Contents

1 Problem 01: Bisection Method

Objective

To find the root of a nonlinear equation $f(x) = 0$ by successive halving of an interval.

Theory

The Bisection Method is based on the Intermediate Value Theorem. If $f(a)$ and $f(b)$ have opposite signs, then there exists at least one root in $[a, b]$. The interval is halved repeatedly until the approximate root is obtained within tolerance ϵ. It is simple and guarantees convergence but converges slowly.

Formula

$$x_n = \frac{a+b}{2}$$

If $f(a)f(x_n) < 0 \Rightarrow b = x_n$, else $a = x_n$.

Implementation in C++

```
#include<bits/stdc++.h>
using namespace std;

#define f(x) pow(x,3)-2*x-5
int main()
{
    float x0,x1;
    cin >> x0 >> x1;
    float e = 0.001;
    float f0,f1,f2,x2;
    int count=0;
    do {
        f0 = f(x0);
        f1 = f(x1);
        x2 = (x0+x1)/2;
        f2 = f(x2);
        if(f2>0) x1=x2;
        else x0=x2;
        count++;
        cout<<count<<" "<<"x: "<<x2<<" "<<"f(x): "<<f2<<"\n";
    }
    while(abs(f2)>e && count<120);
}
```

2 Problem 02: Newton–Raphson Method

Objective

To approximate the root of a nonlinear equation using tangent-line approximation.

Theory

This method starts with an initial guess x_0 and uses the slope of the tangent to approximate the root. It converges quickly if x_0 is close to the actual root, but fails if $f'(x) = 0$ near the root.

Formula

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Implementation in C++

```
#include<bits/stdc++.h>
using namespace std;

#define f(x) pow(x,3)-3*x-5
#define df(x) 3*pow(x,2)-3
int main()
{
    float x0;
    cin >> x0;
    float e = 0.001;
    float fval,dfval,x,f1;
    int count=0;
    do {
        fval = f(x0);
        dfval = df(x0);
        x = x0-(fval/dfval);
        f1 = f(x);
        x0=x;
        count++;
        cout << count <<" X: "<<x << " f(x): "<< f1 << endl;
    }
    while(abs(f1)>e);
    cout << "Root is : "<< x << endl;
}
```

3 Problem 03: Newton Forward Interpolation

Objective

To interpolate values of a function near the beginning of tabulated data with equal spacing.

Theory

Uses a forward difference table constructed from tabulated values.

Formula

$$f(x) = y_0 + u\Delta y_0 + \frac{u(u-1)}{2!}\Delta^2 y_0 + \frac{u(u-1)(u-2)}{3!}\Delta^3 y_0 + \cdots$$

where $u = \frac{x-x_0}{h}$

Implementation in C++

```
#include<bits/stdc++.h>
using namespace std;

#define order 4
#define MAXN 100
int main()
{
    int n;
    cin >> n;
    float x[MAXN+1],y[MAXN+1];
    for(int i=0; i<=n; i++) cin >> x[i] >> y[i];

    float value;
    cin >> value;

    float diff[MAXN+1][order+1];
    for(int i=0; i<=n-1; i++) diff[i][1] = y[i+1]-y[i];

    for(int j=2; j<=order; j++) {
        for(int i=0; i<=n-j; i++) {
            diff[i][j] = diff[i+1][j-1]-diff[i][j-1];
        }
    }
    int idx = 0;
    while(x[idx]<value) idx++;
    idx--;

    float h = x[1]-x[0];
    float u = (value-x[idx])/h;
```

```
    float y_value = y[idx];
    float fact_u = 1.0,multi = 1.0;
    for(int i=1; i<=order; i++) {
        fact_u *= u-i+1;
        multi *= i;
        y_value += (fact_u/multi)*diff[idx][i];
    }
    cout << y_value << endl;
}
```

4 Problem 04: Newton Backward Interpolation

Objective

To interpolate values near the end of tabulated data with equal spacing.

Theory

Uses a backward difference table, starting from the last tabulated value.

