Homework 2 MA 574 - Python

Numerical Differentiation

How to find the derivative of a function numerically?

. 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 for more details and Python implementation.

In [4]:
```
# Forward Difference Formula
def fdiff(f,a,h):
    derivative_forward_diff = (f(a+h)-f(a))/h
    return derivative_forward_diff
```

In [5]:
```
def bdiff(f,a,h):
    derivative_backward_diff = (f(a)-f(a-h))/h
    return derivative_backward_diff
```

In [6]:
```
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

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

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 [7]:

```
'''
Following function could be used to find all kinds of Riemann Sums by adjuting the 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,0,2,600, shift=1.0)
print(L40)
```

```
8.020011111111119
```

Trapezoidal Rule

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

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

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.

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 $h = 0.1,\ 0.01,\ 0.001,\ 0.0001,\ 0.00001.$

In [88]:
```
import math
from tabulate import tabulate
import pandas as pd

hList = [0.1, 0.01, 0.001, 0.0001, 0.00001]
dictDiff = {"forwardDifference":[],"backwardDifference":[],"centralDifference":[]}

def fdiff(f,a,h):
    derivative_forward_diff = (math.tanh(a+h)-math.tanh(a))/h
    return derivative_forward_diff

def bdiff(f,a,h):
    derivative_backward_diff = (math.tanh(a)-math.tanh(a-h))/h
    return derivative_backward_diff

def cdiff(f,a,h):
    derivative_central_diff = (math.tanh(a+h)-math.tanh(a-h))/(2*h)
    return derivative_central_diff

for h in hList:
#     print(fdiff(f,0.5,h))
#     print(bdiff(f,0.5,h))
#     print(cdiff(f,0.5,h))
    dictDiff["forwardDifference"].append(fdiff(f,0.5,h))
    dictDiff["backwardDifference"].append(fdiff(f,0.5,h))
    dictDiff["centralDifference"].append(fdiff(f,0.5,h))
# print(dictDiff)

# df = pd.DataFrame(dictDiff)
# df.index = hList
# df.index.name = "h"

# print(df)


mydata = [
    ["0.1", dictDiff["forwardDifference"][0],dictDiff["backwardDifference"][0],dic
    ["0.01", dictDiff["forwardDifference"][1],dictDiff["backwardDifference"][1],di
    ["0.001", dictDiff["forwardDifference"][2],dictDiff["backwardDifference"][2],d
    ["0.0001", dictDiff["forwardDifference"][3],dictDiff["backwardDifference"][3],
    ["0.00001", dictDiff["forwardDifference"][4],dictDiff["backwardDifference"][4]
]
# create header
head = ["H Value","Forward Difference", "Backward Difference","Central Difference"
```

```
# display table
print(tabulate(mydata, headers=head, tablefmt="grid"))
```

```
+-----------+----------------------+----------------------+----------------------
+
|   H Value |   Forward Difference |   Backward Difference |   Central Difference 
|
+===========+======================+======================+======================
+
|    0.1    |             0.749324 |             0.749324 |             0.749324 
|
+-----------+----------------------+----------------------+----------------------
+
|    0.01   |             0.782804 |             0.782804 |             0.782804 
|
+-----------+----------------------+----------------------+----------------------
+
|    0.001  |             0.786084 |             0.786084 |             0.786084 
|
+-----------+----------------------+----------------------+----------------------
+
|    0.0001 |             0.786411 |             0.786411 |             0.786411 
|
+-----------+----------------------+----------------------+----------------------
+
|    1e-05  |             0.786444 |             0.786444 |             0.786444 
|
+-----------+----------------------+----------------------+----------------------
+
```

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.

In [86]:
```
#Evaluate the integral using scipy (Referance - https://scipy.org/)
from tabulate import tabulate
import math
from scipy import integrate
limit = [-1,1]

def indegrand(x):
    return 1/(1+x**2)

ans = integrate.quad(indegrand,limit[0],limit[1])
print("Integration of 1/(1+x**2) from -1 to 1 is ----->",ans[0])

#Evaluate using Mid-point Riemann sum approximation

def RiemannSum(f,a,b,n,shift):
    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


f = lambda x: 1/(1+x**2)
RS = RiemannSum(f,limit[0],limit[1],20, shift=0.5)

#Evaluate using Trapezoidal Rule approximation
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

## Examples
f = lambda x: 1/(1+x**2)
T20 = Trapezoidal(f,limit[0],limit[1],20)

#Evaluate using Simpson Rule approximation


def Simpson(f,a,b,n):
    deltax = (b-a)/n
    m = n/2
    sum1=0.0;sum2 = 0.0;sum = 0.0
    for i in range(1,int(m)):
        sum1 = sum1 + f(a+((2*i)-1)*deltax)
        sum2 = sum2 + f(a+(2*i)*deltax)
    sum1 = sum1 + f(a+((2*m)-1)*deltax)
    sum = f(a)+4*sum1+2*sum2+f(b)
    return sum*(deltax/3)

f = lambda x: 1/(1+x**2)
S20 = Simpson(f,limit[0],limit[1],20)

# print(RS)
# print(T20)
# print(S20)

mydata = [
    ["Riemann sum approximation", RS],
    ["Trapezoidal Rule approximation", T20],
    ["Simpson's  1/3-rule", S20]
]
# create header
head = ["Name", "Approximation"]

# display table
print(tabulate(mydata, headers=head, tablefmt="grid"))
```

```
Integration of 1/(1+x**2) from -1 to 1 is -----> 1.5707963267948968
+--------------------------------+-----------------+
| Name                           |   Approximation |
+================================+=================+
| Riemann sum approximation      |         1.57121 |
+--------------------------------+-----------------+
| Trapezoidal Rule approximation |         1.56996 |
+--------------------------------+-----------------+
| Simpson's  1/3-rule            |         1.5708  |
+--------------------------------+-----------------+
```

In [83]:

In [ ]: