a simple examination of the relative change in the result between two terms of the process or even when a term falls below a particular number, e.g. in computing sine we could ignore all terms after 10^{-8}. This is usually fine if the series is well behaved and converges smoothly but if the terms can vary in size making the convergence not so smooth some other test must be applied. For example, the series could be truncated if the change in the result was below a certain value over ten terms (or even more).

Consider another irrational number e:

e = 2.71828182845905...

and compare it with the Taylor series of the function exp(x) near the given point x = 0:

$$exp(x) = 1 + x + \frac{x2}{2} + \frac{x3}{6} + \ ...$$

Check a few Taylor series approximations of the number e = exp(1):

order of n	approximation for e	absolute error	relative error
3	2.500000	0.218282	8.030140%
4	2.666667	0.051615	1.898816%
5	2.708333	0.009948	0.365984%

3.6 ADVANTAGE AND DISADVANTAGE

OF

TRUNCATION AND ROUND-OFF ERRORS:-

- As the amount of information is increased to decrease truncation error, the size of each round-off error, which is determined by the number of digits used in the calculation, remains constant. Hence, no amount of additional information can reduce the error below some minimum unless the precision of the calculation is increased.

- The number of arithmetic operations in the calculation increases as the amount of information increases, so that the total error due to round-off increases because it is the combined effect of an increasing number of round-off errors. In these calculations, the error in the answer, which is the sum of the effects of truncation errors and round-off errors, initially decreases as the amount of information used increases and the truncation error decreases, but then increases as the round-off errors become dominant.

- Round-off errors, which are proportional to the size of the value containing the error when a fixed number of significant digits is used, can be as large as 5 in the first neglected place or one-half in the last retained place. When a value is added to a much smaller value, the round-off can be large relative to the smaller value. For example, if 23,456 is added to 10.518 in five digits, the result is 23,467, with a round-off error of -0.482. If 23,456 is subtracted from the answer, the result is 11. Thus, in five-digit arithmetic, (23,456 + 10.518) - 23,456 = 11.0, which has an error in the third place. After the addition, the error was less than one-half in the last place, but the subtraction removed the three leading digits, 234, moving this error to the third place. This phenomenon is called cancellation. It does not cause errors but makes the size of errors already introduced larger relative to the

computed result. Thus, although round-off errors are small, their effect in the final answer can be large, so that one of the tasks of the numerical analyst is to devise or modify computational schemes to minimize the effect of these errors

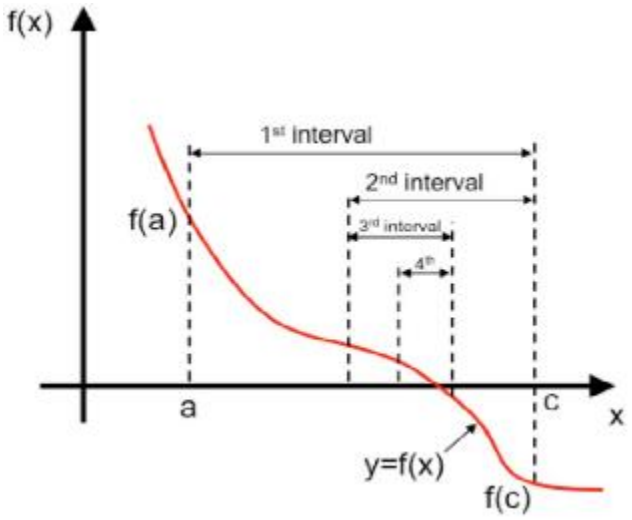
3.7 SUMMARY

Thus one can see that the error from rounding is considerably less than that from chopping. In fact the rounding error never exceeds the chopping error and is less about half the time.

4 ROOT OF EQUATION
BISECTION METHOD

4.1 INTRODUCTION

Bisection method is very simple but time consuming method. In this method, we first define an interval in which our solution of equation. Bisection method uses the bisecting (divide the range by 2) principle. In this method we minimize the range of solution by dividing it by integer 2.



Following are the steps to find the approximate solution of given equation using Bisection method:

Let us assume that we have to find out the roots of f(x), whose solution is lies in the range (a,b), which we have to determine. The only condition for bisection method is that f(a) and f(b) should have opposite signs (f(a) negative and f(b) positive). When f(a) and f(b) are of opposite signs at least one real root between 'a' and 'b' should exist.

For the first approximation we assume that root to be,

x0=(a+b)/2

Then we have to find sign of f(x0).

If f(x0) is negative the root lies between a and x0. If f(x0) is positive the root lies between x0 and b.

Now we have new minimized range, in which our root lies.

The next approximation is given by,

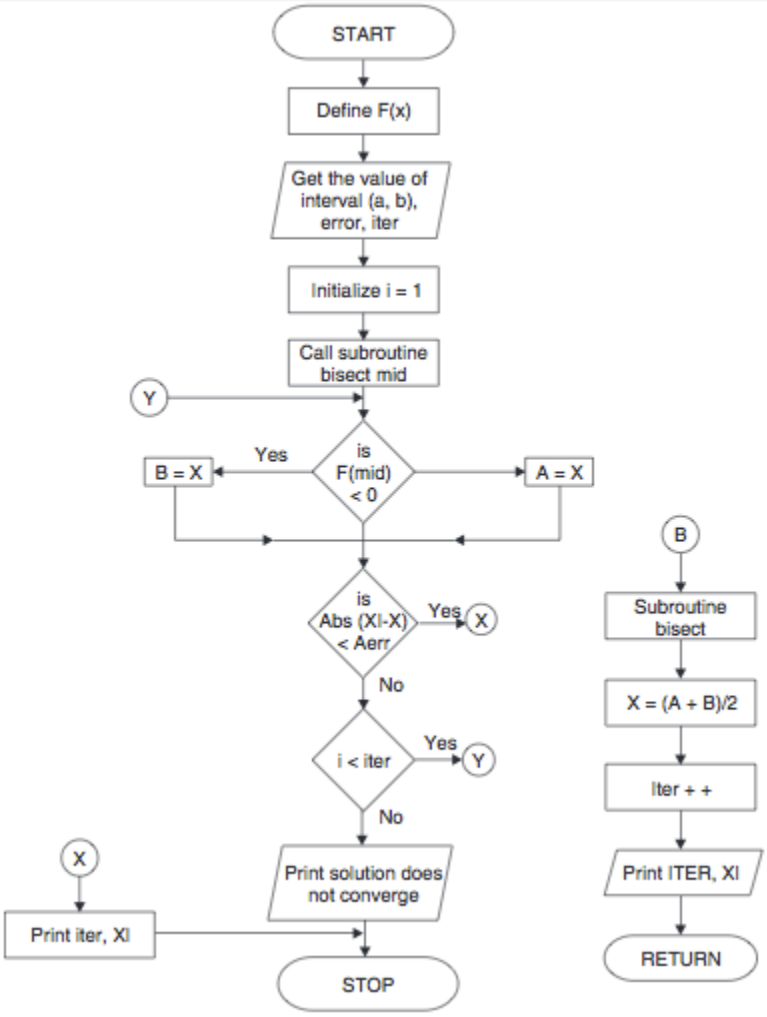
x1 = (a+x0)/2if f(x0) is negative.

x1 = (x0+b)/2 if f(x0) is positive.

4.2 ALGORITHM FOR BISECTION METHOD:

1. Input function and limits.
2. Repeat steps 3 and 4 100 times.
3. x=(a+b)/2
4. If f(x0)<0, a=x else b=x
5. Display x
6. Repeat steps 7 and 8 10 times.
7. Error = x-(a+b)/2
8. Store error values in array
9. Plot error
10. STOP.

4.3 FLOW CHART



The algorithm and flowchart presented above can be used to understand how bisection method works and to write program for bisection method in any programming language.

Example: find the root of f(x) = x2 − 5 between [0, 4] for iteration 3.

Solution

f(0) = 02 − 5 = −5

f(4) = 42 − 5 = 11

a = 0; f(a) = -5

b = 4; f(b) = 11

Iteration 1:

m = (a + b)/2 = 2

f(m) = 22 − 5 = -1

Because f(m) < 0, we replace a with

a = 2; f(a) = -1

b = 4; f(b) = 11

Iteration 2:

m = (a + b)/2 = 3

f(m) = 32 − 5 = 4

Because f(m) > 0, we replace b with m

a = 2; f(a) = -1

b = 3; f(b) = 4

Iteration 3:

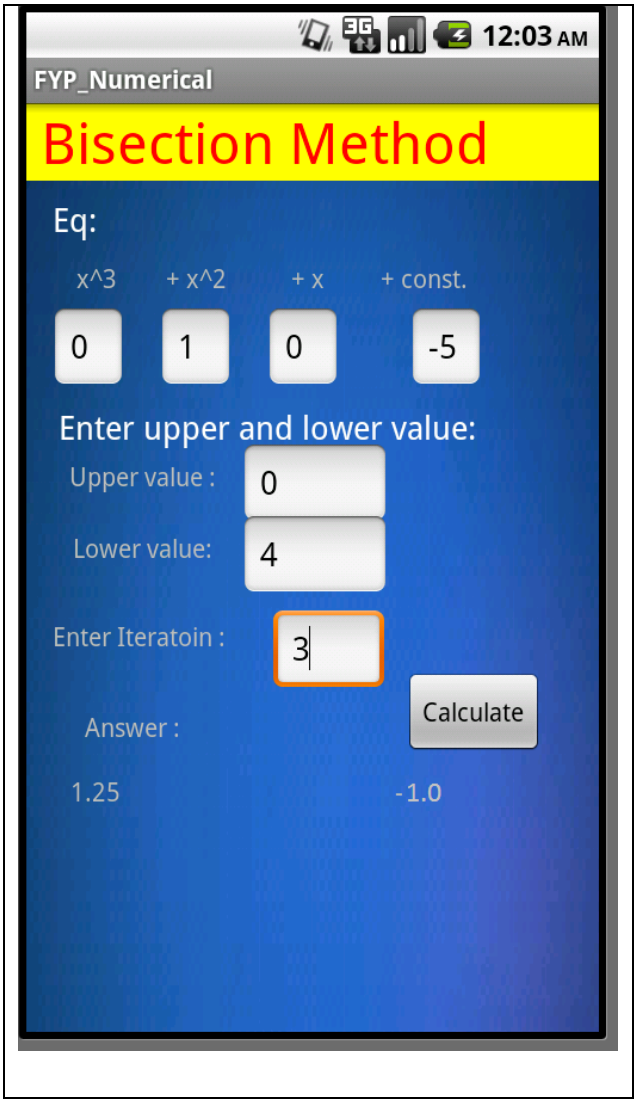
m = (a + b)/2 = 2.5

f(m) = 2.52 − 5 = 1.25

Because f(m) > 0, we replace b with m

a = 2; f(a) = -1

b = 2.5; f(b) = 1.25



Error Analysis:

|Ea|=|(Xnew-Xold)/Xnew|*100

|Ea|=|(2.5-2)/2.5|*100

|Ea|= 20%

Best Estimate and Error Level

The best estimate of the zero of the function f(x) after the first iteration of the Bisection method is the mid point of the initial interval:

$$Estimate of\ the\ zero:\ r = \frac{b+a}{2}$$

$$Error \leq \frac{b-a}{2}$$

Different approaches in selecting the solution can be used. The approach that will be used here is the middle point of the final interval. Doing this we guarantee that the error in estimating the zero of ef of ex is less than or equal to the half of the size of the final interval.

4.4 STOPPING CRITERIA

Two commonly used way to stop the Bisection method.

Stop after a given fixed number of iterations

Or

Stop when the error is less than a specified value.

In fact if one is given the initial interval and the error level, the number of iterations needed to achieve this accuracy can be calculated easily. Similarly if the initial interval

and the number of iterations are given then one can obtain an upper bound on the resulted error.

After n iterations:

$$|error| = |r - c_n| \leq E_a^n = \frac{b-a}{2^n} = \frac{\Delta x^0}{2^n}$$

The above criteria are related. Fixing the number of iteration, you can get a formula relating the error to the number of iterations.

Let c en be the midpoint of the interval after n iterations and let r be the zero of the function

The error after n iterations is less than b minus a over two to the power n.

4.5 CONVERGENCE ANALYSIS

Given ef of x , a, b and epsilon

We need to determine the number of iterations such that the absolute of r minus x is less than epsilon.

R here is the zero of the function and x is the bisection estimate. That is x equals see kay

$$n \geq \frac{\log(b-a) - \log(\varepsilon)}{\log(2)}$$

The number of iterations n is given by the equation

N is greater than or equal to log of b minus a minus log of epsilon all over log of two.

Let

$a = 6,\ b = 7,\ \varepsilon = 0.0005$

How many iterations are needed such that: $|x - r| \leq \varepsilon$?

$$n \geq \frac{\log(b-a) - \log(\varepsilon)}{\log(2)} = \frac{\log(1) - \log(0.0005)}{\log(2)} = 10.9658$$

$$\Rightarrow n \geq 11$$

In this example the formula for the number of iterations is used to answer the question

How many iterations of the bisection method is needed so that the error is less than epsilon

Here ay is 6, bee is 7 and epsilon is 0.0005

Direct substitution results in the inequality n must be greater than or equal to eleven.

4.6 ADVANTAGES

- Simple and easy to implement
- One function evaluation per iteration
- The size of the interval containing the zero is reduced by 50% after each iteration
- The number of iterations can be determined a priori
- No knowledge of the derivative is needed
- The function does not have to be differentiable

4.7 DISADVANTAGE

- Slow to converge
- Good intermediate approximations may be discarded

4.8 CONCLUSION

Bisection method guarantees the convergence of a function f(x) if it is continuous on the interval [a,b] (denoted by x1 and x2 in the above algorithm. For this, f(a) and f(b) should be of opposite nature i.e. opposite signs.

The slow convergence in bisection method is due to the fact that the absolute error is halved at each step. Due to this the method undergoes linear convergence, which is comparatively slower than the Newton-Raphson method, Secant method and False Position method.

5 NEWTON RAPHSON METHOD

5.1 INTRODUCTION

If you've ever tried to find a root of a complicated function algebraically, you may have had some difficulty. Using some basic concepts of calculus, we have ways of numerically evaluating roots of complicated functions. Commonly, we use the Newton-Raphson method. This iterative process follows a set guideline to approximate one root, considering the function, its derivative, and an initial x-value.

Newton's method is commonly used to find the root of an equation. If the root is simple then the extension to Halley's method will increase the order of convergence from quadratic to cubic.

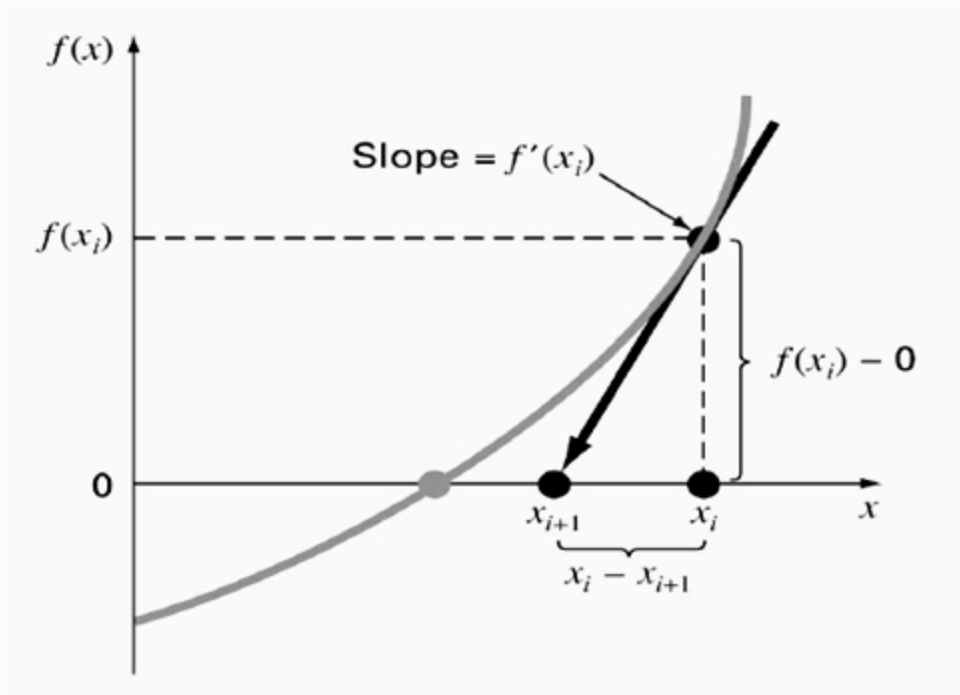
Unlike the earlier methods, this method requires only one appropriate starting point X0 as an initial assumption of the root of the function f(x) = 0. At (x0, f(x0)) a tangent to f(x) = 0 is drawn. Equation of this tangent is given by

$$y=f'(x0)(x-x0)+f(x0)$$

The point of intersection, say x1, of this tangent with x-asis (y = 0) is taken to be the next approximation to the root of f(x) = 0. So on substituting y = 0 in the tangent equation we get

$$x1=x0-f(x0)/f'(x0)$$

if |f(x1)<E|=10-6 (say) we have got an acceptable approximate root of f(x)=0 , otherwise we replace x0 by x1,and draw a tangent to f(x)=0 at (x1,f(x1))and consider its intersection, say x2,with x-axis as an improved approximation to the root of f(x)=0. If |f(x2)|>E we iterate the above process till the convergence criteria is satisfied. This geometrical description of the method may be clearly visualized in the figure below:



The various steps involved in calculating the root of f(x)=0 by Newton Raphson Method are described compactly in the algorithm below.

