**Unit – 1**

**Computer Based Numerical and Statistical Techniques**

**1. ERRORS AND THEIR ANALYSIS**

**Sources of Errors**

Following are the broad sources of errors in numerical analysis:

**(1)Input errors.**

* The input information is rarely exact since it comes from the experiments and **any experiment can give results of only limited accuracy.**
* Moreover, the quantity used can be represented in a **computer for only a limited number of digits**.

**(2)Algorithmic errors.**

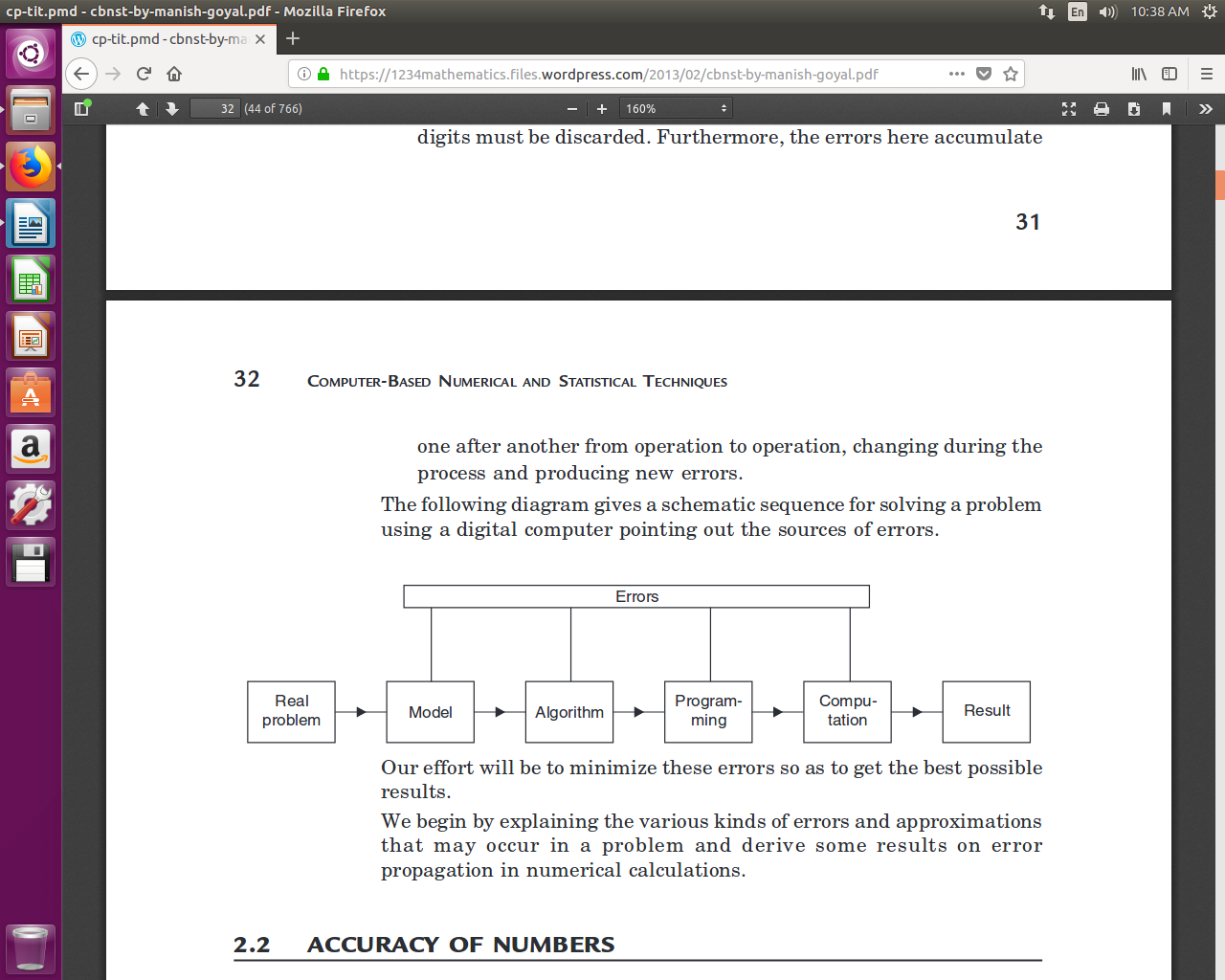
* If direct algorithms based on a **finite sequence of operations are used**, errors due to limited steps don’t amplify the existing errors, but if infinite algorithms are used, exact results are expected only after an infinite number of steps.
* As this cannot be done in practice, **the algorithm has to be stopped after a finite number of steps** and the results are not exact.

**(3)Computational errors.**

* Even when elementary operations such as **multiplication and division** are used, the **number of digits increases greatly** so that the results cannot be held fully in a register available in a given computer.
* In such cases, **a certain number of digits must be discarded**. Furthermore, the errors here accumulate one after another from operation to operation, changing during the process and producing new errors.

The following diagram gives a schematic sequence for solving a problem

using a digital computer pointing out the sources of errors

**Our effort will be to minimize these errors so as to get the best possible results.**

**2. Arithmetic Operations with Normalized Floating Point Number**

**2.1 Adsition and Subtraction**

* If two numbers represented in normalized floating point notation are to be added, the exponents of the two numbers must be made equal and the Mantissa shifted appropriately.
* The operation of subtraction is nothing but the addition of a negative number. Thus the principles are the same.

**Example 1.**

**Add the following floating point numbers:**

(i) .4546 E 5 and .5433 E 5

(ii) .4546 E 5 and .5433 E 7

**Sol.(i)** Here the exponents are equal ∴Mantissas are added

∴Sum = .9979 E 5

**(ii)** Here exponents are not equal. The operand with the larger exponent is

kept as it is

.5433 E 7

+ .0045 E 7

-------------------- | .4546 E 5 = .0045 E 7

.5478 E 7

**Example 2. Subtract the following floating point numbers:**

(i) .9432 E – 4 from .5452 E – 3

(ii) .5424 E 3 from .5452 E 3

Sol.(i)

.5452 E – 3

– .0943 E – 3

.4509 E – 3

**(ii)**

.5452 E 3

– .5424 E 3

.0028 E 3

In a normalized floating point, the mantissa is ≥.1

Hence, the result is .28 E 1

**2.2 Multiplication**

Two numbers are multiplied in the normalized floating point mode by multiplying the mantissas and adding the exponents. After the multiplicationof the mantissas, the resulting mantissa is normalized as in an addition orsubtraction operation, and the exponent is appropriately adjusted.

