Computational Methods Lab
ME 15 L1

Department of Mechanical Engineering
College of Engineering Thalassery

June 24, 2016

Preface

This document is prepared to support the course, Computational Methods Lab (ME 15L1).The course offers a basic introduction to applied numerical methods and computer based problem solving.Topics include direct or iterative methods for the solution of polynomial and transcendental equations, systems of linear algebraic equations, Ordinary differential equations and numerical integration algorithms. This document contains a brief introduction about each of these methods as well as their solution algorithm with at least one solved example. However, detailed mathematical treatment of each method is beyond the scope of this document and the students are requested to refer dedicated books on the topic for better understanding. Some of the books are mentioned in the Bibliography.

For reference, some chapters are also provided with sample computer programmes written in C. These are given for the better understanding of the implementation of solution algorithm to the programme. **Students are encouraged to read them carefully and come up with their own codes.** Students, who turns up with more efficient algorithms will also be appreciated. I welcome any comments suggestions or corrections informed in person or sent by email to shal.arun@gmail.com

Contents

Part I

Root Finding

Chapter 1

Bisection Method

1.1 Principle

Bisection method is used to find the real roots of polynomial, transcendental, linear or non-linear equations. The process is based on the **Intermediate Value Theorem**. According to this theorem,"If a function $f(x) = 0$ is continuous in an interval (a, b), such that $f(a)$ and $f(b)$ are of opposite nature or opposite signs, then there exists at least one or an odd number of roots between a and b".In this method, if a function changes sign over an interval, the value of the function at the midpoint is evaluated. The location of the root is then determined as lying within the subinterval where the sign change occurs. The subinterval then becomes the interval for the next iteration. The process is repeated until the root is known to the required precision.Bisection method is a closed bracket method and requires two initial guesses. It is the simplest method with slow but steady rate of convergence.

In practical problems, the roots may not be reached exactly. In such case, we need to adopt a criterion to decide when to terminate the computations. A convenient criterion is to compute the error e defined by

$$e = \left| \frac{x - xold}{x} \right|$$

where $xold$ is the value of x in the previous iteration

1.2 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 $f2 = f(x2)$, assign: $xold = x1$
4. If $(f1 \times f2) > 0$, then display initial guesses are wrong and go to (11). Otherwise continue.
5. $x = (x1 + x2)/2$
6. If $|(x–xold)/x| < e$, then display x and go to (11).
7. Else, $f = f(x)$
8. If $((f \times f1) > 0)$, then $x1 = x$ and $f1 = f$.
9. Else, $x2 = x$ and $f2 = f$.
10. Go to (5). *Now the loop continues with new values.*
11. Stop

1.3 Example

Find a positive root of the equation $xe^x = 1$, which lies between 0 and 1.

Sol Here, $f(x) = xe^x - 1$ and initial guesses are 0 and 1 respectively. Since $f(0) = -1$ and $f(1) = 1.718$, it follows that a root lies between 0 and 1. Thus, $x_0 = 0.5$. Since $f(0.5)$ is negative, it follows that the root lies between 0.5 and 1. Hence the new root is 0.75, i.e., $x_1 = 0.75$. Using the values of x_0 and x_1, we calculate e_1:

$$e_1 = \left|\frac{x_1 - x_0}{x_1}\right| = 0.333..$$

Again, we find that $f(0.75)$ is positive and hence the root lies between 0.5 and 0.75, i.e. $x_2 = 0.625$. Now, the error is :

$$e_2 = \left|\frac{0.625 - 0.75}{0.625}\right| = 0.20$$

Proceeding in this way, the following table is constructed. The prescribed tolerance is 0.05%

Table 1.1: Bisection Method-Iterations.

