Homework 2 MA 574 - Python

Numerical Differentiation

How to find the derivative of a function numerically?

(https://getlin

. Forward Difference Formula

$$f'(a) \approx \frac{f(a+h) - f(a)}{h}$$

. Backward Difference Formula

$$f'(a) \approx \frac{f(a) - f(a-h)}{h}$$

. Central Difference Formula

$$f'(a) \approx \frac{f(a+h) - f(a-h)}{2h}$$

See [this book (https://pythonnumericalmethods.berkeley.edu/notebooks/chapter20.02-Finite-Difference-Approximating-Derivatives.html)](https://pythonnumericalmethods.berkeley.edu/notebooks/chapter20.02-Finite-Difference-Approximating-Derivatives.html) for more details and Python implementation.

In [23]:
```
# Forward Difference Formula
def fdiff(f,a,h):
    derivative_forward_diff = (f(a+h)-f(a))/h
    return derivative_forward_diff
```

In [ ]:
```
def bdiff(f,a,h):
    derivative_backward_diff = (f(a)-f(a-h))/h
    return derivative_backward_diff
```

In [ ]:
```
def cdiff(f,a,h):
    derivative_central_diff = (f(a+h)-f(a-h))/(2*h)
    return derivative_central_diff
```

Numerical Integration

General Riemann Sum

$$\boxed{\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} f(x_i^*)\,\Delta x,}$$

(https://getlin

where x_i^* is the leftmost point of the i-th subinterval for left Riemann sum, rightmost point for right Riemann sum and middle-point for the mid-point sum.

In [ ]:
```

```

In [1]:
```
'''
Following function could be used to find all kinds of Riemann Sums by adjuting the shift
argument. Set it = 0 for left sum, =1 for right sum and 0.5 for mid point sum.
'''
def RiemannSum(f,a,b,n,shift=0):
    if shift < 0 or shift > 1:
        print("Please provide appropriate value for the shift  from 0 to 1.0.")
        return

    deltax = (b-a)/n
    sum=0.0
    a = a+shift*deltax
    for i in range(n):
        sum = sum + f(a+i*deltax)
    return sum*deltax
```

In [10]:
```
## Example
f = lambda x: 3*x**2
L40 = RiemannSum(f,0,2,40, shift=1.0)
print(L40)
```

```
8.302500000000002
```

Trapezoidal Rule

The trapezoidal rule for numerical approximation of a definite integral could be thought as the average fo the leftand right Riemann sums. (https://getlin

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n-1} \frac{f(x_i) + f(x_{i+1})}{2} \Delta x.$$

Or more efficiently as

$$\boxed{\int_a^b f(x)dx \approx \frac{\Delta x}{2}\left(f(x_0) + 2\left(\sum_{i=1}^{n-1} f(x_i) \right) + f(x_n) \right).}$$

In [11]:
```
def Trapezoidal(f,a,b,n):
    deltax = (b-a)/n
    sum=0.0
    for i in range(1,n):
        sum = sum + f(a+i*deltax)
    sum = f(a)+2*sum+f(b)
    return sum*deltax/2
```

In [12]:
```
## Examples
f = lambda x: 3*x**2
T40 = Trapezoidal(f,0,2,40)
print(T40)
```

```
8.002500000000003
```

Simpson's $1/3$-rule

Number of subintervals n of $[a, b]$ must be even. Let $n = 2m$, then

$$\boxed{\int_a^b f(x)dx \approx \frac{\Delta x}{3}\left[f(x_0) + 4\left(\sum_{i=1}^{m} f(x_{2i-1})\right) + 2\left(\sum_{i=1}^{m-1} f(x_{2i})\right) + f(x_{2m})\right].}$$

If you are interested to find more about Simpson's method and its implimentation in Python, check this online book (https://pythonnumericalmethods.berkeley.edu/notebooks/chapter21.04-Simpsons-Rule.html).

(https://getlin

Question 1 Consider the sigmoid function given by $f(x) = \tanh(x)$. Create a tabular comparison of its exact derivative at $x = 0.5$ with the approximations by the three numerical differentiation formulas provided above for $h = 0.1,\ 0.01,\ 0.001,\ 0.0001,\ 0.00001$.

(https://getlin

In [ ]:

```
# Your Code Here
import numpy as np

# Define the sigmoid function
def sigmoid(x):
    return np.tanh(x)

# Define the exact derivative of the sigmoid function
def exact_derivative(x):
    return 1 - np.tanh(x)**2

# Value of x where we want to calculate the derivative
x = 0.5

# List of different step sizes (h values)
h_values = [0.1, 0.01, 0.001, 0.0001, 0.00001]

# Create a table to display the results
print("h\tExact Derivative\tForward Diff Approx\tCentral Diff Approx\tBackward Diff Approx")
print("-" * 80)

# Calculate and display results for each h value
for h in h_values:
    # Forward difference approximation
    forward_diff_approx = (sigmoid(x + h) - sigmoid(x)) / h

    # Central difference approximation
    central_diff_approx = (sigmoid(x + h) - sigmoid(x - h)) / (2 * h)

    # Backward difference approximation
    backward_diff_approx = (sigmoid(x) - sigmoid(x - h)) / h

    # Calculate the exact derivative
    exact = exact_derivative(x)

    # Display results in the table
    print(f"{h}\t{exact:.6f}\t{forward_diff_approx:.6f}\t{central_diff_approx:.6f}\t{backward_diff_approx:.6f
```

(https://getlin

Question 2 Understand the example of implementation of Trapezoidal rule. Modify this code to find the definite integral by Simpson's rule.

Evaluate the integral of $\int_{-1}^{1} \frac{1}{1+x^2}$ exactly and compare this with the approximations by

1. Mid-point Riemann sum.
2. Trapezoidal Rule
3. Simpson's Rule

Take n=20 above.

(https://getlin

(https://getlin

In [ ]:

```
# Your Code Here
import numpy as np

# Define the function to be integrated
def f(x):
    return 1 / (1 + x**2)

# Define the number of subintervals (n)
n = 20

# Define the interval [a, b]
a = -1
b = 1

# Calculate the step size (h)
h = (b - a) / n

# Initialize sums for Simpson's Rule, Mid-point Riemann sum, and Trapezoidal Rule
integral_simpson = 0
integral_midpoint_riemann = 0
integral_trapezoidal = 0

# Calculate the integral using Simpson's Rule
for i in range(n + 1):
    x_i = a + i * h

    if i == 0 or i == n:
        integral_simpson += f(x_i)
    elif i % 2 == 1:
        integral_simpson += 4 * f(x_i)
    else:
        integral_simpson += 2 * f(x_i)

integral_simpson *= h / 3

# Calculate the integral using Mid-point Riemann sum
for i in range(n):
    x_midpoint = a + (i + 0.5) * h
    integral_midpoint_riemann += f(x_midpoint)

integral_midpoint_riemann *= h
```

(https://getlin

```
# Calculate the integral using Trapezoidal Rule
for i in range(n + 1):
    x_i = a + i * h

    if i == 0 or i == n:
        integral_trapezoidal += f(x_i) / 2
    else:
        integral_trapezoidal += f(x_i)

integral_trapezoidal *= h

# Calculate the exact integral (can be computed symbolically)
exact_integral = np.arctan(b) - np.arctan(a)

# Print the results
print("Exact Integral:", exact_integral)
print("Simpson's Rule Approximation:", integral_simpson)
print("Mid-point Riemann Sum Approximation:", integral_midpoint_riemann)
print("Trapezoidal Rule Approximation:", integral_trapezoidal)
```

(https://getlin