# Contents

1 Fixed Point Iteration / Repeated Substitution Method 3  
  1.1 Implementation in C ......................................................... 3  
  1.2 Output .............................................................................. 4  

2 Bisection Method 5  
  2.1 C Implementation ............................................................... 5  
  2.2 Output .............................................................................. 6  

3 Regula Falsi Method 7  
  3.1 C Implementation ............................................................... 7  
  3.2 Output .............................................................................. 8  

4 Newton Raphson Method 9  
  4.1 C Implementation ............................................................... 9  
  4.2 Output .............................................................................. 10  

5 Secant Method 11  
  5.1 C Implementation ............................................................... 11  
  5.2 Output .............................................................................. 12
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 absolute difference 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

```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("The equation is not convergent");
                ifound=1;
                break;
            }
        printf("%d	%1.2f	%1.2f	%1.2f	%1.2f
",i,x1,f(x1),error);
        printf("-------------------------------------------");
    }while(count<MAX_COUNT);
    return 0;
}
```

1.2 Output

Please enter the initial value: 2

<table>
<thead>
<tr>
<th>i</th>
<th>xi</th>
<th>f(xi)</th>
<th>error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2.00000</td>
<td>-4.00000</td>
<td>-1.00000</td>
</tr>
<tr>
<td>1</td>
<td>4.00000</td>
<td>6.00000</td>
<td>1.50000</td>
</tr>
<tr>
<td>2</td>
<td>2.50000</td>
<td>-2.25000</td>
<td>0.90000</td>
</tr>
<tr>
<td>3</td>
<td>3.40000</td>
<td>2.16000</td>
<td>0.63529</td>
</tr>
<tr>
<td>4</td>
<td>2.76471</td>
<td>-1.12111</td>
<td>0.40551</td>
</tr>
<tr>
<td>5</td>
<td>3.17021</td>
<td>0.88004</td>
<td>0.27760</td>
</tr>
<tr>
<td>6</td>
<td>2.89262</td>
<td>-0.52538</td>
<td>0.18163</td>
</tr>
<tr>
<td>7</td>
<td>3.07425</td>
<td>0.37674</td>
<td>0.12255</td>
</tr>
<tr>
<td>8</td>
<td>2.95170</td>
<td>-0.23918</td>
<td>0.08103</td>
</tr>
<tr>
<td>9</td>
<td>3.03273</td>
<td>0.16471</td>
<td>0.05431</td>
</tr>
<tr>
<td>10</td>
<td>2.97842</td>
<td>-0.10745</td>
<td>0.03608</td>
</tr>
<tr>
<td>11</td>
<td>3.01449</td>
<td>0.07268</td>
<td>0.02411</td>
</tr>
<tr>
<td>12</td>
<td>2.99038</td>
<td>-0.04799</td>
<td>0.01605</td>
</tr>
<tr>
<td>13</td>
<td>3.00643</td>
<td>0.03220</td>
<td>0.01071</td>
</tr>
<tr>
<td>14</td>
<td>2.99572</td>
<td>-0.02137</td>
<td>0.00713</td>
</tr>
<tr>
<td>15</td>
<td>3.00286</td>
<td>0.01429</td>
<td>0.00476</td>
</tr>
<tr>
<td>16</td>
<td>2.99810</td>
<td>-0.00951</td>
<td>0.00317</td>
</tr>
<tr>
<td>17</td>
<td>3.00127</td>
<td>0.00635</td>
<td>0.00211</td>
</tr>
<tr>
<td>18</td>
<td>2.99915</td>
<td>-0.00423</td>
<td>0.00141</td>
</tr>
<tr>
<td>19</td>
<td>3.00056</td>
<td>0.00282</td>
<td>0.00094</td>
</tr>
<tr>
<td>20</td>
<td>2.99962</td>
<td>-0.00188</td>
<td>0.00063</td>
</tr>
<tr>
<td>21</td>
<td>3.00025</td>
<td>0.00025</td>
<td>0.00042</td>
</tr>
</tbody>
</table>

The required root is: 2.999833
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 \( f(a) \cdot f(b) < 0 \)

