Lab Manual for Numerical Analysis

Lab No. 8

INTEGRATION - TRAPEZOIDAL RULE & SIMPSON'S ONE-THIRD RULE

BAHRIA UNIVERSITY KARACHI CAMPUS
Department of Software Engineering

NUMERICAL ANALYSIS

LAB EXPERIMENT # 8

Integration - Trapezoidal rule & Simpson's one-third rule

OBJECTIVE:

This lab aims to instruct students in numerical integration techniques for implementing the **Trapezoidal rule** and **Simpson's one-third rule**. The goal is to equip them with practical skills to compute definite integrals crucial in scientific and engineering analyses.

Introduction

Integration, a fundamental mathematical concept, plays a pivotal role in determining the accumulation, total, or area under a curve within various disciplines. When encountering functions that lack analytical integration or involve complex forms, numerical integration methods become indispensable. These techniques provide approximations for definite integrals, enabling solutions to problems where precise analytical solutions are challenging to attain. Let's explore the essence of integration, the concept of numerical integration, and its practical application within numerical analysis.

Integration involves finding the total or accumulation of quantities represented as the area under a curve. It is used extensively across mathematics, physics, engineering, and other fields to solve problems related to area, volume, probability, and differential equations. However, in scenarios where exact analytical solutions are unattainable due to function complexity or lack of closed-form expressions, numerical integration becomes essential.

Numerical integration entails employing algorithms and techniques such as the Trapezoidal rule, Simpson's rule, or Gaussian quadrature to estimate definite integrals numerically. For instance, in numerical analysis, when dealing with functions that cannot be integrated symbolically, such as in complex physics simulations or financial modeling involving intricate equations, numerical integration methods provide approximations for these integrals. Consider a scenario where a scientist aims to compute the area under a curve representing a physical phenomenon's behavior, or an engineer needing to determine the total force acting on a structure. In such cases, numerical integration methods offer valuable tools to approximate these integrals, providing crucial insights into the underlying systems or structures being studied.

In analysis, numerical integration involves a diverse range of algorithms used to compute the numerical value of definite integrals. This chapter is dedicated to examining and employing two fundamental techniques, namely the Trapezoidal rule and Simpson's one-third rule, within

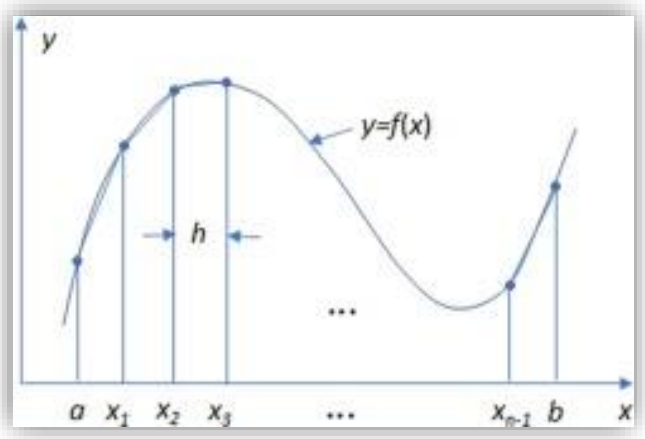
numerical integration. These methods are essential for accurately approximating definite integrals, playing a crucial role in scientific and engineering computations.

Implementation of Trapezoidal rule and Simpson's one-third rule

This section will extensively explore the principles and Python implementation of the Trapezoidal rule and Simpson's one-third rule in numerical integration, emphasizing their crucial role in computing definite integrals within scientific and engineering applications.

a. Trapezoidal rule

The Trapezoidal rule is a numerical method used for approximating definite integrals by dividing the area under a curve into trapezoids and summing their areas. It's based on approximating the curve by straight line segments, forming trapezoids whose collective area provides an estimate of the integral's value.



The principle behind the Trapezoidal rule involves dividing the interval [a, b] into smaller sub-intervals and approximating the curve by straight-line segments between these points. The area under each of these segments resembles a trapezoid. The larger the number of sub-intervals (n), the closer the approximation aligns with the actual integral value. The formula for the Trapezoidal rule sums up these trapezoidal areas over the interval [a, b] to estimate the integral.

