Problem Statement : Implement Bisection method to find a root of polynomial equation.

Problem Analysis: Using The bisection method we can find a root of equation **f(x)=0***,* if **f(x)** is a [continuous function](https://en.wikipedia.org/wiki/Continuous_function) on the interval [**a,b**] and **f(a)** and **f(b)** have opposite signs. The [absolute error](https://en.wikipedia.org/wiki/Approximation_error) is halved at each step so the method [converges linearly](https://en.wikipedia.org/wiki/Rate_of_convergence). Specifically, **x0 =a+b/2** is the midpoint of the initial interval, and **xn** is the midpoint of the interval in the nth step. Then we have to compute absolute value of **f(xi)** (where i = 0,1,2,…,n) and if the value is zero, **xi** is the root. Otherwise, if **f(xi)** & **f(a)** are of opposite signs ,the root lies between **a** & **xi**and we will replace **b** by **xi ,** else we will replace **a** by **xi** and generate next approximation. We will repeat these process until the root is obtained.

Algorithm:

1. Start.
2. Input degree **n** and **(n+1)** co-efficients of the equation **f(x)** **= 0**.
3. Input the desired precision and compute error criterion **E**.
4. Choose two real values for **a** and **b**.
5. If **f(a) \* f(b)>0**, a and b do not bracket any root and go to step 4.
6. Compute **x := (a+b)/2** and compute **f(x).**
7. If **f(x)= 0** or absolute value of **f(x)** **<=** **E** ,then  
    **x** is the required root , go to step 8.  
   else if **f(a)\*f(b) < 0**  
    **set b = x ,** go to step 6.   
   else  
    **set a = x ,** go to step 6.
8. Print **x**.
9. End.

Source Code:

#include<bits/stdc++.h>

#define accuracy 0.0001

using namespace std;

int a,b,c,d,degree;

double m,n,x,z,sum;

int arr[10];

double fun(double x)

{

sum=0;

int i;

z=degree;

for(i=0;i<=degree;i++)

{

sum+=arr[i]\*pow(x,z);

z--;

}

return sum;

}

int main()

{

int i;

cout <<"enter the value of the highest degree of the equation" << endl;

cin >> degree;

cout <<"Enter the co-efficients :" <<endl;

for(i=0;i<=degree;i++)

cin >> arr[i];

cout << "Enter two real number assuming the root is between them" << endl;

cin >> m >> n ;

while(fun(m)\*fun(n)>=0)

{

cout << "Enter two real number assuming the root is between them (again)" << endl;

cin >> m >> n ;

}

while(abs(n-m)>accuracy)

{

x=(m+n)/2;

if(fun(x)==0) break;

else if(fun(m)\*fun(x)<0) n=x;

else m=x;

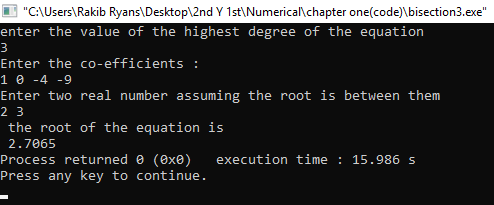
}

printf(" the root of the equation is\n %.4lf",x);

return 0;

}

Sample Input & Output:



Problem Statement : Implement False Position method to find a root of polynomial equation.

Problem Analysis: The false-position method is a modification on the bisection method: if it is known that the root lies on [*a*, *b*], then it is reasonable that we can approximate the function on the interval by interpolating the points (*a*, f(*a*)) and (*b*, f(*b*)).

Algorithm:

1. Start.
2. Input degree **n** and **(n+1)** co-efficients of the equation **f(x)** **= 0**.
3. Input the desired precision and compute error criterion **E**.
4. Choose two real values for **a** and **b**.
5. If **f(a) \* f(b)>0**, a and b do not bracket any root and go to step 4.
6. Compute: **x := (a f(b) – b f(a) / ( f(b) – f(a) )** and compute **f(x).**
7. If **f(x)= 0** or absolute value of **f(x)** **<=** **E** ,then  
    **x** is the required root , go to step 8.  
   else if **f(a)\*f(b) < 0**  
    **set b := x ,** go to step 6.   
   else  
    **set a := x ,** go to step 6.
8. Print **x**.
9. End.

