PHY1112: Lab 9

> Aim, ready, differentiate!

March 12th, 2024

Learning Objectives

1. Practice numerical differentiation and integration

Grade Breakdown

|  |  |  |  |
| --- | --- | --- | --- |
| Part | 1 | 2 | Total |
| Points | 12 | 16 |  |
| Score |  |  |  |

**Question 1: A Central Problem**

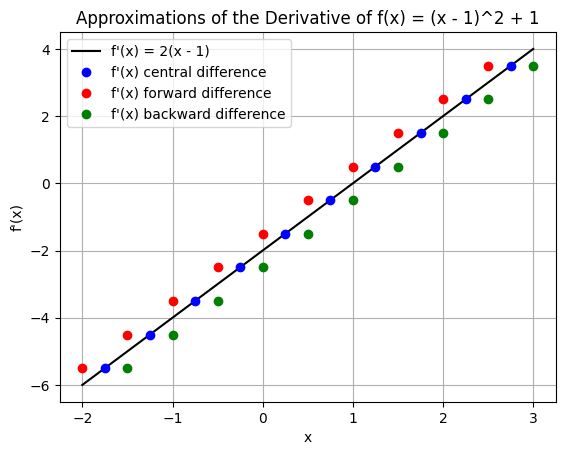
1. (5 points) Write your own custom vectorized python function called central\_difference\_derivative(), that calculates differentiation using the central difference algorithm that we learned in class:

The input of your custom function will be two arrays holding and data sets.

The output will be two arrays with a length that is shorter by 1 that will hold the calculated df/dx data as well as the positions x at which df/dx is evaluated.

1. (2 points). Create a data set for the function we considered in class, 𝑓(𝑥)= (𝑥−1)\*\*2+1.
2. (5 points) Pass the data set from part b) to your custom function of part a), central\_difference\_derivative(). Plot your results as markers, along with the analytical derivative. Comment on your results.

Hint: you can verify that you got the correct result by comparing to the graph shown in lecture.

****

**Figure 1.** A graph of the true derivative of and its approximate differences as defined by the central, forward, and backward differences. The derivative is represented by the black line, while the central difference is represented by the blue markers, forward difference is represented by the orange markers, and the backward difference is represented by the green markers.

**Question 2: A Sine You Should Be Studying**

In this question, we will tackle numerical integration of definite integrals.

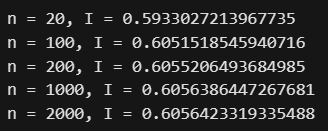
1. (5 points) Write your own vectorized trapezoidal-rule integration function to calculate the definite integral

where the inputs are the **function handle** for the integrand, the limits of integration and , and the number of partitions, . The output will be the value of .

1. (1 point) Consider the integral

What is the expected result for this integral? Do this by hand.

1. (5 points) Use your vectorized trapezoidal-rule integration function to perform numerical integration of the integral in part b)for equal to 20, 100, 200, 1000 and 2000.



1. (5 points) Plot the values of the integral versus that you obtained in part c), as well as a horizontal line at the true value of the integral (obtained from part b)).

What can you say about the convergence toward the correct answer? Relate this to what you know about the order of error.

**A graph with numbers and symbols

Description automatically generated**

**Figure 2.** A plot of the definite integral from 0 to π for with its value by the trapezoidal rule for n = 20, 100, 200, 1000, 2000. The true value of the definite integral is represented by a red horizontal line.

As the number of subintervals (n) increases, the value of the trapezoidal rule converges towards the true value of the definite integral. In terms of the order of error, increasing the number of subintervals decreases the order of error; with a higher n value, there will be more correct decimal places.

**CODE:**

'''

Filename:       lab9.py

Author:         Patrick Geraghty

Date Created:   2024-03-12

Date Modified:  2024-03-12

Description:    Contains the functions central\_diff, forward\_diff\_vectorized, backward\_diff\_vectorized, diff\_plot, trapezoidal\_rule, sin\_function, rule\_test, and trapz\_plot for lab 9.

'''

import numpy as np

import matplotlib.pyplot as plt

# Part 1

*def* central\_diff(*x*, *y*):

    '''

    (np.array, np.array) -> np.array, np.array

    Uses the central difference method to approximate the derivative of f at x.

    Preconditions: x and y are np.arrays of the same length.

    '''

    # Compute the differences in x and y using the same indices seen in class

    df = y[1:] - y[:-1]

    dx = x[1:] - x[:-1]

    # Return the x values and the approximations of the derivative of f at x

    # The x values are shifted by 0.5 \* dx to align with the central difference method

    return x[:-1] + (0.5 \* dx), df/dx

# Define the forward and backward difference methods for comparison

*def* forward\_diff\_vectorized(*x*,*y*):

    '''

    (np.array, np.array) -> np.array, np.array

    Uses the forward difference method to approximate the derivative of f at x.

    '''

    df = y[1:] - y[:-1]

    dx = x[1:] - x[:-1]

    return x[:-1], df/dx

*def* backward\_diff\_vectorized(*x*,*y*):

    '''

    (np.array, np.array) -> np.array, np.array

    Uses the backward difference method to approximate the derivative of f at x.

