WEEK-4 LAB

Question 1:

```
y=[np.cos(math.pow(i,2))*2*i for i in x]
z=[(np.sin(math.pow(i+0.01,2))-np.sin(math.pow(i,2)))/0.01 for i in x]
ax.plot(x,y,"b",label="Actual value of the derivative")
ax.plot(x,z,"r",label="Forward finte difference approximation")
plt.grid(True,which="both")
plt.xticks(x)
ax.set_xlabel("x")
ax.set_ylabel("f'(x)")
ax.set_title('Plot of actual derivative and forward finite difference
of $sin(x^2)$',fontsize=7.5)
ax.legend()
plt.show()
```

- Forward difference formula is used to calculate the derivative of the function when h=0.01
- **y** holds the actual value whereas **z** holds the approximated value by forward difference and finally we plot it

Question 2:

```
y=[np.cos(math.pow(i,2))*2*i for i in x]
FD=[(np.sin(math.pow(i+0.01,2))-np.sin(math.pow(i,2)))/0.01 for i in x]
BD=[(np.sin(math.pow(i,2))-np.sin(math.pow(i-0.01,2)))/0.01 for i in x]
CD=[(np.sin(math.pow(i+0.01,2))-np.sin(math.pow(i-0.01,2)))/(2*0.01)
for i in x]
err_FD=[abs(y[i]-FD[i]) for i in range(len(x))]
err_BD=[abs(y[i]-BD[i]) for i in range(len(x))]
err_CD=[abs(y[i]-CD[i]) for i in range(len(x))]
```

- **Y** holds the actual value of the derivative**. FD, CD, BD** holds the approximations for forward, centered and backward finite difference formulas.
- The lists starting with the "**err**" holds the error terms for those methods
- Finally, we have plotted the error terms corresponding to different values of x

Question 3:

```
h = np.linspace(0.001, 0.01, 100)
x = np.linspace(0, 1, 100)
FD_max_err_list = []
CD_max_err_list = []
# print(x)
for i in h:
    err_list1 = [abs(np.cos(j ** 2) * 2 * j - (np.sin((j + i) ** 2) -
np.sin(j ** 2)) / i) for j in x]
    err_list2 = [abs(np.cos(j ** 2) * 2 * j- (np.sin(math.pow(j + i,
2)) - np.sin(math.pow(j - i, 2))) / (2 * i) )  for j in x]
    max_err1 = max(err_list1)
    max_err2=max(err_list2)
    FD_max_err_list.append(max_err1)
    CD_max_err_list.append(max_err2)
    err_list1.clear()
    err_list2.clear()
```

- In order to get the errors corresponding to a function of **h**. we have sampled 100 values from 0.001 to 0.01 in the list **h**.
- For all such **h'**s we have accumulated two lists namely **FD_max_err_list** and **CD_max_err_list** which contains the maximum error corresponding to different **h's**
- Finally, we have plotted the maximum error with respect to h on the x axis

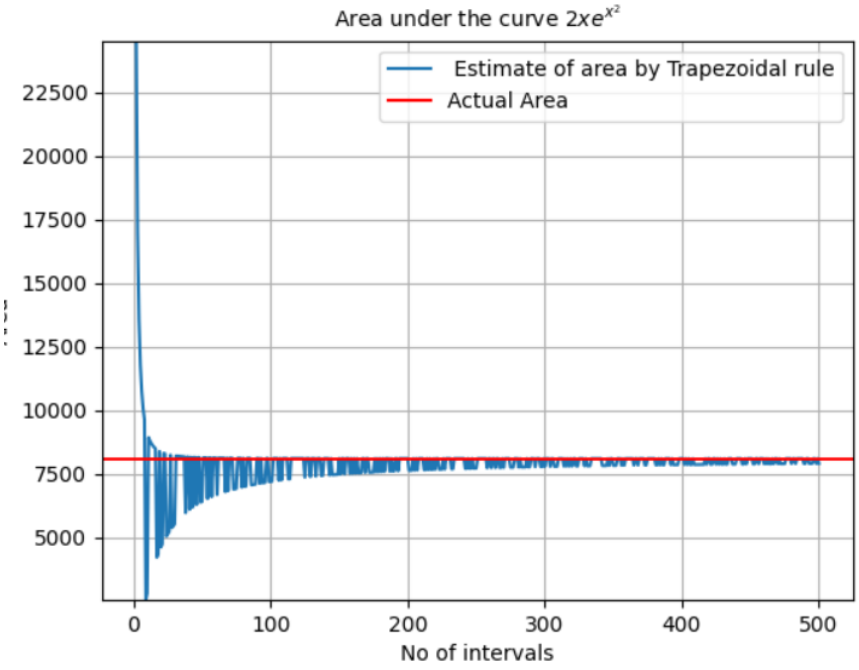
Question 4 :

```
M = 500
I_trap = []
I = 0
x = 1
for i in range(2, M + 1):
    x = 1
    h = 2 / i
    while x+h <= 3:

        I = I + ((2*x*np.e**(x**2))+(2*(x+h)*np.e**((x+h)**2)))/2

        x = x + h
    I_trap.append(h *I)
    I = 0
```

