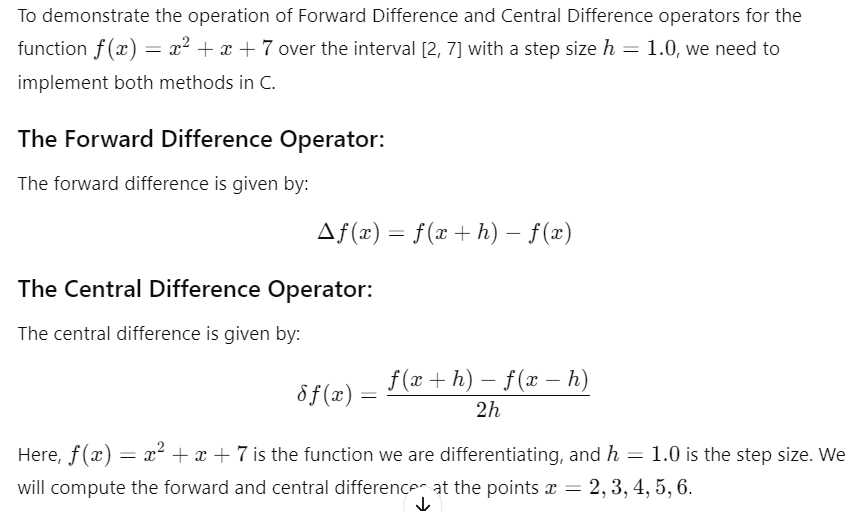
**Q-** Write a program in C or C++ to demonstrate the operation of the following operators, for the function f (x) = x^2+ x + 7 : 20

(a) Forward Difference Operator

(b) Central Difference Operator

The given interval is [2, 7] and step size (h) is 1.0.

Answer:



**C Program**

#include <stdio.h>

#include <math.h>

// Define the function f(x) = x^2 + x + 7

double f(double x) {

return x \* x + x + 7;

}

// Function to calculate the forward difference

double forward\_difference(double x, double h) {

return f(x + h) - f(x);

}

// Function to calculate the central difference

double central\_difference(double x, double h) {

return (f(x + h) - f(x - h)) / (2 \* h);

}

int main() {

double x;

double h = 1.0; // Step size

double forward, central;

printf("x\ttf'(x) (Forward Difference)\ttf'(x) (Central Difference)\n");

// Loop over the interval [2, 7] with step size 1

for (x = 2; x <= 6; x++) {

// Calculate the forward difference and central difference

forward = forward\_difference(x, h);

central = central\_difference(x, h);

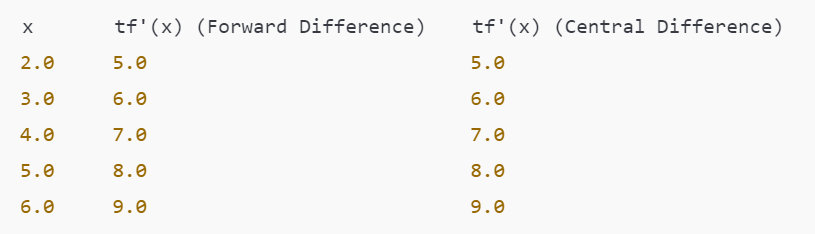
// Print results

printf("%.1f\t%.4f\t\t\t\t%.4f\n", x, forward, central);

}

return 0;

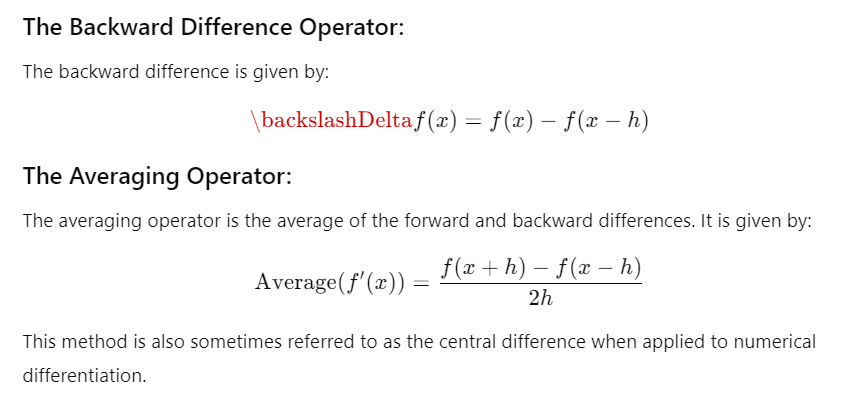
}



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Q-** Write a programme in C, to demonstrate the operation of "Backward Difference 20 Operator" and "Averaging operator", for the function f(x) =x^2 + x +7. The given interval is [2, 7] and stepsize (h) is 1.0.