Example 1.

Multiply the following floating point numbers:

(i) .5543 E 12 and .4111 E – 15 (ii) .1111 E 10 and .1234 E 15

**Sol. (i)** .5543 E 12 × .4111 E – 15 = .2278 E – 3

**(ii)** .1111 E 10 × .1234 E 15 = .01370 E 25= .1370 E 24

**3.Division**

In division, the mantissa of the numerator is divided by that of the denominator.The denominator exponent is subtracted from the numerator exponent. The quotient mantissa is normalized to make the most significant digit non-zero and the exponent is appropriately adjusted. The mantissa of the result ischopped down to 4 digits.

**Example 1.**

**Perform the following operations:**

(i).9998 E 1 ÷.1000 E – 99 (ii).9998 E – 5 ÷.1000 E 98

(iii).1000 E 5 ÷.9999 E 3.

**Sol.(i)** .9998 E 1 ÷.1000 E – 99 = .9998 E 101

Hence the result overflows.

**(ii)** .9998 E – 5 ÷.1000 E 98 = .9998 E – 104

Hence the result underflows.

**(iii)** .1000 E 5 ÷ .9999 E 3 = .1000 E 2.

**3. Bisection Method**

|  |
| --- |
| * **Bisection Method** = a **numerical method** in **Mathematics** to find a **root** of a given ***function*** |

* ***Root* of a function:**

|  |  |
| --- | --- |
| * **Root of a function *f(x)*** = a **value *a*** such that:  |  | | --- | | * ***f(a) = 0*** | |

**Example:**

|  |
| --- |
| **Function: f(x) = x2 - 4**  **Roots: x = -2, x = 2**  **Because:**  **f(-2) = (-2)2 - 4 = 4 - 4 = 0**  **f(2) = (2)2 - 4 = 4 - 4 = 0** |

**A Mathematical Property**

* **Well-known Mathematical Property:**

|  |  |  |
| --- | --- | --- |
| * If a **function *f(x)*** is **continuous** on the **interval [*a*..*b*]** and **sign of *f(a)* ≠ sign of *f(b)***, then:  |  | | --- | | * There is a value ***c* ∈ [*a*..*b*]** such that: ***f(c) = 0***   I.e., there is a **root *c*** in the **interval [*a*..*b*]** |   **Example:**   |  | | --- | |  | |

**The *Bisection* Method**

* The **Bisection Method** is a ***successive* approximation method** that **narrows down** an **interval** that contains a **root of the function *f(x)***
* The **Bisection Method** is ***given*** an **initial interval [*a*..*b*]** that **contains a root**

(We can use the **property** **sign of *f(a)* ≠ sign of *f(b)*** to find such an **initial interval**)

* The **Bisection Method** will ***cut the interval*** into **2 halves** and check **which half interval** contains a **root of the function**
* The **Bisection Method** will keep ***cut the interval*** in halves until the **resulting interval** is **extremely small**

The **root** is then ***approximately equal*** to ***any value*** in the **final (very small) interval**.

**Advantage**

* It is very easy and simple
* It never fails as compared to netwon Raphson or other method

**Disadvantage**

* Biggest disadvantage is the slow convergence rate.
* There's also the inability to detect multiple roots.
* It takes so many iteration

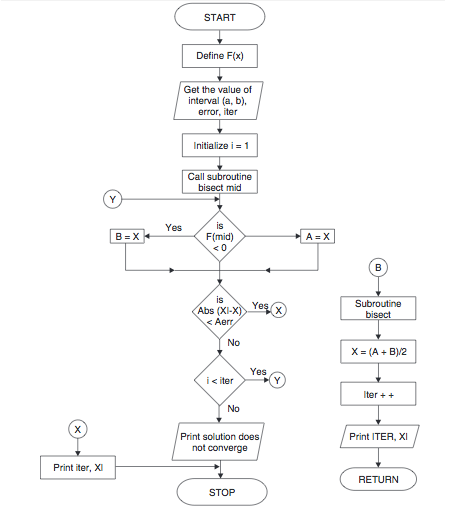
**Example: find the root of *f(x) = x2 − 5* between [0, 4]**

|  |
| --- |
| **There is a root between [0,4] because::**  **f(0) = 02 − 5 = −5**  **f(4) = 42 − 5 = 11**  **Start:**  **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 m**  **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**  **And so on....** |

## **Bisection Method Algorithm:**

1. Start
2. Read x1, x2, e  
   \*Here x1 and x2 are initial guesses  
   e is the absolute error i.e. the desired degree of accuracy\*
3. Compute: f1 = f(x1) and f2 = f(x2)
4. If (f1\*f2) > 0, then display initial guesses are wrong and goto (11).  
   Otherwise continue.
5. x = (x1 + x2)/2
6. If ( [ (x1 – x2)/x ] < e ), then display x and goto (11).  
   \* Here [ ] refers to the modulus sign. \*
7. Else, f = f(x)
8. If ((f\*f1) > 0), then x1 = x and f1 = f.
9. Else, x2 = x and f2 = f.
10. Goto (5).  
    \*Now the loop continues with new values.\*
11. Stop

## **Bisection Method Flowchart:**

****

**Programs:**

**#include<stdio.h>**

//function used is x^3-2x^2+3

double func(double x)

{

    return x\*x\*x - 2\*x\*x + 3;

}

double e=0.01;

double c;

void bisection(double a,double b)

{

    if(func(a) \* func(b) >= 0)

    {

        printf("Incorrect a and b");

        return;

    }

    c = a;

    while ((b-a) >= e)

    {

        c = (a+b)/2;

        if (func(c) == 0.0){

            printf("Root = %lf\n",c);

            break;

        }

        else if (func(c)\*func(a) < 0){

                printf("Root = %lf\n",c);

                b = c;

        }

        else{

                printf("Root = %lf\n",c);

                a = c;

        }

    }

}

int main()

{

    double a,b;

    a=-10;

    b=20;

    printf("The function used is x^3-2x^2+3\n");

    printf("a = %lf\n",a);

    printf("b = %lf\n",b);

    bisection(a,b);

    printf("\n");

    printf("Accurate Root calculated is = %lf\n",c);

    return 0;

