Math104A Homework1

January 14, 2025

Brandon Su

1. Review and state the following theorems of Calculus:

   (a) The Intermediate Value Theorem.
   If $f \in C^1([a,b])$, we denote $M = \max_{x\in[a,b]} f(x)$ and $m = \min_{x\in[a,b]} f(x)$, for any $y \in [m, M]$, there exists $c \in [a,b]$ $s.t.$ $f(c) = y$.

   (b) The Mean Value Theorem.
   If $f \in C^1([a,b])$, there exists a number $c \in (a,b)$ $s.t.$ $f'(c) = \frac{f(b)-f(a)}{b-a}$.

   (c) Rolle's Theorem.
   If $f \in C^1([a,b])$ and $f(a) = f(b) = 0$, then there exists $c \in (a,b)$ $s.t.$ $f'(c) = 0$.

   (d) The Mean Value Theorem for Integrals.
   If f is continuous on $[a,b]$, then there exists a $c \in (a,b)$ $s.t.$

   $$f(c) = \frac{1}{b-a}\int_a^b f(x)dx$$

   (e) The Weighted Mean Value Theorem for Integrals.
   If f and g are continuous on $[a,b]$, and g does not change sign on $[a,b]$, then there exists an $\eta \in (a,b)$ $s.t.$

   $$\int_a^b f(x)g(x)dx = f(\eta)\int_a^b g(x)dx$$

Name: Brandon Su
Time for last modification: Oct. 12, 2024

2. Write a computer code to implement the Composite Trapezoidal Rule quadrature

$$T_h[f] = h\left(\frac{1}{2}f(x_0) + f(x_1) + ... + f(x_{N-1}) + \frac{1}{2}f(x_N)\right), \tag{1}$$

to approximate the definite integral

$$I[f] = \int_a^b f(x)dx, \tag{2}$$

using the equally spaced points $x_0 = a,\ x_1 = x_0 + h,\ x_2 = x_0 + 2h, \dots, x_N = b,$ where $h = (b-a)/N$. Make sure that **all your codes** have a preamble which describes the purpose of the code, all the input variables, the expected output, your name, and the date of the last time you modified the code.

[2]:
```
import numpy as np
import matplotlib as plt
import math
import scipy

'''
Preamble:
Purpose: This part's code will be used to approximate the define integral by␣
↪using the Composite Trapezoidal Rule by defining a function called "ctr".
Inputs:
    f: the function to be integrated
    a: the start of the defined interval, equal to x_0
    b: the end of the defined interval, equal to x_N
    N: the number of subintervals
Expected output:
    The approximated integration of f(x) over the interval [a, b].
'''

def ctr(f, a, b, N):
    h = (b - a)/N
    summation = 0.5 * (f(a) + f(b))
    for i in range(1, N):
        summation += f(a + i * h)
    return summation * h
```

3. To test your code, take $f(x) = xe^{x^2}$ in $[0, 1]$, compute the error $|I[f] - T_h[f]|$ for $h = 1/10, 1/20, 1/40$, and verify that T_h has a convergent trend at the expected quadratic rate.

[3]:
```
'''
Preamble:
Purpose: This part's code will be used to test the approximation method␣
↪implemented in question 2 by a specific function f(x) = x * e^(x^2) on the␣
↪interval [0, 1] and compute the error for different h values.
Inputs:
    f(x): the function that we use to test
    h: length of subintervals
Expected output:
    The errors for approximation for different h values.
    The rate of convergence.
'''

def f(x):
    return x * np.exp(x ** 2)

I_f = 1/2 * (np.exp(1) - 1)
```

```
h_s = [1/10, 1/20, 1/40]
errors = []

for h in h_s:
    N = int(1 / h)
    T_h = ctr(f, 0, 1, N)
    error = abs(I_f - T_h)
    errors.append(error)
    print(f"When h = {h}, T_h = {T_h: .7f}, and the error E_h is {error: .7f}.")
```

```
When h = 0.1, T_h =  0.8650898, and the error E_h is  0.0059489.
When h = 0.05, T_h =  0.8606307, and the error E_h is  0.0014897.
When h = 0.025, T_h =  0.8595135, and the error E_h is  0.0003726.
```

[4]:
```
for i in range(1, len(errors)):
    convergence = errors[i - 1] / errors[i]
    print(f"When h = {h_s[i-1]} becomes h = {h_s[i]}, the rate of convergence␣
 ↪is {convergence: .7f}.")
```

```
When h = 0.1 becomes h = 0.05, the rate of convergence is  3.9932346.
When h = 0.05 becomes h = 0.025, the rate of convergence is  3.9983023.
```

From the above three different simulations of h, we finds that when $h \to 0$, the error $E_h \to 0$ as well. Also, we calculated that the rate of convergence is approximately 4, indicating that T_h exhibits a convergent trend at the expected quadratic rate.