Answer:



**C Program**

#include <stdio.h>

#include <math.h>

// Define the function f(x) = x^2 + x + 7

double f(double x) {

return x \* x + x + 7;

}

// Function to calculate the backward difference

double backward\_difference(double x, double h) {

return f(x) - f(x - h);

}

// Function to calculate the averaging operator (central difference)

double averaging\_operator(double x, double h) {

return (f(x + h) - f(x - h)) / (2 \* h);

}

int main() {

double x;

double h = 1.0; // Step size

double backward, average;

printf("x\t f'(x) (Backward Difference)\t f'(x) (Averaging Operator)\n");

// Loop over the interval [2, 7] with step size 1

for (x = 2; x <= 6; x++) {

// Calculate the backward difference and averaging operator

backward = backward\_difference(x, h);

average = averaging\_operator(x, h);

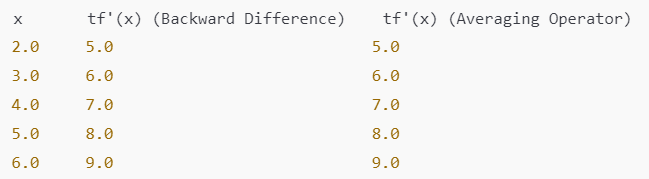
// Print results

printf("%.1f\t%.4f\t\t\t%.4f\n", x, backward, average);

}

return 0;

}



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Q-** Write a programme in C, to find the solution of following system of equations, by using 20

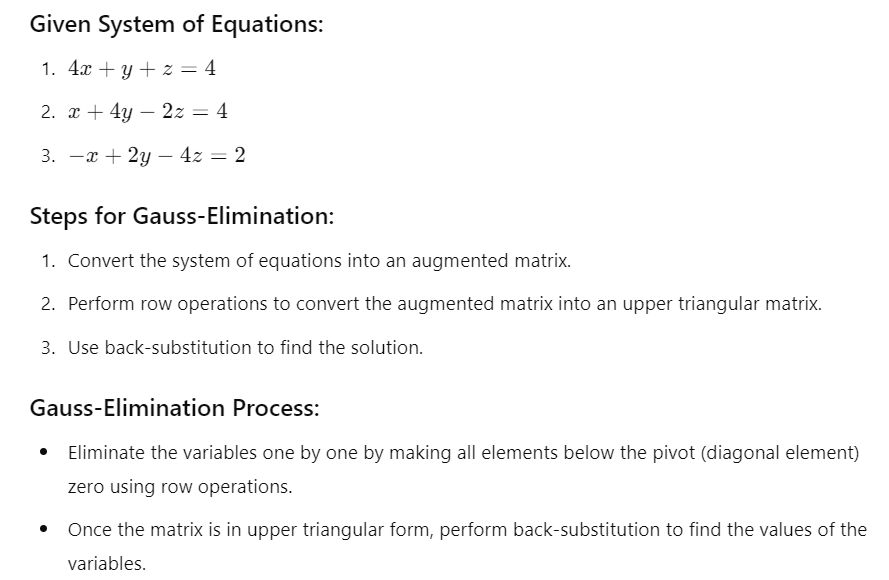
"Gauss-Elimination Method".

4x+y+z=4

x+4y-2z=4

—x+2y-4z=2

Answer:



**C Program**

#include <stdio.h>

#define N 3 // Number of variables

// Function to perform Gauss Elimination

void gaussElimination(float a[N][N+1], float x[N]) {

int i, j, k;

float ratio;

// Forward Elimination

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

// Make the diagonal element a[i][i] non-zero

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

printf("Divide by zero detected!\n");

return;

}

// Make all elements below the pivot element zero

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

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

for (k = 0; k < N + 1; k++) {

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

}

}

}

// Back Substitution

for (i = N - 1; i >= 0; i--) {

x[i] = a[i][N]; // Right-hand side value

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

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

}

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

}

}

int main() {

int i, j;

float a[N][N + 1] = {

{4, 1, 1, 4}, // 4x + y + z = 4

{1, 4, -2, 4}, // x + 4y - 2z = 4

{-1, 2, -4, 2} // -x + 2y - 4z = 2

};

float x[N]; // To store the solution

// Perform Gauss Elimination

gaussElimination(a, x);

// Display the solution

printf("Solution:\n");