Iteration	$x1$	$x2$	x	$f(x)$	e
1	0.000000	1.000000	0.500000	-0.175639	1.000000
2	0.500000	1.000000	0.750000	0.587750	0.333333
3	0.500000	0.750000	0.625000	0.167654	0.200000
4	0.500000	0.625000	0.562500	-0.012782	0.111111
5	0.562500	0.625000	0.593750	0.075142	0.052632
6	0.562500	0.593750	0.578125	0.030619	0.027027
7	0.562500	0.578125	0.570312	0.008780	0.013699
8	0.562500	0.570312	0.566406	-0.002035	0.006897
9	0.566406	0.570312	0.568359	0.003364	0.003436
10	0.566406	0.568359	0.567383	0.000662	0.001721
11	0.566406	0.567383	0.566895	-0.000687	0.000861
12	0.566895	0.567383	0.567139	-0.000013	0.000430

There, after 12 iterations, the error e, finally satisfies the required tolerance,viz.,0.05%. Hence the required root is 0.567 correct to 3 decimal places.

1.4 Sample Programme

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double funct(double x)
{
return x*exp(x)-1;
}
int main(void)
{
int itrmax=100,itr=1;
double x1, x2, x, xold, errormax,e;
//input parameters
printf("\nENTER THE LOWER LIMIT:\t");
scanf("%lf", &x1);
printf("\nENTER THE UPPER LIMIT:\t");
scanf("%lf", &x2);
printf("\nENTER THE MAX ERROR:\t");
scanf("%lf", &errormax);
if(funct(x2) * funct(x) <= 0)
{
do
{
xold = x1;
//computing the mid point
x = (x1 + x2)/2;
//error calcualtion
if (x!=0)
e = fabs(float (x- xold)/x);
else
e = fabs(float (x-xold)/xold);

//selecting the subinterval which contains the root
if (funct(x2) * funct(x) <= 0)
x1 = x;
else
x2 = x;
printf("The root after %2.1d iteration is %lf with error %lf\n",itr,x,e);

if (e <= errormax)
break;
++itr;
}while (itrmax > itr);
printf("\nRoot is: %lf with error +/-%lf after %d iterations\n",x,e,itr);

}
else
printf("\n Invalid limits!");
return 0;
}
```

Result:

```
ENTER THE LOWER LIMIT: 0

ENTER THE UPPER LIMIT: 1

ENTER THE MAX ERROR: 0.0005

The root after  1 iteration is 0.500000 with error 1.000000
The root after  2 iteration is 0.750000 with error 0.333333
The root after  3 iteration is 0.625000 with error 0.200000
The root after  4 iteration is 0.562500 with error 0.111111
The root after  5 iteration is 0.593750 with error 0.052632
The root after  6 iteration is 0.578125 with error 0.027027
The root after  7 iteration is 0.570312 with error 0.013699
The root after  8 iteration is 0.566406 with error 0.006897
The root after  9 iteration is 0.568359 with error 0.003436
The root after 10 iteration is 0.567383 with error 0.001721
The root after 11 iteration is 0.566895 with error 0.000861
The root after 12 iteration is 0.567139 with error 0.000430

Root is: 0.567139 with error +/-0.000430 after 12 iterations
```

Chapter 2

False Position Method

2.1 Principle

False position (also called the linear interpolation method or Regula falsi method) is another well-known bracketing method for finding roots of equations. In Bisection Method, the convergence process is very slow and the rate of convergence depends only on the choice of end point of intervals.False Position Method is very similar to bisection with the exception that it uses a different strategy to come up with its new root estimate. In false position method, rather than bisecting the interval$(x1, x2)$, it locates the root by joining $f(x1)$ and $f(x2)$ with a straight line. The intersection of this line with the x axis represents an improved estimate of the root. Thus, the shape of the function influences the new root estimate

The point of intersection mentioned above and hence the improved estimate of root can be calculated as:

$$x = \frac{x1f(x2) - x2f(x1)}{f(x2) - f(x1)}$$

In practical problems, the roots may not be reached exactly. In such case, we need to adopt a criterion to decide when to terminate the computations. A convenient criterion is to compute the error e defined by

$$e = \left| \frac{x - xold}{x} \right|$$

where $xold$ is the value of x in the previous iteration

2.2 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 $f2 = f(x2)$, assign: $xold = x1$
4. If $(f1 \times f2) > 0$, then display initial guesses are wrong and go to (11). Otherwise continue.
5. $x = \dfrac{x1f(x2) - x2f(x1)}{f(x2) - f(x1)}$
6. If $|(x–xold)/x| < e$, then display $x$ and go to (11).
7. Else, $f = f(x)$
8. If $((f \times f1) > 0)$, then $x1 = x$ and $f1 = f$.
9. Else, $x2 = x$ and $f2 = f$.
10. Go to (5). *Now the loop continues with new values.*
11. Stop
```