}

**4. Interation Method**

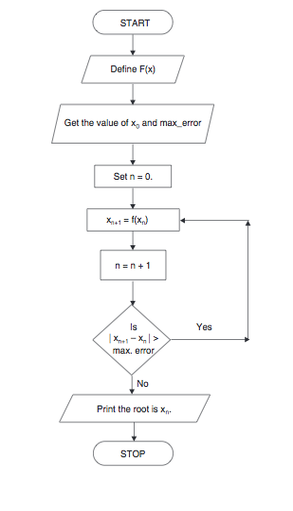
* **Fixed point iteration method** is commonly known as the iteration method.
* It is one of the most common methods used to **find the real roots** of a function.
* Fixed point iteration method is more particularly useful for locating the real roots of an equation given in the form of an infinite series.
* So, this method can be used for finding the solution of arithmetic series, geometric series, [Taylor’s series](http://en.wikipedia.org/wiki/Taylor's_series) and other forms of infinite series.
* This method is linearly convergent with somewhat slower rate of convergence, similar to the [bisection method](https://www.codewithc.com/c-program-for-bisection-method/).
* It is based on modification approach to find the fixed point.
* It is commonly referred to as simple enclosure method or open bracket method.
* Like other methods to find the root of a function, the programming effort for Iteration Method is easy, short and simple.
* Just like the [Newton-Raphson method](https://www.codewithc.com/c-program-for-newton-raphson-method/), it requires only one initial guess, and the equation is solved by the assumed approximation.
* Iterations and modifications are successively continued with the updated approximations of the guess.
* Iterative method gives good accuracy overall just like the other methods.

Features of Fixed Point Iteration Method:

* Type – open bracket
* No. of initial guesses – 1
* Convergence – linear
* Rate of convergence – fast
* Accuracy – good
* Programming effort – easy
* Approach – modification

## **Iteration Method Algorithm:**

1. Start
2. Read values of x0 and e.  
   \*Here x0 is the initial approximation  
   e is the absolute error or the desired degree of accuracy, also the stopping criteria\*
3. Calculate x1 = g(x0)
4. If [x1 – x0] <= e, goto step 6.  
   \*Here [ ] refers to the modulus sign\*
5. Else, assign x0 = x1 and goto step 3.
6. Display x1 as the root.
7. Stop

**Iteration Method Flowchart:**

Below is a source code in C program for iteration method to find the root of (cosx+2)/3. The desired degree of accuracy in the program can be achieved by continuing the iteration i.e. by increasing the maximum number of iterations.

f(x) = (cos(x) +2)/3

## Source Code for Iteration Method in C:

|  |  |
| --- | --- |
|  | #include<stdio.h>  #include<math.h>  float iteration(float);  main()  {      float a[100],b[100],c=100.0;      int j=0;        printf("\nEnter initial guess:\n");      scanf("%f",&a[0]);      printf("\n\n The values of iterations are:\n\n ");      while(c>0.00001)      {          a[j+1]=raj(a[j]);          c=a[j+1]-a[j];          c=fabs(c);          printf("%d\t%f\n",j,a[j]);          j++;        }      printf("\n The root of the given function is %f",a[j]);  }  float iteration(float x)  {      float y;      y=(cos(x)+2)/3;      return y;  } |

# **5. Regula Falsi Method**

* Regula Falsi method, also known as the false position method, is the oldest approach to find the real root of a function.
* It is a closed bracket method and closely resembles the [bisection method](https://www.codewithc.com/c-program-for-bisection-method/).
* Regula falsi method requires two initial guesses of opposite nature.
* Interpolation is done to find the new values for successive iterations, but in this method one interval always remains constant.
* The programming effort for Regula Falsi or False Position Method language is simple and easy.
* The convergence is of first order and it is guaranteed.
* In [manual approach](http://nptel.ac.in/courses/122104019/numerical-analysis/Rathish-kumar/ratish-1/f3node3.html), the method of false position may be slow, but it is found superior to the bisection method.

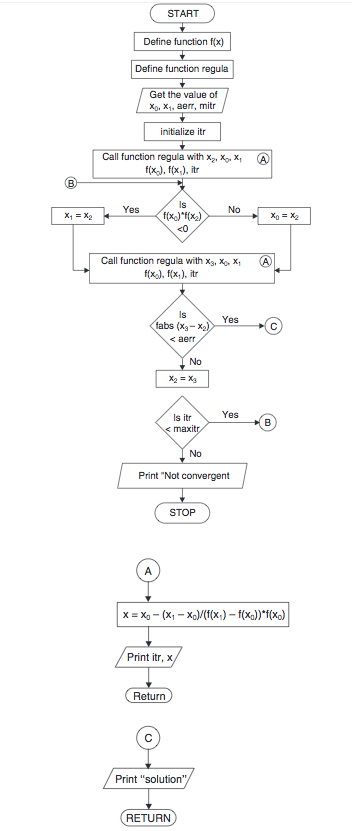
**Features of Regula Falsi Method:**

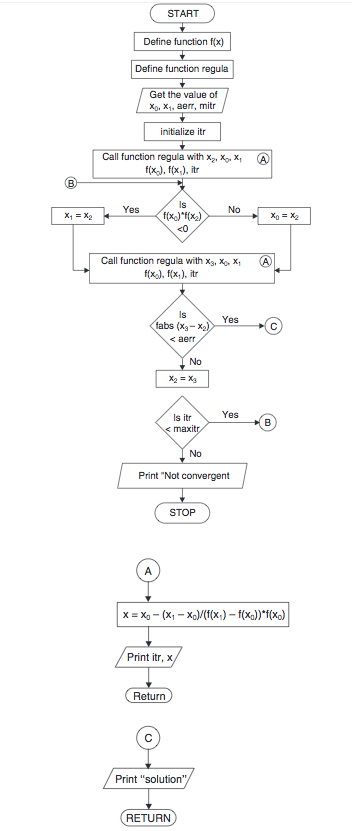
* Type – closed bracket
* No. of initial guesses – 2
* Convergence – linear
* Rate of convergence – slow
* Accuracy – good
* Programming effort – easy
* Approach – interpolation

## **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

## Regula Falsi Method Flowchart:





f(x) = cos(x) – x\*e^x

## Source Code for Regula Falsi Method in C:

|  |  |
| --- | --- |
|  | #include<stdio.h>  #include<math.h>  float f(float x)  {      return cos(x) - x\*exp(x);  }  void regula (float \*x, float x0, float x1, float fx0, float fx1, int \*itr)  {      \*x = x0 - ((x1 - x0) / (fx1 - fx0))\*fx0;      ++(\*itr);      printf("Iteration no. %3d X = %7.5f \n", \*itr, \*x);  }  void main ()  {      int itr = 0, maxmitr;      float x0,x1,x2,x3,allerr;      printf("\nEnter the values of x0, x1, allowed error and maximum iterations:\n");      scanf("%f %f %f %d", &x0, &x1, &allerr, &maxmitr);      regula (&x2, x0, x1, f(x0), f(x1), &itr);      do      {          if (f(x0)\*f(x2) < 0)              x1=x2;          else              x0=x2;          regula (&x3, x0, x1, f(x0), f(x1), &itr);          if (fabs(x3-x2) < allerr)          {              printf("After %d iterations, root = %6.4f\n", itr, x3);              return 0;          }          x2=x3;      }      while (itr<maxmitr);      printf("Solution does not converge or iterations not sufficient:\n");      return 1;  } |