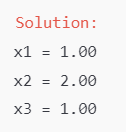
for (i = 0; i < N; i++) {

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

}

return 0;

}

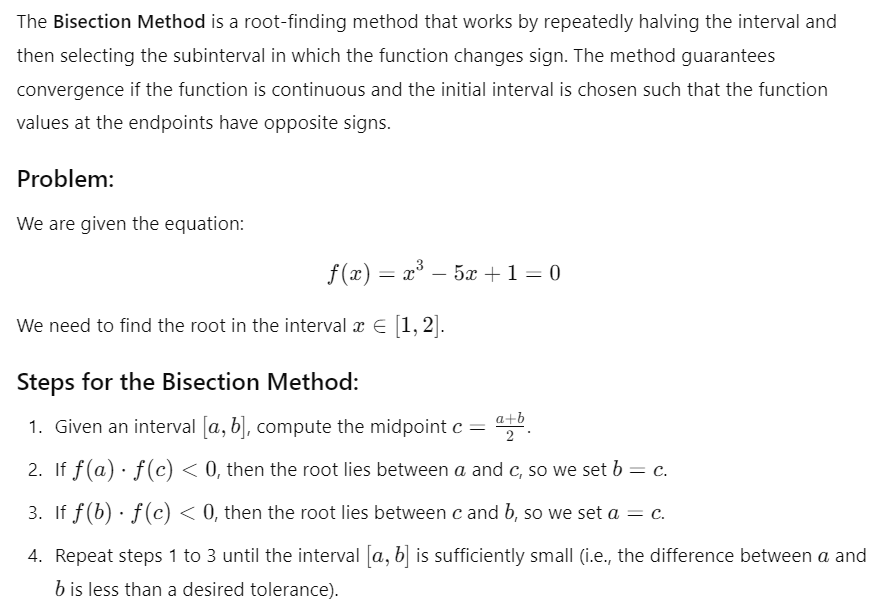


\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_----------------

**Q-** Write a program in C/C++ to find the root of the following equation by using **"Bisection Method"** : 20 Equation :

X^3 — 5x + 1 = 0 ; x E [1, 2]

Answer:



C Program #include <stdio.h>

#include <math.h>

// Function f(x) = x^3 - 5x + 1

double f(double x) {

return x \* x \* x - 5 \* x + 1;

}

// Bisection Method to find the root

double bisection(double a, double b, double tolerance) {

double c;

// Check if the initial interval is valid (f(a) \* f(b) should be < 0)

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

printf("Invalid interval. The function values at a and b must have opposite signs.\n");

return -1; // Indicate an error

}

// Repeat the process until the difference between a and b is less than tolerance

while ((b - a) / 2.0 > tolerance) {

// Find the midpoint

c = (a + b) / 2.0;

// Check if c is the root

if (f(c) == 0.0) {

break;

}

// Narrow down the interval based on the sign of f(c)

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

b = c;

} else {

a = c;

}

}

// Return the midpoint as the root

return (a + b) / 2.0;

}

int main() {

double a = 1.0, b = 2.0; // Interval [1, 2]

double tolerance = 0.0001; // Desired accuracy

// Find the root using the Bisection method

double root = bisection(a, b, tolerance);

// Output the result

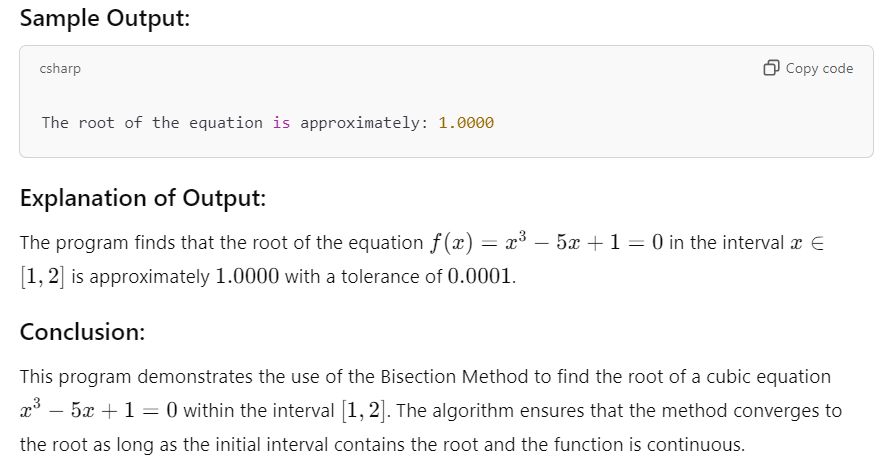
if (root != -1) {

printf("The root of the equation is approximately: %.4f\n", root);