2.3 Example

Find a positive root of the equation $xe^x = 1$, which lies between 0 and 1.

Sol Here, $f(x) = xe^x - 1$ and initial guesses are 0 and 1 respectively. Since $f(0) = -1$ and $f(1) = 1.718$, it follows that a root lies between 0 and 1.Thus, $x_0 = 0.368$. Since $f(0.368)$ is negative, it follows that the root lies between 0.368 and 1. Hence the new root is 0.503, i.e., $x_1 = 0.503$. Using the values of x_0 and x_1, we calculate e_1:

$$e_1 = \left| \frac{x_1 - x_0}{x_1} \right| = 0.269$$

Again, we find that $f(0.503)$ is negative and hence the root lies between 0.503 and 1, i.e. $x_2 = 0.547$. Now, the error is :

$$e_2 = \left| \frac{0.547 - 0.503}{0.547} \right| = 0.08$$

Proceeding in this way, the following table is constructed. The prescribed tolerance is 0.05%

Table 2.1: False Position Method-Iterations.

Iteration	$x1$	$x2$	x	$f(x)$	e
1	0.000000	1.000000	0.367879	-0.468536	1.000000
2	0.367879	1.000000	0.503314	-0.167420	0.269086
3	0.503314	1.000000	0.547412	-0.053649	0.080557
4	0.547412	1.000000	0.561115	-0.016575	0.024421
5	0.561115	1.000000	0.565308	-0.005063	0.007418
6	0.565308	1.000000	0.566585	-0.001541	0.002254
7	0.566585	1.000000	0.566974	-0.000469	0.000685
8	0.566974	1.000000	0.567092	-0.000142	0.000208

There, after 8 iterations, the error e, finally satisfies the required tolerance,viz.,0.05%. Hence the required root is 0.567 correct to 3 decimal places.

2.4 Sample Programme

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double funct(double x)
{
return x*exp(x)-1;
}
int main(void)
{
int itrmax=100,itr=1;
double x1, x2, x, xold, errormax,e;
//input parameters
printf("\nENTER THE LOWER LIMIT:\t");
scanf("%lf", &x1);
printf("\nENTER THE UPPER LIMIT:\t");
scanf("%lf", &x2);
printf("\nENTER THE MAX ERROR:\t");
scanf("%lf", &errormax);
if(funct(x2) * funct(x) <= 0)
{
do
{
xold = x1;
//computing the root estimate
x = (x1*funct(x2)-x2*funct(x1))/(funct(x2)-funct(x1));
//error calcualtion
if (x!=0)
e = fabs(float (x- xold)/x);
else
e = fabs(float (x-xold)/xold);

//selecting the subinterval which contains the root
if (funct(x2) * funct(x) <= 0)
x1 = x;
else
x2 = x;
printf("The root after %2.1d iteration is %lf with error %lf\n",itr,x,e);

if (e <= errormax)
break;
++itr;
}while (itrmax > itr);
printf("\nRoot is: %lf with error +/-%lf after %d iterations\n",x,e,itr);

}
else
printf("\n Invalid limits!");
return 0;
}
```

Result:

```
ENTER THE LOWER LIMIT: 0

ENTER THE UPPER LIMIT: 1

ENTER THE MAX ERROR: 0.0005

ENTERED LOWER & UPPER LIMITS ARE:  0.000000 and 1.000000


The root after  1 iteration is 0.367879 with error 1.000000
The root after  2 iteration is 0.503314 with error 0.269086
The root after  3 iteration is 0.547412 with error 0.080557
The root after  4 iteration is 0.561115 with error 0.024421
The root after  5 iteration is 0.565308 with error 0.007418
The root after  6 iteration is 0.566585 with error 0.002254
The root after  7 iteration is 0.566974 with error 0.000685
The root after  8 iteration is 0.567092 with error 0.000208

