Numerical Methods in C Documentation

Release 1

Vineet Kumar

December 20, 2015

Contents

Iterative Solutions to Non Linear Equations:

CHAPTER 1

Fixed Point Iteration / Repeated Substitution Method

This is most easiest of all method. The logic is very simple. Given an equation, take an initial guess and and find the functional value for that guess, in the subsequent iteration the result obtained in last iteration will be new guess. Continue this process until get the required accuracy. This means that absoulte differebce between two subsequent functional values in two iterations is too small to be neglected.

But all functions don't converge. If for given function

$$|f'(x)| < 1$$

Then this method will converge to actual solution or in other cases, it will diverge.

1.1 Implementation in C

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

#define g(x) sqrt(x+6)
#define gd(a0) (0.5/sqrt(a0+6))
#define f(x) (x*x-x-6)
#define MAX_COUNT 5000
int main()
{
int count=0;
float x0=0,x1=0,error=0;
char iffound=0;
printf("Please enter the initial value: ");
scanf("%f",&x0);
do{
        x1=g(x0);
        error=fabs(x1-x0);
        if(count==0)
        {
                if(gd(x0)>1){
                        printf("\nThe equation is not convergent");
                        iffound=1;
                        break;
                }
                else{
                        printf("\n   i    xi      f(xi)          error");
                        printf("\n-------------------------------------------");
```

```
                }
        }
        printf("\n %3d  %3.5f    %3.5f  %3.5f",count,x0,f(x0),count==0?0:error);
        x0=x1;
        count++;
}while(error>0.0005 && count<MAX_COUNT);

if(!iffound)
        printf("\nThe required root is: %f\n",x0);
return 0;
}
```

1.2 Output

```
Please enter the initial value: 2

  i      xi      f(xi)          error
-----------------------------------------
  0    2.00000  -4.00000       -1.00000
  1    4.00000  6.00000        1.50000
  2    2.50000  -2.25000       0.90000
  3    3.40000  2.16000        0.63529
  4    2.76471  -1.12111       0.40551
  5    3.17021  0.88004        0.27760
  6    2.89262  -0.52538       0.18163
  7    3.07425  0.37674        0.12255
  8    2.95170  -0.23918       0.08103
  9    3.03273  0.16471        0.05431
 10    2.97842  -0.10745       0.03608
 11    3.01449  0.07268        0.02411
 12    2.99038  -0.04799       0.01605
 13    3.00643  0.03220        0.01071
 14    2.99572  -0.02137       0.00713
 15    3.00286  0.01429        0.00476
 16    2.99810  -0.00951       0.00317
 17    3.00127  0.00635        0.00211
 18    2.99915  -0.00423       0.00141
 19    3.00056  0.00282        0.00094
 20    2.99962  -0.00188       0.00063
 21    3.00025  0.00125        0.00042
The required root is: 2.999833
```

CHAPTER 2

Bisection Method

This is also an iterative method. To find root, repeatedly bisect an interval (containing the root) and then selects a subinterval in which a root must lie for further processing. Algorithm is quite simple and robust, only requirement is that initial search interval must encapsulates the actual root.

```
Given a function f (x) continuous on an interval [a,b] and f (a) * f (b) < 0
Do
     c = (a+b)/2
     if f (a) * f (c) < 0 then  b = c
                          else  a = c
while (none of the convergence criteria C1, C2 or C3 is satisfied)
```

where the criteria for convergence are :-

- C1. Fixing a priori the total number of bisection iterations N i.e., the length of the interval or the maximum error after N iterations in this case is less than $|b-a|/2N$.
- C2. By testing the condition $|c_i - c_{i-1}|$ (where i are the iteration number) less than some tolerance limit, say epsilon, fixed threshold.
- C3. By testing the condition $|f(c_i)|$ less than some tolerance limit alpha again fixed threshold.

2.1 C Implementation

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define f(x) ((x*x*x)-18)
int main(){
float a=0,b=0,error=0,m,mold;
int i=0;
printf("Input Interval: ");
scanf("%f %f",&a,&b);
if((f(a)*f(b))>0){
        printf("Invalid Interval Exit!");       //to test whether search interval
        exit(1);                                                    //is okay or not
}
else if(f(a)==0 || f(b)==0){
        printf("Root is one of interval bounds. Root is %f\n",f(a)==0?a:b);
        exit(0);
}
printf("Ite\ta\t\tb\t\tm\t\tf(m)\t\terror\n");
```

```
do{
        mold=m;
        m=(a+b)/2;
        printf("%2d\t%4.6f\t%4.6f\t%4.6f\t%4.6f\t",i++,a,b,m,f(m));
        if(f(m)==0){
                printf("Root is %4.6f\n",m);
        }else if ((f(a)*f(m))<0){
                b=m;
        }else a=m;
        error=fabs(m-mold);
        if(i==1){
                printf("----\n");
        }else printf("%4.6f\n",error);
}while(error>0.00005);
printf("Approximate Root is %4.6f",m);
return 0;
}
```