5.2 NEWTON RAPHSON METHOD ALGORITHM:

1. Start
2. Read x, e, n, d

   *x is the initial guess

   e is the absolute error i.e the desired degree of accuracy

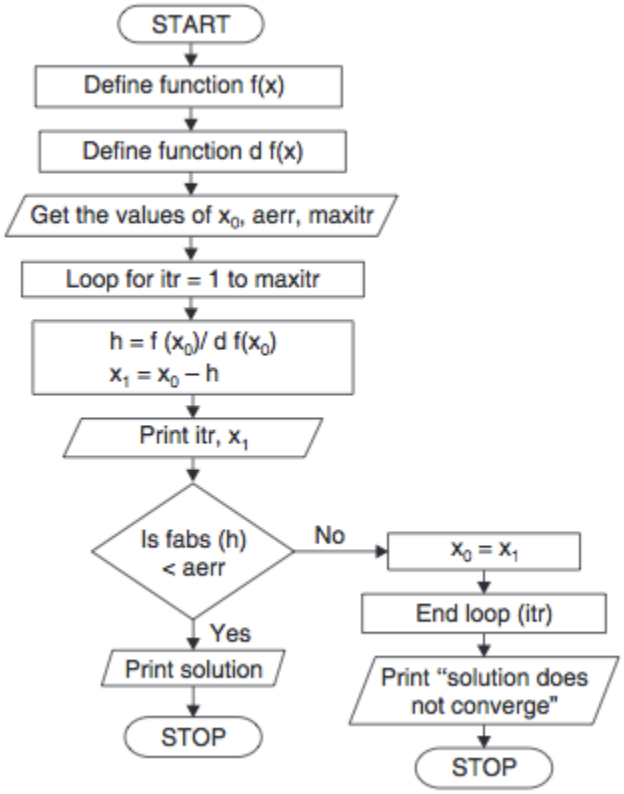
   n is for operating loop

   d is for checking slope*
3. Do for i =1 to n in step of 2
4. f = f(x)
5. f1 = f'(x)
6. If ([f1] < d), then display too small slope and goto 11.

   [ ] is used as modulus sign
7. x1 = x – f/f1

8. If ([(x1 – x)/x1] < e), the display the root as x1 and goto 11.
   [ ] is used as modulus sign
9. x = x1 and end loop
10. Display method does not converge due to oscillation.
11. Stop

5.3 NEWTON RAPHSON METHOD FLOWCHART:



These algorithm and flowchart can be used to write source code for Newton's method in any high level programming language.

5.4 LIMITATIONS OF NEWTON-RAPHSON METHOD:

Although the Newton Raphson method is considered fast, there are some limitations. These are listed below:

- Finding the f'(x) i.e. the first derivative of f(x) can be difficult in cases where f(x) is complicated.

- When f'(xn) i.e. the first derivative of f(xn) tends to zero, Newton Raphson gives no solution.
- Infinite oscillation resulting in slow convergence near local maxima or minima.
- If the initial guess is far from the desired root, then the method may converge to some other roots. So, Newton Raphson method is quite sensitive to the starting value.
- The method cannot be applied suitably when the graph of f(x) is nearly horizontal while crossing the x-axis.
- If root jumping occurs, the intended solution is not obtained.

Example:

Solve $2x^3 - 2.5x - 5 = 0$ for the root in [1,2] by Newton Raphson method.

Solution:

Given

$f(x) = 2x^3 - 2.5x - 5 = 0$

$f'(x) = 6x^2 - 2.5$

$Take\ x0 = 2\ and\ \ epsilon\ = 0$

$f(x0) = 6$

$f'(x0) = 29.5$

$$x1 = x0\ - \frac{f(x0)}{f'(x0)}$$

$$= 2 - \left(\frac{6.0}{29.5}\right)$$

$$= 1.709302187$$

$Since, \quad |f(x1)| = 0.8910911679 > 10^{-6}$

Repeat the process until optimal solution not found.

Find a zero of the function $f(x) = x^3 - 2x^2 + x - 3$, $x_0 = 4$

$f'(x) = 3x^2 - 4x + 1$

Iteration1:

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

$$= 4 - \frac{33}{33}$$

$$= 3$$

Iteration2:

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

$$= 3 - \frac{9}{16}$$

$$= 2.4375$$

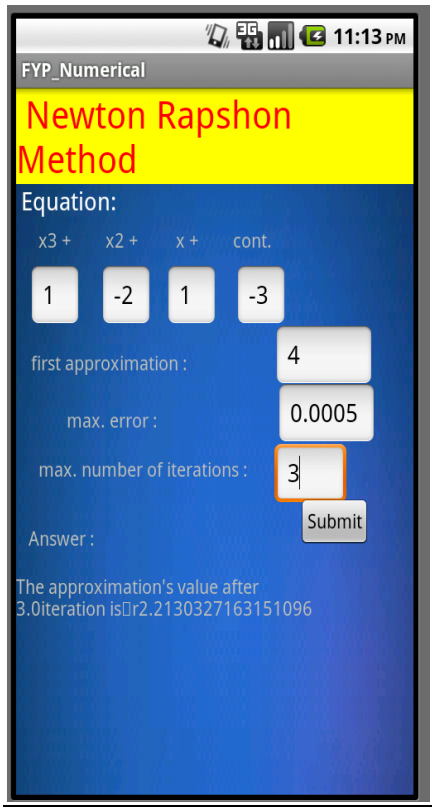
Iteration3:

$$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)}$$

$$= 2.4375 - \frac{2.0369}{9.0742}$$

$$= 2.2130$$

Repeat the process until optimal solution not found.



5.5 ***CONVERGENCE ANALYSIS***

Theorem:

Let $f(x), f'(x)$ *and* $f''(x)$ be continuous at $x \approx r$ where $f(r) = 0$. *If* $f'(r) \neq 0$ then there exists $\delta > 0$ such that $|x_0 - r| \leq \delta \Rightarrow \frac{|x_{k+1} - r|}{|x_k - r|^2} \leq C$

$$C = \frac{1}{2} \frac{\max_{|x_0 - r| \leq \delta} |f''(x)|}{\min_{|x_0 - r| \leq \delta} |f'(x)|}$$

When the guess is close enough to a simple root of the function then Newton's method is guaranteed to converge quadratically.

Quadratic convergence means that the number of correct digits is nearly doubled at each iteration.

5.6 PROBLEMS WITH NEWTON'S METHOD

If the initial guess of the root is far from the root the method may not converge.

Newton's method converges linearly near multiple zeros { $f(r) = f'(r) = 0$ }. In such a case, modified algorithms can be used to regain the quadratic convergence.

5.7 ADVANTAGES OF NEWTON-RAPHSON

- One of the fastest convergences to the root
- Converges on the root quadraticly
- Near a root, the number of significant digits approximately doubles with each step.
- This leads to the ability of the Newton-Raphson Method to "polish" a root from another convergence technique
- Easy to convert to multiple dimensions
- Can be used to "polish" a root found by other methods

5.8 DISADVANTAGES OF NEWTON-RAPHSON

- Must find the derivative
- Poor global convergence properties
- Dependent on initial guess
- May be too far from local root
- May encounter a zero derivative
- May loop indefinitely

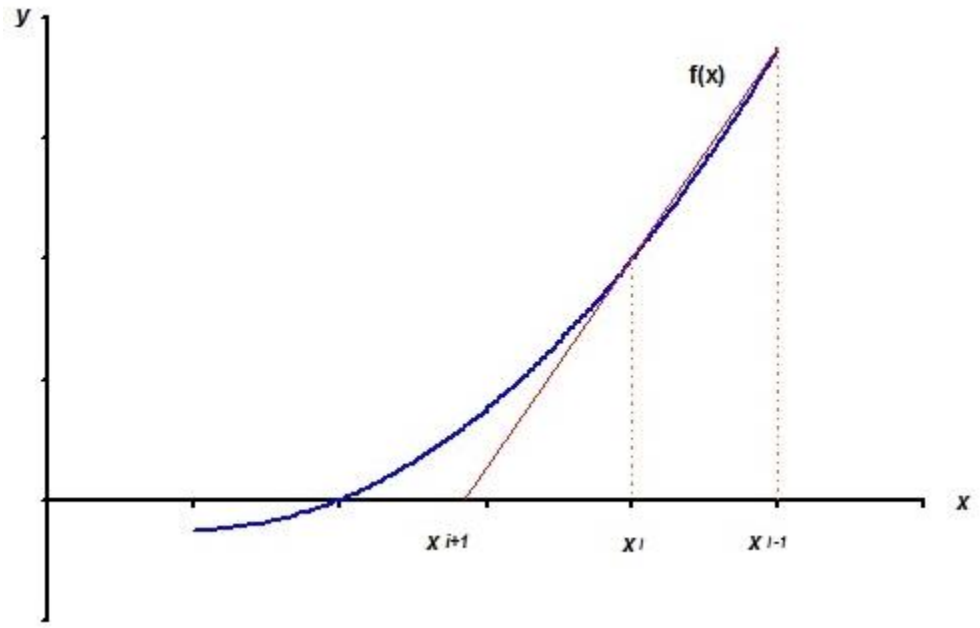
5.9 NEWTON-RAPHSON METHOD SUMMARY

The Newton method to find the root of equations is widely used. The derivation of the formula has been described, and an example shown step by step. It has some limitations, as it cannot by itself detect when no real roots exist, and the result to which the iteration process leads is highly dependent on the choice of the very first guess, x1.

6 SECANT METHOD

6.1 INTRODUCTION

Although the Newton-Raphson method is very powerfull to solve non-linear equations, evaluating of the function derivative is the major difficulty of this method. To overcome this deficiency, the secant method starts the iteration by employing two starting points and approximates the function derivative by evaluating of the slope of the line passing through these points.



Let x0 and x1 are two initial approximations for the root 's' of f(x) = 0 and f(x0) & f(x1) respectively, are their function values. If x 2 is the point of intersection of x-axis and the line-joining the points (x0, f(x0)) and (x1, f(x1)) then x2 is closer to 's' than x0 and x1. The equation relating x0, x1 and x2 is found by considering the slope 'm'

$$m = \frac{f(x1) - f(x0)}{x1 - x0} = \frac{f(x2) - f(x1)}{x2 - x1} = \frac{0 - f(x1)}{x2 - x1}$$

$$x2 - x1 = \frac{-f(x1) * (x1 - x0)}{f(x1) - f(x0)}$$

$$x2 \;=\; x1 - \left(\frac{f(x1) * \;(x1 - x0)}{f(x1) - \;f(x0)}\right)$$

or in general the iterative process can be written as

$$xi + 1 = \; xi \; - \frac{\left(f(xi) * \;\; (xi \; - \; xi - 1\,)\right)}{f(xi) - \; f(xi - 1)}$$

6.2 SECANT METHOD ALGORITHM:

1. Start
2. Get values of x0, x1 and e

   *Here x0 and x1 are the two initial guesses

   e is the stopping criteria, absolute error or the desired degree of accuracy*
3. Compute f(x0) and f(x1)
4. Compute x2 = [x0*f(x1) – x1*f(x0)] / [f(x1) – f(x0)]
5. Test for accuracy of x2

   If [(x2 – x1)/x2] > e, *Here [ ] is used as modulus sign*

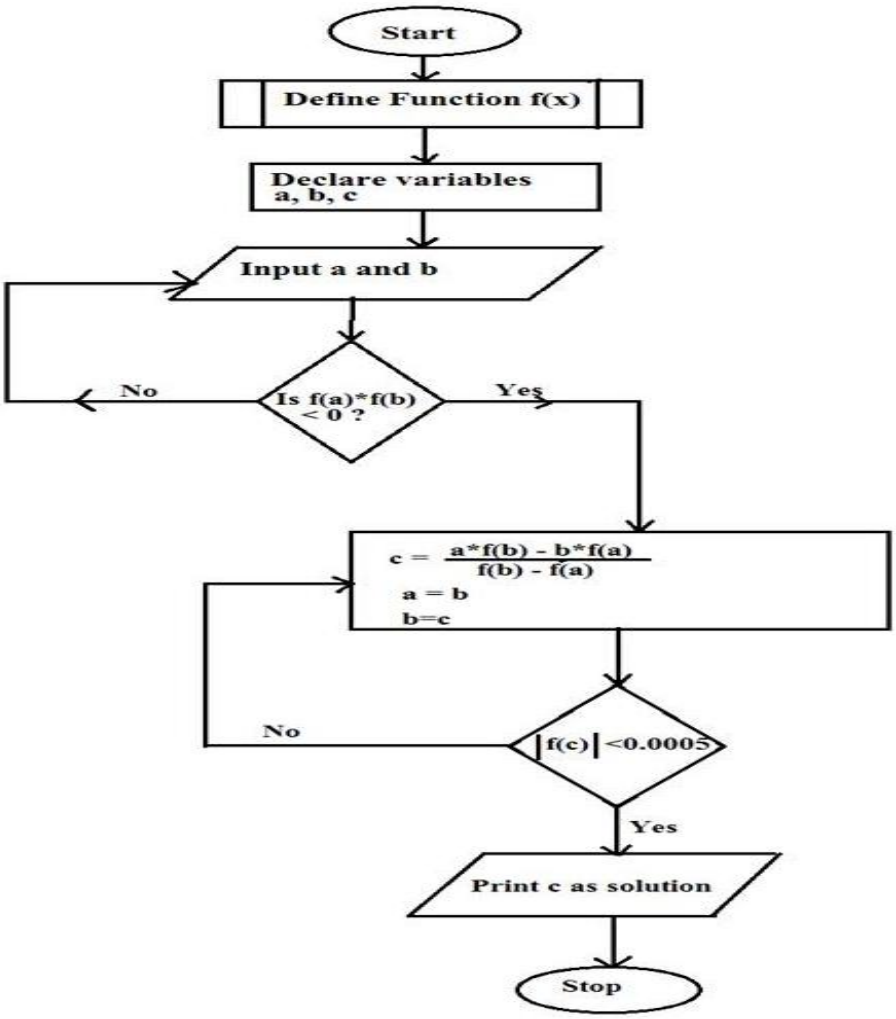
   then assign x0 = x1 and x1 = x2

   goto step 4

   Else,

   goto step 6
6. Display the required root as x2.
7. Stop

6.3 ***SECANT METHOD FLOWCHART:***



Example: Secant Method Solution of

$f\ (x)\ x2\ = -\ 2, x0\ =\ 1, x1\ =\ 2$

Solution

$y0\ =\ f\ (x0)\ = -1, y1\ =\ f\ (x1)\ =\ 2$

$$x2 = \frac{x0y1 - x1y0}{y1 - y0}$$

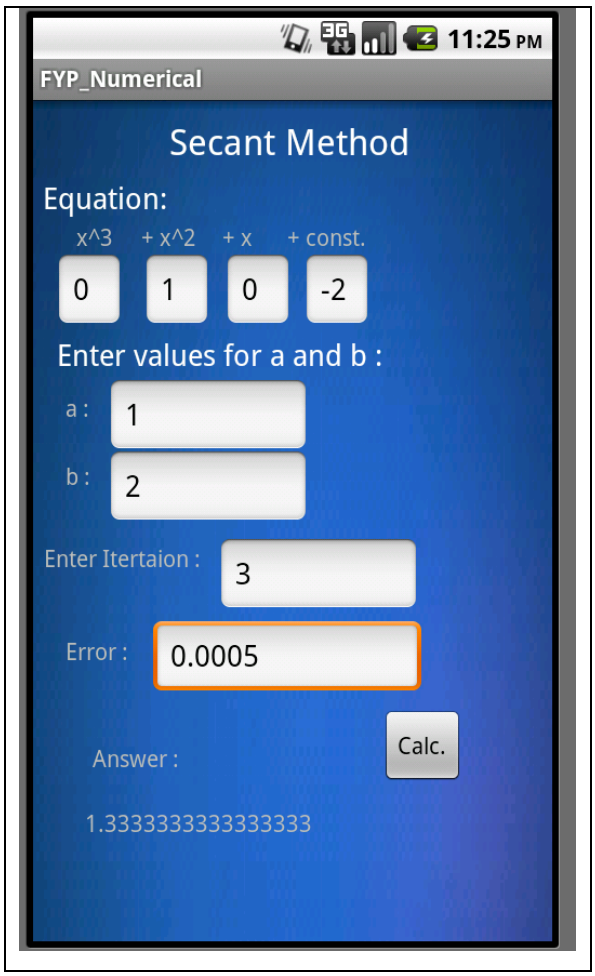
$$= \frac{(1)(2) - (2)(-1)}{3}$$

$$= \frac{4}{3} \text{»} 1.3333$$

$$y2 = f(x2) \approx -0.2222$$

$$x3 = \frac{x1y2 - x2y1}{y2 - y1}$$

$$\approx \frac{(2)-(0.2222)-(1.3333)(2)}{(-0.2222)-(2)} = 1.33333$$



6.4 CONVERGENCE ANALYSIS

With a combination of algebraic manipulation and the mean-value theorem from calculus, we can show

$$\alpha - Xn + 1 = (\alpha - Xn)(\alpha - Xn - 1)\left[\frac{-f''(\xi n)}{2f'(\zeta n)}\right]$$

with ξn and ζn unknown points. The point ξn is located between the minimum and maximum of xn−1, xn, and α; and ζn is located between the minimum and maximum of xn−1 and xn. Recall for Newton's method that the Newton iterates satisfied

$$\alpha - Xn + 1 = (\alpha - Xn)^2\left[\frac{-f''(\xi n)}{2f'(\zeta n)}\right]$$

which closely resembles above.

