[1]#include <stdio.h>

#include <math.h>

int main() {

double actual\_value, measured\_value;

printf("Enter the actual value of u: ");

scanf("%lf", &actual\_value);

printf("Enter the measured value of u: ");

scanf("%lf", &measured\_value);

double absolute\_error = fabs(measured\_value - actual\_value);

double relative\_error;

if (actual\_value != 0.0)

relative\_error = fabs((measured\_value - actual\_value) / actual\_value);

else

relative\_error = INFINITY;

double percentage\_error;

if (actual\_value != 0.0)

percentage\_error = relative\_error \* 100.0;

else

percentage\_error = INFINITY;

printf("Actual value of u: %.10lf\n", actual\_value);

printf("Measured value of u: %.10lf\n", measured\_value);

printf("Absolute error: %.10lf\n", absolute\_error);

printf("Relative error: %.10lf\n", relative\_error);

printf("Percentage error: %.10lf%%\n", percentage\_error);

return 0;

}

[2] #include<stdio.h>

#include<math.h>

#define EPSILON 0.00001

double f(double x){

return (x\*x\*x)-(2\*x)-5;

}

void bisection(double a, double b){

if(f(a)\*f(b)>=0){

printf("Invalid Interval. No root founf\n");

return;

}

int iteration = 0;

double c;

do{

c=(a+b)/2;

printf("iteration %d: interval[%lf %lf],Root:%lf\n", iteration,a,b,c);

if (f(c) == 0.0) {

printf("Found exact root: %lf\n", c);

break;

} else if (f(c) \* f(a) < 0) {

b = c;

} else {

a = c;

}

iteration++;

} while (fabs(b - a) >= EPSILON);

printf("Final approx root: %lf\n", c);

}

int main() {

double a, b;

printf("Enter the interval [a, b]: ");

scanf("%lf %lf", &a, &b);

bisection(a, b);

return 0;

}

[3] #include <stdio.h>

#include <math.h>

#define EPSILON 0.00001

double f(double x) {

return (x \* x \* x) - (2 \* x) - 5;

}

double regulaFalsi(double a, double b) {

double c;

int iteration = 0;

do {

c = (a \* f(b) - b \* f(a)) / (f(b) - f(a));

printf("Iteration %d: Approximation = %lf\n", iteration, c);

if (f(c) == 0.0) {

printf("Found exact root: %lf\n", c);

break;

} else if (f(c) \* f(a) < 0) {

b = c;

} else {

a = c;

}

iteration++;

} while (fabs(f(c)) >= EPSILON);

printf("Final approx root: %lf\n", c);

return c;

}

int main() {

double a, b;

printf("Enter the initial interval [a, b]: ");

scanf("%lf %lf", &a, &b);

if (f(a) \* f(b) >= 0) {

printf("Invalid interval. No root found.\n");

return 1;

}

regulaFalsi(a, b);

return 0;

}

[4] #include <stdio.h>

#include <math.h>

#define EPSILON 0.00001

double f(double x) {

return (x \* x \* x) - (2 \* x) - 5;}

double f\_prime(double x) {

return 3 \* (x \* x) - 2;}

double newtonRaphson(double initial\_guess) {

double x = initial\_guess;

int iteration = 0;

do {

double delta\_x = f(x) / f\_prime(x);

x = x - delta\_x;

printf("Iteration %d: Approximation = %lf\n", iteration, x);

if (f(x) == 0.0) {

printf("Found exact root: %lf\n", x);

break;

}

iteration++;

} while (fabs(f(x)) >= EPSILON);

printf("Final approx root: %lf\n", x);

return x;}

int main() {

double initial\_guess;

printf("Enter the initial guess: ");

scanf("%lf", &initial\_guess);

newtonRaphson(initial\_guess);

return 0;

