**False position method**

#include<iostream>

#include<iomanip>

#include<math.h>

/\*

Defining equation to be solved.

Change this equation to solve another problem.

\*/

#define f(x) ((x)\*(x) - (x) - 2)

using namespace std;

int main()

{

/\* Declaring required variables \*/

float x0, x1, x, f0, f1, f, e;

int step = 1;

/\* Setting precision and writing floating point values in fixed-point notation. \*/

cout<< setprecision(6)<< fixed;

/\* Inputs \*/

up:

cout<<"Enter first guess: ";

cin>>x0;

cout<<"Enter second guess: ";

cin>>x1;

cout<<"Enter tolerable error: ";

cin>>e;

/\* Calculating Functional Value \*/

f0 = f(x0);

f1 = f(x1);

/\* Checking whether given guesses brackets the root or not. \*/

if( f0 \* f1 > 0.0)

{

cout<<"Incorrect Initial Guesses."<< endl;

goto up;

}

/\* Implementing False Position Method \*/

cout<< endl<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<< endl;

cout<<"False Position Method"<< endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<< endl;

do

{

/\* Applying False Position Method \*/

/\* x is next approximated root using False Position method \*/

x = x0 - (x0-x1) \* f0/ (f0-f1);

f = f(x);

cout<<"Iteration-"<< step<<":\t x = "<< setw(10)<< x<<" and f(x) = "<< setw(10)<< f(x)<< endl;

if( f0 \* f < 0)

{

x1 = x;

f1 = f;

}

else

{

x0 = x;

f0 = f;

}

step = step + 1;

}while(fabs(f)>e);

cout<< endl<<"Root is: "<< x<< endl;

return 0;

}

**Newton-Raphson method**

#include<iostream>

#include<iomanip>

#include<math.h>

#include<stdlib.h>

/\* Defining equation to be solved.

Change this equation to solve another problem. \*/

#define f(x) ((x)\*(x) - 3\*(x) + 2)

#define g(x) (2\*(x) - 3)

/\* Defining derivative of g(x).

As you change f(x), change this function also. \*/

using namespace std;

int main()

{

float x0, x1, f0, f1, g0, e;

int step = 1, N;

/\* Setting precision and writing floating point values in fixed-point notation. \*/

cout<< setprecision(6)<< fixed;

/\* Inputs \*/

cout<<"Enter initial guess: ";

cin>>x0;

cout<<"Enter tolerable error: ";

cin>>e;

cout<<"Enter maximum iteration: ";

cin>>N;

/\* Implementing Newton Raphson Method \*/

cout<< endl<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<< endl;

cout<<"Newton Raphson Method"<< endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<< endl;

do

{

g0 = g(x0);

f0 = f(x0);

if(g0 == 0.0)

{

cout<<"Mathematical Error.";

exit(0);

}

x1 = x0 - f0/g0;

cout<<"Iteration-"<< step<<":\t x = "<< setw(10)<< x1<<" and f(x) = "<< setw(10)<< f(x1)<< endl;

x0 = x1;

step = step+1;

if(step > N)

{

cout<<"Not Convergent.";

exit(0);

}

f1 = f(x1);

}while(fabs(f1)>e);

cout<< endl<<"Root is: "<< x1;

return 0;

}

**Secant metod**

#include<iostream>

#include<iomanip>

#include<math.h>

#include<stdlib.h>

/\* Defining equation to be solved.