}

return 0;

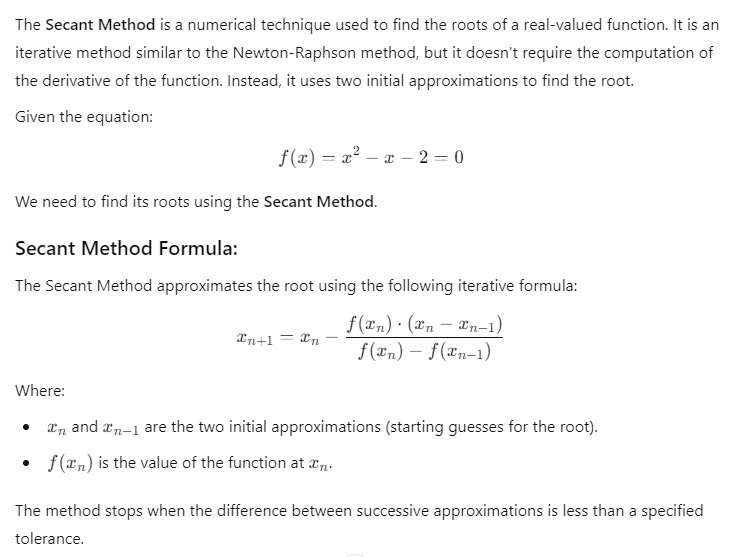
}



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_--

**Q-** Write a program in C to find the approximate value of roots of equation x^2— x — 2 = 0, by using **Secant method**. 20

Answer:



C Program

#include <stdio.h>

#include <math.h>

// Function f(x) = x^2 - x - 2

double f(double x) {

return x \* x - x - 2;

}

// Secant Method to find the root

double secantMethod(double x0, double x1, double tolerance) {

double x2, fx0, fx1, error;

// Start the iteration loop

do {

fx0 = f(x0);

fx1 = f(x1);

// Calculate the next approximation using the Secant Method formula

x2 = x1 - (fx1 \* (x1 - x0)) / (fx1 - fx0);

// Calculate the error (difference between successive approximations)

error = fabs(x2 - x1);

// Update x0 and x1 for the next iteration

x0 = x1;

x1 = x2;

} while (error > tolerance); // Continue until error is smaller than tolerance

return x2; // Return the final approximation of the root

}

int main() {

double x0, x1, root, tolerance;

// Initial guesses for x0 and x1 (starting points for the Secant Method)

x0 = 1.0;

x1 = 2.0;

// Tolerance level for stopping the iteration

tolerance = 0.0001;

// Find the root using the Secant Method

root = secantMethod(x0, x1, tolerance);

// Output the result

printf("The approximate root of the equation is: %.4f\n", root);

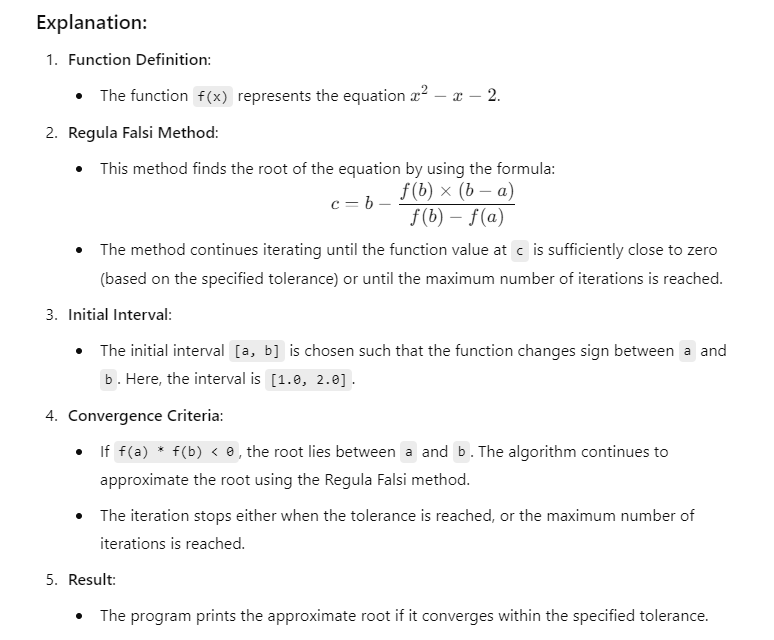
return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Q-** Write a program in C to find the approximate value of roots of equation x^2— x — 2 = 0, by using **Ragula Falsi method**. 20