Source Code:

#include<bits/stdc++.h>

#define accuracy 0.05

using namespace std;

int a,b,c,d,degree;

double m,n,x,z,sum,k,loop=0;

int arr[100];

double f(double x)

{

sum=0;

int i; z=degree;

for(i=0;i<=degree;i++)

{

sum+=arr[i]\*pow(x,z);

z--;

}

return sum;

}

int main()

{

int i;

cout <<"enter the value of the highest degree of the equation" << endl;

cin >> degree;

cout <<"Enter the co-efficients :" <<endl;

for(i=0;i<=degree;i++)

cin >> arr[i];

cout << "Enter two real number assuming the root is between them" << endl;

cin >> m >> n ;

while(f(m)\*f(n)>=0)

{

cout << "Enter two real number assuming the root is between them (again)" << endl;

cin >> m >> n ;

}

while(1)

{

x=(m\*f(n)-n\*f(m))/float(f(n)-f(m));

if(loop>0)

{

if(abs( (( 1-(k/x) )\*100) <=accuracy ) ) break;

}

if(f(x)==0) break;

if(f(x)\*f(m)<0) n=x;

else m=x;

k=x;

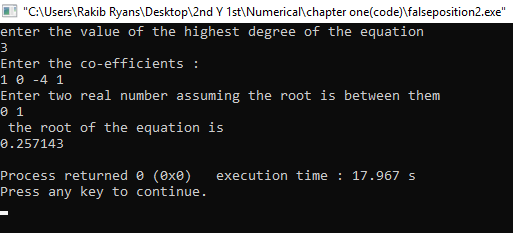
loop++;

}

cout << " the root of the equation is"<<endl<< x << endl;

return 0;}

Sample Input & Output:



Problem Statement : Implement Secant method to find a root of polynomial equation.

Algorithm:

1. Start.
2. Input degree **n** and **(n+1)** co-efficients of the equation **f(x)=0**.
3. Input the desired precision and compute error criterion **E**.
4. Choose two real values for **x-1**and **x0**.
5. Compute: **xi+1 := ( xi-1 fi – xi fi-1 ) / ( fi – fi-1 )** and compute **f(xi+1).**
6. If **f(xi+1)= 0** or absolute value of **f(xi+1)** **<=** **E** ,then  
    **x** is the required root , go to step 8.  
   else if **fi-1 \* fi < 0**  
    **set fi := xi+1 ,** go to step 5.   
   else  
    **set fi-1 := xi+1 ,** go to step 5.
7. Print **xi+1**.
8. End.

Source Code:

#include<bits/stdc++.h>

#define accuracy 0.05

using namespace std;

int a,b,c,d,degree;

double m,n,x,z,sum,k,loop=0;

int arr[100];

double f(double x)

{

sum=0;

int i; z=degree;

for(i=0;i<=degree;i++)

{

sum+=arr[i]\*pow(x,z);

z--;

}

return sum;

}

int main()

{

int i;

cout <<"enter the value of the highest degree of the equation" << endl;

cin >> degree;

cout <<"Enter the co-efficients :" <<endl;

for(i=0;i<=degree;i++)

cin >> arr[i];

cout << "Enter two real number assuming the root is between them" << endl;

cin >> m >> n ;

while(f(m)\*f(n)>=0)

{

cout << "Enter two real number assuming the root is between them (again)" << endl;

cin >> m >> n ;

}

while(1)

{

x=(m\*f(n)-n\*f(m))/float(f(n)-f(m));

if(loop>1)

{

if(abs( (( 1-(k/x) )\*100) <=accuracy ) ) break;

}

if(f(x)==0) break;

m=n;

n=x;

k=x;

loop++;

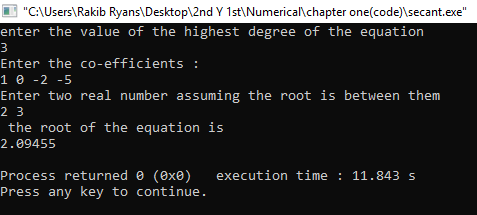
}

cout << " the root of the equation is"<<endl<< x << endl;

return 0;

}

Sample Input & Output:



Problem Statement : Implement Newton-Raphson method to find a root of polynomial equation.