2.2 Output

```
 Input Interval: 1 3
Ite     a               b               m               f(m)            error
0       1.000000        3.000000        2.000000        -10.000000      ----
1       2.000000        3.000000        2.500000        -2.375000       0.500000
2       2.500000        3.000000        2.750000        2.796875        0.250000
3       2.500000        2.750000        2.625000        0.087891        0.125000
4       2.500000        2.625000        2.562500        -1.173584       0.062500
5       2.562500        2.625000        2.593750        -0.550446       0.031250
6       2.593750        2.625000        2.609375        -0.233189       0.015625
7       2.609375        2.625000        2.617188        -0.073128       0.007812
8       2.617188        2.625000        2.621094        0.007261        0.003906
9       2.617188        2.621094        2.619141        -0.032963       0.001953
10      2.619141        2.621094        2.620117        -0.012859       0.000977
11      2.620117        2.621094        2.620605        -0.002802       0.000488
12      2.620605        2.621094        2.620850        0.002230        0.000244
13      2.620605        2.620850        2.620728        -0.000286       0.000122
14      2.620728        2.620850        2.620789        0.000973        0.000061
15      2.620728        2.620789        2.620758        0.000343        0.000031
Approximate Root is 2.620758
```

CHAPTER 3

Regula Falsi Method

This method is improvement over slow convergence of bisection method. To find root, input is search Interval containing the root [a,b], then tangent is drawn joining (a,f(a)) & (b,f(b)). The point where the tangent touches the x-axis is point of interest.

```
Given a function f (x) continuos on an interval [a,b] such that f (a) * f (b) < 0
Do
       c = (a*f(b) - b*f(a))/(f(b) - f(a))
   if f (a) * f (c) < 0        then  b = c
                            else  a = c
while (none of the convergence criterion C1, C2 or C3 is satisfied)
```

The criteria for convergence are still same:-

- C1. Fixing a priori the total number of iterations N i.e., the length of the interval or the maximum error after N iterations in this case is less than $|b-a|/2N$.
- C2. By testing the condition $|c_i - c_{i-1}|$ (where i are the iteration number) less than some tolerance limit, say epsilon, fixed threshold.
- C3. By testing the condition $|f(c_i)|$ less than some tolerance limit alpha again fixed threshold.

3.1 C Implementation

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define f(x) ((x*x*x)-18)
int main(){
float a=0,b=0,error=0,c,cold;
int i=0;
printf("Input Interval: ");
scanf("%f %f",&a,&b);
if((f(a)*f(b))>0){
        printf("Invalid Interval Exit!");
        exit(1);
}
else if(f(a)==0 || f(b)==0){
        printf("Root is one of interval bounds. Root is %f\n",f(a)==0?a:b);
        exit(0);
}
printf("Ite\ta\t\tb\t\tc\t\tf(c)\t\terror\n");
```

```
do{
        cold=c;
        c=(((a*f(b))-(b*f(a)))/(f(b)-f(a)));
        printf("%2d\t%4.6f\t%4.6f\t%4.6f\t%4.6f\t",i++,a,b,c,f(c));
        if(f(c)==0){
                break;
        }else if(f(a)*f(c)<0){
                b=c;
        }else a=c;
                error=fabs(c-cold);
        if(i==1){
                printf("----\n");
        }else printf("%4.6f\n",error);

}while(error>0.00005);
printf(" Root is %4.6f \n",c);
return 0;
}
```

3.2 Output

```
Input Interval: 1 3
Ite     a               b               c               f(c)            error
0       1.000000        3.000000        2.307692        -5.710514       ----
1       2.307692        3.000000        2.576441        -0.897459       0.268749
2       2.576441        3.000000        2.614847        -0.121172       0.038406
3       2.614847        3.000000        2.619964        -0.016010       0.005117
4       2.619964        3.000000        2.620639        -0.002108       0.000675
5       2.620639        3.000000        2.620728        -0.000275       0.000089
6       2.620728        3.000000        2.620739        -0.000040       0.000011
Root is 2.620739
```

CHAPTER 4

Newton Raphson Method

This is fairly good method, which doesnt requires any search interval. It only needs an initial guess. But lack of interval is compensated by First order derivative of function. the algorithm is fairly simple and gives close the accurate results in most of the cases