}

[5] #include <stdio.h>

#include <math.h>

// Function to calculate the equation cos(x) - 3x + 1

double equation(double x) {

return cos(x) - 3 \* x + 1;}

int main() {

double a, b, c, epsilon;

printf("Enter the initial interval [a, b]: ");

scanf("%lf %lf", &a, &b);

printf("Enter the tolerance (epsilon): ");

scanf("%lf", &epsilon);

int iterations = 0;

while ((b - a) >= epsilon) {

c = (a + b) / 2; // Calculate midpoint

if (equation(c) == 0.0) {

break; // Found the root exactly

} else if (equation(a) \* equation(c) < 0) {

b = c;

} else {

a = c;

}

iterations++;

}

printf("Root found at x = %lf after %d iterations\n", c, iterations);

return 0;

}

[6] #include <stdio.h>

#include <math.h>

// Function to calculate the equation cos(x) - 3x + 1

double equation(double x) {

return cos(x) - 3 \* x + 1;

}

int main() {

double x0, x1, x2, epsilon;

printf("Enter initial guesses (x0 and x1): ");

scanf("%lf %lf", &x0, &x1);

printf("Enter the tolerance (epsilon): ");

scanf("%lf", &epsilon);

int iterations = 0;

do {

x2 = x1 - (equation(x1) \* (x1 - x0)) / (equation(x1) - equation(x0));

iterations++;

if (fabs(x2 - x1) < epsilon) {

printf("Root found at x = %lf after %d iterations\n", x2, iterations);

break;

}

x0 = x1;

x1 = x2;

} while (iterations < 1000);

if (iterations >= 1000) {

printf("Root not found within 1000 iterations.\n");

}

return 0;

}

[13] #include <stdio.h>

double func(double x) {

// Define your function here

return x \* x;

}double trapezoidalRule(double a, double b, int n) {

double h = (b - a) / n;

double result = (func(a) + func(b)) / 2.0;

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

double x = a + i \* h;

result += func(x); }

result \*= h;

return result;}

int main() {

double a, b;

int n;

printf("Enter the lower limit of integration (a): ");

scanf("%lf", &a);

printf("Enter the upper limit of integration (b):");

scanf("%lf", &b);

printf("Enter the number of subintervals (n): ");

scanf("%d", &n);

double result = trapezoidalRule(a, b, n);

printf("Numerical integration using Trapezoidal Rule: %lf\n", result);

return 0;

}

[7] #include <stdio.h>

#define N 10 // Maximum number of equations/variables

void gaussElimination(float matrix[N][N + 1], int n) {

int i, j, k;

float ratio, temp;

printf("Upper Triangular Matrix:\n");

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

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

printf("%.2f\t", matrix[i][j]);

printf("\n");

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

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

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

matrix[j][k] -= ratio \* matrix[i][k];}}

printf("\nSolution:\n");

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

printf("x%d = %.2f\n", i + 1, matrix[i][n]);}

int main() {

int n;

printf("Enter the number of equations/variables (up to %d): ", N);

scanf("%d", &n);

float matrix[N][N + 1];

printf("Enter the augmented matrix coefficients (row-wise):\n");

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

printf("Enter coefficients for equation %d: ", i + 1);

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

scanf("%f", &matrix[i][j]);}

gaussElimination(matrix, n);

return 0;

}

[8] #include <stdio.h>

#define MAX\_SIZE 10

void gaussJordan(double mat[MAX\_SIZE][MAX\_SIZE + 1], int n) {

int i, j, k;

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

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

if (i != j) {

double ratio = mat[j][i] / mat[i][i];

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

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

}

}

}

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

double divisor = mat[i][i];

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

mat[i][j] /= divisor;}}

int main() {

int n;

printf("Enter the number of variables in the system: ");

scanf("%d", &n);

double mat[MAX\_SIZE][MAX\_SIZE + 1];

printf("Enter the augmented matrix (coefficients and constants):\n");

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

for (int j = 0; j <= n; j++) {

printf("Enter element mat[%d][%d]: ", i, j);

scanf("%lf", &mat[i][j]);}}

gaussJordan(mat, n);

printf("The solutions are:\n");