Problem Analysis: This method requires only one appropriate starting point $ x_{0}$ as an initial assumption of the root of the function $ f(x)=0$. At $ (x_{0},f(x_{0}))$ a tangent to $ f(x)=0$ is drawn. Equation of this tangent is given by

$\displaystyle y=f'(x_{0})(x-x_{0})+f(x_{0})$

The point of intersection, say https://nptel.ac.in/courses/122104019/numerical-analysis/Rathish-kumar/ratish-1/img124a.gif, 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

$\displaystyle x_{1}=x_{0}-\frac{f(x_{0})}{f'(x_{0})}$

Algorithm:

1. Start.
2. Input degree **n** and **(n+1)** co efficients of the equation **f(x)=0**.
3. Choose the initial approximation to the root **x0**.
4. Input the desired precision and compute error criterion **E**.
5. Compute: **f(xi)** and **f*′*(xi).**
6. Compute: **xi+1 := xi – ( f(xi)** **/** **f*′*(xi) )** and compute **f(xi+1).**
7. If absolute value of **f(xi+1)** **<=** **E** ,then  
    **xi+1** is the required root , go to step 8.  
   else  
   set **xi := xi+1 ,** go to step 5.
8. Print **xi+1**.
9. End.

Source Code:

#include<bits/stdc++.h>

#define accuracy 0.05

using namespace std;

int a,b,c,d,degree;

double m,n,x,z,sum,sum1,sum2,k,loop=0;

int arr[100];

double f(double x)

{

sum1=0,sum2=0,sum=0;

int i;

z=degree;

for(i=0;i<=degree;i++)

{

sum1+=arr[i]\*pow(x,z);

z--;

}

z=degree;

for(i=0;i<degree;i++)

{

sum2+=arr[i]\*z\*pow(x,z-1);

z--;

}

sum=x-(sum1/sum2);

return sum;

}

int main()

{

int i;

cout <<"enter the value of the highest degree of the equation" << endl;

cin >> degree;

cout <<"Enter the co-efficients :" <<endl;

for(i=0;i<=degree;i++)

cin >> arr[i];

cout << "Enter a real number assuming the root " << endl;

cin >> m ;

while(1)

{

x=f(m);

if(loop>0)

{

if(abs( (( 1-(k/x) )\*100) <=accuracy ) ) break;

}

m=x;

k=x;

loop++;

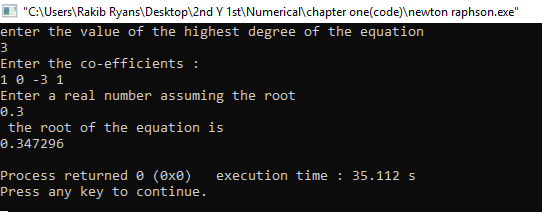
}

cout << " the root of the equation is"<<endl<< x << endl;

return 0;

}

**Sample Input Output:**



roblem Statement : Implement Generalized Newton’s method to find a root of polynomial equation.

Algorithm:

1. Start.
2. Input degree **n** and **(n+1)** co efficients of the equation **f(x)=0**.
3. Choose the initial approximation to the root **x0**.
4. Input the desired precision and compute error criterion **E**.
5. Determine multiplicity **p.**
6. Compute: **xi+1 := xi – p ( f(xi)** **/** **f*′*(xi) )** and **xi+1 := xi – (p-1) ( f(xi)** **/** **f*′*(xi) ).**
7. If absolute value of **f(xi+1)** **<=** **E** for any of the **xi+1**,then  
   corresponding **xi+1** is the required root , go to step 8.  
   else  
   set **xi := xi+1**; **xi+1** is the value for which **f(xi+1)** is less than the other,go to step 6.
8. Print **xi+1**.
9. End.