Using, it can be shown that xn converges to α,and moreover,

$$limn \to \frac{\infty(|\alpha - xn + 1|)}{|\alpha - xn|^r}$$

$$= \left|\frac{f''(\alpha)}{2f'(\alpha)}\right|^{r-1} \equiv c$$

where $\frac{1}{2(1+sqrt(5))} = 1.62$. This assumes that x0 and x1 are chosen sufficiently close to α; and how close this is will vary with the function f . In addition, the above result assumes f (x) has two continuous derivatives for all x in some interval about α. The above says that when we are close to α, that

$$|\alpha - Xn + 1| \approx c\,|\alpha - xn|^r$$

This looks very much like the Newton result

$$\alpha - xn + 1 \approx M\,(\alpha - xn)^2,$$

$$M = \frac{-f\text{“}(\alpha)}{2f'(\alpha)}$$

and $c = |M|^{r-1}$ .Both the secant and Newton methods converge at faster than a linear rate, and they are called superlinear methods.

The secant method converge slower than Newton's method; but it is still quite rapid. It is rapid enough that we can prove

$$limn \to \infty \frac{(|Xn+1 - Xn|)}{|\alpha - Xn|} = 1$$

and therefore,

$$|\alpha - Xn| \approx |Xn+1 - Xn|$$

is a good error estimator.

6.5 ADVANTAGES & DISADVANTAGES

Advantages of secant method:

- It converges at faster than a linear rate, so that it is more rapidly convergent than the bisection method.
- It does not require use of the derivative of the function, something that is not available in a number of applications.
- It requires only one function evaluation per iteration, as compared with Newton's method which requires two.

6.6 DISADVANTAGES OF SECANT METHOD:

- It may not converge.
- There is no guaranteed error bound for the computed iterates.
- It is likely to have difficulty if f 0(α) = 0. This means the x-axis is tangent to the graph of y = f (x) at x = α.
- Newton's method generalizes more easily to new methods for solving simultaneous systems of nonlinear equations.

6.7 CONCLUSION

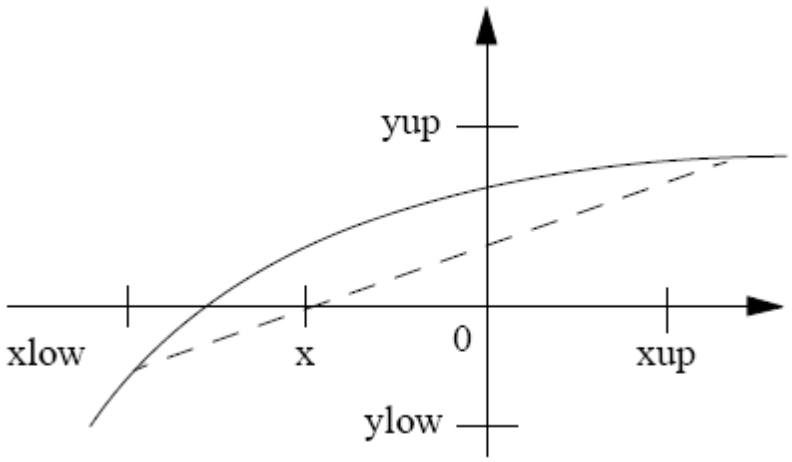
Secant method is an improvement over the Regula-Falsi method, as successive approximations are done using a secant line passing through the points during each iteration. Following the secant method algorithm and flowchart given above, it is not compulsory that the approximated interval should contain the root.

Secant method is faster than other numerical methods, except the Newton Raphson method. Its rate of convergence is 1.62, which is quite fast and high. However, convergence is not always guaranteed in this method. But, overall, this method proves to be the most economical one to find the root of a function.

7 FALSE-POSITION METHOD (REGULA FALSI)

7.1 INTRODUCTION

Improvement on bisection search by interpolating next *x* position, instead of having the interval.



By similar triangles:

$$\frac{yup - 0}{xup - x} = \frac{0 - ylow}{x - xlow}$$

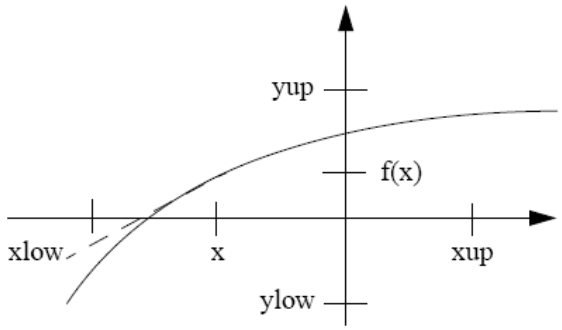
Or

$$x = \frac{xlow * yup - xup * ylow}{yup - ylow}$$

Then use this calculation in place of xmid in the Bisection Algorithm.

• False Position Method usually converges more rapidly than Bisection approach.

• Can improve false position method by adjusting interpolation line:

Each interpolation line (after first) is now drawn from (x, f(x)) to either (xlow, ylow/2) or to (xup, yup/2), depending on region containing root.	

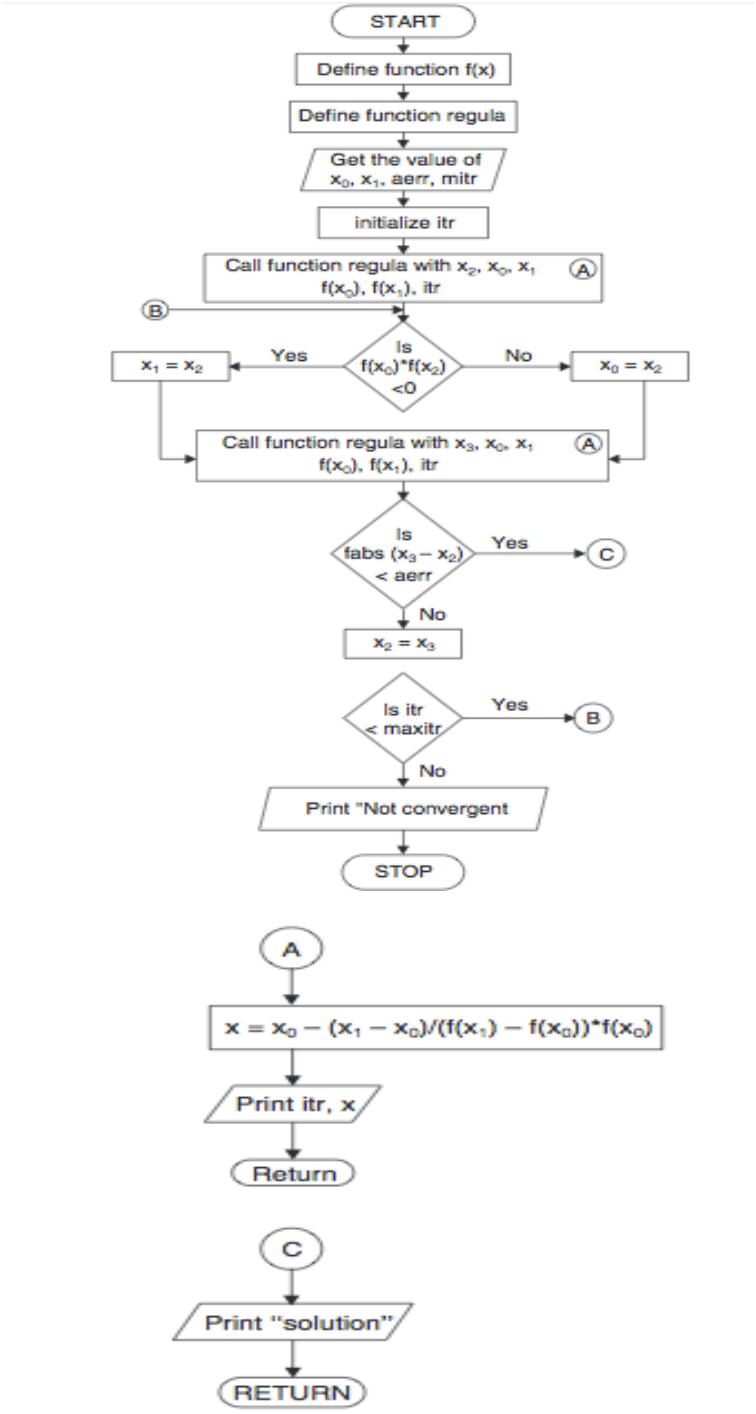
For continuous functions (single-valued), both Bisection and False Position are guaranteed to converge to root.

7.2 REGULA FALSI METHOD ALGORITHM:

1. Start
2. Read values of x0, x1 and e
   *Here x0 and x1 are the two initial guesses
   e is the degree of accuracy or the absolute error i.e. the stopping criteria*
3. Computer function values f(x0) and f(x1)
4. Check whether the product of f(x0) and f(x1) is negative or not.
   If it is positive take another initial guesses.
   If it is negative then goto step 5.
5. Determine:
   x = [x0*f(x1) – x1*f(x0)] / (f(x1) – f(x0))
6. Check whether the product of f(x1) and f(x) is negative or not.
   If it is negative, then assign x0 = x;
   If it is positive, assign x1 = x;
7. Check whether the value of f(x) is greater than 0.00001 or not.
   If yes, goto step 5.
   If no, goto step 8.
   Here the value 0.00001 is the desired degree of accuracy, and hence the stopping criteria.
8. Display the root as x.

9. Stop

7.3 REGULA FALSI METHOD FLOWCHART:



7.4 ADVANTAGES

- Simple and easy to implement
- One function evaluation per iteration
- The size of the interval containing the zero is reduced by 50% after each iteration
- The number of iterations can be determined a priori
- No knowledge of the derivative is needed
- The function does not have to be differentiable

7.5 DISADVANTAGE

- Slow to converge
- Good intermediate approximations may be discarded

7.6 CONCLUSION

False-Position Method guarantees the convergence of a function f(x) if it is continuous on the interval [a,b] (denoted by x1 and x2 in the above algorithm. For this, f(a) and f(b) should be of opposite nature i.e. opposite signs.

The slow convergence in False-Position method is due to the fact that the absolute error is halved at each step but faster than Bisection Method. Due to this the method undergoes linear convergence, which is comparatively slower than the Newton-Raphson method, and Secant method.

7.7 SUMMARY OF ROOT-FINDING METHODS

Method	Input	Converge?	Rate Converge
Bisection	xlow, xup	yes	linear
False Pos.	xlow, xup	yes	better
Secant	“any” 2 vals	maybe	better yet
Newton’s	any” 1 val	maybe	usually best

LINEAR ALGERAIC EQUATION

8 GAUSS-SEIDEL

8.1 INTRODUCTION

Although it seems that the Gauss elimination method gives an exact solution, the accuracy of this method is not very good in large systems. The main reason of inaccuracy in the gauss elimination method is round-off error because of huge mathematical operations of this technique. Furthermore, this method is very time consuming in large systems. Many of systems of linear algebraic equations which should be solved in engineering problems are large and there are lots of zeros in their coefficient matrix. To solve this kinds of problems, iterative methods often is used. Gauss-Seidel one of the iterative techniques, is very well-known because of its good performance in solving engineering problems. For a system of linear equation as follows:

$$a11x1 + a12x2 + \ldots + a1nxn = c1$$

$$a21x1 + a22x2 + \ldots + a2nxn = c2$$

.

.

.

$$an1x1 + an2x2 + \ldots + annxn = cn$$

where xi are unknowns and aij are coefficients of unknowns and ci are equations' constants. The Gauss-Seidel method needs a starting point as the first guess.

With the Jacobi method, the values of xi^k obtained in the kth iteration remain unchanged until the entire (k+1)th iteration has been calculated. With the Gauss-Seidel method, we use the new values xi^{k+1} as soon as they are known. For example, once we have

computed xi^{k+1} from the first equation, its value is then used in the second equation to obtain the new x_2^{k+1} and so on.

For each $k \geq 1$, generate the components xi^k of xi^k from by xi^{k-1}

$$x_i^k = \frac{1}{aii}\left[-\sum_{j=1}^{i-1}\left(aijx_j^k\right) - \sum_{j=i+1}^{n}\left(aijx_j^{k-1}\right) + bi\right], for\ i = 1,2,\dots n$$

Namly

$$a_{11}x_1^k = -a_{12}x_2^{k-1} - \cdots - a_{1n}x_n^{k-1} + b1$$

$$a_{21}x_1^k + a_{22}x_2^k = -a_{23}x_3^{k-1} - \cdots - a_{2n}x_n^{k-1} + b2$$

.

.

.

$$a_{n1}x_1^k + a_{n2}x_2^k + \cdots a_{nn}x_n^{k-1} = bn$$

8.2 MATRIX FORM OF GAUSS-SEIDEL METHOD.

$$(D - L)x^k = U_{x^{k-1}} + b$$

$$x^k = (D - L)^{-1}U_{x^{k-1}} + (D - L)^{-1}b$$

Define $$T_g = (D - L)^{-1}U$$

And $$c_g = (D - L)^{-1}b$$

Gauss-Seidel method can be written as

$$x^k = T_g x^{k-1} + c_g \quad k = 1,2,3,\dots.$$

8.3 ALGORITHM

Input: A=[aij],b ,XO=x^0 tolerance TOL, maximum number of iterations N.

Step 1 Set k=1

Step 2 while ($k \leq N$) do Steps 3-6

Step 3 For $i = 1,2, \dots, n$

$$x_i = \frac{1}{aii}\left[-\sum_{j=1}^{i-1}(aijx_j) - \sum_{j=i+1}^{n}(aijXO_j) + bi\right],$$

Step 4 If ||x - XO || < TOL, then OUTPUT ($x_1, x_2, \dots . x_n$);

Stop

Step 5 Set k=k+1.