Change this equation to solve another problem. \*/

#define f(x) x\*x - 4\*x - 10

using namespace std;

int main()

{

float x0, x1, x2, f0, f1, f2, e;

int step = 1, N;

/\* Setting precision and writing floating point values in fixed-point notation. \*/

cout<< setprecision(6)<< fixed;

/\* Inputs \*/

cout<<"Enter first guess: ";

cin>>x0;

cout<<"Enter second guess: ";

cin>>x1;

cout<<"Enter tolerable error: ";

cin>>e;

cout<<"Enter maximum iteration: ";

cin>>N;

/\* Implementing Secant Method \*/

cout<< endl<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<< endl;

cout<<"Secant Method"<< endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<< endl;

do

{

f0 = f(x0);

f1 = f(x1);

if(f0 == f1)

{

cout<<"Mathematical Error.";

exit(0);

}

x2 = x1 - (x1 - x0) \* f1/(f1-f0);

f2 = f(x2);

cout<<"Iteration-"<< step<<":\t x2 = "<< setw(10)<< x2<<" and f(x2) = "<< setw(10)<< f(x2)<< endl;

x0 = x1;

f0 = f1;

x1 = x2;

f1 = f2;

step = step + 1;

if(step > N)

{

cout<<"Not Convergent.";

exit(0);

}

}while(fabs(f2)>e);

cout<< endl<<"Root is: "<< x2;

return 0;

}

**Fixed point method**

#include<iostream>

#include<iomanip>

#include<math.h>

#include<stdlib.h>

/\* Define function f(x) which

is to be solved \*/

#define f(x) x\*x+x-2

/\* Write f(x) as x = g(x) and

define g(x) here \*/

#define g(x) 2\*x+1

using namespace std;

int main()

{

int step=1, N;

float x0, x1, e;

/\* Setting precision and writing floating point values in fixed-point notation. \*/

cout<< setprecision(6)<< fixed;

/\* Inputs \*/

cout<<"Enter initial guess: ";

cin>>x0;

cout<<"Enter tolerable error: ";

cin>>e;

cout<<"Enter maximum iteration: ";

cin>>N;

/\* Implementing Fixed Point Iteration \*/

cout<< endl<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<< endl;

cout<<"Fixed Point Iteration Method"<< endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<< endl;

do

{

x1 = g(x0);

cout<<"Iteration-"<< step<<":\t x1 = "<< setw(10)<< x1<<" and f(x1) = "<< setw(10)<< f(x1)<< endl;

step = step + 1;

if(step>N)

{

cout<<"Not Convergent.";

exit(0);

}

x0 = x1;

}while( fabs(f(x1)) > e);

cout<< endl<<"Root is "<< x1;

return(0);

}

**Gauss elimination method**

**Confution**

#include <iostream>

#include <iomanip>

#include <cmath>

#include <cstdlib>

#define SIZE 10

using namespace std;

int main()

{

float a[SIZE][SIZE], x[SIZE], ratio;

int i, j, k, n;

/\* Setting precision and writing floating point values in fixed-point notation. \*/

cout << setprecision(3) << fixed;

/\* Inputs \*/

/\* 1. Reading number of unknowns \*/

cout << "Enter number of unknowns: ";

cin >> n;

/\* 2. Reading Augmented Matrix \*/

cout << "Enter Coefficients of Augmented Matrix: " << endl;

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

{

for (j = 1; j <= n + 1; j++)

{

cout << "a[" << i << "][" << j << "]= ";

cin >> a[i][j];

}

}

/\* Applying Gauss Elimination \*/

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

{

if (a[i][i] == 0.0)

{

cout << "Mathematical Error!";

exit(0);

}

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

{

ratio = a[j][i] / a[i][i];

for (k = 1; k <= n + 1; k++)

{

a[j][k] = a[j][k] - ratio \* a[i][k];

}

}

// Print current state of the matrix

cout << "Matrix after iteration " << i << ":" << endl;

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

{

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

{

cout << setw(8) << a[r][c] << " ";

}

cout << endl;

}

cout << endl;

}

/\* Obtaining Solution by Back Substitution Method \*/

x[n] = a[n][n + 1] / a[n][n];

for (i = n - 1; i >= 1; i--)

{

x[i] = a[i][n + 1];

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

{

x[i] = x[i] - a[i][j] \* x[j];

}

x[i] = x[i] / a[i][i];

}

/\* Displaying Solution \*/

cout << endl

<< "Solution: " << endl;