Source Code:

#include<iostream>

#include<cmath>

#include<iomanip>

using namespace std;

double func(double v,double \*coeff,int n)

{

double sum = 0;

for(int i=0;i<=n;i++)

{

sum += coeff[i] \* pow(v,n-i);

}

return sum;

}

double derivative(double v,double coeff[],int n)

{

double sum1 = 0;

for(int i=0;i<n;i++)

{

sum1 += (coeff[i]\*(n-i)) \* pow(v,n-i-1);

}

return sum1;

}

int main()

{

cout << "Enter the degree of the equation\n";

int n;

cin >> n;

double coeff[n+1];

cout << "\nEnter "<<n+1<<" co-efficients of the equation\n";

for(int i=0;i<=n;i++)

cin >> coeff[i];

cout << "\nEnter the approximate root\n";

double x0;

cin >> x0;

cout << "\nCorrection to how many decimal places\n";

double dec;

cin >> dec;

double e = 1.0 / pow(10,dec);

int p = n-1;

double xi[2];

while (1)

{

xi[0] = x0 - p \* (func(x0,arr,n)/derivative(x0,arr,n));

xi[1] = x0 - (p-1) \* (func(x0,arr,n)/derivative(x0,arr,n));

if (fabs( func(xi[0],arr,n) ) <= e)

{

cout <<fixed<<setprecision(dec)<< "The value of root is : " << xi[0];

break;

}

else if (fabs( func(xi[1],arr,n) ) <= e)

{

cout <<fixed<<setprecision(dec)<< "The value of root is : " << xi[1];

break;

}

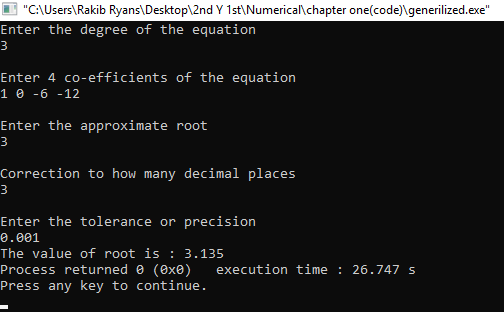
x0 = ( func(xi[0],arr,n)< func(xi[1],arr,n) )? xi[0] : xi[1];

}

return 0;

}

**Sample Input Output:**



Problem Statement : Implement Iteration method to find a root of polynomial equation.

Algorithm:

1. Start.
2. Input degree **n** and **(n+1)** co-efficients of the equation **f(x)** **= 0**.
3. Choose two real values for **a** and **b**.
4. Input the desired precision and compute error criterion **E**.
5. Compute: **k** **:=** absolute value of **max( f*′*(a)**, **f*′*(b) )**
6. Compute: **accuracy** **:=** absolute value of **[(1–k) / k ] E**.
7. Compute: **x1 := (a+b)/2** and **x2 := f(x1)**;
8. If absolute value of **(x2 – x1) >** **accuracy**, go to step 10.
9. Else **x2** is the required root , go to step 11.
10. Set : **x1 := x2** and **x2 := f(x1) ,** go to step 9.
11. Print **x2**.
12. End.

Source Code:

#include<iostream>

#include<cmath>

#include<iomanip>

using namespace std;

double func(double x,double \*coeff,int n)

{

double sum = 0;

double arr[n+1];

for(int i=1;i<=n;i++)

arr[i] = -1.0 \* (coeff[i]/coeff[0]);

for(int i=1;i<=n;i++)

sum += arr[i] \* pow(x,n-i);

sum = pow(sum,1.0/n);

return sum;

}

double derivative(double x,double \*coeff,int n)

{

double sum1 = 0;

double arr1[n+1];

for(int i=1;i<=n;i++)

arr1[i] = -1.0 \* (coeff[i]/coeff[0]);

for(int i=1;i<=n;i++)

sum1 += arr1[i] \* pow(x,n-i);

sum1 = (1.0/n) \* pow(sum1,((1.0/n)-1));

double sum2 = 0;

for(int i=1;i<=n-1;i++)

sum2 += (arr1[i]\*(n-i)) \* pow(x,n-i-1);

sum1 = sum1 \* sum2;

return sum1;

}

int main()