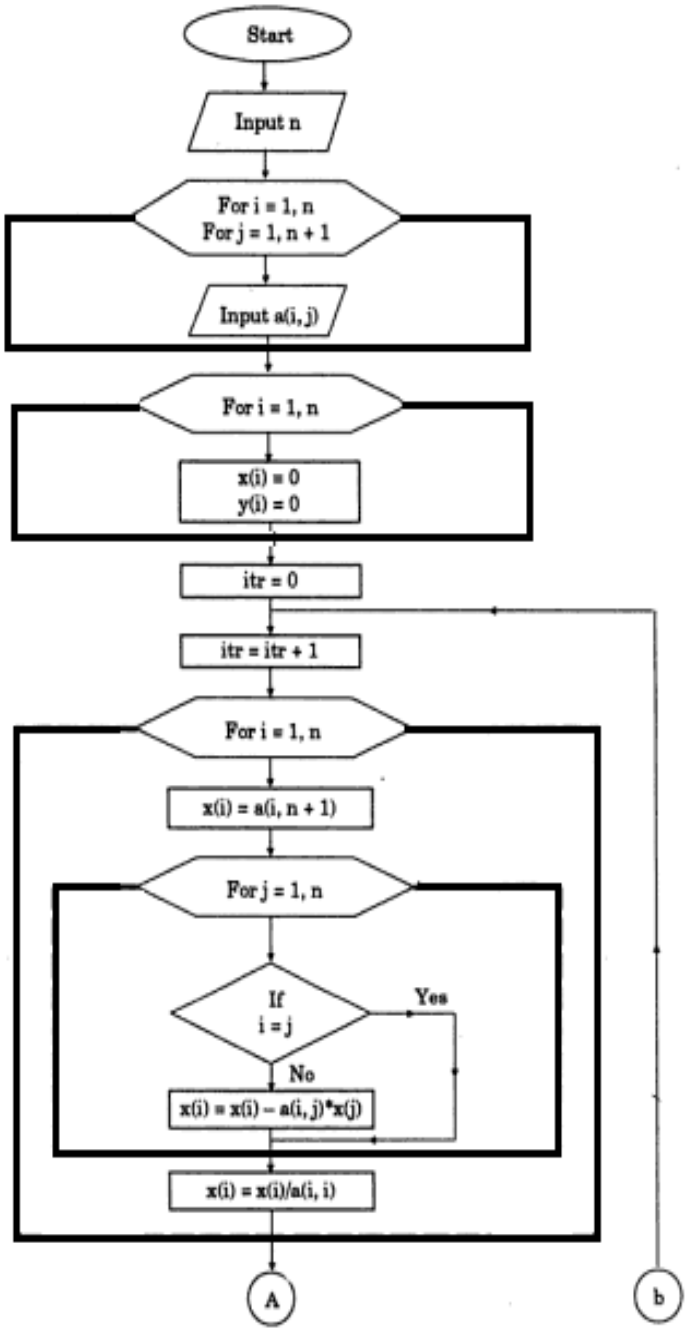
Step 6 For I = 1,2,…,n

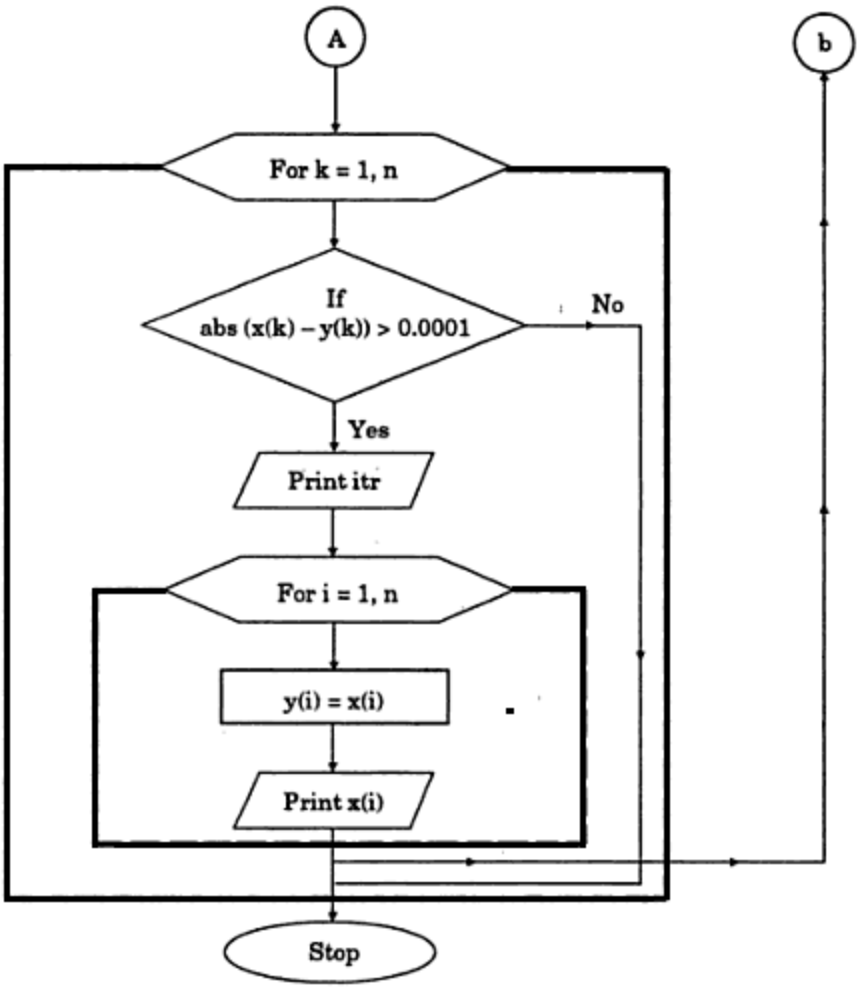
Set XOi=xi

Step 7 OUTPUT ($x_1, x_2, \dots . x_n$);

STOP.

8.4 FLOWCHART





Example

$4x1 + x2 - x3 = 3$

$2x1 + 7x2 + x3 = 19$

$x1 - 3x2 + 12x3 = 31$

Let

$$x1 = -\frac{1}{4}x2 + \frac{1}{4}x3 + \frac{3}{4}$$

$$x2 = -\frac{2}{7}x1 - \frac{1}{7}x3 + \frac{19}{7}$$

$$x3 = -\frac{1}{12}x1 + \frac{1}{4}x2 + \frac{31}{12}$$

Solution:

We have

$$\underline{x} = \begin{pmatrix} 0 & -\frac{1}{4} & \frac{1}{4} \\ -\frac{2}{7} & 0 & -\frac{1}{7} \\ -\frac{1}{12} & \frac{1}{4} & 0 \end{pmatrix} \begin{pmatrix} x1 \\ x2 \\ x3 \end{pmatrix} + \begin{pmatrix} \frac{3}{4} \\ \frac{19}{7} \\ \frac{31}{12} \end{pmatrix} = G\underline{x} + \underline{r}$$

The iteration formulas are

$$x1^{(k+1)} = -\frac{1}{4}x2^k + \frac{1}{4}x3^k + \frac{3}{4}$$

$$x2^{(k+1)} = -\frac{2}{7}x1^{k+1} - \frac{1}{7}x3^k + \frac{19}{7}$$

$$x3^{(k+1)} = -\frac{1}{12}x1^{k+1} + \frac{1}{4}x2^{k+1} + \frac{31}{12}$$

If we start from $x1^{(0)} = x2^{(0)} = x3^{(0)} = 0$ and apply the iteration formulas, we obtain

$k \quad x1^k \quad x2^k \quad x3^k$

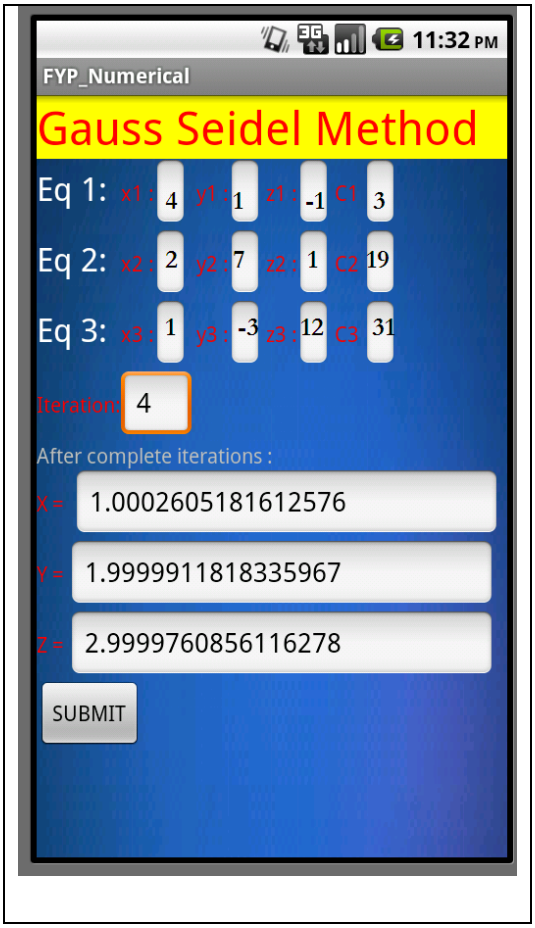
0 0 0 0

1 0,75 2,50 3,15

2 0,91 **2, 00** 3,01

3 1,00 2,00 3,00

4 1,00 2,00 3,00



The exact solution is: $x1 = 1,\ x2 = 2,\ x3 = 3.$

For instance, when *k*=2, we have $x_2^{(2)}$**= 2,00**:

$$x2^{(2)} = -\frac{2}{7}x1^{(2)} - \frac{1}{7}x3^{(1)} + \frac{19}{7}$$

$$= -\frac{2}{7}\cdot 0{,}72 - \frac{1}{7}\cdot 3{,}15 + \frac{19}{7} = \mathbf{2,00}$$

8.5 CONVERGENCE

Let the iteration method be written as

$$x^k = T_g x^{k-1} + c_g \quad for\ each\ k = 1,2,3,\ldots.$$

If the spectral radius satisfies $(T) < 1$, then $(1-T)^{-1}$exists, and

$$(I-T)^{-1} = I + T + T^2 + \cdots = \sum_{j=0}^{\infty} T^j$$

For any $x^0 \in R^n$, the sequence $\{x^k\}_{k=0}^{\infty}$defined by

$$x^k = T_g x^{k-1} + c_g \quad for\ each\ k \geq 1$$

converges to the unique solution of $x = Tx + c$ if and only if $\rho(T) < 1$

Proof (only show $\rho(T) < 1$ is sufficient condition)

$$x = Tx^{k-1} + c = T(Tx^{k-2} + c) + \ c = .. = T^k x^0 + (T^{k-1} + \cdots + T + 1)c$$

Since $\rho(T) < 1$,$\lim_{k\to\infty} T^k x^0 = 0$

$$\lim_{k\to\infty} x^k = 0 + \lim_{k\to\infty} \left(\sum_{j=0}^{k-1} T^j \right) c = (I - T)^{-1} c$$

If ||T<1 || for any natural matrix norm and c is a given vector, then the sequence $\{x^k\}_{k=0}^{\infty}$ defined by $x = Tx^{k-1} + c$ converges, for any$x^0 \in R^n$, to a vector $x \ \in R^n$ with x = Tx + c, and the following error bound hold:

i-$\left|\left|x - x^k\right|\right| \leq \left||T|\right|^k ||x^0 - \ x||$

ii-$\left|\left|x - x^k\right|\right| \leq \left|\left|\frac{||T||^k}{1-||T||}\right|\right| \ \ ||x^1 - \ x^0||$

If A is strictly diagonally dominant, then for any choice of x^0 both the Jacobi and Gauss-Seidel methods give sequences $\{x^k\}_{k=0}^{\infty}$ that converges to the unique solution of Ax=b.

8.6 GAUSS-SEIDEL ADVANTAGES

- Each iteration is relatively fast (computational order is proportional to number of branches + number of buses in the system).
- Relatively easy to program.

8.7 GAUSS-SEIDEL DISADVANTAGES

- Tends to converge relatively slowly, although this can be improved with acceleration.
- Has tendency to fail to find solutions, particularly on large systems.
- Tends to diverge on cases with negative branch reactances (common with compensated lines)

- Need to program using complex numbers.

9 CRAMER'S RULE

9.1 INTRODUCTION

Given a system of linear equations, Cramer's Rule is a handy way to solve for just one of the variables without having to solve the whole system of equations. They don't usually teach Cramer's Rule this way, but this is supposed to be the point of the Rule: instead of solving the entire system of equations, you can use Cramer's to solve for just one single variable.

It involves a quantity called the determinant. Every *m*×*m* matrix has a unique determinant. The determinant is a single number. To find the determinant of a m×m *matrix* , multiply the numbers on the downward diagonal and subtract the product of the numbers on the upward diagonal.

9.2 CRAMER'S RULE - TWO EQUATIONS

If we are given a pair of simultaneous equations

$a1x + b1y = d1$

$a2x + b2y = d2$

then *x*, and *y* can be found from

$$x = \frac{\begin{pmatrix} d1 & b1 \\ d2 & b2 \end{pmatrix}}{\begin{pmatrix} a1 & b1 \\ a2 & b2 \end{pmatrix}}$$

$$y = \frac{\begin{pmatrix} a1 & d1 \\ a2 & d2 \end{pmatrix}}{\begin{pmatrix} a1 & b1 \\ a2 & b2 \end{pmatrix}}$$

Solution

Using Cramer's rule we can write the solution as the ratio of two determinants.

$$x = \frac{\begin{pmatrix} -14 & 4 \\ 11 & -3 \end{pmatrix}}{\begin{pmatrix} 3 & 4 \\ -2 & -3 \end{pmatrix}}$$

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

$$= 2,$$

$$x = \frac{\begin{pmatrix} 3 & -14 \\ -2 & 11 \end{pmatrix}}{\begin{pmatrix} 3 & 4 \\ -2 & -3 \end{pmatrix}}$$

$$= \frac{5}{-1}$$

$$= -5$$

The solution of the simultaneous equations is then $x = 2$, $y = -5$.

9.3 CRAMER'S RULE - THREE EQUATIONS

To find the determinant of a 3×3 matrix, copy the first two columns of the matrix to the right of the original matrix. Next, multiply the numbers on the three downward diagonals, and add these products together. Multiply the numbers on the upward diagonals, and add *these* products together. Then subtract the sum of the products of the upward diagonals from the sum of the product of the downward diagonals (subtract the second number from the first number):

$a1x + b1y + c1z = d1$

$a2x + b2y + c2z = d2$

$a3x + b3y + c3z = d3$

then *x*, *y* and *z* can be found from

$$x = \frac{\begin{vmatrix} d_1 & b_1 & c_1 \\ d_2 & b_2 & c_2 \\ d_3 & b_3 & c_3 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix}} \qquad y = \frac{\begin{vmatrix} a_1 & d_1 & c_1 \\ a_2 & d_2 & c_2 \\ a_3 & d_3 & c_3 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix}} \qquad z = \frac{\begin{vmatrix} a_1 & b_1 & d_1 \\ a_2 & b_2 & d_2 \\ a_3 & b_3 & d_3 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix}}$$

Cramer's Rule states that:

$$x = \frac{detD_x}{detD}$$

$$y = \frac{detD_y}{detD}$$

$$z = \frac{detD_z}{detD}$$

Note: If $detD = 0$ and $detD_x$, $detD_y$, or $detD_z \neq 0$, the system is inconsistent. If $detD = 0$ and $detD_x = detD_y = detD_z = 0$, the system has multiple solutions.

Thus, to solve a system of three equations with three variables using Cramer's Rule,

1. Arrange the system in the following form:

   $a_1 x + b_1 y + c_1 z = d_1$
   $a_2 x + b_2 y + c_2 z = d_2$
   $a_3 x + b_3 y + c_3 z = d_3$

2. Create D, D_x, D_y, and D_z.

3. Find $detD$, $detD_x$, $detD_y$, and $detD_z$.

4. $x = \frac{detD_x}{detD}$, $y = \frac{detD_y}{detD}$, and $z = \frac{detD_z}{detD}$.

Solve the following equation and find the value of x, y, z.

$3x + y + z = 3$

$2x + 2y + 5z = -1$

$x - 3y - 4z = 2$

$$\Delta = \begin{vmatrix} 3 & 1 & 1 \\ 2 & 2 & 5 \\ 1 & -3 & -4 \end{vmatrix}$$

= 3(-8 + 15) – 1(-8 - 5) + 1(-6 - 2)

= 21 + 13 – 8

= 26

Here Determinant is not equal to zero, Hence Cramer's rule can be applicable.