Do
\[
\begin{align*}
c &= \frac{(a+b)}{2} \\
\text{if } f(a) \cdot f(c) < 0 \text{ then } b &= c \\
\text{else } a &= c
\end{align*}
\]
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!\n");
        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\tb\tm\tf(m)\terror\n");
    while (none of the convergence criteria C1, C2 or C3 is satisfied)
    {
        c = (a+b)/2;
        if f(a) * f(c) < 0 then b = c
        else a = c
    }
    printf("\n");
}
```

```c
do
  mold=m;
  m=(a+b)/2;
  printf("%2d\t%4.6f\t%4.6f\t%4.6f\t%4.6f\n",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

<table>
<thead>
<tr>
<th>Ite</th>
<th>a</th>
<th>b</th>
<th>m</th>
<th>f(m)</th>
<th>error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.000000</td>
<td>3.000000</td>
<td>2.000000</td>
<td>-10.000000</td>
<td>----</td>
</tr>
<tr>
<td>1</td>
<td>2.000000</td>
<td>3.000000</td>
<td>2.500000</td>
<td>-2.375000</td>
<td>0.500000</td>
</tr>
<tr>
<td>2</td>
<td>2.500000</td>
<td>3.000000</td>
<td>2.750000</td>
<td>2.796875</td>
<td>0.250000</td>
</tr>
<tr>
<td>3</td>
<td>2.500000</td>
<td>2.750000</td>
<td>2.625000</td>
<td>0.087891</td>
<td>0.125000</td>
</tr>
<tr>
<td>4</td>
<td>2.500000</td>
<td>2.625000</td>
<td>2.562500</td>
<td>-1.173584</td>
<td>0.062500</td>
</tr>
<tr>
<td>5</td>
<td>2.562500</td>
<td>2.625000</td>
<td>2.593750</td>
<td>-0.550446</td>
<td>0.031250</td>
</tr>
<tr>
<td>6</td>
<td>2.593750</td>
<td>2.625000</td>
<td>2.609375</td>
<td>-0.233189</td>
<td>0.015625</td>
</tr>
<tr>
<td>7</td>
<td>2.609375</td>
<td>2.625000</td>
<td>2.617188</td>
<td>-0.073128</td>
<td>0.007812</td>
</tr>
<tr>
<td>8</td>
<td>2.617188</td>
<td>2.625000</td>
<td>2.621094</td>
<td>0.007261</td>
<td>0.003906</td>
</tr>
<tr>
<td>9</td>
<td>2.617188</td>
<td>2.621094</td>
<td>2.619141</td>
<td>-0.032963</td>
<td>0.001953</td>
</tr>
<tr>
<td>10</td>
<td>2.619141</td>
<td>2.621094</td>
<td>2.620117</td>
<td>-0.012859</td>
<td>0.000977</td>
</tr>
<tr>
<td>11</td>
<td>2.620117</td>
<td>2.621094</td>
<td>2.620605</td>
<td>-0.002802</td>
<td>0.000488</td>
</tr>
<tr>
<td>12</td>
<td>2.620605</td>
<td>2.621094</td>
<td>2.620850</td>
<td>0.002230</td>
<td>0.000244</td>
</tr>
<tr>
<td>13</td>
<td>2.620605</td>
<td>2.620850</td>
<td>2.620728</td>
<td>-0.000268</td>
<td>0.000122</td>
</tr>
<tr>
<td>14</td>
<td>2.620728</td>
<td>2.620850</td>
<td>2.620789</td>
<td>0.000973</td>
<td>0.000061</td>
</tr>
<tr>
<td>15</td>
<td>2.620728</td>
<td>2.620789</td>
<td>2.620758</td>
<td>0.000343</td>
<td>0.000031</td>
</tr>
</tbody>
</table>

Approximate Root is 2.620758
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.

\[
\text{Given a function } f(x) \text{ continuos on an interval } [a,b] \text{ such that } f(a) \times f(b) < 0
\]

\[
\text{Do}
\]
\[
\begin{align*}
  c &= \frac{a \times f(b) - b \times f(a)}{f(b) - f(a)} \\
  \text{if } f(a) \times f(c) < 0 & \quad \text{then } b = c \\
  \text{else } a &= c
\end{align*}
\]