For an integral ∫f(x) dx from a to b, the Trapezoidal rule's formula is given by:

$$\int_a^b f(x)dx \approx \frac{h}{2}\left[f(a) + 2\left\{\sum_{i=1}^{n-1} f(a+ih)\right\} + f(b)\right]$$

Where;

- $h = \frac{b-a}{n}$

This formula computes the area of each trapezoid formed by the function values at each sub-interval and sums these areas to approximate the integral's value.

Implementation in Python

- The following is the Python code that implements the Trapezoidal rule for approximating the definite integral of a given function. The function function(x) represents $\frac{1}{1+x^2}$. The code calculates the approximate integral value within the **range [0, 1]** using **6 sub-intervals.** Adjust the **start_point, end_point,** and **sub_intervals** variables to change the range and precision of the integral approximation.

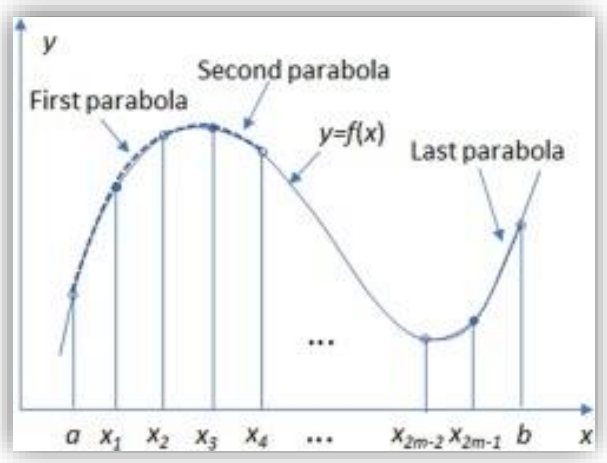
```
def function(x):
    return (1 / (1 + x * x))

def trapezoidal_rule(a, b, n):
    h = (b - a) / n
    result = (function(a) + function(b))
    i = 1
    while i < n:
        result += 2 * function(a + i * h)
        i += 1
    return ((h / 2) * result)

# Definite integral range
start_point = 0
end_point = 1

# Number of sub-intervals for accuracy
sub_intervals = 6
print("Approximated integral value:", "%.4f"
      % trapezoidal_rule (start_point, end_point, sub_intervals))
```

- Meanwhile, the following code defines a **trapezoidal_rule_from_points** function that calculates the approximate integral value using the Trapezoidal rule based on the given points instead of function.

```
def trapezoidal_rule_from_points(points):
    n = len(points)
    integral = points[0][1]+points[-1][1] #Adding 1st & last y-values

    for i in range(1, n - 1):
        integral += 2 * points[i][1]  #Adding rest of the y-values

    h = points[1][0] - points[0][0]  # Calculating the interval width
    integral *= h / 2  # Multiplying by h/2 to compute the integral

    return integral

# Example usage with given points [(x0, y0), (x1, y1), ..., (xn, yn)]
given_points = [(0,1), (0.2,0.96), (0.4,0.87), (0.6,0.7), (0.8,0.45)]
result = trapezoidal_rule_from_points(given_points)
print("Approximated integral value using given points:", result)
```

b. Simpson's one-third rule

Simpson's 1/3 Rule is a numerical method used to approximate definite integrals by fitting parabolic arcs (quadratic polynomials) to small intervals of the function. It's a type of numerical integration that provides a more accurate estimation of integrals compared to the Trapezoidal rule by utilizing quadratic approximations of the function curve.



The core principle of Simpson's 1/3 Rule involves segmenting the interval [a, b] into smaller sub-intervals (usually even in number) and applying quadratic interpolation over pairs of adjacent sub-intervals. Each pair of intervals, along with their function values at both ends and midpoint, forms a quadratic polynomial, resulting in a series of parabolic arcs. These arcs' collective areas under the curve within the interval [a, b] are summed to estimate the integral.