# **6. Newton Raphson Method**

* Newton-Raphson method, also known as the Newton’s Method, is the simplest and fastest approach to find the root of a function.
* It is an open bracket method and requires only one initial guess.
* Newton Raphson method presented here is a programming approach which can be used to find the real roots of not only a nonlinear function, but also those of algebraic and [transcendental equation](http://en.wikipedia.org/wiki/Transcendental_equation)s.
* Newton’s method is often used to improve the result or value of the root obtained from other methods.
* This method is more useful when the first derivative of f(x) is a large value.
* The programming effort for Newton Raphson Method is relatively simple and fast.
* The convergence is the fastest of all the root finding methods discussed in [Numerical Methods Tutorial](https://www.codewithc.com/numerical-methods-tutorial/) section – the [bisection method](https://www.codewithc.com/c-program-for-bisection-method/), the [secant method](https://www.codewithc.com/c-program-for-secant-method/) and the [regula-falsi method](https://www.codewithc.com/c-program-for-regula-falsi-method/).

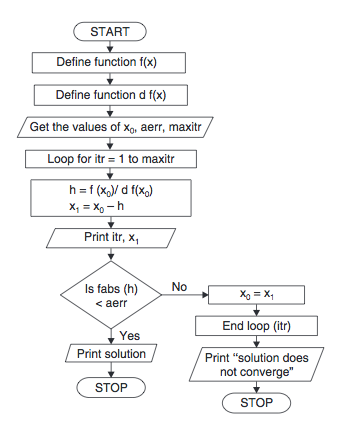
**Features of Newton Raphson Method:**

* Type – open bracket
* No. of initial guesses – 1
* Convergence – quadratic
* Rate of convergence – faster
* Accuracy – good
* Programming effort – easy
* Approach – Taylor’s series

## **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

## **Newton Raphson Method Flowchart:**



f(x) = x\*log10(x) – 1.2

## Source Code for Newton Raphson Method in C:

|  |  |
| --- | --- |
|  | #include<stdio.h>  #include<math.h>  float f(float x)  {      return x\*log10(x) - 1.2;  }  float df (float x)  {      return log10(x) + 0.43429;  }  void main()  {      int itr, maxmitr;      float h, x0, x1, allerr;      printf("\nEnter x0, allowed error and maximum iterations\n");      scanf("%f %f %d", &x0, &allerr, &maxmitr);      for (itr=1; itr<=maxmitr; itr++)      {          h=f(x0)/df(x0);          x1=x0-h;          printf(" At Iteration no. %3d, x = %9.6f\n", itr, x1);          if (fabs(h) < allerr)          {              printf("After %3d iterations, root = %8.6f\n", itr, x1);              return 0;          }          x0=x1;      }      printf(" The required solution does not converge or iterations are insufficient\n");      return 1;  } |