$X = \frac{\Delta x}{\Delta}, y = \frac{\Delta y}{\Delta}, z = \frac{\Delta z}{\Delta}$

$$\Delta_x = \begin{vmatrix} 3 & 1 & 1 \\ -1 & 2 & 5 \\ 2 & -3 & -4 \end{vmatrix}$$

=3(-8 + 15) – 1(4 - 10) + 1(3 - 4)

= 21 + 6 – 1

Δ_x= 26

$$\Delta_y = \begin{vmatrix} 3 & 3 & 1 \\ 2 & -1 & 5 \\ 1 & 2 & -4 \end{vmatrix}$$

$= 3(4 - 10) – 3(-8 – 5) + 1(4 + 1)$

$=-18 + 39 + 5$

$\Delta_y = 26$

$$\Delta_z = \begin{vmatrix} 3 & 1 & 3 \\ 2 & 2 & -1 \\ 1 & -3 & 2 \end{vmatrix}$$

$= 3(4 - 3) – 1(4 + 1) + 3(-6 -2)$

$= 3 – 5 – 24$

$\Delta_z = -26$

By using Cramer's rule:

$$x = \frac{\Delta 1}{\Delta}$$

$$x = \frac{26}{26}$$

$$x = 1$$

$$y = \frac{\Delta 2}{\Delta}$$

$$y = \frac{26}{26}$$

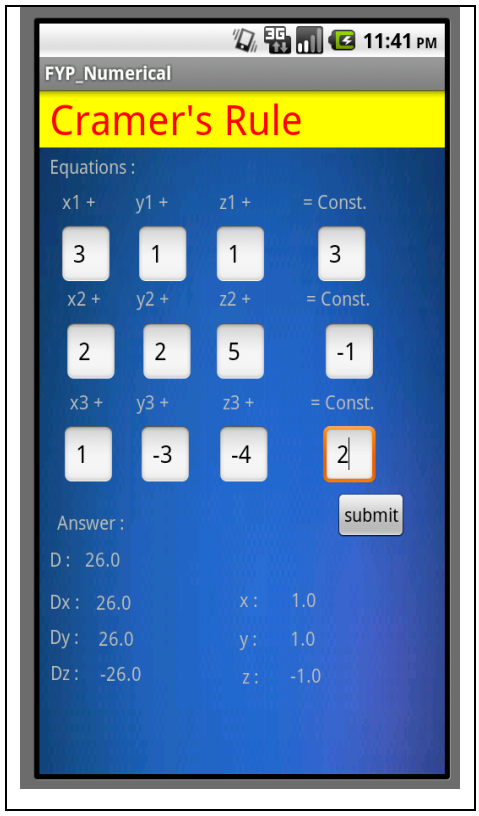
$$y = 1$$

$$z = \frac{\Delta 3}{\Delta}$$

$$z = \frac{26}{-26}$$

$$z = -1$$

Therefore the required x = 1, y = 1 & z = -1



9.4 ***ADVANTAGES***

- Cramer's rule has the advantage that less tedious computations are necessary if we are only interested in the solution value of one of the variables.
- The advantage of this method is that it allows you to solve for all the variables at once.
- Easy to remember steps

9.5 ***DISADVANTAGES***

- Computationally intensive compared to other methods: the most efficient ways of calculating the determinant of an n × n matrix require (n − 1)(n!) operations. So Cramer's rule would require
- (n − 1)((n + 1)!) total operations. For 8 equations, that works out to 7(9!) = 2540160 operations, or around 700 hours if you can perform one operation per second.
- Roundoff error may become significant on large problems with non-integer coefficients.

INTERPOLATION

10 WHAT IS INTERPOLATION?

10.1 INTRODUCTION

Many times, data is given only at discrete points such as .So, how then does one find the value of *y* at any other value of *x* ? Well, a continuous function *f x* may be used to represent the *n*1 data values with *fx* passing through the *n* 1 points (Figure 1). Then one can find the value of *y* at any other value of *x* .

This is called *interpolation*.

Of course, if *x* falls outside the range of *x* for which the data is given, it is no longer interpolation but instead is called *extrapolation*.

So what kind of function *f x* should one choose? A polynomial is a common choice for an interpolating function because polynomials are easy to

(A) evaluate,

(B) differentiate, and

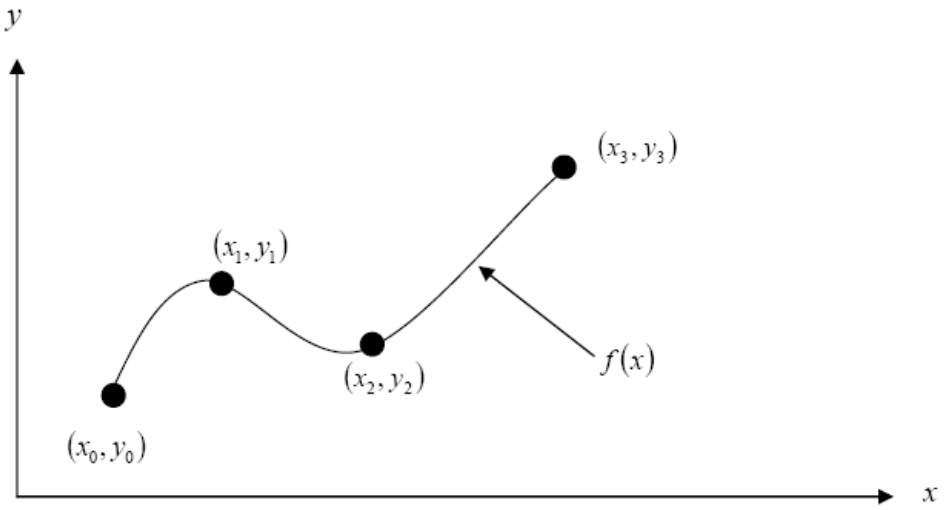
(C) integrate,

relative to other choices such as a trigonometric and exponential series.

Polynomial interpolation involves finding a polynomial of order *n* that passes through the *n* 1 points. One of the methods of interpolation is called Newton's divided difference polynomial method. Other methods include the direct method and the Lagrangian interpolation method.

10.2 NEWTON'S DIVIDED DIFFERENCE POLYNOMIAL METHOD

To illustrate this method, linear and quadratic interpolation is presented first. Then, the general form of Newton's divided difference polynomial method is presented.



10.3 GENERAL FORM OF NEWTON'S DIVIDED DIFFERENCE POLYNOMIAL

In the two previous cases, we found linear and quadratic interpolants for Newton's divided difference method. Let us revisit the quadratic polynomial interpolant formula.

Let us assume that the function f(x) is linear then we have

$$\frac{f(xi) - f(xj)}{xi - xj}$$

where $\mathbf{x_i}$ and $\mathbf{x_j}$ are any two tabular points, is independent of $\mathbf{x_i}$ and $\mathbf{x_j}$. This ratio is called the first divided difference of **f(x)** relative to $\mathbf{x_i}$ and $\mathbf{x_j}$ and is denoted by $\mathbf{f\,[x_i, x_j]}$.

$$f\,[xi, xj] = \frac{f(xi) - f(xj)}{xi - xj} = f\,[xj, xi]$$

Since the ratio is independent of xi *and* xj *we can write*

$$f\,[x0, x] = f\,[x0, x1]$$

$$\frac{f(x) - f(x0)}{x - x0} = f\,[x0, x1]$$

$$f(x) = f(x0) + (x - x0)f[x0, x1]$$

$$= \frac{1}{x0 - x}\begin{vmatrix} f(x0) & x0 - x \\ f(x1) & x1 - x \end{vmatrix} = \frac{f1 - f0}{x1 - x0}x + \frac{f0x1 - f1x0}{x1 - x0}$$

So if **f(x)** is approximated with a linear polynomial then the function value at any point **x** can be calculated by using **f(x)** ≈ **$P_1(x) = f(x_0) + (x - x_1) f[x_0, x_1]$**

where **$f[x_0, x_1]$** is the first divided difference of **f** relative to **x_0** and **x_1**.

Similarly if **f(x)** is a second degree polynomial then the secant slope defined above is not constant but a linear function of **x.** Hence we have

$$\frac{\boldsymbol{f[x1, x2] - f[x0, x1]}}{\boldsymbol{x2 - x0}}$$

is independent of **x_0, x_1** and **x_2**. This ratio is defined as second divided difference of **f** relative to**x_0, x_1** and **x_2**. The secind divided difference are denoted as

$$f[x0, x1, x2] = \frac{f[x1, x2] - f[x0, x1]}{x2 - x0}$$

Now again since **$f[x_0, x_1, x_2]$** is independent of **x_0, x_1** and **x_2** we have

$$f[x1, x0, x] = f[x0, x1, x2]$$

$$\frac{f[x0, x] - f[x1, x0]}{x - x1} = f[x0, x1, x2]$$

$$f[x0, x] = f[x0, x1] + (x - x1)f[x0, x1, x2]$$

$$\frac{f[x] - f[x0]}{x - x0} = f[x0, x1] + (x - x1)f[x0, x1, x2]$$

$$f(x) = f[x0] + (x - x0)f[x0, x1] + (x - x0)(x - x1)f[x0, x1, x2]$$

Example:

Given the following data table, evaluate f(2.4) using 3rd order Newton's Divided Difference interpolation polynomial.

i	0	1	2	3	4
xi	0	1	2	3	4
Yi=f(xi)	1	2.25	3.75	4.25	5.81

Solution:

Here n=5. For constructing 3rd order Newton Divided Difference polynomial we need only four points. Let us use the first four points. The 3rd Newton Divided Difference polynomial is given by:

$$p_3(x) = \sum_{k=0}^{3} a_k \prod_{j=0}^{\text{k-1}} (x - x_j) = \sum_{k=0}^{3} f[x_0 \ldots x_k] \prod_{j=0}^{\text{k-1}} (x - x_j)$$

$$= f[x0] + f[x0, x1](x - x0) + f[x0, x1, x2](x - x0)(x - x1)$$

$$+ f[x0, x1, x2, x3](x - x0)(x - x1)(x - x2)$$

$$therefore\ a0 = f[x0] = 1$$

$$a1 = f[x0, x1] = \frac{f(x1) - f(x0)}{(x1 - x0)} = \frac{2.25 - 1}{1 - 0} = 1.25$$

$$f[x1, x2] = \frac{f[x2] - f[x1]}{x2 - x1} = \frac{3.75 - 2.25}{2 - 1} = 1.5$$

$$a2 = f[x0, x1, x2] = \frac{f[x1, x2] - f[x0, x1]}{x2 - x0} = \frac{1.5 - 1.25}{2 - 0} = 0.125$$

$$f[x2, x3] = \frac{f[x3] - f[x2]}{x3 - x2} = \frac{4.25 - 3.75}{3 - 2} = \frac{0.5}{1} = 0.5$$

$$f[x1, x2, x3] = \frac{f[x2, x3] - f[x1, x2]}{x3 - x1} = \frac{0.5 - 1.5}{3} - 1 = -0.5\$$$

$$a3 = f[x0, x1, x2, x3] = \frac{f[x1, x2, x3] - f[x0, x1, x2]}{x3 - x0}$$

$$= \frac{-0.5 - 0.125}{3 - 0} = -\frac{0.625}{3} = -0.20833$$

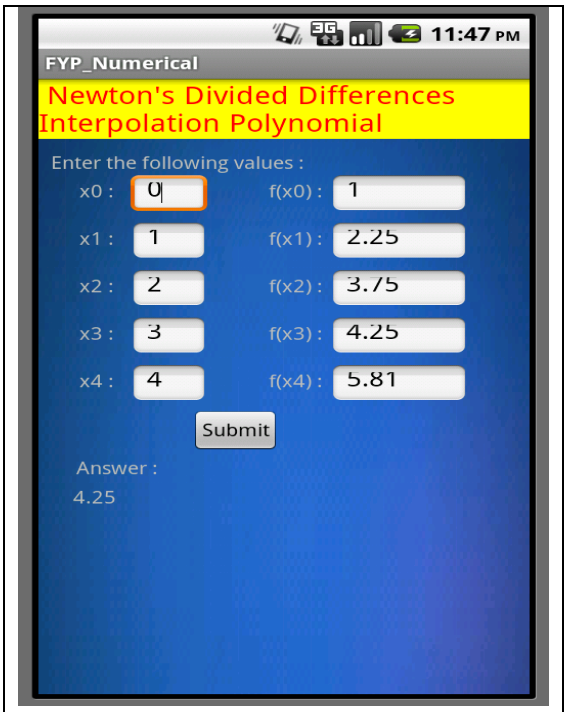
$$p3(x) = 1 + 1.25(x - 0) + 0.125(x - 0)(x - 1)$$
$$+ (-0.20833)(x - 0)(x - 1)(x - 2)$$

$$therefore\ f(2.4) = p3(2.4)$$

$$= 1 + 1.25(2.4 - 0) + 0.125(2.4 - 0)(2.4 - 1)$$
$$+ (-0.20833)(2.4 - 0)(2.4 - 1)(2.4 - 2)$$

$$= 1 + (1.25)(2.4) + 0.125(2.4)(1.4) - 0.20833(2.4)(1.4)(0.4)$$

$$= 4.2200032$$


11:47 PM
FYP_Numerical
Newton's Divided Differences Interpolation Polynomial
Enter the following values :
x0 : 0
f(x0) : 1
x1 : 1
f(x1) : 2.25
x2 : 2
f(x2) : 3.75
x3 : 3
f(x3) : 4.25
x4 : 4
f(x4) : 5.81
Submit
Answer :
4.25

11 LAGRANGE INTERPOLATION

11.1 INTRODUCTION

To construct a polynomial of degree n passing through n+1 data points (x0 , y0), (x1 , y1), ... , (xn, yn) we start by constructing a set of basis polynomials Ln,k (x) with the property that

$$Ln,k\ (xj\ )\ =\ \begin{cases}1 & when\ j\ =\ k\\ 0 & when\ j\ \neq\ k\end{cases}$$

These basis polynomials are easy to construct. For example for a sequence of x values {x0 , x1 , x2 , x3} we would have the four basis polynomials

$$L3,0(x) = \frac{(\ x\ -\ x1)(\ x\ -\ x2)(x\ -\ x3\ )}{(\ x0\ -\ x1)(\ x0\ -\ x2)(x0\ -\ x3\ )}$$

$$L3,1(x) = \frac{(\ x\ -\ x0)(\ x\ -\ x2)(x\ -\ x3\ )}{(\ x1\ -\ x0)(\ x1\ -\ x2)(x1\ -\ x3\ )}$$

$$L3,2(x) = \frac{(\ x\ -\ x0)(\ x\ -\ x1)(x\ -\ x3\ )}{(\ x2\ -\ x0)(\ x2\ -\ x1)(x2\ -\ x3\ )}$$

$$L3,3(x) = \frac{(\ x\ -\ x0)(\ x\ -\ x1)(x\ -\ x2\ )}{(\ x3\ -\ x0)(\ x3\ -\ x1)(x3\ -\ x2\ )}$$