Root is: 0.567092 with error +/-0.000208 after 8 iterations
```

Chapter 3

Successive Iteration Method

3.1 Principle

For the bracketing methods like Bisection and False position, the root is located within an interval prescribed by a lower and an upper bound. Repeated application of these methods always results in closer estimates of the true value of the root. In contrast, Successive iteration method described here require only a single starting value for computing the root. So this method is usually classified under open methods. As such, Open methods sometimes diverge or move away from the true root as the computation progresses. However, when they converge they usually do so much more quickly than the bracketing methods.

Open methods employ a formula to predict the root. Such a formula can be developed for simple fixed-point iteration (or, as it is also called, one-point iteration or successive substitution) by rearranging the function $f(x) = 0$ so that x is on the left-hand side of the equation by algebraic manipulation or by simply adding x to both sides of the original equation:

$$x = g(x)$$

The utility of the above equation is that it provides a formula to predict a new value of x as a function of an old value of x. Thus, given an initial guess at the root x_i , this equation can be used to compute a new estimate x_{i+1} as expressed by the iterative formula:

$$x_{i+1} = g(x_i)$$

As discussed earlier, error e at $i + 1^{th}$ step can be defined by:

$$e_i = \left| \frac{x_{i+1} - x_i}{x_{i+1}} \right|$$

3.2 Algorithm

1. Start
2. Read values of x_0 and e.*Here* x_0 *is the initial approximation,* e *is the absolute error or the desired degree of accuracy, also the stopping criteria*
3. Calculate $x_1 = g(x_0)$
4. If $|(x_1 – x_0)/x_1| < e$, then go to step 6.
5. Else, assign $x_0 = x_1$ and go to step 3.
6. Display x_1 as the root.
7. Stop

3.3 Example

Find a real root of the equation $x^3 + x^2 - 1 = 0$ on the interval $[0, 1]$ with an accuracy of 10^{-4}

Sol To find this root, we rewrite the given equation in the form

$$x = \frac{1}{\sqrt{x+1}}$$

Starting with an initial guess of $x_0 = 0.5$, the iterative equation $x_{i+1} = \frac{1}{\sqrt{x_i+1}}$ can be used to compute:

$$x_1 = \frac{1}{\sqrt{0.5+1}} = 0.816497$$

Proceeding in this way, the following table is constructed.The prescribed accuracy is 10^{-4}

Table 3.1: Successive Iteration Method-Iterations.

Iteration	x_i	x_{i+1}	e
1	0.500000	0.816497	0.387628
2	0.816497	0.741964	0.100453
3	0.741964	0.757671	0.020730
4	0.757671	0.754278	0.004498
5	0.754278	0.755007	0.000966
6	0.755007	0.754850	0.000208
7	0.754850	0.754884	0.000045

Chapter 4

Newton Raphson Method

4.1 Principle

Newton-Raphson method is also an open method which requires only a single approximation to find the root of an equation. It is one of the most widely used root finding formulas.

Let x_0 be an approximate root of $f(x) = 0$ and let $x_1 = x_0 + h$ be the correct root so that $f(x_1) = 0$. Expanding $f(x_0 + h)$ by Taylor's series, we obtain

$$f(x_0) + hf'(x_0) + \frac{h^2}{2!}f''(x_0) + ... = 0$$

Neglecting the second and higher order derivatives, we have

$$f(x_0) + hf''(x_0) = 0$$

which gives

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

A better approximation than x_0 is therefore given by x_1, where

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

Successive approximations are given by

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_{i+1})}.$$

The Newton-Raphson method can also be interpreted graphically.When the initial guess at the root is x_0 , a tangent can be extended from the point $[x_0, f(x_0)]$. The point where this tangent crosses the x axis usually represents an improved estimate of the root x_1.

4.2 Algorithm

1. Start
2. Read values of x_0 and e.*Here* x_0 *is the initial approximation,* e *is the absolute error or the desired degree of accuracy, also the stopping criteria*
3. Calculate $f = f(x_0)$ and $df = f'(x_0)$
4. $x_1 = x_0 - \dfrac{f}{df}$
5. If $|(x_1–x_0)/x_1| < e$, then go to step 7.
6. Else, assign $x_0 = x_1$ and go to step 3.
7. Display x_1 as the root.
8. Stop

4.3 Example

Use the Newton-Raphson method to estimate the root of $f(x) = e^{-x} - x$ employing an initial guess of $x_0 = 0$

Sol The first derivative of the function can be evaluated as

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

which can be substituted along with the original function into Newton-Raphson formula to give

$$x_{i+1} = x_i - \frac{e^{-x_i} - x_i}{-e^{-x_i} - 1)}$$

Starting with an initial guess of $x_0 = 0$, this iterative equation can be applied to compute

Table 4.1: Newton-Raphson Method-Iterations.

Iteration	x_0	$f(x_0)$	$f'(x_0)$	x_1	e
1	0.000000	1.000000	-2.000000	0.500000	1.000000
2	0.500000	0.106531	-1.606531	0.566311	0.117093
3	0.566311	0.001305	-1.567616	0.567143	0.001467
4	0.567143	0.000000	-1.567143	0.567143	0.000000

Thus, the approach rapidly converges on the true root.The relative error at each iteration decreases much faster than it does in successive iteration.

4.4 Sample Programme