**7. Rate of Convergence**

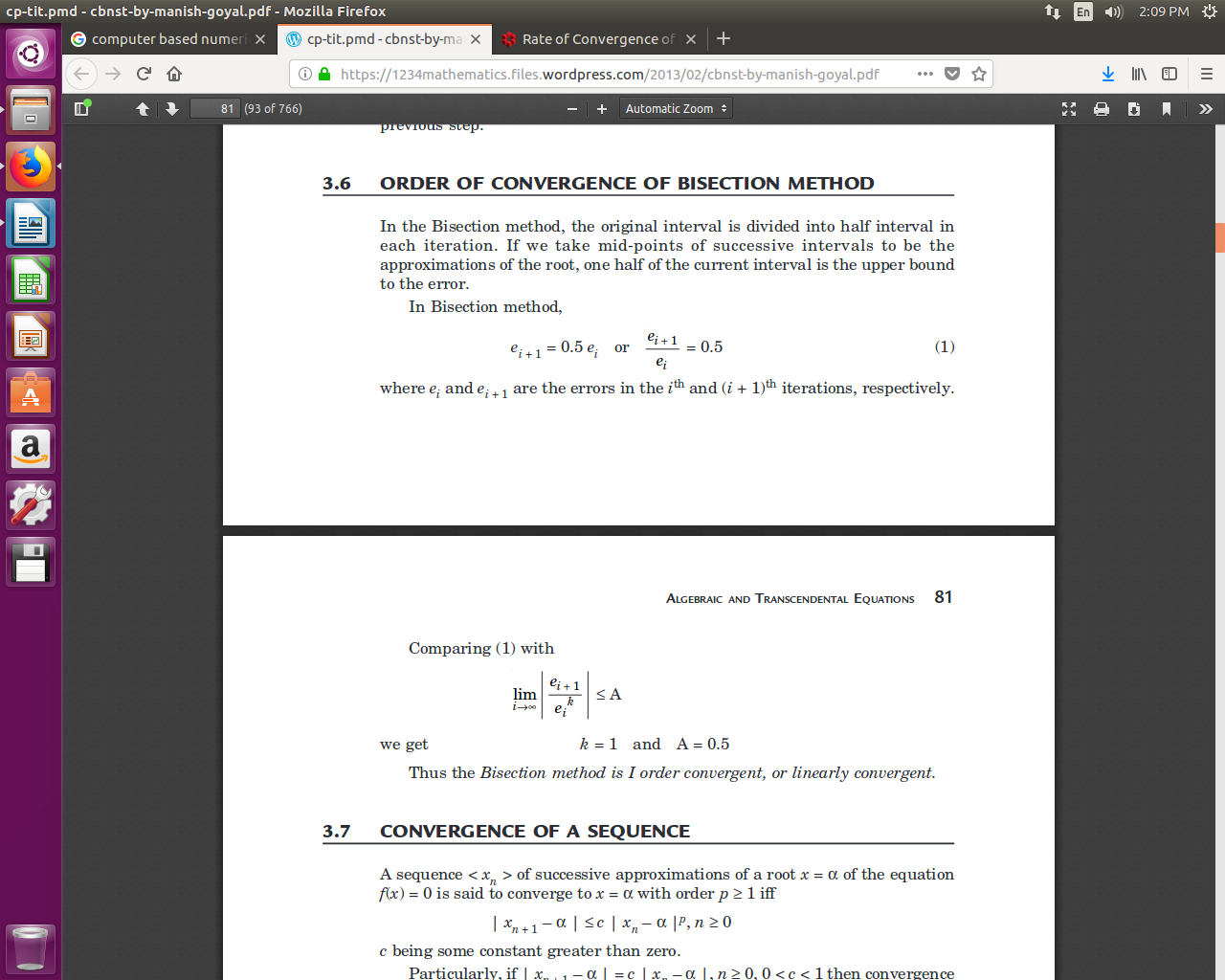
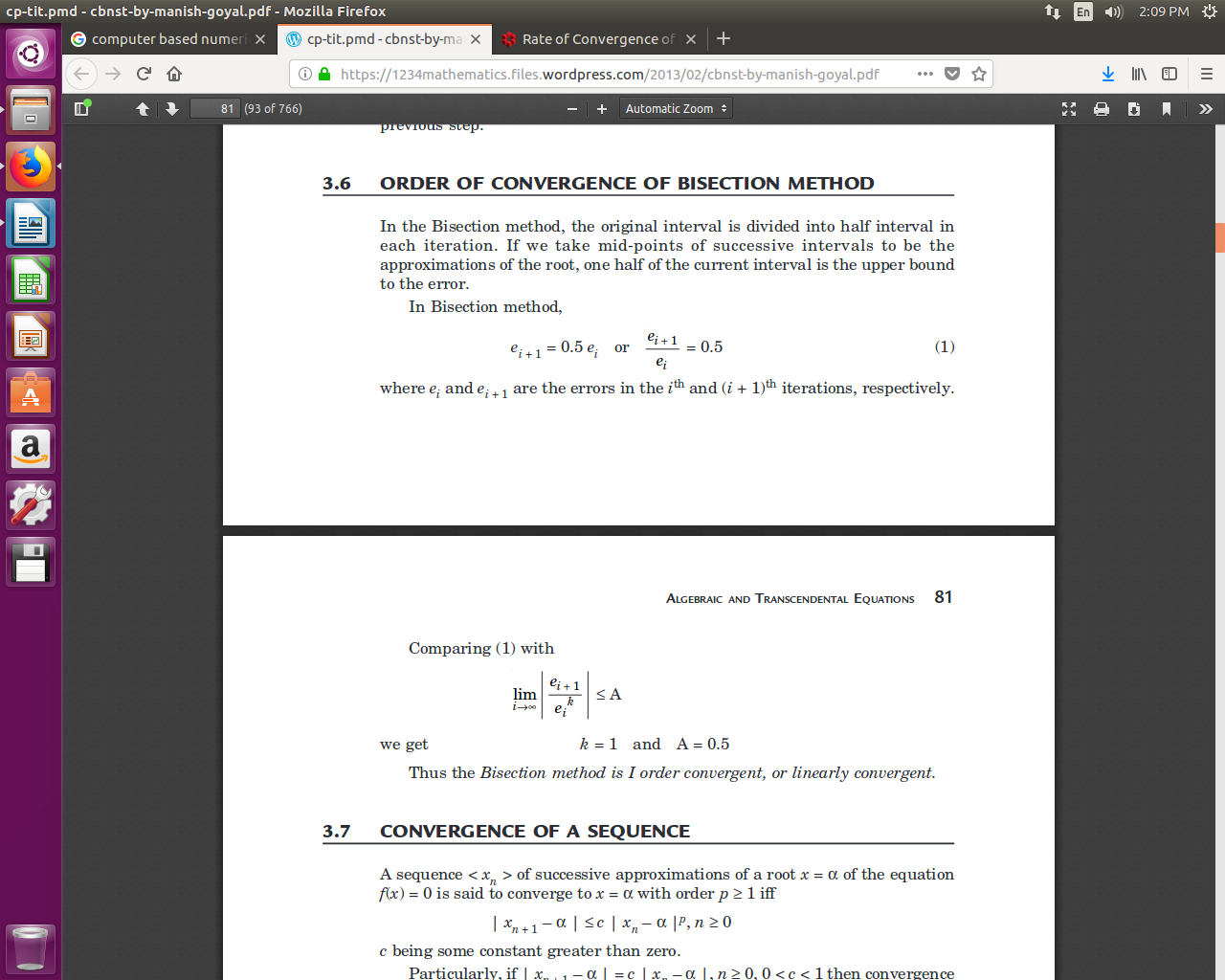
* We study different numerical methods to find a root of a equation?
* Because different method converge to the root with different speed.
* Rate of Convergence measures how fast of a sequence converges