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

printf("x%d = %.2lf\n", i + 1, mat[i][n]);

return 0;

}

[9] #include <stdio.h>

#include <math.h>

#define MAX\_SIZE 10

#define EPSILON 0.00001

void gaussSeidel(double mat[MAX\_SIZE][MAX\_SIZE + 1], int n) {

double x[MAX\_SIZE];

double new\_x[MAX\_SIZE];

int i, j, k;

double error;

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

x[i] = 0; // Initialize the variables

do {

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

new\_x[i] = mat[i][n];

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

if (j != i)

new\_x[i] -= mat[i][j] \* x[j];

}

new\_x[i] /= mat[i][i];

}

error = 0;

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

error += fabs(new\_x[i] - x[i]);

x[i] = new\_x[i];

}

} while (error > EPSILON);

printf("The solutions are:\n");

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

printf("x%d = %.5lf\n", i + 1, x[i]);

}

int main() {

int n;

printf("Enter the number of variables in the system: ");

scanf("%d", &n);

double mat[MAX\_SIZE][MAX\_SIZE + 1];

printf("Enter the augmented matrix (coefficients and constants):\n");

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

for (int j = 0; j <= n; j++) {

printf("Enter element mat[%d][%d]: ", i, j);

scanf("%lf", &mat[i][j]);}}

gaussSeidel(mat, n);

return 0;}

[10]#include <stdio.h>

int factorial(int n) {

if (n == 0 || n == 1) {

return 1;

} else {

return n \* factorial(n - 1);

}

}

double newtonForwardInterpolation(int n, double x[], double y[], double value) {

double result = 0.0;

double p = (value - x[0]) / (x[1] - x[0]);

double term;

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

term = p;

for (int j = 0; j < i; j++) {

term \*= (p - j);

}

term = term \* y[i] / factorial(i);

result += term;

}

return result;

}

int main() {

int n;

printf("Enter the number of data points: ");

scanf("%d", &n);

double x[n], y[n];

printf("Enter the data points:\n");

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

printf("x[%d] = ", i);

scanf("%lf", &x[i]);

printf("y[%d] = ", i);

scanf("%lf", &y[i]);

}

double value;

printf("Enter the value to interpolate: ");

scanf("%lf", &value);

double result = newtonForwardInterpolation(n, x, y, value);

printf("Interpolated value at x = %lf is %lf\n", value, result);

return 0;

}

[11] #include <stdio.h>

int factorial(int n) {

if (n == 0 || n == 1) {

return 1;

} else {

return n \* factorial(n - 1);

}

}

double newtonBackwardInterpolation(int n, double x[], double y[], double value) {

double result = 0.0;

double p = (value - x[n - 1]) / (x[1] - x[0]);

double term;

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

term = p;

for (int j = 0; j < i; j++) {

term \*= (p + j + 1);

}

term = term \* y[n - i - 1] / factorial(i);

result += term;

}

return result;

}

int main() {

int n;

printf("Enter the number of data points: ");

scanf("%d", &n);

double x[n], y[n];

printf("Enter the data points:\n");

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

printf("x[%d] = ", i);

scanf("%lf", &x[i]);

printf("y[%d] = ", i);

scanf("%lf", &y[i]);

}

double value;

printf("Enter the value to interpolate: ");

scanf("%lf", &value);

double result = newtonBackwardInterpolation(n, x, y, value);

printf("Interpolated value at x = %lf is %lf\n", value, result);

return 0;

}

[12] #include <stdio.h>

double lagrangeInterpolation(int n, double x[], double y[], double value) {

double result = 0.0;

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

double term = y[i];

for (int j = 0; j < n; j++) {

if (j != i) {

term = term \* (value - x[j]) / (x[i] - x[j]);

} }

result += term;

} return result;

}

int main() {

int n;

printf("Enter the number of data points: ");

scanf("%d", &n);

double x[n], y[n];

printf("Enter the data points:\n");