If **f(x)** is approximated with an **N**th degree polynomial then the**N**th divided difference of **f(x)** constant and **(N+1)**th divided difference is zero. That is

$$\boldsymbol{f\ [x0, x1, \ldots xn, x]\ =\ 0}$$

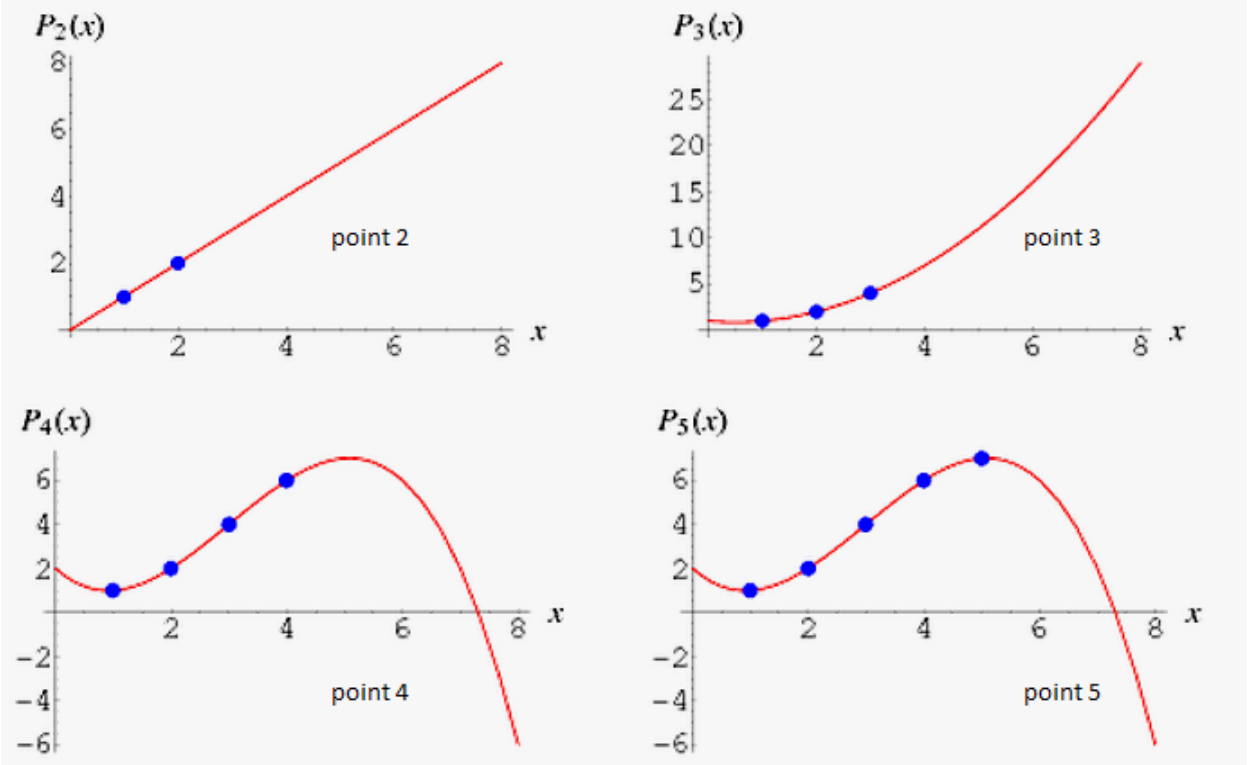
From the second property of divided difference we can write

$$\frac{f0}{(x0 - x1)\ldots(x0 - xn)(x0 - x)} + \frac{fn}{(xn - x0)\ldots(xn - xn-1)(xn - x)} + \ldots + \frac{fx}{(x - x0)\ldots(x - xn)} = 0$$

$$f(x) = \frac{(x - x1)\ldots(x - xn)}{(x0 - x1)\ldots(x0 - xn)} f0 + \ldots + \frac{(x - x0)\ldots(x - xn-1)}{(xn - x0)\ldots(xn - xn-1)} fn$$

$$\sum_{i=0}^{n} \left(\prod_{\substack{j=0 \\ j \neq 1}}^{n} \left(\frac{x - xi}{xi - xj} \right) \right) fi$$

Since Lagrange's interpolation is also an **N**th degree polynomial approximation to **f(x)** and the **N**th degree polynomial passing through **(N+1)** points is unique hence the Lagrange's and Newton's divided difference approximations are one and the same. However, Lagrange's formula is more convinent to use in computer programming and Newton's divided difference formula is more suited for hand calculations.

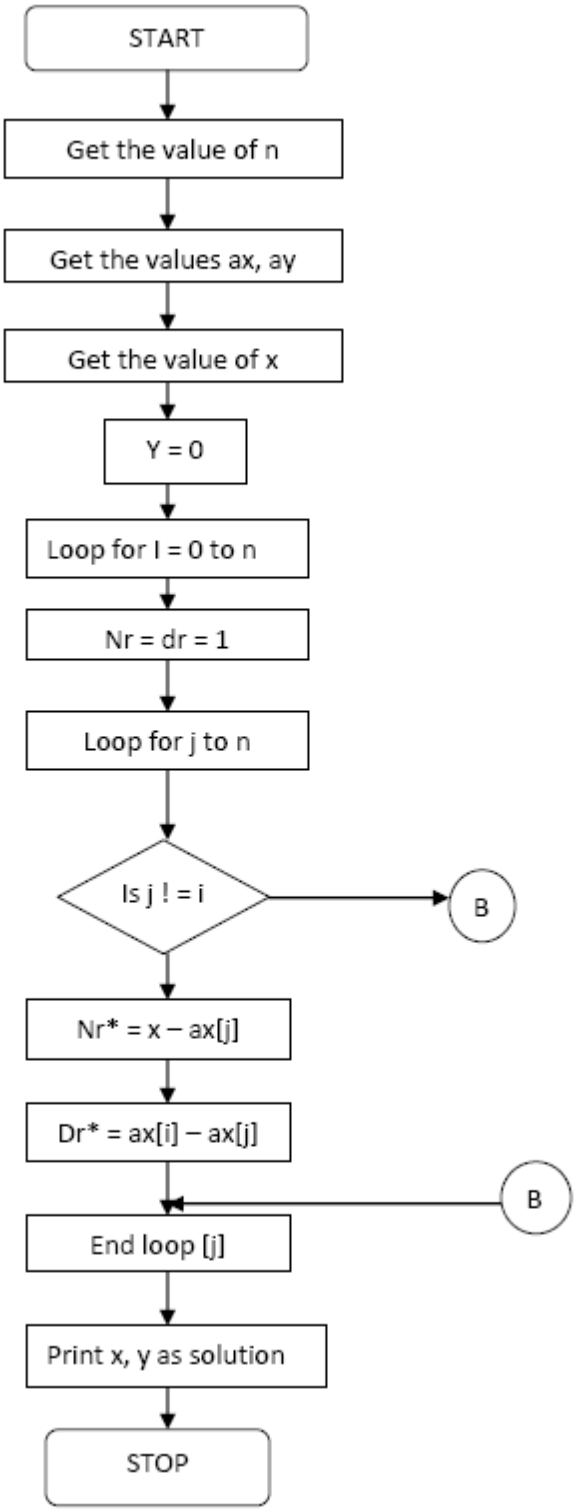


11.2 ALGORITHM

To implement the Lagrange Polynomial Interpolating Polynomial Algorithm to a set of data it is a matter of following the formula to the right for the set of data:

$\{(x_1,y_1),(x_2,y_2),(x_3,y_3)\ldots(x_m,y_m)\}$
alldiffx ← True
For[i=1, $i \le m$, i++
    For[j=1, j<i, j++,
        alldiffx=alldiffx and ($x_i \neq x_j$)]]
If alldiffx
    psum=0
    For[i=1, $i \le m$, i++
        pprod=y_i
        For[j=1, j<i, j++,
            If i≠j then pprod=pprod*($x - x_j$)/($x_i - x_j$)]
        psum=psum+pprod]
(* else *)
    Print[“Data set can not be interpolated by a function”]

11.3 FLOWCHART



Example : Compute **f(0.3)** for the data

Where

x	**0**	**1**	**3**	**4**	**7**
f	**1**	**3**	**49**	**129**	**813**

Solution

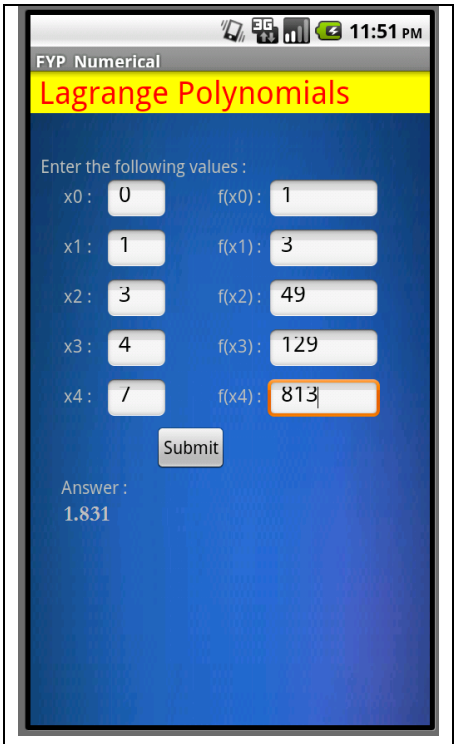
$$f(x) = \frac{(x - x_1)(x - x_2)(x - x_3)(x - x_4)}{(x_0 - x_1)(x_0 - x_2)(x_0 - x_3)(x_0 - x_4)} f_0 + \ldots + \frac{(x - x_0)(x - x_1)(x - x_2)(x - x_3)}{(x_4 - x_0)(x_4 - x_1)(x_4 - x_2)(x_4 - x_3)} f_4$$

$$= \frac{(0.3 - 1)(0.3 - 3)(0.3 - 4)(0.3 - 7)}{(-1)(-3)(-4)(-7)} 1 + \frac{(0.3 - 0)(0.3 - 3)(0.3 - 4)(0.3 - 7)}{1 \times (-2)(-3)(-6)} 3 +$$

$$\frac{(0.3 - 0)(0.3 - 1)(0.3 - 4)(0.3 - 7)}{3 \times 2 \times (-1)(-4)} 49 + \frac{(0.3 - 0)(0.3 - 1)(0.3 - 3)(0.3 - 7)}{4 \times 3 \times 1 (-3)} 129 +$$

$$\frac{(0.3 - 0)(0.3 - 1)(0.3 - 3)(0.3 - 4)}{7 \times 6 \times 4 \times 3} 813$$

= **1.831**



11.4 ADVANTAGES OF LAGRANGE INTERPOLATION

- The interpolation polynomial can be written down without the solution of a linear system of equations.
- The basis polynomials lj(t) depend only on t1 ….. tn and not on the values yi .
- The data may be unequally spaced.
- They are “perfect” in the curve fitting sense because they pass through all the data points.
- They are easy to use for evaluation, differentiation and integration.

11.5 DISADVANTAGES OF LAGRANGE INTERPOLATION

- It is extremely expensive to evaluate the polynomial.
- The Lagrange polynomial is only used for theoretical considerations.
- All of the work must be redone for each degree polynomia
- Extreme caution must be used to extrapolate for values outside the intervals of the independent variable for which data has been gathered.

- They tend to oscillate severely near the endpoints of the data range. This leads to errors when interpolating near the endpoints and extreme error when estimating the derivative near the endpoints.

11.6 SUMMARY

An advantage of Lagrange interpolation is that the method does not need evenly spaced values in *x*. However, it is usually preferable to search for the nearest value in the table and then use the lowest-order interpolation consistent with the functional form of the data. High-order polynomials that match many entries in the table simultaneously can introduce undesirable rapid fluctuations between tabulated values. If used for extrapolation with a high-order polynomial, this method may give serious errors.

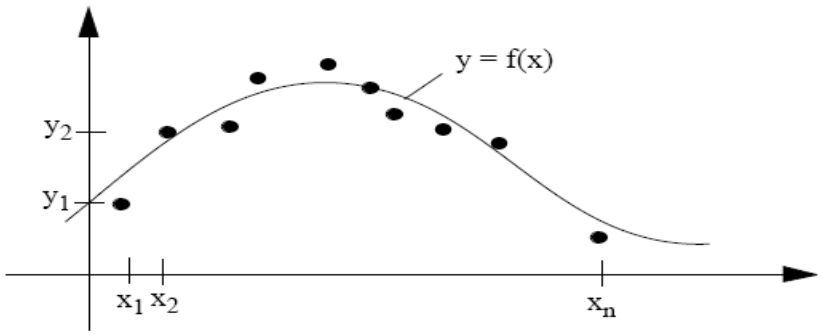
12 LEAST-SQUARES DATA FITTING

12.1 INTRODUCTION

Given a set of data values (generated from an experiment or a computer simulation), we often want to determine a function that "best" approximates the data set, using:

- A Linear Function
- A Polynomial Function
- or Other Function

Example - 2D Data Set, One Independent Variable



We can the use f(x) to:

Interpolate between x positions

$$xk < x < xk+1$$

Extrapolate to y values beyond range of data

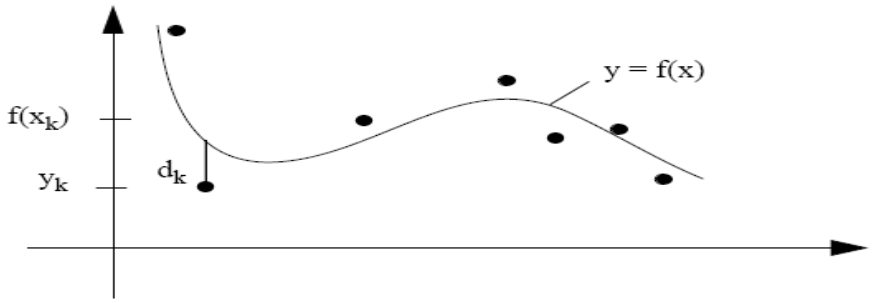
$$x < x1, x < xn$$

12.2 STEPS IN LEAST-SQUARES DATA FITTING

1. Select a function type (linear, quadratic, etc.).
2. Determine function parameters by minimizing "distance" of the function from the data points.

For a function of one independent variable, we set of expressions for the deviations:

dk = yk - f(xk)



Parameters of f(x) are then determined by minimizing the

12.3 SUM OF DEVIATIONS SQUARED:

$$E = \sum_{k=1}^{n} d_k^2 = \sum_{k=1}^{n} [y_k - f(x_k)]^2$$

The particular form of this expression depends on the structure of f(x).

12.4 LINEAR LEAST-SQUARES DATA FITTING

$f\,x = a0 + a1x$

To determine values for parameters in the linear equation, we minimize sum of deviations squared:

$$E = \sum_{k=1}^{n} [yk - f(zk)]^2$$

$$E = \sum_{k=1}^{n} [y^2k - 2ykf(zk) + f^2(xk)]$$

$$E = \sum_{k=1}^{n} [y^2k - 2yk(a0 + a1x1) + a^20 + 2a0a1xk + a^21x^2k]$$

Since E is a function of two variables ($a0$ and $a1$), we minimize by setting:

$$\delta/ao = 0$$

$$\delta/a1 = 0$$

yielding

$$E = \sum_{k=1}^{n} [-2yk + 2a0 + 2a1xk] = 0$$

$$E = \sum_{k=1}^{n} [-2ykxk + 2a0xk + 2a1x^2k] = 0$$

Rewriting the simultaneous equation is standard form, we have

$$na0 + (\sum_{k=1}^{n} xk)a1 = (\sum_{k=1}^{n} yk)$$

$$\left(\sum_{k=1}^{n} xk\right)a0 + \left(\sum_{k=1}^{n} x^2k\right)a1 = \sum_{k=1}^{n} xk\,yk$$

Using Cramer's Rule with

$$d = \begin{vmatrix} n & \sum_k xk \\ \sum_k xk & \sum_k x^2k \end{vmatrix} = n\left(\sum_{k=1}^{n} x^2k\right) - \left(\sum_{k=1}^{n} xk\right)^2$$

we can write the solution for the linear equation parameter as

$$a0 = \frac{(\sum kx^{2k})(\sum kyk) - (\sum kxk)(\sum kxkyk)}{D}$$

$$a1 = \frac{n(\sum kxkyk) - (\sum kxk)(\sum kyk)}{D}$$

Now consider a linear least-squares fit with a function of several variables:

$$f\,x1\;x2\;(\,,\,,\ldots,\,xm) = a0 + a1x1 + a2x2 + \ldots + amxm$$

Each of the n points in a data set will now be m+1 dimensional:

$(x11, x21,\ldots, xm1, y1)$

$(x12, x22,\ldots, xm2, y2)$

$(x1n, y2n,\ldots, xmn, yn)$

Also, E is a function of m+1 variables

$$E = Ea0, a1, \ldots, am$$

and the m+1 equations to solve are

$$\partial E/\partial a\,j = 0, \quad j = 0, 1, 2,\ldots, m$$

We can then use Gaussian Elimination, or some other linear-equation method, to solve this set of simultaneous linear equations. Polynomial Least-Squares Data Fitting.

In this case, we fit the data with a curve of the form

$$f\,(x) = a0 + a1x + a2x\text{^}2 + \ldots + amxm$$

assuming we have a single independent variable x and n data points:

$(x1, y1), (x2, y2), \ldots, (xn, yn)$

The solution is obtained similarly to that for a linear-function of several variables:

i.e., we form the sum of deviations squared:

$$E = \sum_{k=1}^{n} [yk - f(xk)]^2$$

and solve the m+1 equations:

$$\partial E/\partial a\,j = 0, \qquad j = 0, 1, 2,\ldots, m$$

The simultaneous equations for a polynomial fit of a data set, using a function of a single independent variable, can be written in matrix form as

$$\begin{bmatrix} n & \sum x_k & \sum x_k^2 & \cdots & \sum x_k^m \\ \sum x_k & \sum x_k^2 & \sum x_k^3 & \cdots & \sum x_k^{m+1} \\ \sum x_k^2 & \sum x_k^3 & \sum x_k^4 & \cdots & \sum x_k^{m+2} \\ \\ \sum x_k^m & \sum x_k^{m+1} & \sum x_k^{m+2} & \cdots & \sum x_k^{2m} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \\ a_n \end{bmatrix} = \begin{bmatrix} \sum y_k \\ \sum x_k y_k \\ \sum x_k^2 y_k \\ \\ \sum x_k^m y_k \end{bmatrix}$$

Or *Mcoeff A = B*

where the coefficients matrix (*Mcoeff*) is an (*m*+1) by (*m*+1) array of the sums of powers of *xk*, and B is a *m*+1 column vector of sums of powers of *xk* multiplied by *yk*. In the coefficients matrix, the power of *xk* in each position is equal to the sum of the row and column numbers, assuming rows and columns are numbered from 0 to *m*.