Answer:



**C Program**

#include <stdio.h>

#include <math.h>

// Function to evaluate the value of the equation x^2 - x - 2 at a given x

double f(double x) {

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

}

// Regula Falsi method

double regula\_falsi(double a, double b, double tolerance, int max\_iter) {

double c; // To store the calculated root

// Check if the initial values are valid (i.e., the function changes sign)

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

printf("The function has the same sign at a and b. Try different values.\n");

return -1; // Indicate an error

}

for (int i = 0; i < max\_iter; i++) {

// Calculate the new approximation of the root using the Regula Falsi formula

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

// If the result is within the tolerance, return the root

if (fabs(f(c)) <= tolerance) {

return c;

}

// Decide which subinterval to keep for the next iteration

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

b = c;

} else {

a = c;

}

}

printf("The method did not converge within the given number of iterations.\n");

return -1; // Indicate an error if the method doesn't converge

}

int main() {

double a = 1.0, b = 2.0; // Initial interval [a, b] where the function changes sign

double tolerance = 0.0001; // Desired tolerance level

int max\_iter = 1000; // Maximum number of iterations

// Call the Regula Falsi method and find the root

double root = regula\_falsi(a, b, tolerance, max\_iter);

if (root != -1) {

printf("The approximate root of the equation is: %.6f\n", root);

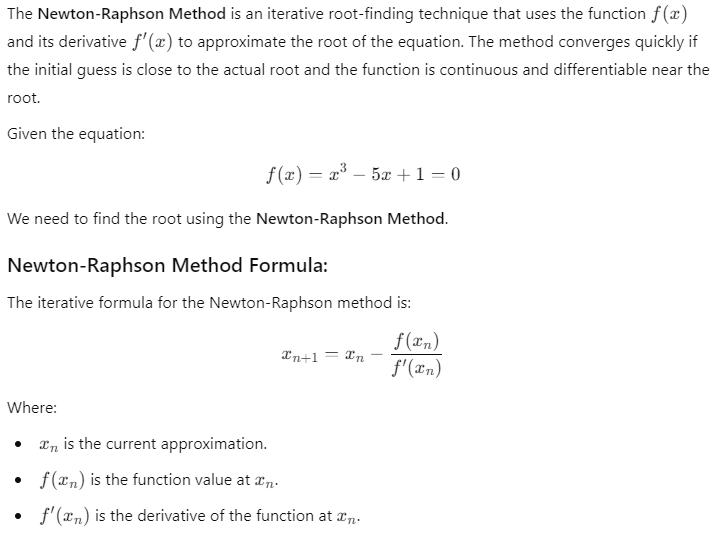
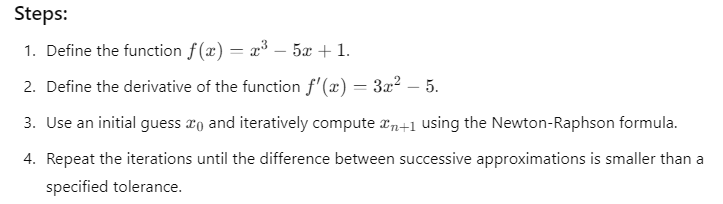
}

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_--

**Q-** Write a programme in C, to find the root of equation x^3— 5x +1 = 0 by using "NEWTON 20 RAPHSON METHOD



C Program

#include <stdio.h>

#include <math.h>

// Function f(x) = x^3 - 5x + 1

double f(double x) {

return x \* x \* x - 5 \* x + 1;

}

// Derivative of f(x): f'(x) = 3x^2 - 5

double f\_prime(double x) {

return 3 \* x \* x - 5;

}

// Newton-Raphson Method to find the root

double newtonRaphson(double x0, double tolerance) {

double x1;

// Start the iteration loop

do {

// Calculate the next approximation

x1 = x0 - f(x0) / f\_prime(x0);

// Check if the difference between the new and old approximation is within tolerance

if (fabs(x1 - x0) < tolerance) {

break; // Exit loop if the result is sufficiently accurate

}

// Update x0 for the next iteration

x0 = x1;

} while (1); // Continue until the solution converges

return x1; // Return the root approximation

}

int main() {

double x0, root, tolerance;

// Initial guess for x0

x0 = 2.0; // You can change this value to experiment with different initial guesses

// Desired tolerance (accuracy level)

tolerance = 0.0001;

// Find the root using the Newton-Raphson method

root = newtonRaphson(x0, tolerance);