    '''

    dx = x[1:] - x[:-1]

    df = y[1:] - y[:-1]

    return x[1:], df/dx

# plot the approximations of the derivative of f(x) = (x - 1)^2 + 1

*def* diff\_plot():

    # initialize the figure

    plt.figure(1)

    # specify the x and y values

    # set the x values to align with those seen in class for easy comparison

    x = np.linspace(-2, 3, *num*=11)

    y = np.square(x - 1) + 1

    # compute the true derivative of f at x

    dy = 2 \* (x - 1)

    # compute the approximations of the derivative of f at x using the central difference method, forward difference method, and backward difference method

    x1, y1 = central\_diff(x, y)

    x2, y2 = forward\_diff\_vectorized(x, y)

    x3, y3 = backward\_diff\_vectorized(x, y)

    # plot the true derivative of f at x and the approximations of the derivative of f at x using the central difference method, forward difference method, and backward difference method

    plt.plot(x, dy, 'k', *label*='f\'(x) = 2(x - 1)')

    plt.plot(x1, y1, 'bo', *label*='f\'(x) central difference')

    plt.plot(x2, y2, 'ro', *label*='f\'(x) forward difference')

    plt.plot(x3, y3,'go', *label*='f\'(x) backward difference')

    # add a title, x-axis label, y-axis label, and legend to the plot

    plt.title('Approximations of the Derivative of f(x) = (x - 1)^2 + 1')

    plt.xlabel('x')

    plt.ylabel('f\'(x)')

    plt.grid()

    plt.legend()

    plt.show()

# Part 2

*def* trapezoidal\_rule(*f*, *a*, *b*, *n*):

    '''

    (function, float, float, int) -> float

    Approximate the definite integral of the function f(x) over the interval [a, b] using the trapezoidal rule with n subintervals.

    Preconditions: f is a function, a and b are numbers, and n is an int.

    '''

    # Compute the width of each subinterval

    dx = (b - a) / n

    # Compute the values of the function at the left and right endpoints of each subinterval

    x = np.linspace(a, b, *num*= n + 1)

    y = f(x)

    # Compute the area of each trapezoid and sum them to approximate the definite integral of f over [a, b] using the trapezoidal rule formula

    return dx \* (np.sum(y) - 0.5 \* (y[0] + y[-1]))

# define the function f(x) = sin(pi \* x)

*def* sin\_function(*x*):

    '''

    (float) -> float

    Returns the sine of pi \* x.

    '''

    return np.sin(np.pi \* x)

# test the trapezoidal\_rule function

*def* rule\_test():

    '''

    () -> None

    Tests the trapezoidal\_rule function with the sin\_function.

    '''

    print('n = 20, I =', trapezoidal\_rule(sin\_function, 0, np.pi, 20))

    print('n = 100, I =', trapezoidal\_rule(sin\_function, 0, np.pi, 100))

    print('n = 200, I =', trapezoidal\_rule(sin\_function, 0, np.pi, 200))

    print('n = 1000, I =', trapezoidal\_rule(sin\_function, 0, np.pi, 1000))

    print('n = 2000, I =', trapezoidal\_rule(sin\_function, 0, np.pi, 2000))

# plot the true value of the definite integral from 0 to pi of sin(pi \* x) with its trapezoidal rule values for n = 20, 100, 200, 1000, and 2000

*def* trapz\_plot():

    '''

    () -> None

    Plots the true value of the definite integral from 0 to pi of sin(pi \* x) with its trapezoidal rule values for n = 20, 100, 200, 1000, and 2000.

    Preconditions: None

    '''

    # initialize the figure

    plt.figure(2)

    # plot the trapezoidal rule values for n = 20, 100, 200, 1000, and 2000

    plt.plot(20, trapezoidal\_rule(sin\_function, 0, np.pi, 20), 'bx', *label*=*f*'n = 20, I = {trapezoidal\_rule(sin\_function, 0, np.pi, 20) *:.6f*}', *markersize*=10)

    plt.plot(100, trapezoidal\_rule(sin\_function, 0, np.pi, 100), 'rx', *label*=*f*'n = 100, I = {trapezoidal\_rule(sin\_function, 0, np.pi, 100) *:.6f*}', *markersize*=10)

    plt.plot(200, trapezoidal\_rule(sin\_function, 0, np.pi, 200), 'gx', *label*=*f*'n = 200, I = {trapezoidal\_rule(sin\_function, 0, np.pi, 200) *:.6f*}', *markersize*=10)

    plt.plot(1000, trapezoidal\_rule(sin\_function, 0, np.pi, 1000), 'kx', *label*=*f*'n = 1000, I = {trapezoidal\_rule(sin\_function, 0, np.pi, 1000) *:.6f*}', *markersize*=10)

    plt.plot(2000, trapezoidal\_rule(sin\_function, 0, np.pi, 2000), 'mx', *label*=*f*'n = 2000, I = {trapezoidal\_rule(sin\_function, 0, np.pi, 2000) *:.6f*}', *markersize*=10)

    # plot the true value of the definite integral from 0 to pi of sin(pi \* x) as a horizontal line

    plt.axhline(*y*=(1/np.pi) \* (-np.cos(np.square(np.pi)) + 1), *color*='r', *linestyle*='-')

    # add a title, x-axis label, y-axis label, and legend to the plot

    plt.title('Trapezoidal Rule Approximations of the Definite Integral of sin(pi \* x) from 0 to pi')

    plt.xlabel('n')

    plt.ylabel('I')

    plt.grid()

    plt.legend()

    plt.show()