- We have specified a value of for the maximum no of intervals here **M=500.**
- To get the values of the integration with respect to different intervals we use the trapezoidal rule which estimates the value and stores it in **I_trap** list
- We plot the different values of the integration with respect to the no of intervals and notice as the no of intervals get bigger the accuracy of the estimate increases. Below is the sample output:



Question 5:

```
f= lambda x: 2*x*np.exp(x**2)
u=np.linspace(0,3,200)
#print(u)
for i in range(1,len(u)):
    I_quad=scipy.integrate.quad(f,0,u[i])
    x=np.linspace(0,u[i],100)
    y=f(x)
    I_trapz=scipy.integrate.trapz(y,x)

    I_simps=scipy.integrate.simps(y,x)
    I_real_list.append(np.exp(u[i]**2)-np.exp(1))
    I_trapz_list.append(I_trapz)
    I_simps_list.append(I_simps)
    I_quad_list.append(I_quad[0])
x=[u[i] for i in range(1,len(u))]
```

- Here we have divided the interval [0,3] into several small intervals of the form [0,u] where u varies and is less than equal to 3.
- We have used python's SciPy. Integrate module to evaluate the area under the curve in three different ways: - **I_quad_list** holds the area calculated by the quadrature method, I**_trapz_list** holds the area evaluated via trapezoidal method and **I_simps_list** hold s the area evaluated via Simpson's method

Question 6:

```
def derivative(self):
    derivative_list=[]
    for i in range(1,len(self.list_co)):
        derivative_list.append((i)*self.list_co[i])
    self.list_co=derivative_list
    return self
def area(self,a,b):
   self.a=a
   self.b=b
   Integration_list = [0]
   for i in range( len(self.list_co)):
       Integration_list.append(self.list_co[i]/(i+1))
   p=Polynomial(Integration_list)
   return "Area in the interval "+ str([self.a,self.b])+" is:
"+str((p[self.b]-p[self.a]))
```

- We have added two different methods **area and derivative** to the already created class Polynomial which basically uses the differentiation and integration formula of the polynomials to evaluate the co-efficient
- Note when finding the area, we use integration of the polynomial in the interval and hence we start the list with a 0 which is similar to the constant term to get when we integrate a function. However, the constant does not create any ambiguity when finding out the area as it will vanish when we subtract the upper limit value of the polynomial from the lower limit value.

Question 7:

```
# Approximate the integration with taylor series polynomial
fn = lambda n: 2 ** (n / 2) * np.sin(n * np.pi / 4)  # nth order
derivative of the function e^x sin(x) at x=0
x = [fn(i) / math.factorial(i) for i in range(7)]  # Initializing the
taylor polynomial up to  x^7
p = Polynomial(x)  # creating the polynomial
print(p.area(0, 1 / 2))  # calculating the area/integration with our
area method
```

- We approximate the integration with the help of Taylor series polynomial
- We calculate the derivative of the function at x=0 using **fn**
- **Creating the polynomial for Taylor series with degree up to 7 which guarantees the error stays below 10^{-6}**
- Using **area** method of the polynomial to calculate the value of the integration
- Below is the calculation of the error term.