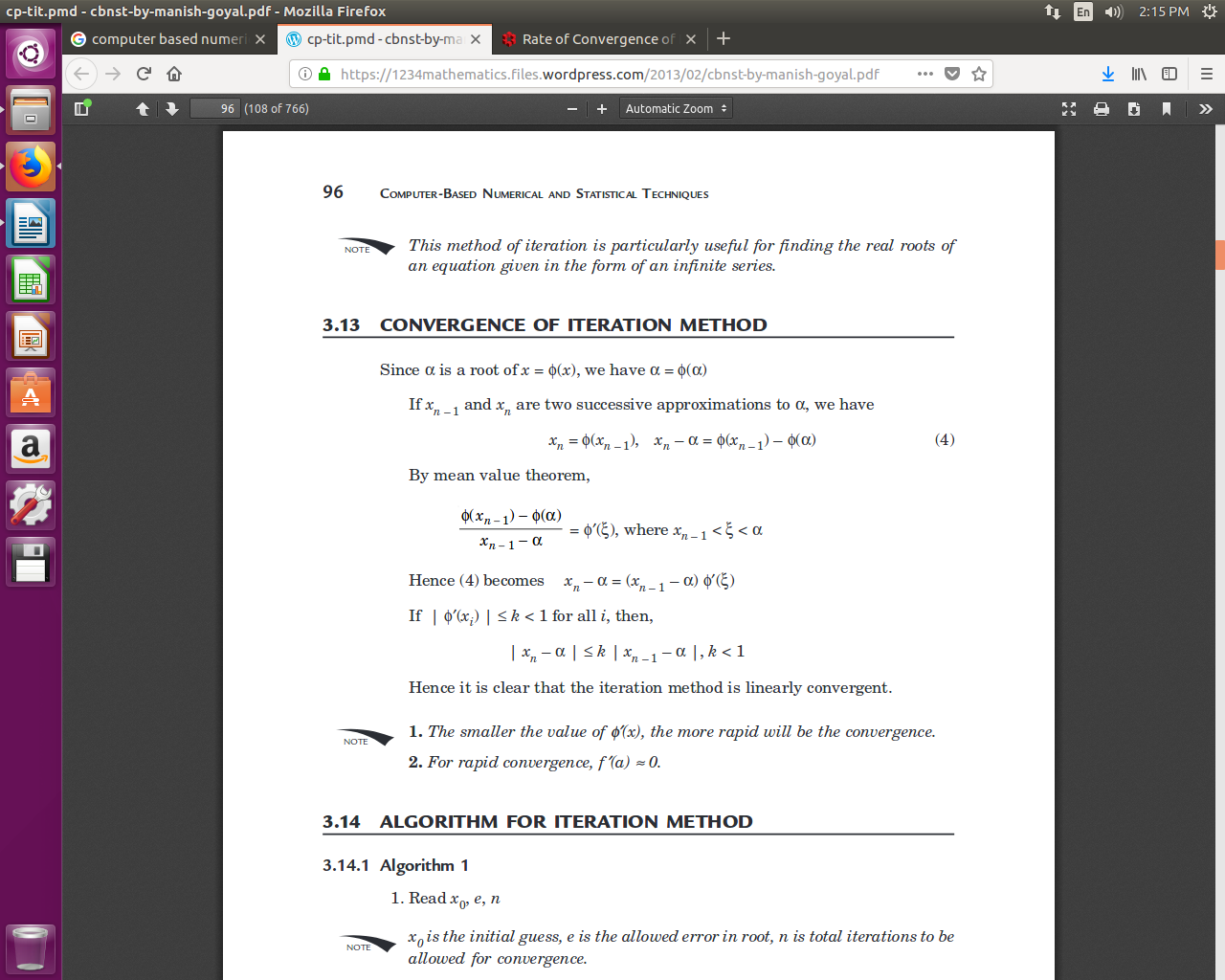
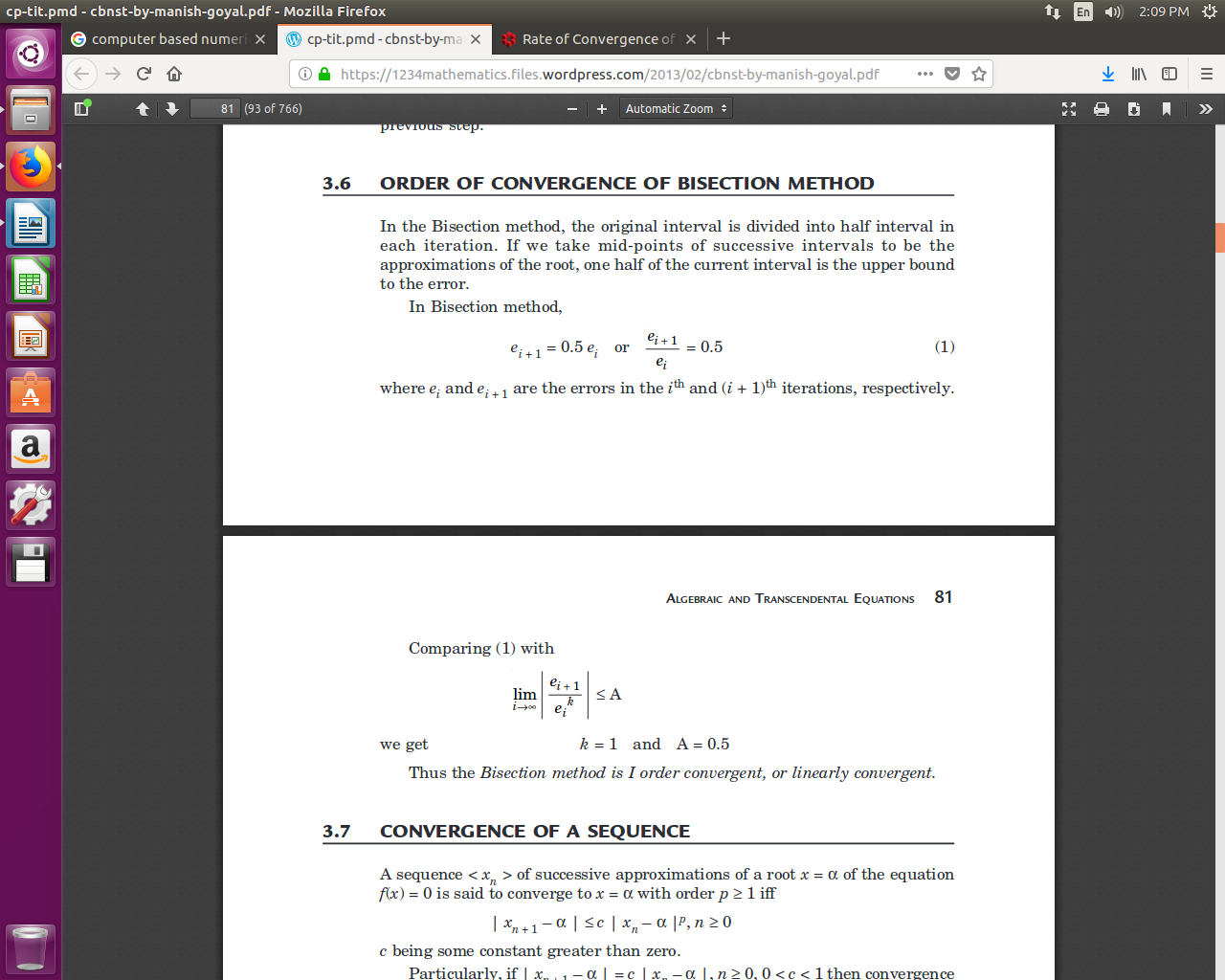
**7.1 ORDER OF CONVERGENCE OF BISECTION METHOD**

In the Bisection method, the original interval is divided into half interval in each iteration. If we take mid-points of successive intervals to be the approximations of the root, one half of the current interval is the upper bound

to the error.

