EM215 Numerical Methods
Numerical Integration and Differentiation
Lab1/Assignment 1

Numerical Quadrature

Lab Objective: Learn the basics of Newton -Cotes quadrature and Gaussian quadrature and its application to numerical integration.
Compare the accuracy and speed of selected quadrature schemes.

This lab is to understand the basic properties and convergence properties of Newton-quotes quadrature formulas. As we discussed in class:

The numerical approximation $X(h)$ converges to X if $e(h) \to 0$ as $h \to 0$. The order of the approximation is p if there exists a positive constant K such that

$$e(h) \leq Kh^p$$

The Big O-notation we can simply write

$$e(h) = O(h^p) \qquad \text{as } h \to 0.$$

This is often used when we are not directly interested in any expression for the constant K, we only need to know it exists.

How are we going to numerically verify this?

We will assume that for sufficiently small h we have $e(h) \approx Ch^p$.

Suppose the error in the $k+1$ th step is $e(h_{k+1})$. Then we have:

$$\begin{aligned} e(h_{k+1}) &\approx Ch_{k+1}^p \\ e(h_k) &\approx Ch_k^p \end{aligned}$$

With little algebra and taking the logarithm we get:

$$\frac{e(h_{k+1})}{e(h_k)} \approx \left(\frac{h_{k+1}}{h_k}\right)^p$$

$$p \approx \frac{\log\left(e(h_{k+1})/e(h_k)\right)}{\log\left(h_{k+1}/h_k\right)}$$

We call p the experimental order of convergence. But also note that:

$$e(h) \approx Ch^p \quad \Rightarrow \quad \log e(h) \approx \log C + p \log h$$

$y \simeq mx + C$

EM215 Numerical Methods
Numerical Integration and Differentiation
Lab1/Assignment 1

a plot of *e(h)* as a function of h using a logarithmic scale on both axes (a log-log plot) will be a straight line with slope p. Such a plot is referred to as an error plot or a convergence plot.

Q1. Study the given codes and explain that the absolute error for the composite trapezoidal rule decays at the rate of $1/n^2$, and the absolute error for the composite Simpson's rule decays at the rate of $1/n^4$, where n is the number of subintervals.

Q2. Determine n that ensures the composite Simpson's rule approximates

$$\int_1^2 x\log x \, dx$$

with an absolute error of at most 10^{-6}. Numerically verify if the value for n is reasonable.

Q3. Consider the error function that occur in many areas of engineering and statistics:

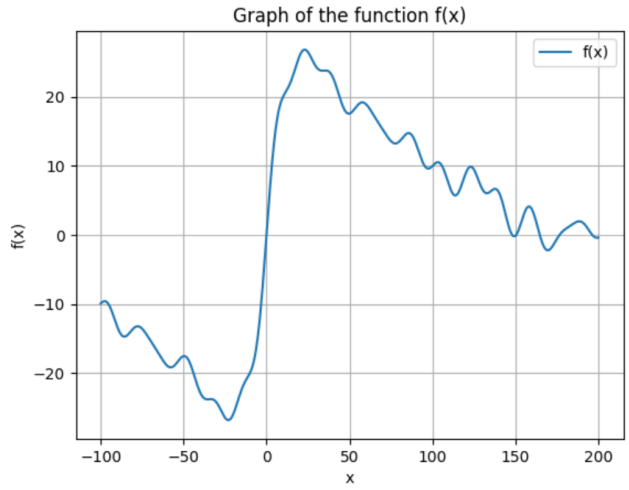
$$errf(a) = \frac{2}{\sqrt{\pi}} \int_0^a e^{-x^2} dx$$

a) Find the value of the integral for a = 1.5 with scipy.integrate.quad and use this as the exact value of the integral.
b) Plot $errf(a)$ for $y \in [-2,2\ ]$.
c) Evaluate the integral using the Gauss integration scheme with 2, 3, 4 and 5 integration points. Calculate the relative error in each case.
d) Calculate the number of trapeziums required by the trapezoidal rule to produce an error similar to that produced by the 3-point Gauss integration scheme, then evaluate the integral using the trapezoidal rule with the obtained number of trapeziums.

EM215 Numerical Methods
Numerical Integration and Differentiation
Lab1/Assignment 1

Numerical Differentiation

Q1



.