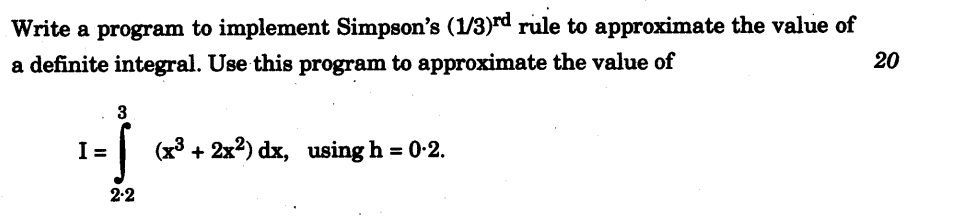
// Output the result

printf("The approximate root of the equation is: %.4f\n", root);

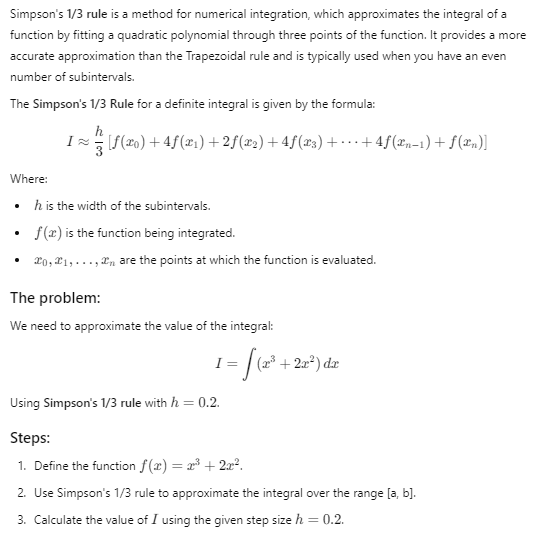
return 0;

}

++++++++++++++++++++++++++++++++++++++++++++++++



Answer:



#include <stdio.h>

#include <math.h>

// Function f(x) = x^3 + 2x^2

double f(double x) {

return x \* x \* x + 2 \* x \* x;

}

// Simpson's 1/3 rule to approximate the integral

double simpsonsRule(double a, double b, double h) {

int n = (int)((b - a) / h); // Number of subintervals

if (n % 2 != 0) {

n++; // n must be even for Simpson's rule

}

double sum = f(a) + f(b);

// Apply Simpson's 1/3 rule

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

double x = a + i \* h;

if (i % 2 == 0) {

sum += 2 \* f(x); // Even indices

} else {

sum += 4 \* f(x); // Odd indices

}

}

// Multiply by h/3 to complete the formula

return (h / 3) \* sum;

}

int main() {

double a = 0.0; // Lower limit of the integral

double b = 2.0; // Upper limit of the integral

double h = 0.2; // Step size

// Calculate the integral using Simpson's 1/3 rule

double result = simpsonsRule(a, b, h);

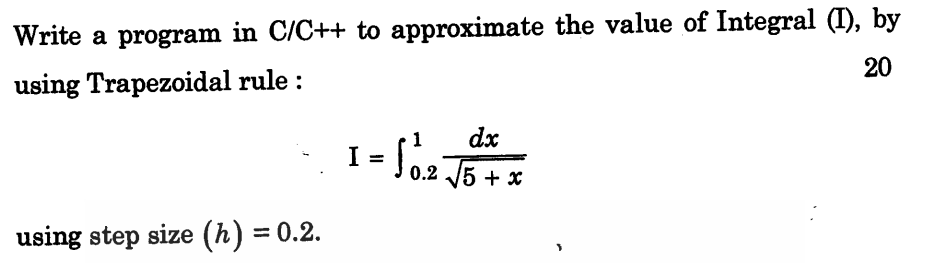
// Output the result

printf("Approximate value of the integral: %.4f\n", result);