{

cout << "Enter the degree of the equation\n";

int n;

cin >> n;

double coeff[n+1];

cout << "\nEnter "<<n+1<<" co-efficients of the equation\n";

for(int i=0;i<=n;i++)

cin >> coeff[i];

cout << "\nEnter boundary values or terminal points\n";

double a,b;

cin >> a >> b;

cout << "\nCorrection to how many decimal places\n";

double dec;

cin >> dec;

double e = 1.0 / pow(10,dec);

double k = fabs(max(derivative(a,coeff,n),derivative(b,coeff,n)));

double accuracy = fabs(((1.0-k) / k ) \* e);

double x1 = (a+b)/2;

double x2 = func(x1,coeff,n);

while (fabs(x2-x1)>accuracy)

{

x1 = x2;

x2 = func(x1,coeff,n);

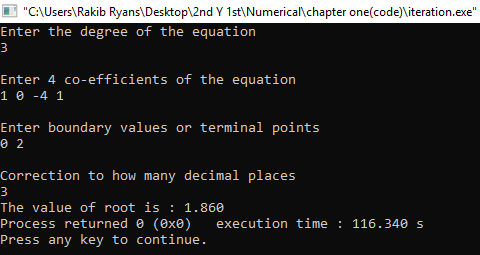
}

cout <<fixed<<setprecision(dec)<< "The value of root is : " << x2<<endl;

return 0;

}

**Sample Input Output:**



**Problem No. 7**

**Problem Name:** Use of Newtons forward difference interpulation formulae.

**Driscription:** Interpolation is the technique of estimating the value of a function for any intermediate value of the independent variable, while the process of computing the value of the function outside the given range is called extrapolation. Interpolation is the process of deriving a simple function from a set of discrete data points so that the function passes through all the given data points and and can be used to estimate data points in-between the given ones. It is necessary because in science and engineering we often need to deal with discrete experimental data. I was taught that the forward formula should be used when calculating the value of a point near x0 and the backward one when calculating near xn.

However, the interpolation polynomial is unique, so the value should be the same. using an equation we can clculate newtons forward difference interpulation :

given the set of (n+1) values , viz...(x0,y0),(x1,y1)....(xn,yn) of x and y,it is required to find yn(x) , a polynominal of the nth degree such that and yn(x) agree at the tabulated points. let the value of x be equidistant , x=x0+ (p-i)h; i=1,2,3..n x=x0+ph. p=(x-x0)/h. yn(x)=y0+pΔy0+p(p−1)/2!Δ2y0+⋯+p(p−1)(p−2)...(p−n+1)n!Δny0

**source code:**

#include<bits/stdc++.h>

using namespace std;

double x[10],y[10],d1[10],d2[10],d3[10],d4[10],d5[10],p;

double fun(void)

{

double sum;

sum=y[0]+p\*d1[0]+p\*(p-1)\*d2[0]\*(1/2.0)+p\*(p-1)\*(p-2)\*d3[0]\*(1/6.0)+p\*(p-1)\*(p-2)\*(p-3)\*d4[0]\*(1/24.0)+p\*(p-1)\*(p-2)\*(p-3)\*(p-4)\*d5[0]\*(1/120.0);

return sum;

}

int main()

{

int i,n;

double iv,dv;

cout<<"Enter the number of input set(x & y)[Maximum 6 set]:"<<endl;

cin>>n;

cout<<"Enter the value of x & y respectively:"<<endl;

for(i=0; i<n; i++)

cin>>x[i]>>y[i];

for(i=0; i<n-1; i++)

d1[i]=y[i+1]-y[i];

for(i=0; i<n-2; i++)

d2[i]=d1[i+1]-d1[i];

for(i=0; i<n-3; i++)

d3[i]=d2[i+1]-d2[i];

for(i=0; i<n-4; i++)

d4[i]=d3[i+1]-d3[i];

for(i=0; i<n-5; i++)

d5[i]=d4[i+1]-d4[i];

cout<<"Enter the desired value of x:"<<endl;

cin>>iv;

p=(iv-x[0])/abs(x[1]-x[0]);

dv=fun();

cout<<"y"<<iv<<"="<<dv<<endl;

return 0;

}

**Sample output:**