Shown above is the graph of a function described by the python code below:

```
import numpy as np

from math import sin


# Define the function

def f(x):

    n = 27

    t = 0

    for i in range(50):

        t += sin(10 * x / n)

        n = (n // 2) if n % 2 == 0 else 3 * n + 1

    return t
```

Use the forward difference formula with $h = 10^{-8}$ to estimate $f'(2)$.

EM215 Numerical Methods
Numerical Integration and Differentiation
Lab1/Assignment 1

Q2. Suppose You have obtained a signal $f=\cos(x)$. But this signal is spoiled by some noise given by a small sine wave:

$f_{\epsilon,\omega}(x)=\cos(x)+\epsilon\sin(\omega x)$. where $0<\epsilon\ll 1$ is a very small number and ω is a large number.

For $\epsilon=0.01$ and $\omega=100$, first plot in the same graph the function $f=\cos(x)$ and the function with noise $f_{\epsilon,\omega}(x)=\cos(x)+\epsilon\sin(\omega x)$.

Then in a different graph plot the derivatives f' and $f'_{\epsilon,\omega}$ and comment on your observations.

Q3. Suppose a formula for the first derivative using finite difference formula is given as:

$$f'(x_i) \approx \frac{-f(x_{i+3}) + 9\,f(x_{i+1}) - 8f(x_i)}{6h}$$

Where the points x_i, x_{i+1}, x_{i+2}, x_{i+3} are all equally spaced with step size h. Determine the order of the approximation.

Q4. The following table gives the values of $f(x) = \sin x$: Estimate $f'(0.1)$; $f'(0.3)$ using an appropriate three-point formula.

X	f(x)
0.1	0.09983
0.2	0.19867
0.3	0.29552
0.4	0.38942

EM215 Numerical Methods
Numerical Integration and Differentiation
Lab1/Assignment 1

Understanding the Trapezoidal Rule Integration Code

Overview

This document is designed to help you understand the provided Python code that implements numerical integration using the trapezoidal rule. The code also compares the numerical integration results to the exact integral value and visualizes the error and the function being integrated.

Code Breakdown

1. Function Definition: trapezoidal

```
def trapezoidal(func, a, b, N):
    """
    Numerical quadrature based on the trapezoidal rule.

    Parameters:
    func : function
        The function to integrate, handle to y = f(x).
    a : float
        The lower bound of the integration interval.
    b : float
        The upper bound of the integration interval.
    N : int
        The number of subintervals (N+1 points).
    """
    from numpy import linspace, sum

    # Quadrature nodes
    x = linspace(a, b, N + 1)
    h = x[1] - x[0]

    # Quadrature weights: internal nodes: w = 1, boundary nodes: w = 0.5
    I = sum(func(x[1:-1])) + 0.5 * (func(x[0]) + func(x[-1]))

    return I * h
```

Explanation:

- **Parameters:**
  - func: The function to integrate.
  - a: The lower bound of the integration interval.
  - b: The upper bound of the integration interval.
  - N: The number of subintervals.
- **Process:**
  - Calculate x, the quadrature nodes, as N + 1 evenly spaced points between a and b.
  - Calculate h, the width of each subinterval.
  - Compute the integral using the trapezoidal rule: sum the function values at the internal nodes and add half the function values at the boundary nodes. Multiply the result by h to get the integral approximation.

2. Main Execution Block

```
if __name__ == "__main__":
    import matplotlib.pyplot as plt
    from scipy import integrate
    from numpy import array, linspace, size, log, polyfit
    import numpy as np

    # Define a function and an interval
    f = lambda x: x**2
    left = 0.0
    right = 1.0

    # Exact integration with scipy.integrate.quad
    exact, e = integrate.quad(f, left, right)
    print(f"Exact integral: {exact}")

    # Trapezoid rule for different number of quadrature points
    N = linspace(2, 101, 100)
    res = array([trapezoidal(f, left, right, int(n)) for n in N])
    err = abs(res - exact)

    # Print the trapezoidal approximations
    for n, approximation in zip(N, res):
        print(f"N = {int(n)}: Trapezoidal approximation = {approximation}")

    # Plotting error vs. number of quadrature points
    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)
    plt.loglog(N, err, 'o')
    plt.xlabel('Number of Quadrature Points')
```

```
plt.ylabel('Error')
plt.title('Error in Trapezoidal Rule Integration')

# Plotting the function
x_vals = np.linspace(left, right, 400)
y_vals = f(x_vals)

plt.subplot(1, 2, 2)
plt.plot(x_vals, y_vals, label='f(x) = x^2')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Function Plot')
plt.legend()

plt.tight_layout()
plt.show()

# Linear fit to determine convergence order
p = polyfit(log(N), log(err), 1)

# Output the convergence order
print("Convergence order:", -p[0])
```