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

printf("x[%d] = ", i);

scanf("%lf", &x[i]);

printf("y[%d] = ", i);

scanf("%lf", &y[i]); }

double value;

printf("Enter the value to interpolate: ");

scanf("%lf", &value);

double result = lagrangeInterpolation(n, x, y, value);

printf("Interpolated value at x = %lf is %lf\n", value, result);

return 0;

}

[14]#include <stdio.h>

double func(double x) {

return x \* x;

}double simpsonsRule(double a, double b, int n) {

double h = (b - a) / n;

double result = func(a) + func(b);

for (int i = 1; i < n; i += 2) {

double x = a + i \* h;

result += 4 \* func(x);

}

for (int i = 2; i < n - 1; i += 2) {

double x = a + i \* h;

result += 2 \* func(x);

}

result \*= h / 3.0;

return result;

}

int main() {

double a, b;

int n;

printf("Enter the lower limit of integration (a): ");

scanf("%lf", &a);

printf("Enter the upper limit of integration (b):");

scanf("%lf", &b);

printf("Enter the number of subintervals (n, even): ");

scanf("%d", &n);

if (n % 2 != 0) {

printf("Error: Number of subintervals should be even for Simpson's 1/3 Rule.\n");

return 1;

} double result = simpsonsRule(a, b, n);

printf("Numerical integration using Simpson's 1/3 Rule: %lf\n", result);

return 0;

}

[15] #include <stdio.h>

double func(double x) {

// Define your function here

return x \* x;

}

double simpsons3by8Rule(double a, double b, int n) {

double h = (b - a) / n;

double result = func(a) + func(b);

for (int i = 1; i < n; i += 3) {

double x = a + i \* h;

result += 3 \* func(x);

}

for (int i = 2; i < n; i += 3) {

double x = a + i \* h;

result += 3 \* func(x);

}

for (int i = 3; i < n - 1; i += 3) {

double x = a + i \* h;

result += 2 \* func(x);

}

result \*= (3 \* h) / 8.0;

return result;

}

int main() {

double a, b;

int n;

printf("Enter the lower limit of integration (a): ");

scanf("%lf", &a);

printf("Enter the upper limit of integration (b): ");

scanf("%lf", &b);

printf("Enter the number of subintervals (n, multiple of 3): ");

scanf("%d", &n);

if (n % 3 != 0) {

printf("Error: Number of subintervals should be a multiple of 3 for Simpson's 3/8 Rule.\n");

return 1;

}

double result = simpsons3by8Rule(a, b, n);

printf("Numerical integration using Simpson's 3/8 Rule: %lf\n", result);

return 0;

}

[16] #include <stdio.h>

double ode(double t, double y) {

return -2 \* t \* y;

}

void eulerMethod(double t0, double y0, double h, double endTime) {

double t = t0;

double y = y0;

while (t <= endTime) {

printf("t = %lf, y = %lf\n", t, y);

y = y + h \* ode(t, y);

t = t + h;

}}

int main() {

double t0, y0, h, endTime;

printf("Enter initial time (t0): ");

scanf("%lf", &t0);

printf("Enter initial value of y (y0): ");

scanf("%lf", &y0);

printf("Enter step size (h): ");

scanf("%lf", &h);

printf("Enter end time: ");

scanf("%lf", &endTime);

eulerMethod(t0, y0, h, endTime);

return 0;

}

[17] #include <stdio.h>

double ode(double t, double y) {

return -2 \* t \* y;

}void rungeKuttaMethod(double t0, double y0, double h, double endTime) {

double t = t0;

double y = y0;

while (t <= endTime) {

printf("t = %lf, y = %lf\n", t, y);

double k1 = h \* ode(t, y);

double k2 = h \* ode(t + h / 2.0, y + k1 / 2.0);

double k3 = h \* ode(t + h / 2.0, y + k2 / 2.0);

double k4 = h \* ode(t + h, y + k3);

y = y + (k1 + 2 \* k2 + 2 \* k3 + k4) / 6.0;

t = t + h;