*12.5 **ALGORITHM***

Polynomial f(x) Least-Squares Data Fit

x, y = n-element data vectors

m = degree of polynomial

b = m+1 element vector of sums of *xk*

1 *yk* over *k*

xPowers = vector of sums of powers of *xk*, with 2m+1 elements

mCoeff = matrix of sums of powers

Input n, x, y, m

Initialize b, xPowers to 0

xPowes(0) = n

Do {p = 1, 2m}, {k = 1, n}

xPowers(p) = xPowers(p) + x(k)^p

Endofdo

Do {k=1,n}

b(0) = b(0) + y(k)

Endofdo

Do {p = 1, m}, {k = 1, n}

b(p) = b(p) + y(k) * x(k)^p

Endofdo

Do {row = 0, m}, {col = 0, m}

mCoeff(row, col) = xPowers(row + col)

Endofdo

gauss (0, m, mCoeff, b)

Routine “gauss” is a simultaneous linear-equation solver, and parameters 0, m are the lower and upper bounds on arrays mCoeff and b).

12.6 SUMMARY OF LEAST-SQUARES DATA FITTING

• Most often used to fit a set of data points with Straight Line, Polynomial of Degree < 5

Given a functional form f(x) and a set of n data points, we set up the sum of deviations squared**:**

$$E = \sum_{k=1}^{n} [yk - f(xk)]^2$$

Determine values for parameters

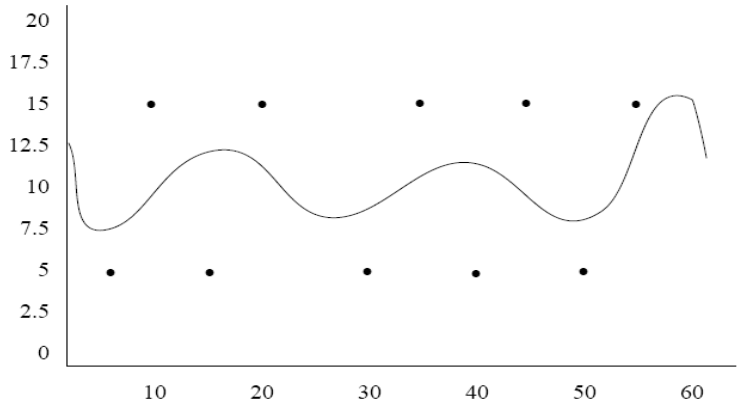
a j, *j* = 0, 1, 2, …, *m*

by solving the set of m+1 simultaneous equations:

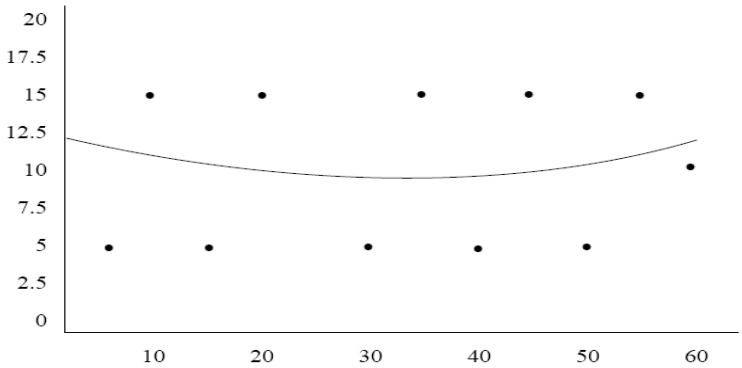
∂E/∂a j= 0, *j* = 0, 1, 2,…, *m*

And, we can generalize the above procedures for functions of several variables.

• When fitting a set of data points, we usually want a "smooth" function**.** With higher-order polynomials, can get extraneous wiggles with curve trying to connect all data points:



Typically have some error associated with each data point, so what we want is something like this:



• Also, when degree of polynomial is large (> 6, 7), system of equations can become ill-conditioned (large round-off errors).

• In some cases, transcendental functions (trig, log plots, exponential, etc.) are useful for data-fitting.

• Can get measure of how well a particular function fits the data by applying "Goodness-of Fit" tests.

ORDINARY DIFFERENTIAL EQUATION

13 EULER'S METHOD

13.1 INTRODUCTION

In order to develop a technique for solving first order initial value problems numerically, we should first agree upon some notation. Euler's method is considered to be one of the oldest and simplest methods to find the numerical solution of ordinary differential equation or the initial value problems. Here, short and simple algorithm and flowchart for Euler's method has been presented, which can be used to write program for the method in any high level programming language.

Through Euler's method, you can find a clear expression for y in terms of a finite number of elementary functions represented with x. The initial values of y and x are known, and for these an ordinary differential equation is considered.

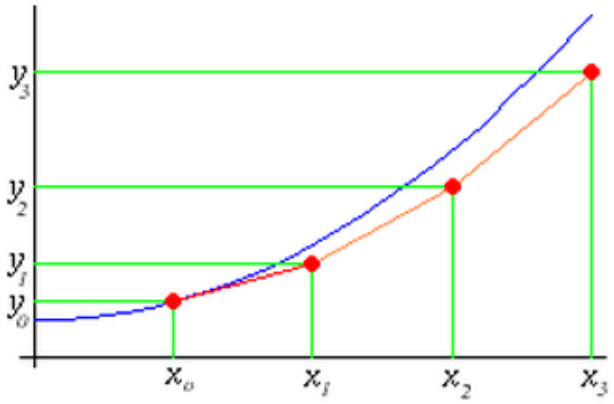
Now, lets look at the mathematics and algorithm behind the Euler's method. A sequence of short lines is approximated to find the curve of solution; this means considering tangent line in each interval.

We will assume that the problem in question can be algebraically manipulated into the form:

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

$$y(xo) = yo$$

Our goal is to find a numerical solution, i.e. we must find a set of points which lie along the initial value problem's solution. If we look at this problem, as stated above, we should realize that we actually already know one point on the solution, namely the one defined by the initial condition, (xo, yo).



So the list of points in our approximate numerical solution now has four members:

- (xo, yo): an exact value, known to lie on the solution curve.
- (x1, y1): an approximate value, known to lie on the solution curve's tangent line through (xo, yo).
- (x2, y2): an approximate value, known to lie on the solution curve's pseudo-tangent line through (x1, y1).
- (x3, y3): an approximate value, known to lie on the solution curve's pseudo-tangent line through (x2, y2).

We're solving the initial value problem:

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

$$y(xo) = yo$$

As we just saw in the graphical description of the method, the basic idea is to use a known point as a "starter," and then use the tangent line through this known point to jump to a new point. Rather than focus on a particular point in the sequence of points we're going to generate, let's be generic. Let's use the names:

- (xn, yn) for the known point
- (xn+1, yn+1) for the new point

The formula relating xn and xn+1 is obvious:

$$xn + 1 \; = \; xn \; + \; h$$

Also, we know from basic algebra that $slope \; = \frac{rise}{run}$, so applying this idea to the triangle in our picture, the formula becomes:

$$f(xn, yn) = \frac{\Delta y}{h}$$

which can be rearranged to solve for Δy giving us:

$$\Delta y \; = \; h \, f(xn, yn)$$

But, we're really after a formula for yn+1. Looking at the picture, it's obvious that:

$$yn + 1 \; = \; yn \; + \; \Delta y$$

And, replacing Δy by our new formula, this becomes:

$$yn + 1 \; = \; yn \; + \; h \, f(xn, yn)$$

In order to use Euler's Method to generate a numerical solution to an initial value problem of the form:

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

$$y(xo) \; = \; yo$$

we decide upon what interval, starting at the initial condition, we desire to find the solution. We chop this interval into small subdivisions of length h. Then, using the initial condition as our starting point, we generate the rest of the solution by using the iterative formulas:

$$xn + 1 \; = \; xn \; + \; h$$

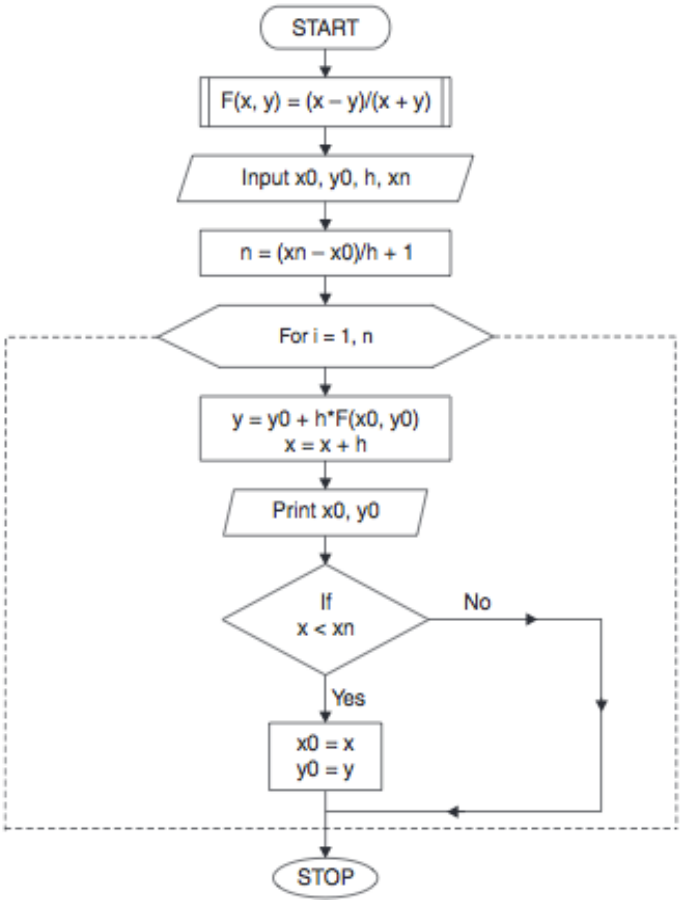
$$yn + 1 \; = \; yn \; + \; h \, f(xn, yn)$$

to find the coordinates of the points in our numerical solution. We terminate this process when we have reached the right end of the desired interval.

13.2 EULER'S METHOD ALGORITHM:

1. Start
2. Define function
3. Get the values of x0, y0, h and xn
   *Here x0 and y0 are the initial conditions
   h is the interval
   xn is the required value
4. n = (xn – x0)/h + 1
5. Start loop from i=1 to n
6. y = y0 + h*f(x0,y0)
   x = x + h
7. Print values of y0 and x0
8. Check if x < xn
   If yes, assign x0 = x and y0 = y
   If no, goto 9.
9. End loop i
10. Stop

13.3 EULER'S METHOD FLOWCHART:



Example

$$y' = x + 2y$$

$$y(0) = 0$$

numerically, finding a value for the solution at x = 1, and using steps of size h = 0.25.

Applying the Method

Clearly, the description of the problem implies that the interval we'll be finding a solution on is [0,1]. The differential equation given tells us the formula for f(x, y) required by the Euler Method, namely:

$$f(x, y) = x + 2y$$

and the initial condition tells us the values of the coordinates of our starting point:

$$xo = 0$$

$$yo = 0$$

We now use the Euler method formulas to generate values for x1 and y1.

Iteration 0

$x1 = xo + h$

$x1 = 0 + 0.25$

$x1 = 0.25$

and

$y1 = yo + h\, f(xo, yo)$

$y1 = yo + h\,(xo + 2yo)$

$y1 = 0 + 0.25\,(0 + 2 * 0)$

$y1 = 0$

Therefore,

$x1 = 0.25$

$y1 = 0$

Iteration 1

$x2 = x1 + h$

$x2 = 0.25 + 0.25$

$x2 = 0.5$

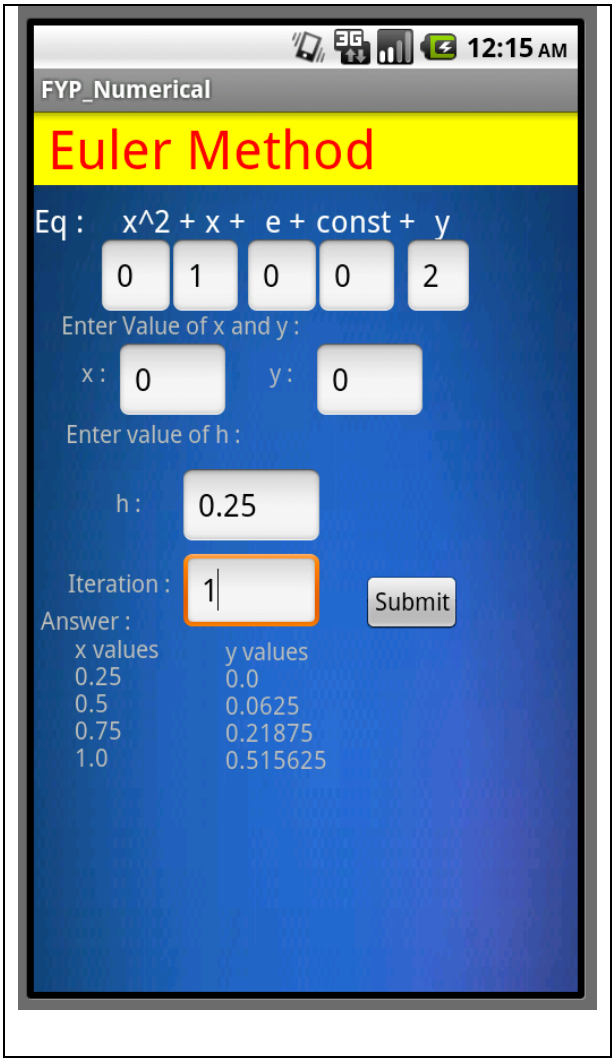
and

$y2 = y1 + h\,f(x1, y1)$

$y2 = y1 + h\,(x1 + 2y1)$

$y2 = 0 + 0.25\,(0.25 + 2 * 0)$

$y2 = 0.0625$



Iteration 2

$x3 = x2 + h$

$x3 = 0.5 + 0.25$

$x3 = 0.75$

and

$y3 = y2 + h\,f(x2, y2)$

$y3 = y2 + h\,(x2 + 2y2)$

$y3 = 0.0625 + 0.25\,(0.5 + 2 * 0.0625)$

$y3 = 0.21875$

Therefore,

$x3 = 0.75$

$y3 = 0.21875$

Iteration 3

$x4 = x3 + h$

$x4 = 0.75 + 0.25$

$x4 = 1$

and

$y4 = y3 + h\, f(x3, y3)$

$y4 = y3 + h\,(x3 + 2y3)$

$y4 = 0.21875 + 0.25\,(0.75 + 2 * 0.21875)$

$y4 = 0.515625$

Therefore,

$x4 = 1$

$y4 = 0.515625$

13.4 TYPES OF ERRORS

Local truncation error: Error due to the use of truncated Taylor series to compute x(t+h) in one step.

Global Truncation error: Accumulated truncation over many steps.

Round off error: Error due to finite number of bits used in representation of numbers. This error could be accumulated and magnified in succeeding steps.

13.5 ADVANTAGES:

- Simplest of all linear multi-step method to obtain the approximated solution of the specified initial value problem.
- It has the one-step techniques, and it can be easily programmed.
- The derivation of Euler's method can be revealed by constructing Taylor series.
- Approximated solution can be acquired at each step before progressing to the next step.

- The error analysis, which involve local and global truncation error, and remainder term can be obtained smoothly.

13.6 DISADVANTAGES:

- It is less accurate and numerically unstable.
- Approximation error is proportional to the step size h. Hence, good approximation is obtained with a very small h. This requires a large number of time discretization leading to a larger computation time.

Usually applicable to explicit differential equations.

13.7 CONVERGENCE AND ACCURACY OF EULER'S METHOD

It is easy to see that Euler's method converges for the special case of the equation $y' = \lambda y$ with solution$y(T) = yoe^{\lambda T}$. For this example,

$$yn = yo(1 + h\lambda)^n = \text{yo}(1 + \frac{\text{T}\lambda}{\text{n}})^{\text{n}}$$

Recalling that $\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n = e$, we see that$\lim_{n\to\infty} yn = yoe^{\lambda T}$

Error estimates show

The global error at T is O(h) (first order accuracy).