```
#include<stdio.h>
#include<math.h>
float f(float x)
{
return exp(-x)-x;
}
float df (float x)
{
return -1*exp(-x)-1;
}
int main()
{
int itr, maxmitr=100;
float x0, x1, e, emax;
printf("\nEnter x0 and allowed error \n");
scanf("%f %f", &x0, &emax);
for (itr=1; itr<=maxmitr; itr++)
{
x1=x0-f(x0)/df(x0);
if (x1!=0)
e = fabs(double (x1- x0)/x1);
else
e = fabs(double (x1- x0)/x0);
printf(" \n At Iteration no. %3d,\tx = %9.6f,\twith error = %9.6f,", itr, x1, e);

if (e < emax)
{
printf("\nAfter %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;
}
```

Result:

```
Enter x0 and allowed error
0
0.0001

At Iteration no.   1,x =  0.500000,with error =  1.000000,
At Iteration no.   2,x =  0.566311,with error =  0.117093,
At Iteration no.   3,x =  0.567143,with error =  0.001467,
At Iteration no.   4,x =  0.567143,with error =  0.000000,
After   4 iterations, root = 0.567143
```

Chapter 5

Secant Method

5.1 Principle

Part II

Systems of Equations

Chapter 6

Gauss Elimination Method

6.1 Principle

Chapter 7

Gauss Jordan method

7.1 Principle

Chapter 8

Gauss-Seidel Method

8.1 Principle

Part III

Numerical Integration

Chapter 9

Trapezoidal rule

9.1 Principle

Chapter 10

Simpson's 1/3 rule

10.1 Principle

Chapter 11

Simpson's 3/8 rule

11.1 Principle

Chapter 12

Gauss quadrature formulae

12.1 Principle

Part IV

Ordinary Differential Equations

Chapter 13

Euler Method

13.1 Principle

Chapter 14

Runge-Kutta method

14.1 Principle

Chapter 15

Numerical solution of boundary value problems

15.1 Principle

Part V

File operations in C and Computer Graphics

Bibliography

[1] Chapra and Canale *Numerical methods for scientist and engineers* McGraw Hill.

[2] Froberg *Introduction to numerical analysis* Addison Wesley.

[3] Hildebrand *Introduction to Numerical Analysis* Tata McGraw Hill.

[4] Kandaswamy *Numerical Analysis* S Chand.