Explanation:

1. **Imports:** Necessary libraries for plotting and numerical calculations are imported.
2. **Function Definition and Interval:**
   - f = lambda x: x**2 defines the function to be integrated.
   - left and right set the bounds of the integration interval.
3. **Exact Integration:**
   - Uses scipy.integrate.quad to compute the exact value of the integral for comparison.
4. **Numerical Integration:**
   - N is defined as an array of 100 points between 2 and 101.
   - The trapezoidal function is applied to each value in N to compute the approximations.
   - The absolute error of each approximation compared to the exact integral is calculated.
5. **Print Results:**
   - Prints the trapezoidal approximation for each value in N.
6. **Plotting:**
   - Creates a subplot for the error vs. the number of quadrature points using a log-log plot.
   - Creates a subplot to plot the function f(x).
   - plt.tight_layout() adjusts subplot parameters for a cleaner layout.

- Displays the plots.

7. **Convergence Order:**
   - Performs a linear fit on the log-log plot data to determine the convergence order.
   - Prints the convergence order.

Running the Code

To run this code, you need Python installed along with the required libraries (numpy, matplotlib, scipy). Save the code in a .py file and execute it using a Python interpreter or run it in a Jupyter notebook. Ensure all the necessary libraries are installed using pip

```
pip install numpy matplotlib scipy
```

Understanding the Gauss Quadrature Integration Code

Introduction

This report explains the Python code provided for performing numerical integration using Gauss quadrature. The code includes functions to compute Gauss quadrature nodes and weights, apply the Gauss quadrature method for integration, and evaluate the error of numerical quadrature methods against exact integrals. Additionally, the code visualizes the error and the convergence order of the quadrature method.

Code Breakdown

1. Imports

The code imports several libraries necessary for numerical calculations, integration, and plotting:

```
import matplotlib.pyplot as plt
from numpy import linspace, arctan, power, array, sqrt, abs, log, dot, zeros, diag, size, polyfit
from numpy.linalg import eigh
from scipy import integrate
```

- matplotlib.pyplot: For plotting graphs.
- numpy: For numerical operations.
- scipy.integrate: For computing exact integrals for comparison.

2. Gauss Quadrature Nodes and Weights: gaussquad

EM215 Numerical Methods
Numerical Integration and Differentiation
Lab1/Assignment 1

(In the class we did not discuss this particular algorithm. Therefore you need not know the details and may use to find Gauss quadrature points and weights

Golub–Welsch[edit]

In 1969, Golub and Welsch published their method for computing Gaussian quadrature rules given the three term recurrence relation that the underlying orthogonal polynomials satisfy. They reduce the problem of computing the nodes of a Gaussian quadrature rule to the problem of finding the eigenvalues of a particular symmetric tridiagonal matrix. The QR algorithm is used to find the eigenvalues of this matrix. (Ref: https://en.wikipedia.org/wiki/Gauss%E2%80%93Legendre_quadrature)

```
def gaussquad(n):
    """
    Calculate Gauss quadrature nodes and weights.

    Parameters:
    n : int
        The number of quadrature points.

    Returns:
    x : ndarray
        The quadrature nodes.
    w : ndarray
        The quadrature weights.
    """
    b = zeros(n-1)
    for i in range(size(b)):
        b[i] = (i + 1) / sqrt(4 * (i + 1) * (i + 1) - 1)

    J = diag(b, -1) + diag(b, 1)
    x, ev = eigh(J)
    w = 2 * ev[0]**2
    return x, w
```

- **Parameters:**
  - n: Number of quadrature points.
- **Returns:**
  - x: Gauss quadrature nodes.
  - w: Gauss quadrature weights.
- **Process:**
  - Compute the off-diagonal elements b for the Jacobi matrix J.
  - Form the Jacobi matrix J using diag.

- Compute the eigenvalues (x) and eigenvectors (ev) of J using eigh.
- Calculate the weights (w) using the square of the first element of the eigenvectors.