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

{

cout << "x[" << i << "] = " << x[i] << endl;

}

return 0;

}

**Gauss elimination with pivorting**

#include <iostream>

#include <iomanip>

#include <cmath>

#define SIZE 10

using namespace std;

int main()

{

float a[SIZE][SIZE + 1], x[SIZE];

int i, j, k, n, iter = 0;

/\* Setting precision and writing floating point values in fixed-point notation. \*/

cout << setprecision(3) << fixed;

/\* Inputs \*/

/\* 1. Reading number of unknowns \*/

cout << "Enter number of unknowns: ";

cin >> n;

/\* 2. Reading Augmented Matrix \*/

cout << "Enter Coefficients of Augmented Matrix: " << endl;

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

{

for (j = 1; j <= n + 1; j++)

{

cout << "a[" << i << "][" << j << "]= ";

cin >> a[i][j];

}

}

/\* Applying Gauss Elimination with Pivoting \*/

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

{

// Early Pivoting

int pivot\_row = i;

float max\_val = abs(a[i][i]);

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

{

if (abs(a[j][i]) > max\_val)

{

max\_val = abs(a[j][i]);

pivot\_row = j;

}

}

if (pivot\_row != i)

{

// Swap rows

for (k = 1; k <= n + 1; k++)

{

swap(a[i][k], a[pivot\_row][k]);

}

}

if (a[i][i] == 0.0)

{

cout << "Mathematical Error!";

exit(0);

}

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

{

float ratio = a[j][i] / a[i][i];

for (k = 1; k <= n + 1; k++)

{

a[j][k] = a[j][k] - ratio \* a[i][k];

}

}

// Print intermediate steps

cout << "Iteration-" << ++iter << ":\n";

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

{

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

{

cout << setw(8) << a[r][c] << " ";

}

cout << endl;

}

cout << endl;

}

/\* Back Substitution Method \*/

x[n] = a[n][n + 1] / a[n][n];

for (i = n - 1; i >= 1; i--)

{

float sum = 0;

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

{

sum += a[i][j] \* x[j];

}

x[i] = (a[i][n + 1] - sum) / a[i][i];

}

/\* Displaying Solution \*/

cout << endl

<< "Solution: " << endl;

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

{

cout << "x[" << i << "] = " << x[i] << endl;

}

return 0;

}

**Jecobi iteration method**

#include<iostream>

#include<iomanip>

#include<math.h>

/\* Arrange systems of linear

equations to be solved in

diagonally dominant form

and form equation for each

unknown and define here

\*/

/\* In this example we are solving

3x + 20y - z = -18

2x - 3y + 20z = 25

20x + y - 2z = 17

\*/

/\* Arranging given system of linear

equations in diagonally dominant

form:

20x + y - 2z = 17

3x + 20y -z = -18

2x - 3y + 20z = 25

\*/

/\* Equations:

x = (17-y+2z)/20

y = (-18-3x+z)/20

z = (25-2x+3y)/20

\*/

/\* Defining function \*/

#define f1(x,y,z) (5-y-z)/2

#define f2(x,y,z) (15-3\*x-2\*z)/5

#define f3(x,y,z) (8-2\*x-y)/4

using namespace std;

/\* Main function \*/

int main()

{

float x0=0, y0=0, z0=0, x1, y1, z1, e1, e2, e3, e;

int step=1;

/\* Setting precision and writing floating point values in fixed-point notation. \*/

cout<< setprecision(6)<< fixed;

/\* Input \*/

/\* Reading tolerable error \*/

cout<<"Enter tolerable error: ";

cin>>e;

cout<< endl<<"Count\tx\t\ty\t\tz"<< endl;

do

{

/\* Calculation \*/

x1 = f1(x0,y0,z0);

y1 = f2(x0,y0,z0);

z1 = f3(x0,y0,z0);

cout<< step<<"\t"<< x1<<"\t"<< y1<<"\t"<< z1<< endl;