The error grows exponentially in time.

The error increases with M. This suggests that one should take smaller steps where the solution is changing more rapidly. We will return to this below.

The error analysis above ignores round-off error. If one assumes that a fixed error is added at each time step, then the error estimate of is modified to $O(h) + O(\in mach\ h^{-1})$. That is, taking more steps reduces the discretization error, but increases the round-off error. Therefore, there is a point of diminishing returns where the total error increases as h decreases. Better results require not more effort, but more efficiency. The key is to

take more terms of the Taylor series and reduce the discretization error to $O(h^p)$, with $p > 1$.

13.8 CONCLUSION

The Euler method is a first-order method, which means that the local error (error per step) is proportional to the square of the step size, and the global error (error at a given time) is proportional to the step size. The Euler method often serves as the basis to construct more complex methods.

14 RUNGE KUTTA METHOD

14.1 INTRODUCTION

There are two main reasons why Euler's method is not generally used in scientific computing. Firstly, the truncation error per step associated with this method is far larger than those associated with other, more advanced, methods (for a given value of h). Secondly, Euler's method is too prone to numerical instabilities.

The methods most commonly employed by scientists to integrate o.d.e.s were first developed by the German mathematicians C.D.T. Runge and M.W. Kutta in the latter half of the nineteenth century.The basic reasoning behind so-called Runge-Kutta methods is outlined in the following.

The main reason that Euler's method has such a large truncation error per step is that in evolving the solution from Xn to Xn+1the method only evaluates derivatives at the beginning of the interval: i.e., at x_n. The method is, therefore, very asymmetric with respect to the beginning and the end of the interval. We can construct a more symmetric integration method by making an Euler-like trial step to the midpoint of the interval, and then using the values of both x and y at the midpoint to make the real step across the interval.

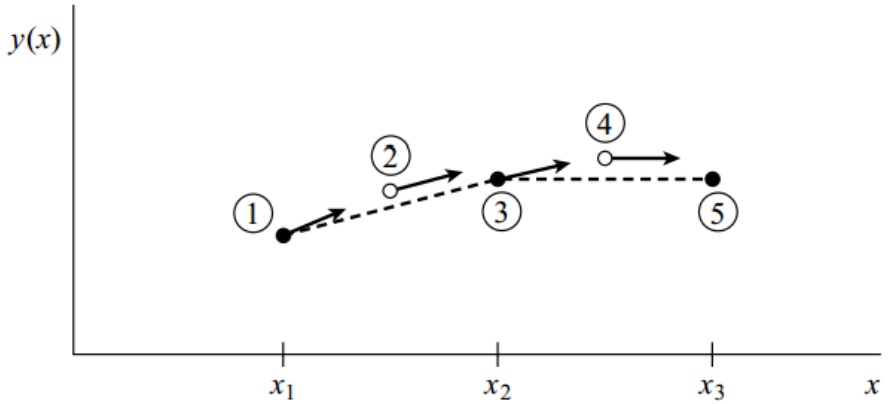
Consider, however, the use of a step like (16.1.1) to take a “trial” step to the midpoint of the interval. Then use the value of both x and y at that midpoint to compute the “real” step across the whole interval. Figure 16.1.2 illustrates the idea. In equations,

$$k1 = hf(xn, yn)$$

$$k2 = hf\left(xn + \frac{1}{2}h, yn + \frac{1}{2}k1\right)$$

$$yn + 1 = yn + k2 + O(h^3)$$

As indicated in the error term, this symmetrization cancels out the first-order error term, making the method second order. [A method is conventionally called nth order if its error term is $O(h^{n+1})$. In fact, (16.1.2) is called the second-order Runge-Kutta or midpoint method.



The most popular of the fixed step size methods is the fourth-order Runge-Kutta method. This is the default method in many integrators. The algorithm involves four evaluations of the right-hand sides and gives accracy that is $O(h^4)$.

$$k1 = hf(xn, yn)$$

$$k2 = hf\left(xn + \frac{h}{2}, yn + \frac{k1}{2}\right)$$

$$k3 = hf\left(xn + \frac{h}{2}, yn + \frac{k2}{2}\right)$$

$$k4 = hf(xn + h, yn + k3)$$

$$yn + 1 = yn + \frac{k1}{6} + \frac{k2}{3} + \frac{k3}{3} + \frac{k4}{6} + O\,(h^5)$$

While it seems that higher order rules require more work and more evaluations of the right-hand sides, for the same accuracy requirements, they are much more efficient. For example to get an error of 0.0001 requires h=0.0001 for Eulers method and h=0.1 for

RK4. Thus to advance one time step requires 10000 evaluations of the right-hand side using Euler and 10 setps of 4 evaluations or 40 evaluations for Runge-Kutta.

14.2 THE RUNGE-KUTTA ALGORITHM

Easily extended to a set of first order differential equations. You wind up with essentially the same formulas shown above, but all the variables (except for time) are vectors.

To give an idea of how it works, here is an example where we expand the vector notation. That is, instead of using the highly compact vector notation, we write out all the components of the vector.

Suppose there are only two variables, x, y and two differential equations

$$x' = f(t,x,y)$$

$$y' = g(t,x,y)$$

Again we let x_n be the value of x at time t_n, and similarly for y_n. Then the formulas for the Runge-Kutta algorithm are

$$x_{n+1} = x_n + (h/6)(k_1 + 2k_2 + 2k_3 + k_4)$$

$$y_{n+1} = y_n + (h/6)(j_1 + 2j_2 + 2j_3 + j_4)$$

where

$$k_1 = f(t_n, x_n, y_n)$$

$$j_1 = g(t_n, x_n, y_n)$$

$$k_2 = f(t_n + h/2, x_n + (h/2)k_1, y_n + (h/2)j_1)$$

$$j_2 = g(t_n + h/2, x_n + (h/2)k_1, y_n + (h/2)j_1)$$

$$k_3 = f(t_n + h/2, x_n + (h/2)k_2, y_n + (h/2)j_2)$$

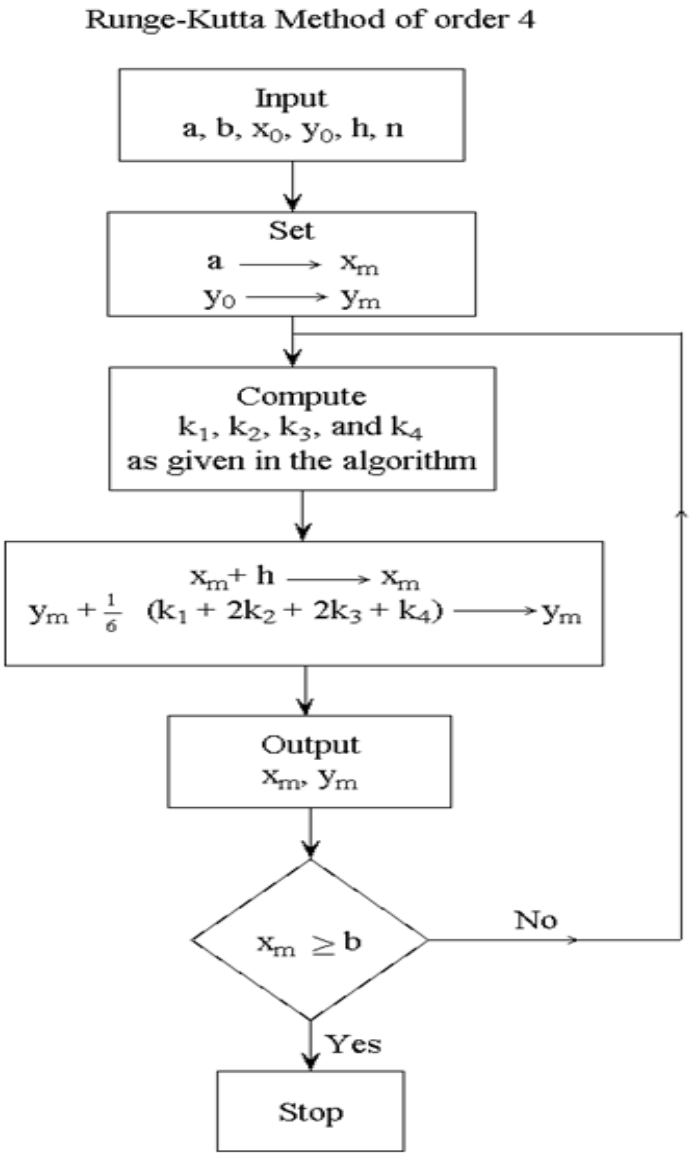
$$j_3 = g(t_n + h/2, x_n + (h/2)k_2, y_n + (h/2)j_2)$$

$$k_4 = f(t_n + h, x_n + h\,k_3, y_n + h\,j_3)$$

$$j_4 = g(t_n + h, x_n + h\,k_3, y_n + h\,j_3)$$

Given starting values x_0,y_0 we can plug them into the formula to find x_1,y_1. Then we can plug in x_1,y_1 to find x_2,y_2 and so on.

14.3 FLOWCHART



Example

The exact solution for this problem is $= t^2 + 2t + 1 - \frac{1}{2}e^t$, and we are interested in the value of y for $0 \leq t \leq 2$.

h = 0.5. From t = 0 to t = 2 with step size h = 0.5, it takes 4 steps: t0 = 0, t1 = 0.5, t2 = 1, t3 = 1.5, t4 = 2.

Step 0 t0 = 0, w0 = 0.5.

Step 1 t1 = 0.5

$$k1 = hf(t0, w0) = 0.5f(0, 0.5) = 0.75$$

$$k2 = hf\left(t0 + \frac{h}{2}, w0 + \frac{k1}{2}\right) = 0.5f(0.25, 0.875) = 0.90625$$

$$K3 = hf\left(t0 + \frac{h}{2}, w0 + \frac{k2}{2}\right) = 0.5f(0.25, 0.953125) = 0.9453125$$

$$K4 = hf(t0 + h, w0 + K3) = 0.5f(0.5, 1.4453125) = 1.09765625$$

$$w1 = w0 + \frac{k1 + 2k2 + 2k3 + k4}{6} = 1.425130208333333$$

Step 2 t2 = 1

$$k1 = hf(t1, w1) = 0.5f(0.5, 1.425130208333333) = 1.087565104166667$$

$$k2 = hf\left(t1 + \frac{h}{2}, w1 + \frac{k1}{2}\right) = 0.5f(0.75, 1.968912760416667)$$
$$= 1.203206380208333$$

$$K3 = hf\left(t1 + \frac{h}{2}, w1 + \frac{k2}{2}\right) = 0.5f(0.75, 2.0267333984375)$$
$$= 1.23211669921875$$

$$K4 = hf(t1 + h, w1 + K3) = 0.5f(1, 2.657246907552083)$$
$$= 1.328623453776042$$

$$w2 = w1 + \frac{k1 + 2k2 + 2k3 + k4}{6} = 2.639602661132812$$

Step 3 t3 = 1.5

$$k1 = hf(t2, w2) = 0.5f(1, 2.639602661132812) = 1.319801330566406$$

$$k2 = hf\left(t2 + \frac{h}{2}, w2 + \frac{k1}{2}\right) = 0.5f(1.25, 3.299503326416016)$$
$$= 1.368501663208008$$

$$K3 = hf\left(t2 + \frac{h}{2}, w2 + \frac{k2}{2}\right) = 0.5f(1.25, 3.323853492736816)$$
$$= 1.380676746368408$$

$$K4 = hf(t2 + h, w2 + K3) = 0.5f(1.5, 4.020279407501221)$$
$$= 1.385139703750610$$

$$w3 = w2 + \frac{k1 + 2k2 + 2k3 + k4}{6} = 4.006818970044454$$

Step 4 t4 = 2

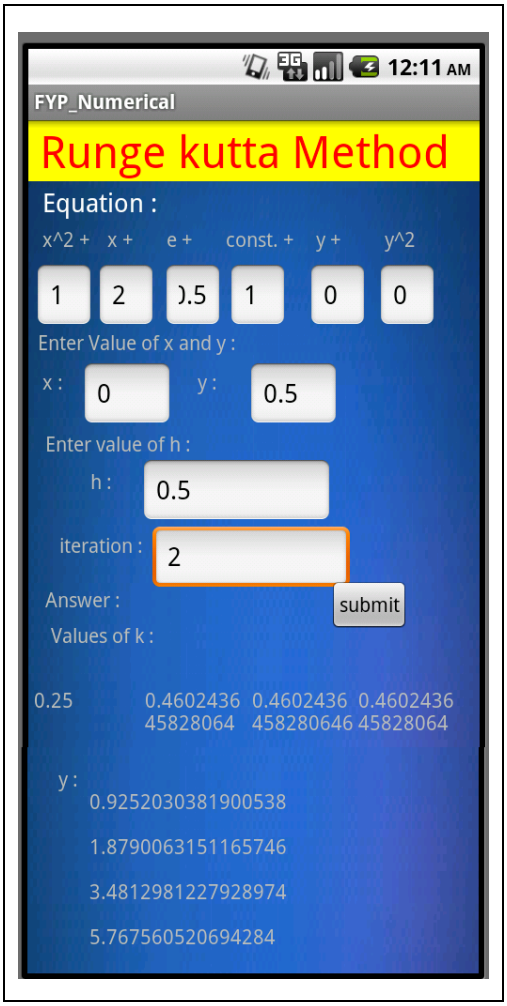
$$k1 = hf(t3, w3) = 0.5f(1.5, 4.006818970044454) = 1.378409485022227$$

$$k2 = hf\left(t3 + \frac{h}{2}, w3 + \frac{k1}{2}\right) = 0.5f(1.75, 4.696023712555567)$$
$$= 1.316761856277783$$

$$K3 = hf\left(t3 + \frac{h}{2}, w3 + \frac{k2}{2}\right) = 0.5f(1.75, 4.665199898183346)$$
$$= 1.301349949091673$$

$$K4 = hf(t3 + h, w3 + K3) = 0.5f(2, 5.308168919136127)$$
$$= 1.154084459568063$$

$$w4 = w3 + \frac{k1 + 2k2 + 2k3 + k4}{6} = 5.301605229265987$$



14.4 ADVANTAGES:

- The idea of families of Runge Kutta method is too complicated, but higher order provides much better approximated solutions than Euler method.
- The most popular Runge-Kutta method is the method of order four.
- It is good choice to get more accurate and more efficient solutions for solving the specified ordinary differential equations.
- The approximated solution converge faster to exact solution and the order of RK4 is 4 and the truncation error is $O(h^5)$.

14.5 DISADVANTAGES:

- Method is re-evaluating the function f at each time to obtain the predictable solution.
- It requires four evaluation per step.
- The computation of function may take long time.
- The derivation of Runge-Kutta method is obtained from Taylor series, but it is tedious to calculate higher derivative.
- To avoid this, the function f is evaluated at more points.

14.6 CONCLUSION

- Runge Kutta methods generate an accurate solution without the need to calculate high order derivatives.
- Second order RK have local truncation error of order $O(h^3)$.
- Fourth order RK have local truncation error of order $O(h^5)$.
- N function evaluations are needed in the Nth order RK method.

15 REFERENCES.

- *Burden and Faires, Numerical Analysis, Ninth Edition, Brooks/Cole Publishing Company, 2011.*
- *Numerical Method for Engineering, Chapra*
- *James F. Epperson, 2002, An Introduction to Numerical Methods and Analysis, John Wiley N.Y.,*
- *K. Atkinson, 1989, An Introduction to Numerical Analysis, John Wiley & Sons, Inc., Second Edition.*
- *Buchanan and Turner, 1992, Numerical Methods and Analysis, McGraw-Hill, N.Y*
- *Isaacson and Keller, 1994, Analysis of Numerical Methods, John Wiley N.Y..*
- *Beginning Android 4 Application Development Wrox-2012*
- *Prentice Hall Android for Programmers, An App-Driven Approach 2012*
- *http://en.wikipedia.org/wiki/Numerical_analysis*
- *https://mat.iitm.ac.in/home/sryedida/public_html/caimna/index1.html*
- *http://math.stanford.edu/~dlevy/books/numerical.pdf*
- *http://www.math.umn.edu/~olver/num.html*
- *http://www.math.ust.hk/~machas/numerical-methods.pdf*