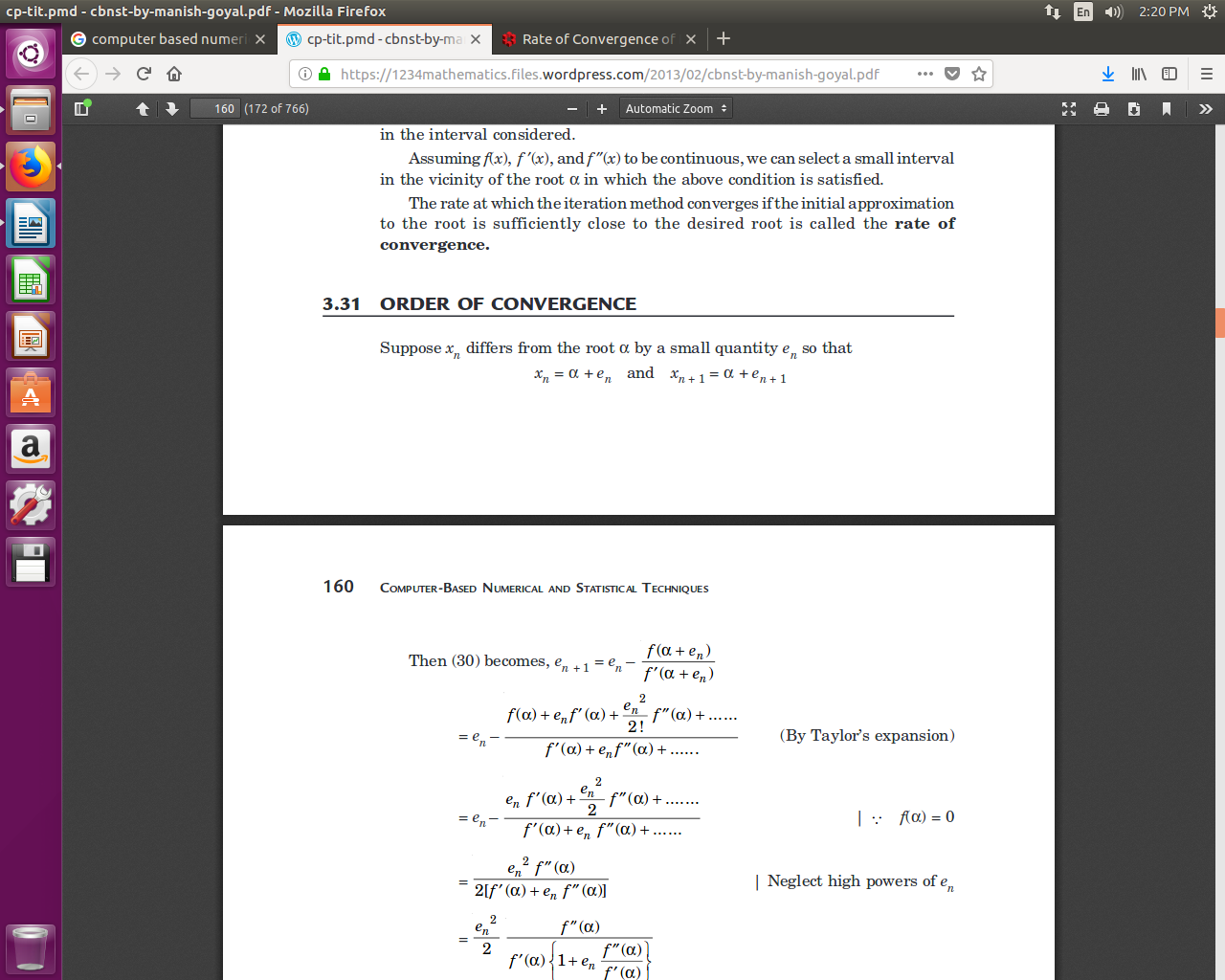
In Bisection method,

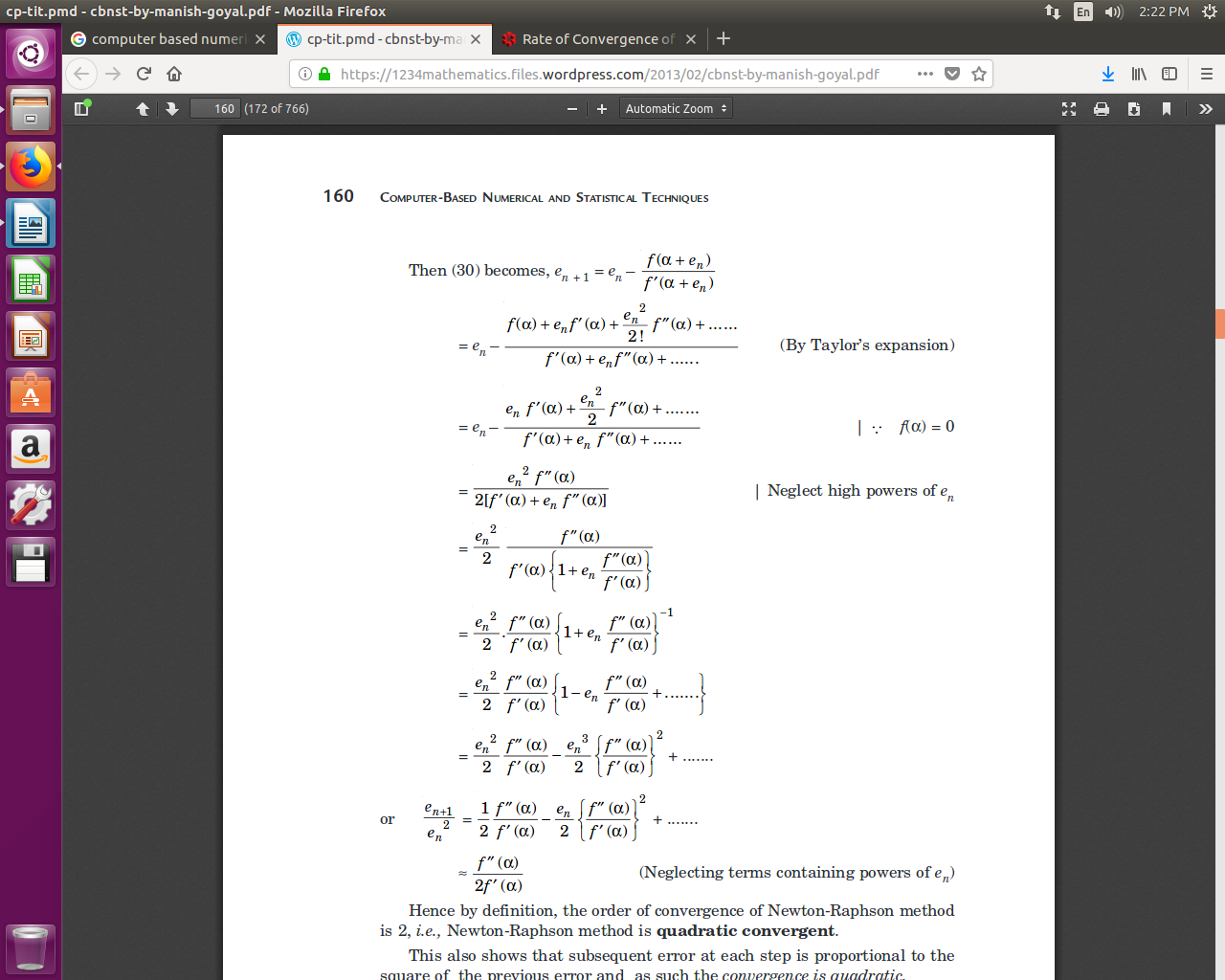




**ORDER OF CONVERGENCE**

**Suppose** xn differs from the root α by a small quantity en so that

****

****

Hence by definition, the order of convergence of Newton-Raphson method is 2,i.e.,Newton-Raphson method is **quadratic convergent**.

This also shows that subsequent error at each step is proportional to the square of the previous error and as such the convergence is quadratic.

Hence, if at the first iteration we have an answer correct to one decimal place, then it should be correct to two places at the second iteration, and to four places at the third iteration.

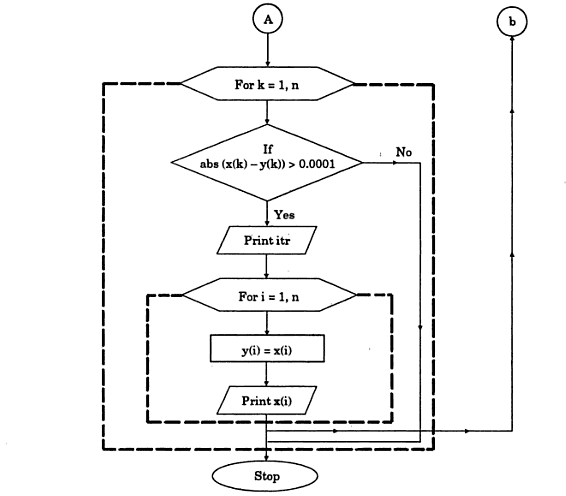
This means that the number of correct decimal places at each iteration is almost doubled

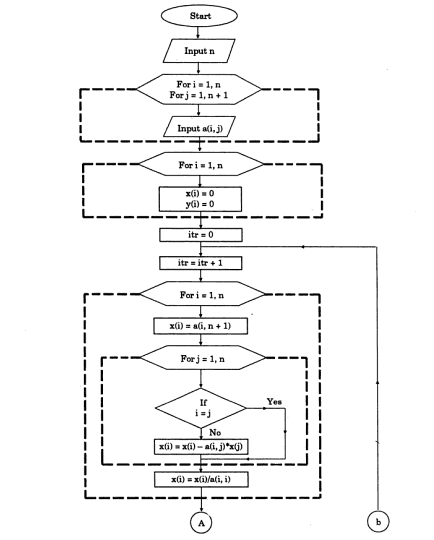
**8.****Gauss–Seidel method**

* In numerical linear algebra, the **Gauss–Seidel method**, also known as the **Liebmann method** or the **method of successive displacement.**
* It is an iterative method used to solve a linear system of equations.
* It is named after the German mathematicians Carl Friedrich Gauss and Philipp Ludwig von Seidel, and is similar to the Jacobi method.
* Though it can be applied to any matrix with non-zero elements on the diagonals, convergence is only guaranteed if the matrix is either diagonally dominant, or symmetric and positive definite.
* It was only mentioned in a private letter from Gauss to his student Gerling in 1823. A publication was not delivered before 1874 by Seidel.
* In case of iterative methods such as Gauss Jacobi and Gauss-Seidel iteration method, we start with an **approximate solution of equation and iterate it till we don’t get the result of desired accuracy**.
* The manual computation iterative method is quite lengthy. But, the program in high level languages run fast and effectively.
* Gauss-Seidel method has been **designed for the solution of linear simultaneous algebraic equations based on the principle of iteration**.