/\* Error \*/

e1 = fabs(x0-x1);

e2 = fabs(y0-y1);

e3 = fabs(z0-z1);

step++;

/\* Set value for next iteration \*/

x0 = x1;

y0 = y1;

z0 = z1;

}while(e1>e && e2>e && e3>e);

cout<< endl<<"Solution: x = "<< x1<<", y = "<< y1<<" and z = "<< z1;

return 0;

}

**Gauss seidul method**

#include<iostream>

#include<iomanip>

#include<math.h>

/\* Arrange systems of linear

equations to be solved in

diagonally dominant form

and form equation for each

unknown and define here

\*/

/\* In this example we are solving

3x + 20y - z = -18

2x - 3y + 20z = 25

20x + y - 2z = 17

\*/

/\* Arranging given system of linear

equations in diagonally dominant

form:

20x + y - 2z = 17

3x + 20y -z = -18

2x - 3y + 20z = 25

\*/

/\* Equations:

x = (17-y+2z)/20

y = (-18-3x+z)/20

z = (25-2x+3y)/20

\*/

/\* Defining function \*/

#define f1(x,y,z) (5-y-z)/2

#define f2(x,y,z) (15-3\*x-2\*z)/5

#define f3(x,y,z) (8-2\*x-y)/4

using namespace std;

/\* Main function \*/

int main()

{

float x0=0, y0=0, z0=0, x1, y1, z1, e1, e2, e3, e;

int step=1;

/\* Setting precision and writing floating point values in fixed-point notation. \*/

cout<< setprecision(6)<< fixed;

/\* Input \*/

/\* Reading tolerable error \*/

cout<<"Enter tolerable error: ";

cin>>e;

cout<< endl<<"Count\tx\t\ty\t\tz"<< endl;

do

{

/\* Calculation \*/

x1 = f1(x0,y0,z0);

y1 = f2(x1,y0,z0);

z1 = f3(x1,y1,z0);

cout<< step<<"\t"<< x1<<"\t"<< y1<<"\t"<< z1<< endl;

/\* Error \*/

e1 = fabs(x0-x1);

e2 = fabs(y0-y1);

e3 = fabs(z0-z1);

step++;

/\* Set value for next iteration \*/

x0 = x1;

y0 = y1;

z0 = z1;

}while(e1>e && e2>e && e3>e);

cout<< endl<<"Solution: x = "<< x1<<", y = "<< y1<<" and z = "<< z1;

return 0;

}

**Euler’s method**

#include <iostream>

/\* Defining the ordinary differential equation to be solved \*/

/\* In this example, we are solving dy/dx = 3x^2 + 1 \*/

#define f(x, y) (3 \* x \* x + 1)

using namespace std;

int main()

{

float x0, y0, xn, h, yn;

cout << "Enter Initial Condition" << endl;

cout << "x0 = ";

cin >> x0;

cout << "y0 = ";

cin >> y0;

cout << "Enter calculation point xn = ";

cin >> xn;

cout << "Enter step size h = ";

cin >> h;

/\* Euler's Method \*/

cout << "x\t\ty" << endl;

cout << "------------------" << endl;

while (x0 < xn)

{

cout << x0 << "\t" << y0 << endl;

yn = y0 + h \* f(x0, y0);

y0 = yn;

x0 = x0 + h;

}

/\* Displaying the result \*/

cout << "Estimated value of y(" << xn << ") = " << yn;

return 0;

}

**Heun’s method**

#include <iostream>

/\* Defining the ordinary differential equation to be solved \*/

/\* In this example, we are solving dy/dx = (2\*y)/x \*/

#define f(x, y) (2 \* y / x)

using namespace std;

int main()

{

float x0, y0, xn, h, yn, slope1, slope2;

cout << "Enter Initial Condition" << endl;

cout << "x0 = ";

cin >> x0;

cout << "y0 = ";

cin >> y0;

cout << "Enter calculation point xn = ";

cin >> xn;

cout << "Enter step size h = ";

cin >> h;