4. Consider the definite integral

$$I[e^{-x^2}] = \int_0^1 e^{-x^2} dx, \tag{3}$$

We cannot calculate its exact value but we can compute accurate approximations to it using $T_h[e^{-x^2}]$. Let

$$q(h) = \frac{T_{h/2}[e^{-x^2}] - T_h[e^{-x^2}]}{T_{h/4}[e^{-x^2}] - T_{h/2}[e^{-x^2}]}. \tag{4}$$

Using your code, find a value of h for which $q(h)$ is approximately equal to 4. (a) Get an *approximation* of the error, $I[e^{-x^2}] - T_h[e^{-x^2}]$, for that particular value of h. (b) Use this error approximation to obtain the *extrapolated*, improved, approximation

$$S_h[e^{-x^2}] = T_h[e^{-x^2}] + \frac{4}{3}\left(T_{h/2}[e^{-x^2}] - T_h[e^{-x^2}]\right). \tag{5}$$

Explain why $S_h[e^{-x^2}]$ is more accurate and converges faster to $I[e^{-x^2}]$ than $T_h[e^{-x^2}]$.

[5]:
```python
'''
Preamble:
Purpose: This part's code first set up the function e^(-x^2) and find its exact␣
 ↪value over the interval [0,1]. Then we are going to find an h that makes␣
 ↪q(h) equal to 4. After that, we will get an approximation of the error.␣
 ↪Finally, we will obtain the extrapolated and improved approximation S_h.
Inputs:
    f(x): the function that's going to be integrated
    h: length of subintervals
Expected output:
    h value for which q(h) is approximately equal to 4
    Improved approximation: S_h
'''

def f(x):
    return np.exp(-x**2)

def q(h):
    T_h2 = ctr(f, 0, 1, int(1 /(h * 1/2)))
    T_h = ctr(f, 0, 1, int(1 / h))
    T_h4 = ctr(f, 0, 1, int(1 / (h * 1/4)))
    return (T_h2 - T_h) / (T_h4 - T_h2)

all_h = np.linspace(0.001, 0.1, 1000)
h_4 = None
for h in all_h:
    if abs(q(h) - 4) < 0.000001:
        h_4 = h
        break

print(f"The value of h for q(h) is approximately equal to 4 is: {h_4}.")
```

```
The value of h for q(h) is approximately equal to 4 is: 0.0013963963963963964.
```

[7]:
```python
a = 0
b = 1
I = scipy.integrate.quad(f, a, b)[0]
T_h = ctr(f, 0, 1, int(1 / h_4))
Error = I - T_h
print(f"The exact integral we get is {I}, and the approximation by T_h is␣
 ↪{T_h}, so, the approximation error is {Error}.")
```

```
The exact integral we get is 0.7468241328124271, and the approximation by T_h is
0.7468240132132328, so, the approximation error is 1.1959919432591448e-07.
```

[10]:
```python
def S(h):
    return (ctr(f, 0, 1, int(1/h)) + 4/3 * (ctr(f, 0, 1, int(1/(h/2))) - ctr(f,␣
 ↪0, 1, int(1 / h))))
```

```python
S_h = S(h_4)
print(f"The extrapolated, improved appromixation S_h is {S_h}.")
```

```
The extrapolated, improved appromixation S_h is 0.7468241328124309.
```

$S_h[e^{-x^2}]$ is more accurate because it applies the Richardson Extrapolation, which improves upon T_h at two different step sizes: h and $h/2$. This process cancels out a large part of the error in $T_h[e^{-x^2}]$. And $S_h[e^{-x^2}]$ converges faster to $I[e^{-x^2}]$ because its error decreases at the rate of $o(h^4)$, which means the error decreases much faster than the error for the Composite Trapezoidal Rule (which behaves like $o(h^2)$).