}}

int main() {

double t0, y0, h, endTime;

printf("Enter initial time (t0): ");

scanf("%lf", &t0);

printf("Enter initial value of y (y0): ");

scanf("%lf", &y0);

printf("Enter step size (h): ");

scanf("%lf", &h);

printf("Enter end time: ");

scanf("%lf", &endTime);

rungeKuttaMethod(t0, y0, h, endTime);

return 0;

}

[18] #include <stdio.h>

void linearCurveFitting(int n, double x[], double y[]) {

double sumX = 0, sumY = 0, sumXY = 0, sumXSquare = 0;

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

sumX += x[i];

sumY += y[i];

sumXY += x[i] \* y[i];

sumXSquare += x[i] \* x[i];

}

double m = (n \* sumXY - sumX \* sumY) / (n \* sumXSquare - sumX \* sumX);

double b = (sumY - m \* sumX) / n;

printf("Linear Fit Equation: y = %.4fx + %.4f\n", m, b);

}

int main() {

int n;

printf("Enter the number of data points: ");

scanf("%d", &n);

double x[n], y[n];

printf("Enter the data points:\n");

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

printf("x[%d] = ", i);

scanf("%lf", &x[i]);

printf("y[%d] = ", i);

scanf("%lf", &y[i]);

}

linearCurveFitting(n, x, y);

return 0;

}

[19] #include <stdio.h>

void parabolicCurveFitting(int n, double x[], double y[]) {

double sumX = 0, sumY = 0, sumXSquare = 0, sumXCube = 0, sumXQuad = 0, sumXY = 0;

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

double xSquare = x[i] \* x[i];

sumX += x[i];

sumY += y[i];

sumXSquare += xSquare;

sumXCube += xSquare \* x[i];

sumXQuad += xSquare \* xSquare;

sumXY += x[i] \* y[i];

}

double a = (sumY \* sumXSquare - sumX \* sumXY + sumXCube) /

(n \* sumXQuad - sumXSquare \* sumXSquare);

double b = (n \* sumXY - sumX \* sumY) / (n \* sumXSquare - sumX \* sumX);

double c = (sumY - a \* sumXSquare - b \* sumX) / n;

printf("Parabolic Fit Equation: y = %.4fx^2 + %.4fx + %.4f\n", a, b, c);

}

int main() {

int n;

printf("Enter the number of data points: ");

scanf("%d", &n);

double x[n], y[n];

printf("Enter the data points:\n");

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

printf("x[%d] = ", i);

scanf("%lf", &x[i]);

printf("y[%d] = ", i);

scanf("%lf", &y[i]);

}

parabolicCurveFitting(n, x, y);

return 0;

}

[20]

#include <stdio.h>

void linearRegression(int n, double x[], double y[], double \*slope, double \*intercept) {

double sumX = 0, sumY = 0, sumXY = 0, sumXSquare = 0;

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

sumX += x[i];

sumY += y[i];

sumXY += x[i] \* y[i];

sumXSquare += x[i] \* x[i];

}

\*slope = (n \* sumXY - sumX \* sumY) / (n \* sumXSquare - sumX \* sumX);

\*intercept = (sumY - \*slope \* sumX) / n;

}

int main() {

int n;

printf("Enter the number of data points: ");

scanf("%d", &n);

double x[n], y[n];

printf("Enter the data points:\n");

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

printf("x[%d] = ", i);

scanf("%lf", &x[i]);

printf("y[%d] = ", i);

scanf("%lf", &y[i]);

}

double slope\_YonX, intercept\_YonX, slope\_XonY, intercept\_XonY;

linearRegression(n, x, y, &slope\_YonX, &intercept\_YonX);

printf("Regression Line for Y on X (Y|X): y = %.4fx + %.4f\n", slope\_YonX, intercept\_YonX);

linearRegression(n, y, x, &slope\_XonY, &intercept\_XonY);

printf("Regression Line for X on Y (X|Y): x = %.4fy + %.4f\n", slope\_XonY, intercept\_XonY);

return 0;

}