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("Iter\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\n",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

<table>
<thead>
<tr>
<th>Ite</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>f(c)</th>
<th>error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.000000</td>
<td>3.000000</td>
<td>2.307692</td>
<td>-5.710514</td>
<td>----</td>
</tr>
<tr>
<td>1</td>
<td>2.307692</td>
<td>3.000000</td>
<td>2.576441</td>
<td>-0.897459</td>
<td>0.268749</td>
</tr>
<tr>
<td>2</td>
<td>2.576441</td>
<td>3.000000</td>
<td>2.614847</td>
<td>-0.121172</td>
<td>0.038406</td>
</tr>
<tr>
<td>3</td>
<td>2.614847</td>
<td>3.000000</td>
<td>2.619964</td>
<td>-0.016010</td>
<td>0.005117</td>
</tr>
<tr>
<td>4</td>
<td>2.619964</td>
<td>3.000000</td>
<td>2.620639</td>
<td>-0.002108</td>
<td>0.000675</td>
</tr>
<tr>
<td>5</td>
<td>2.620639</td>
<td>3.000000</td>
<td>2.620728</td>
<td>-0.000275</td>
<td>0.000089</td>
</tr>
<tr>
<td>6</td>
<td>2.620728</td>
<td>3.000000</td>
<td>2.620739</td>
<td>-0.000040</td>
<td>0.000011</td>
</tr>
</tbody>
</table>

Root is 2.620739
Newton Raphson Method

This is fairly good method, which doesn’t require 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.

\[
do\quad r+1=r - \frac{f(r)}{f'(r)}
\while (\text{none of the convergence criterion C1 or C2 is met})
\]

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 tX1 tError 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%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

<table>
<thead>
<tr>
<th>Ite</th>
<th>X0</th>
<th>X1</th>
<th>Error</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.000000</td>
<td>2.833333</td>
<td>0.83333</td>
<td>--------</td>
</tr>
<tr>
<td>2</td>
<td>2.833333</td>
<td>2.636294</td>
<td>0.19704</td>
<td>8.909258</td>
</tr>
<tr>
<td>3</td>
<td>2.636294</td>
<td>2.620833</td>
<td>0.01546</td>
<td>2.566843</td>
</tr>
</tbody>
</table>

Root is 2.620833
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 $x_0$ and $x_1$
Do
\[ x_{i+1} = x_i - \frac{f(x_i) * (x_i - x_{i-1})}{f(x_i) - f(x_{i-1})} \]
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\t\tX0\t\tX1\t\t\t\t|f(X0)|\t\t\t|f(X1)|\t\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\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

<table>
<thead>
<tr>
<th>It</th>
<th>X0</th>
<th>X1</th>
<th>f(X0)</th>
<th>f(X1)</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.000000</td>
<td>2.000000</td>
<td>-17.00000</td>
<td>-10.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>1</td>
<td>2.000000</td>
<td>3.428571</td>
<td>-10.00000</td>
<td>22.303208</td>
<td>1.428571</td>
</tr>
<tr>
<td>2</td>
<td>3.428571</td>
<td>2.442238</td>
<td>22.303208</td>
<td>-3.433201</td>
<td>0.986333</td>
</tr>
<tr>
<td>3</td>
<td>2.442238</td>
<td>2.573814</td>
<td>-3.433201</td>
<td>-0.949728</td>
<td>0.131575</td>
</tr>
<tr>
<td>4</td>
<td>2.573814</td>
<td>2.624131</td>
<td>-0.949728</td>
<td>0.069927</td>
<td>0.050317</td>
</tr>
<tr>
<td>5</td>
<td>2.624131</td>
<td>2.620680</td>
<td>0.069927</td>
<td>-0.001263</td>
<td>0.003451</td>
</tr>
<tr>
<td>6</td>
<td>2.620680</td>
<td>2.620741</td>
<td>-0.001263</td>
<td>-0.000001</td>
<td>0.000061</td>
</tr>
</tbody>
</table>

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

\[ x^2 = 25 \]
\[ y^2 - y = 6 \]
\[ x^2 - \sin x = 1 \]

These powers and variables 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 characteristics properly specified

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


12 Chapter 5. Secant Method
(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!