```
do
               r+1=r-  f(r)/f'(r)
while (none of the convergence criterion C1 or C2 is met)
```

Convergence criterias are:-

- C1. Fixing apriori the total number of iterations N.
- C2. By testing the condition $|x_{i+1} - x_i|$ (where i is the iteration number) less than some tolerance limit, say epsilon, fixed threshold.

4.1 C Implementation

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define f(x) ((x*x*x)-18)
#define fd(x) (3*x*x)
#define fdd(x) 6*x
int main(){
float x0,x1,error,errorold,converge,order;
int i=0;
printf("Input the approximation : ");
scanf("%f",&x0);
converge= (f(x0)*fdd(x0))/(fd(x0)*fd(x0));
if(converge >1)
        exit(1);
printf("Ite\tX0\t\tX1\t\tError\t\tOrder\n");
do{
        errorold=error;
        x1=x0-(f(x0)/fd(x0));
        if(f(x1)==0){
                break;
        }
        error=fabs(x1-x0);
        printf("%2d\t%4.6f\t%4.6f\t%4.6f\t",++i,x0,x1,error);
        if(i==1||error==0||errorold==1){
```

```
                printf("-----\n");
        }
        else {order=log(error)/log(errorold);
        printf("%4.6f\n",order);
        }
        x0=x1;
}while(error>0.00005);
printf("Root is %4.6f",x0);
return 0;
}
```

4.2 Output

```
Ite     X0              X1              Error           Order
1       2.000000        2.833333        0.833333        -----
2       2.833333        2.636294        0.197040        8.909258
3       2.636294        2.620833        0.015461        2.566843
```

Root is 2.620833

CHAPTER 5

Secant Method

Newton Raphson is good general purpose root finding method, but sometimes if function is very complicated then computing derivates will take much computational time, so to overcome this issue, in secant method we approximate the first order derivative term f'(r). Algorithm is more or less similar to secant method

```
Given an equation f(x) = 0
Let the initial guesses be x0 and x1
   Do
      xi+1= xi -       ( f(xi) * (xi - xi-1) ) / ( f(xi) - f(xi-1) )
       while (none of the convergence criterion C1 or C2 is met)
```

Convergence criterias are:-

- C1. Fixing apriori the total number of iterations N.
- C2. By testing the condition $|x_{i+1} - x_i|$ (where i is the iteration number) less than some tolerance limit, say epsilon, fixed threshold.

5.1 C Implementation

```c
#include<stdio.h>
#include<math.h>
#define f(x) (pow(x,3)-18)
int main(){
float x0,x1,x2,error;
int i=0;
printf("Input Two Approximations: ");
scanf("%f %f",&x0,&x1);
printf("Ite\tX0\t\tX1\t\tf(X0)\t\tf(X1)\t\tError\n");
do{
        x2=((x0*f(x1))-((x1)*f(x0)))/((f(x1)-f(x0)));
        printf("%2d\t%4.6f\t%4.6f\t%4.6f\t%4.6f\t%4.6f\n",i++,x0,x1,f(x0),f(x1),error);
        error=fabs((x2)-(x1));
        x0=x1;
        x1=x2;
}while(error>0.00005);
return 0;
}
```

5.2 Output

```
Input Two Approximations: 1 2
Ite     X0              X1                  f(X0)            f(X1)           Error
0       1.000000        2.000000        -17.000000      -10.000000      0.000000
1       2.000000        3.428571        -10.000000      22.303208       1.428571
2       3.428571        2.442238        22.303208       -3.433201       0.986333
3       2.442238        2.573814        -3.433201       -0.949728       0.131575
4       2.573814        2.624131        -0.949728       0.069927        0.050317
5       2.624131        2.620680        0.069927        -0.001263       0.003451
6       2.620680        2.620741        -0.001263       -0.000001       0.000061
```

With this discussion, this series on solution of non linear equations with Iterative Methods concludes.

In this unit we shall discuss 5 methods for solutions of non linear simulataneous equation namely-

- Fixed Point Iteration
- Bisection Method
- Regula Falsi Method
- Newton Raphson Method
- Secant Method

First thing first, well all the codes illustrated in this tutorial are tested and compiled on a linux machine. To compile a C code, fire up a terminal by CTRL+ALT+T and type

```
gcc -o test test.c
```

where test.c is the name of program we want to compile. To execute a program

```
./test
```

Disclaimer: Well Please refer to a standard Text Book for detailed coverage of Theory, In this tutorial only minimal theoretical information will be put up which is essential for understanding the working of the method.

So, lets dive in...

Non-linear Equations is a set of equations in which unknowns appear as variables of a polynomial of degree higher than one. Examples are

$$x^2 = 25$$
$$y^2 - y = 6$$
$$x^2 - sinx = 1$$

These powers and vaiables may get complicated in that case, In that case manual hand computation will be too troublesome, so we can use numerical techniques to do the computations on computers and get results.

A computer code trying to solve a particular problem should have these characterstics properly specified

```
(1) Algorithm or Method Formula
        There are two type of Methods
+------------------------------+
|                              |
Iterative Methods              Direct Methods
```

```
(2) Stopping Condition: In case of Iterative methods we get closer
to actual solution in each iteration,
so we may need to define a sufficient and necessary
condition which will stop further iterations and prints the
results in desired accuracy.
```

To navigate back to this page, click on Tables of Content link on your left. If you would like to contribute feel free to fork the repo!