Formula

$$f(x) = y_n + u\nabla y_n + \frac{u(u+1)}{2!}\nabla^2 y_n + \frac{u(u+1)(u+2)}{3!}\nabla^3 y_n + \cdots$$

where $u = \frac{x-x_n}{h}$

Implementation in C++

```
#include<bits/stdc++.h>
using namespace std;

#define order 4
#define MAXN 100
int main()
{
    int n;
    cin >> n;
    float x[MAXN+1],y[MAXN+1];
    for(int i=1; i<=n; i++) cin >> x[i] >> y[i];

    float value;
    cin >> value;

    float diff[MAXN+1][order+1];
    for(int i=n; i>=0; i--) diff[i][1] = y[i]-y[i-1];

    for(int j=2; j<=order; j++) {
        for(int i=n; i>j; i--) {
            diff[i][j] = diff[i][j-1]-diff[i-1][j-1];
        }
    }
    int idx = n;
    while(x[idx]>value) idx--;
    idx++;

    float h = x[2]-x[1];
    float u = (value-x[idx])/h;

    float y_value = y[idx];
```

```
    float fact_u = 1.0,multi = 1.0;
    for(int i=1; i<=order; i++) {
        fact_u *= u+i-1;
        multi *= i;
        y_value += (fact_u/multi)*diff[idx][i];
    }
    cout << y_value << endl;
}
```

5 Problem 05: Gauss–Seidel Method

Objective

To solve a system of linear equations iteratively.

Theory

This method updates each variable immediately after computation, using it in subsequent calculations of the same iteration. It converges faster than Jacobi when the matrix is diagonally dominant.

Formula

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)} \right)$$

Implementation in C++

```
#include<bits/stdc++.h>
using namespace std;

#define f1(x,y,z)  (7+y)/2
#define f2(x,y,z)  (1+x+z)/2
#define f3(x,y,z)  (1+y)/2
int main()
{
    float e;
    cin >> e;
    float x0=0,y0=0,z0=0,x1,y1,z1,e1,e2,e3;
    int step=1;
    do {
        x1 = f1(x0,y0,z0);
        y1 = f2(x1,y0,z0);
        z1 = f3(x1,y1,z0);

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

        e1 = abs(x0-x1);
        e2 = abs(y0-y1);
        e3 = abs(z0-z1);
        step++;
        x0 = x1; y0 = y1; z0 = z1;
    }
    while(e1>e && e2>e && e3>e);

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

6 Problem 06: Jacobi Method

Objective

To solve a system of linear equations iteratively.

Theory

Each variable is computed using only the values from the previous iteration. It converges slower than Gauss–Seidel but is easier to implement.

Formula

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right)$$

Implementation in C++

```
#include<bits/stdc++.h>
using namespace std;

#define f1(x,y,z) (85-6*y+z)/27
#define f2(x,y,z) (72-6*x-3*z)/15
#define f3(x,y,z) (110-x-y)/54
int main()
{
    float e;
    cin >> e;
    float x0=0,y0=0,z0=0,x1,y1,z1,e1,e2,e3;
    int step=1;
    do {
        x1 = f1(x0,y0,z0);
        y1 = f2(x0,y0,z0);
        z1 = f3(x0,y0,z0);

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

        e1 = abs(x0-x1);
        e2 = abs(y0-y1);
        e3 = abs(z0-z1);
        step++;
        x0 = x1; y0 = y1; z0 = z1;
    }
    while(e1>e && e2>e && e3>e);

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

7 Problem 07: Trapezoidal Rule

Objective

To approximate the definite integral of a function.

Theory

Approximates the area under a curve using trapezoids. Accuracy improves with smaller intervals.

Formula

$$I \approx \frac{h}{2}\left[f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n)\right], \quad h = \frac{b-a}{n}$$

Implementation in C++

```
#include<bits/stdc++.h>
using namespace std;

#define f(x) 3*exp(x)*sin(x)
int main()
{
    double lower,upper;
    cin >> lower >> upper;
    double interval;
    cin >> interval;

    double h = (upper-lower)/interval;
    double ans = f(lower)+f(upper);
    ans = ans/2;
    for(double i=1; i<interval; i++) {
        double x = lower + i*h;
        ans += f(x);
    }
    cout << ans*h << endl;
}
```

8 Problem 08: Simpson's 1/3 Rule

Objective

To approximate definite integrals using quadratic interpolation.