/\* Heun's Method \*/

cout << "x\t\ty" << endl;

cout << "------------------" << endl;

while (x0 < xn)

{

cout << x0 << "\t" << y0 << endl;

slope1 = f(x0, y0);

slope2 = f(x0 + h, y0 + h \* slope1);

yn = y0 + (h / 2) \* (slope1 + slope2);

y0 = yn;

x0 = x0 + h;

}

/\* Displaying the result \*/

cout << "Estimated value of y(" << xn << ") = " << yn;

return 0;

}

**Baisection method**

#include <iostream>

#include <cmath>

#define EPSILON 0.00001

using namespace std;

/\* Function to calculate the value of the equation \*/

double equation(double x)

{

return x \* x - 4 \* x - 10;

}

/\* Bisection method \*/

double bisection(double a, double b)

{

if (equation(a) \* equation(b) >= 0)

{

cout << "Invalid initial guesses. Please try again with different values." << endl;

return 0.0;

}

double c;

int step = 1;

cout << "Step\t\ta\t\tb\t\tc\t\tRoot" << endl;

cout << "---------------------------------------------" << endl;

while ((b - a) >= EPSILON)

{

c = (a + b) / 2;

cout << step << "\t\t" << a << "\t\t" << b << "\t\t" << c << "\t\t";

if (equation(c) == 0.0)

return c;

if (equation(c) \* equation(a) < 0)

b = c;

else

a = c;

cout << (a + b) / 2 << endl;

step++;

}

return (a + b) / 2;

}

int main()

{

double a, b;

cout << "Enter initial guesses (a and b) where the root lies between: ";

cin >> a >> b;

double root = bisection(a, b);

cout << "Approximate root of the equation: " << root << endl;

return 0;

}

**Gauss Jordan method**

#include <iostream>

#include <iomanip>

#include <cmath>

#include <cstdlib>

#define SIZE 10

using namespace std;

int main()

{

float a[SIZE][SIZE], x[SIZE], ratio;

int i, j, k, n;

/\* Setting precision and writing floating point values in fixed-point notation. \*/

cout << setprecision(3) << fixed;

/\* Inputs \*/

/\* 1. Reading number of unknowns \*/

cout << "Enter number of unknowns: ";

cin >> n;

/\* 2. Reading Augmented Matrix \*/

cout << "Enter Coefficients of Augmented Matrix: " << endl;

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

{

for (j = 0; j < n + 1; j++)

{

cout << "a[" << i + 1 << "][" << j + 1 << "] = ";

cin >> a[i][j];

}

}

/\* Displaying Augmented Matrix before Gauss-Jordan elimination \*/

cout << "Augmented Matrix before Gauss-Jordan elimination:" << endl;

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

{

for (j = 0; j < n + 1; j++)

{

cout << setw(10) << a[i][j] << " ";

}

cout << endl;

}

/\* Applying Gauss-Jordan elimination \*/

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

{

if (a[i][i] == 0.0)

{

cout << "Mathematical Error!";

exit(0);

}

/\* Perform row operations to make the diagonal element equal to 1 \*/

ratio = a[i][i];

for (j = 0; j < n + 1; j++)

{

a[i][j] /= ratio;

}

/\* Perform row operations to make other elements in the column equal to 0 \*/

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

{

if (k != i)

{

ratio = a[k][i];

for (j = 0; j < n + 1; j++)

{

a[k][j] -= ratio \* a[i][j];

}

}

}

/\* Print the augmented matrix at each step \*/

cout << endl

<< "Step " << i + 1 << ":" << endl;

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

{

for (j = 0; j < n + 1; j++)

{

cout << setw(10) << a[k][j] << " ";

}

cout << endl;

}

}

/\* Obtaining Solution \*/

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

{

x[i] = a[i][n];

}

/\* Displaying Solution \*/

cout << endl

<< "Solution:" << endl;

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

{

cout << "x[" << i + 1 << "] = " << x[i] << endl;

}

return 0;

}