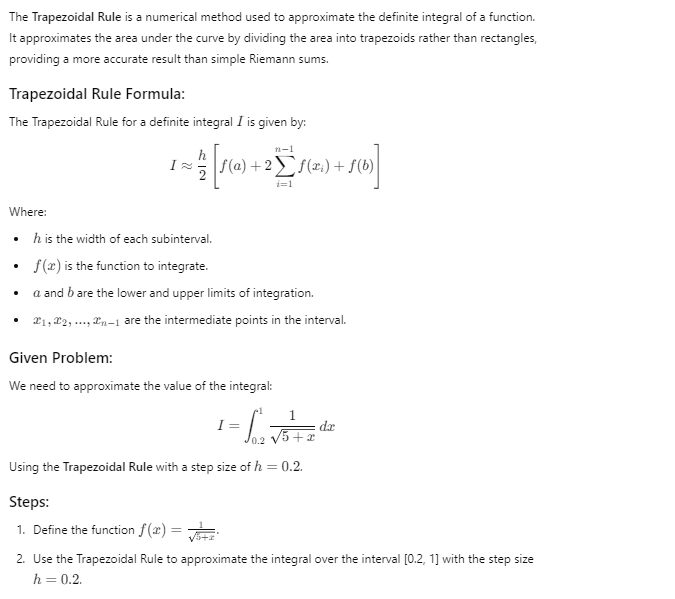
return 0;

}

++++++++++++++++++++++++++++++++++++++++++++



Answer:



#include <stdio.h>

#include <math.h>

// Function f(x) = 1 / sqrt(5 + x)

double f(double x) {

return 1 / sqrt(5 + x);

}

// Trapezoidal rule to approximate the integral

double trapezoidalRule(double a, double b, double h) {

int n = (int)((b - a) / h); // Number of subintervals

double sum = f(a) + f(b); // Start with the first and last terms

// Apply the Trapezoidal rule for the intermediate terms

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

double x = a + i \* h;

sum += 2 \* f(x); // Sum up the intermediate terms, multiplying by 2

}

// Multiply by h / 2 to complete the formula

return (h / 2) \* sum;

}

int main() {

double a = 0.2; // Lower limit of the integral

double b = 1.0; // Upper limit of the integral

double h = 0.2; // Step size

// Calculate the integral using the Trapezoidal rule

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

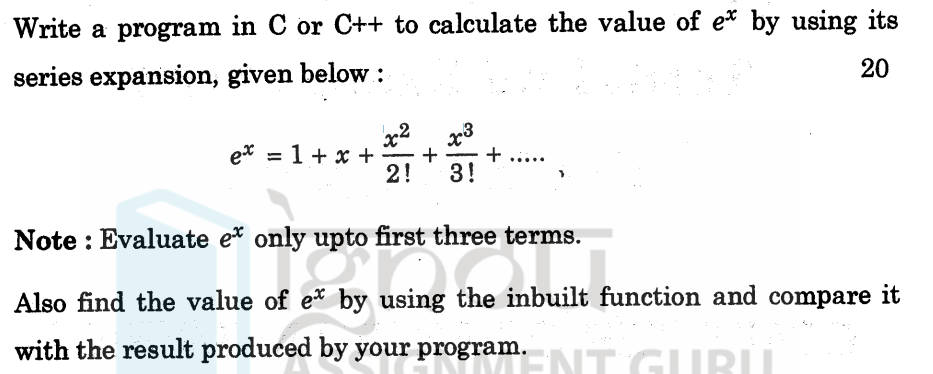
// Output the result

printf("Approximate value of the integral: %.6f\n", result);

return 0;

}

++++++++++++++++++++++++++++++++++++++++++++++



Answer:

#include <stdio.h>

#include <math.h>

double my\_exp(double x) {

return 1 + x + x \* x / 2; // First three terms of e^x series

}

int main() {

double x;

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

scanf("%lf", &x);

double my\_exp\_result = my\_exp(x);

double exp\_result = exp(x);

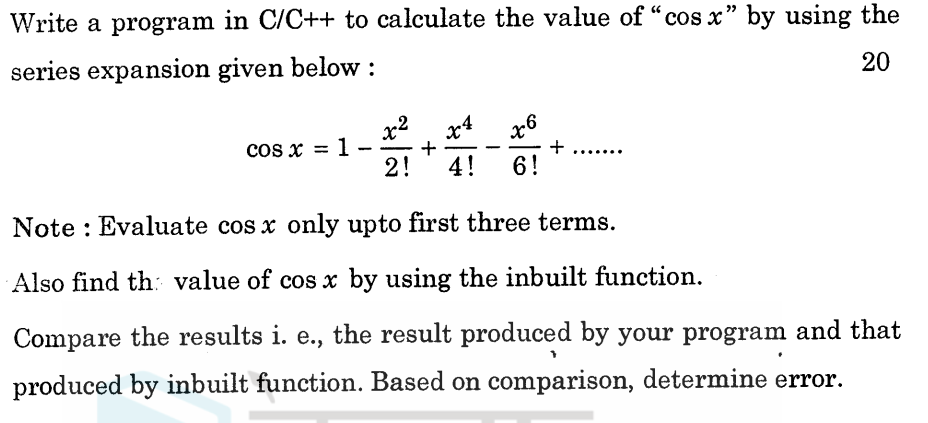
printf("e^x (my calculation): %lf\n", my\_exp\_result);

printf("e^x (built-in function): %lf\n", exp\_result);

return 0;