## Gauss-Seidel Method Algorithm:

1. Start
2. Declare the variables and read the order of the matrix n
3. Read the stopping criteria er
4. Read the coefficients aim as  
   Do for i=1 to n  
   Do for j=1 to n  
   Read a[i][j]  
   Repeat for j  
   Repeat for i
5. Read the coefficients b[i] for i=1 to n
6. Initialize x0[i] = 0 for i=1 to n
7. Set key=0
8. For i=1 to n  
   Set sum = b[i]  
   For j=1 to n  
   If (j not equal to i)  
   Set sum = sum – a[i][j] \* x0[j]  
   Repeat j  
   x[i] = sum/a[i][i]  
   If absolute value of ((x[i] – x0[i]) / x[i]) > er, then  
   Set key = 1  
   Set x0[i] = x[i]  
   Repeat i
9. If key = 1, then  
   Goto step 6  
   Otherwise print results

**Gauss-Seidel Method Flowchart:**

****

**x=1/a1(d1-b1y-c1z) y=1/b2(d2-a2x-c2z) z=1/c3 ( d3-a3x-b3y)**

**Source Code for Gauss-Seidel Method in C:**

|  |  |
| --- | --- |
|  | #include<stdio.h>  #include<math.h>  #define X 2  main()  {      float x[X][X+1],a[X], ae, max,t,s,e;      int i,j,r,mxit;      for(i=0;i<X;i++) a[i]=0;      puts(" Eneter the elemrnts of augmented matrix rowwise\n");      for(i=0;i<X;i++)      {      for(j=0;j<X+1;j++)      {      scanf("%f",&x[i][j]);      }      }      printf(" Eneter the allowed error and maximum number of iteration: ");      scanf("%f%d",&ae,&mxit);      printf("Iteration\tx[1]\tx[2]\n");      for(r=1;r<=mxit;r++)      {          max=0;          for(i=0;i<X;i++)          {              s=0;              for(j=0;j<X;j++)              if(j!=i) s+=x[i][j]\*a[j];              t=(x[i][X]-s)/x[i][i];              e=fabs(a[i]-t);              a[i]=t;          }          printf(" %5d\t",r);          for(i=0;i<X;i++)          printf(" %9.4f\t",a[i]);          printf("\n");          if(max<ae)          {              printf(" Converses in %3d iteration\n", r);              for(i=0;i<X;i++)              printf("a[%3d]=%7.4f\n", i+1,a[i]);              return 0;          }            }      } |