3. Gauss Quadrature Integration: quad_gauss

```
def quad_gauss(f, a, b, deg):
    """
    Numerical integration using Gauss quadrature.

    Parameters:
    f : function
        The function to integrate, handle to y = f(x).
    a : float
        The lower bound of the integration interval.
    b : float
        The upper bound of the integration interval.
    deg : int
        The degree of the polynomial (number of Gauss points).

    Returns:
    result : float
        The integral of the function over [a, b].
    """
    # Get Gauss points for [-1, 1]
    gx, w = gaussquad(deg)

    # Transform to [a, b]
    x = 0.5 * (b - a) * gx + 0.5 * (a + b)
    y = f(x)

    # Return the integral approximation
    return 0.5 * (b - a) * dot(w, y)
```

- **Parameters:**
  - f: The function to integrate.
  - a: Lower bound of the integration interval.
  - b: Upper bound of the integration interval.
  - deg: Degree of the polynomial (number of Gauss points).
- **Returns:**
  - result: The integral of the function over [a, b].
- **Process:**
  - Obtain Gauss quadrature nodes and weights for the interval [-1, 1].
  - Transform the nodes to the interval [a, b].
  - Evaluate the function at the transformed nodes.

- Compute the integral approximation using the quadrature weights and function values.

4. Numerical Quadrature: numquad

```
def numquad(f, a, b, N):
  """
  Stub function for numquad. Replace with actual implementation.
  This function should return nvals and results.
  """
  nvals = linspace(2, N, N - 1, dtype=int)
  results = [quad_gauss(f, a, b, n) for n in nvals]
  return nvals, results
```

- **Parameters:**
  - f: The function to integrate.
  - a: Lower bound of the integration interval.
  - b: Upper bound of the integration interval.
  - N: The number of points for evaluation.
- **Returns:**
  - nvals: Array of the number of quadrature nodes.
  - results: Array of integration results for each number of nodes.
- **Process:**
  - Generate nvals, an array of integers from 2 to N.
  - Compute the Gauss quadrature integration results for each value in nvals.

5. Error Analysis: numquaderrs

```
def numquaderrs():

  """Numerical quadrature on [0, 1]"""
  N = 20

  plt.figure()
  f = lambda x: 1 / (1 + power(5.0 * x, 2))
  left = 0.0
  right = 1.0

  exact, e = integrate.quad(f, left, right)
  nvals, gaures = numquad(f, 0, 1, N)

  plt.loglog(nvals, abs(array(gaures) - exact), 'r+-', label='Gauss quadrature')
  plt.title('Numerical quadrature of function 1/(1+(5x)^2)')
  plt.xlabel('Number of quadrature nodes')
  plt.ylabel('|quadrature error|')
```

```
plt.legend(loc="lower left")
plt.show()

p = polyfit(log(nvals), log(abs(array(gaures) - exact)), 1)
print("Convergence order:", -p[0])

plt.figure()
f = lambda x: sqrt(x)
left = 0.0
right = 1.0

exact, e = integrate.quad(f, left, right)
nvals, gaures = numquad(f, 0, 1, N)

plt.loglog(nvals, abs(array(gaures) - exact), 'r+-', label='Gauss quadrature')
plt.axis([1, 25, 0.000001, 1])
plt.title('Numerical quadrature of function sqrt(x)')
plt.xlabel('Number of quadrature nodes')
plt.ylabel('|quadrature error|')
plt.legend(loc="lower left")
plt.show()

p = polyfit(log(nvals), log(abs(array(gaures) - exact)), 1)
print("Convergence order:", -p[0])
```

- **Process:**
  - Define the number of points N.
  - Create a plot to analyze the error for the function $f(x) = 1/1+5x^2$
    - Compute the exact integral using scipy.integrate.quad.
    - Compute the Gauss quadrature integration results for different numbers of nodes.
    - Plot the error on a log-log scale.
    - Fit a line to the log-log error data to determine the convergence order.
  - Repeat the above steps for the function $f(x) = \sqrt{x}$

6. Main Execution Block

```
if __name__ == "__main__":
    numquaderrs()
```

- **Process:**
  - Calls numquaderrs() to perform the error analysis and visualization.

Conclusion

EM215 Numerical Methods
Numerical Integration and Differentiation
Lab1/Assignment 1

This code provides a comprehensive method for performing numerical integration using Gauss quadrature. It includes functions to calculate quadrature nodes and weights, apply the Gauss quadrature method, and evaluate the error and convergence order of the numerical integration method. The code also includes visualization to help understand the performance of the integration method.