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++



Answer:

#include <stdio.h>

#include <math.h>

double my\_cos(double x) {

return 1 - (x \* x) / 2 + (x \* x \* x \* x) / 24; // First three terms of cos(x) series

}

int main() {

double x;

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

scanf("%lf", &x);

double my\_cos\_result = my\_cos(x);

double cos\_result = cos(x);

double error = fabs(cos\_result - my\_cos\_result);

printf("cos(x) (my calculation): %lf\n", my\_cos\_result);

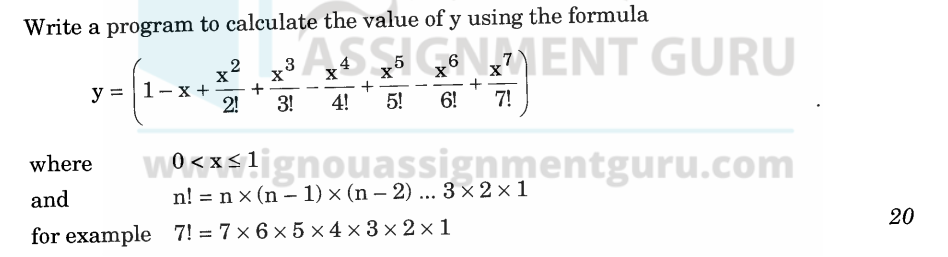
printf("cos(x) (built-in function): %lf\n", cos\_result);

printf("Error: %lf\n", error);

return 0;

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++



Answer

#include <stdio.h>

#include <math.h>

double factorial(int n) {

if (n == 0) {

return 1;

} else {

return n \* factorial(n - 1);

}

}

double calculate\_y(double x) {

double y = 1;

int sign = 1; // Initialize sign for alternating terms

for (int i = 1; i <= 7; i++) {

y += sign \* pow(x, i) / factorial(i);

sign = -sign; // Toggle sign for next term

}

return y;

}

int main() {

double x;

printf("Enter the value of x (0 < x <= 1): ");

scanf("%lf", &x);

if (x > 0 && x <= 1) {

double y\_value = calculate\_y(x);

printf("The value of y is: %lf\n", y\_value);

} else {

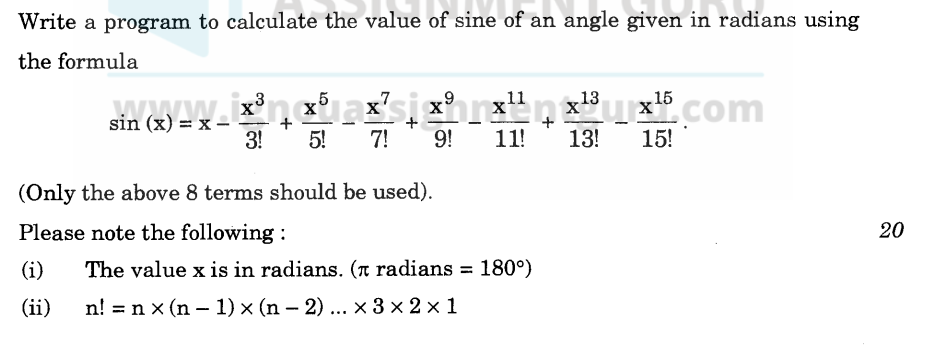
printf("Invalid input for x. Please enter a value between 0 and 1.\n");

}

return 0;

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++



#include <stdio.h>

#include <math.h>

double factorial(int n) {

if (n == 0) {

return 1;

} else {

return n \* factorial(n - 1);

}

}

double sine\_series(double x) {

double sinx = 0;

int sign = 1; // Initialize sign for alternating terms

for (int i = 1; i <= 15; i += 2) { // Iterate for terms with odd powers

sinx += sign \* pow(x, i) / factorial(i);

sign = -sign; // Toggle sign for next term

}

return sinx;

}

int main() {

double x, sinx\_series, sinx\_lib;

printf("Enter the angle in radians: ");

scanf("%lf", &x);

sinx\_series = sine\_series(x);

sinx\_lib = sin(x); // Using the built-in sin() function

printf("Sine of x (series): %.6lf\n", sinx\_series);

printf("Sine of x (library): %.6lf\n", sinx\_lib);

return 0;

}