Theory

Works with an even number of intervals by fitting parabolas.

Formula

$$I \approx \frac{h}{3}\left[f(x_0) + 4\sum_{\text{odd } i} f(x_i) + 2\sum_{\text{even } i} f(x_i) + f(x_n)\right]$$

where n must be even.

Implementation in C++

```
#include<bits/stdc++.h>
using namespace std;

#define f(x) exp(x)
int main()
{
    double lower,upper;
    cin >> lower >> upper;
    double interval;
    cin >> interval;

    double h = (upper-lower)/interval;
    double ans = f(lower)+f(upper);
    for(double i=1; i<interval; i++) {
        double x = lower + i*h;
        if((int)i%2!=0) ans += 4*f(x);
        else ans += 2*f(x);
    }
    cout << ans*h/3 << endl;
}
```

9 Problem 09: Simpson's 3/8 Rule

Objective

To approximate definite integrals using cubic interpolation.

Theory

Fitting a cubic polynomial through four points gives more accuracy.

Formula

$$I \approx \frac{3h}{8} [f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)]$$

Implementation in C++

```
#include<bits/stdc++.h>
using namespace std;

#define f(x) exp(x)
int main()
{
    double lower,upper;
    cin >> lower >> upper;
    double interval;
    cin >> interval;

    double h = (upper-lower)/interval;
    double ans = f(lower)+f(upper);
    for(double i=1; i<interval; i++) {
        double x = lower + i*h;
        if((int)i%3!=0) ans += 3*f(x);
        else ans += 2*f(x);
    }
    cout << ans*h*3/8 << endl;
}
```

10 Problem 10: Euler's Modified Method

Objective

To solve first-order ODEs with improved accuracy over simple Euler's method.

Theory

Computes slopes at both the beginning and end of the interval, and averages them.

Formula

$$y_{n+1} = y_n + \frac{h}{2}\left[f(x_n, y_n) + f(x_{n+1}, y_n + hf(x_n, y_n))\right]$$

Implementation in C++

```
#include <bits/stdc++.h>
using namespace std;

#define f(x,y) (x + y)

int main() {
    double x, y, h, xn;
    int n;

    cin >> x >> y >> h >> xn;
    n = (xn - x) / h;

    for (int i = 0; i < n; i++) {
        double k1 = f(x, y);
        double k2 = f(x + h, y + h * k1);
        y = y + (h / 2.0) * (k1 + k2);
        x = x + h;
        cout << "x = " << x << "\t y = " << y << endl;
    }

    cout << "Approximate solution at x = " << xn << " is y = " << y << endl;
    return 0;
}
```

11 Problem 11: Runge–Kutta Method (4th Order)

Objective

To solve ODEs numerically with high accuracy.

Theory

Uses four slope evaluations per step and combines them with weighted average.

Formula

$$
\begin{aligned}
k_1 &= hf(x_n, y_n) \\
k_2 &= hf\left(x_n + \tfrac{h}{2}, y_n + \tfrac{k_1}{2}\right) \\
k_3 &= hf\left(x_n + \tfrac{h}{2}, y_n + \tfrac{k_2}{2}\right) \\
k_4 &= hf(x_n + h, y_n + k_3) \\
y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)
\end{aligned}
$$

Implementation in C++

```
#include<bits/stdc++.h>
using namespace std;

#define f(x,y) (3*x+y/2)
int main()
{
    float x0,y0;
    cin >> x0 >> y0;
    float xn;
    cin >> xn;
    float interval;
    cin >> interval;
    float h = (xn-x0)/interval;
    float yn,k1,k2,k3,k4,k;
    for(int i=1; i<=interval; i++) {
        k1 = h*(f(x0,y0));
        k2 = h*(f((x0+h/2),(y0+k1/2)));
        k3 = h*(f((x0+h/2),(y0+k2/2)));
        k4 = h*(f((x0+h),(y0+k3)));
        k = (k1+2*k2+2*k3+k4)/6;
        yn = y0+k;
        cout << x0 << " "<<y0 << " "<<yn << endl;
        y0=yn;
        x0 = x0+i*h;
    }
    cout<<"Value of y at x = "<< xn<< " is " << yn << endl;
}
```