For an integral ∫f(x) dx from a to b using Simpson's 1/3 Rule with an even number of intervals (n), the formula is given by:

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(a) + 4\left\{\sum_{i=1,3,5,\ldots}^{n-1} f(xi)\right\} + 2\left\{\sum_{i=2,4,6,\ldots}^{n-2} f(xi)\right\} + f(b)\right]$$

Where;

- $h = \frac{b-a}{n}$ represents the width of each sub-interval.
- xi denotes the points within the interval [a, b].
- In this rule, n must be EVEN.

This formula computes the area of each trapezoid formed by the function values at each sub-interval.

This formula combines the function values at the endpoints of the interval with four times the function values at the odd indices (excluding the endpoints) and two times the function values at the even indices. This weighting scheme aligns with the coefficients of the quadratic polynomial for each pair of intervals.

Implementation in Python

- The following code provides a function **simpsons_one_third_rule** to approximate the integral using Simpson's 1/3 rule for a given function, represented by math.log(x).

```
import math

# Function to calculate f(x)
def calculate_function_value(x):
    return math.log(x)

# Function to approximate integral using Simpson's 1/3 rule
def simpsons_one_third_rule(lower_limit, upper_limit, intervals):

    # Calculating the value of h
    h = (upper_limit - lower_limit) / intervals

    # Lists to store x and f(x) values
    x_values = []
    fx_values = []


    # Calculating values of x and f(x)
    i = 0
    while i <= intervals:
        x_values.append(lower_limit + i * h)
        fx_values.append(calculate_function_value(x_values[i]))
        i += 1


    # Calculating result using Simpson's 1/3 rule
    result = 0
    i = 0
    while i <= intervals:
        if i == 0 or i == intervals:
            result += fx_values[i]
        elif i % 2 != 0:
            result += 4 * fx_values[i]
        else:
            result += 2 * fx_values[i]
        i += 1
    result *= h / 3
    return result


lower_limit = 4  # Lower limit
upper_limit = 5.2  # Upper limit
num_intervals = 6  # Number of intervals
print("%.6f" % simpsons_one_third_rule(lower_limit, upper_limit,
num_intervals))
```

- Meanwhile the following code performs Simpson's 1/3 rule directly on given data points using function **simpsons_one_third_rule_from_points**

```
# Function to perform Simpson's 1/3 rule from given data points
def simpsons_one_third_rule_from_points(data_points):
    n = len(data_points) - 1  # Number of intervals
    if n % 2 != 0:  # Check if the number of intervals is even
        print("Number of intervals must be even for Simpson's
              1/3 rule.")
        return None

    h = data_points[1][0] - data_points[0][0]  # Interval width

    # Calculating result using Simpson's 1/3 rule
    result=data_points[0][1]+data_points[-1][1] #1st & last values

    # Intermediate values
    for i in range(1, n):
        if i % 2 != 0:
            result += 4 * data_points[i][1]
        else:
            result += 2 * data_points[i][1]

    result *= h / 3
    return result

# Given data points in the format [(x0, y0), (x1, y1),..,(xn, yn)]
given_data_points = [(4, 1.386), (4.2, 1.435), (4.4, 1.481),
                     (4.6, 1.526), (4.8, 1.568), (5, 1.609),
                     (5.2, 1.648)]

approximated_integral = simpsons_one_third_rule_from_points
                        (given_data_points)

if approximated_integral is not None:
    print("Approximated integral value using given data points:",
          "%.6f" % approximated_integral)
```

Lab Tasks

1. Write a Python program to determine the area enclosed by the function $f(x) = e^x$ over the interval from x=0 to x=2 using the Trapezoidal Rule with two intervals.

(Output: 6.9128)

2. Write a Python program utilizing Simpson's 1/3 Rule to compute the definite integral $\int_{e^2}^{2e^2} lnx\ dx$ where n = 4 represents the number of intervals used in the approximation.

(Output: 17.6322)

3. Determine the approximation of the area beneath the curve represented by y = f(x) over the interval from x=−4 to x=2 using the Trapezoidal Rule with n=6 subintervals. The values of the function f(x) are provided within following table.

x	-4	-3	-2	-1	0	1	2
f(x)	0	4	5	3	10	11	2

(Output: 34.00)