Homework 1

In [1]:
```
import numpy as np
# nbviewer link >>
# https://nbviewer.jupyter.org/github/ShashankKumbhare/PHYS_5319_MM3/blob/master/Homewor
```

In [ ]:
```
```

Analytical Integration of xe^{-x}

$I_a = 1 - 6e^{-5}$
I_a = 0.959572318

In [2]:
```
I_a = 0.959572318
```

In [ ]:
```
```

Problem 1

Write a program to integrate xe^{-x} with x ranging [0, +5] by the trapezoid algorithm.
Choose the number of sub-intervals, N-1, to be 50, 100, 200, and 500.
And compare with the analytic value.

Solution Problem 1

Trapezoidal integration given by,

$$\int_a^b f(x)\,dx \approx \frac{1}{2}[f(a) + f(b)] + \sum_{i=1}^{N-1} f(x_i).h$$

In [3]:
```
##########################################################################
######################## trapezoidal #####################################
def trapezoidal(func_math, a, b, n, I_a):

    """
    General info:
        This function returns a numerical integration by Trapezoidal's rule of a mathema
    Arguments:
        func_math : a mathematical function that we want to integrate
        a         : lower limit
        b         : higher limit
        n         : number of subintervals is n (i.e. n = 0,1,2,....,n)
        I_a       : analytical integration (for comparision)
    """

    if n < 1:

        print("Not enough points for trapezoidal integration.")

    else:

        f_a = func(a)
        f_b = func(b)
        h   = (b-a) / n
        I_n = (f_a + f_b) / 2

        # Loop for adding n-1 terms >>
        for i in range(1, n):
            x_i = a + i*h
            I_n = I_n + func(x_i) * h

        # Printing results >>
        print("For n = ", n, ": I_n = ", I_n, ", I_n-I_a = ", I_n-I_a, ", (I_n-I_a)/I_a

    return None
######################## trapezoidal #####################################
##########################################################################

##########################################################################
######################## func ############################################
def func(x):
    return( x*np.exp(-x) )
######################## func ############################################
##########################################################################
```

Final Solution Problem 1

In [4]:
```
trapezoidal(func, 0, 5, 10, I_a)
trapezoidal(func, 0, 5, 20, I_a)
trapezoidal(func, 0, 5, 50, I_a)
trapezoidal(func, 0, 5, 100, I_a)
trapezoidal(func, 0, 5, 200, I_a)
trapezoidal(func, 0, 5, 500, I_a)
```

```
For n =  10 : I_n =  0.9468589481687687 , I_n-I_a =  -0.01271336983123128 , (I_n-I_a)/I
_a =  -0.013248996029532483
For n =  20 : I_n =  0.9668735702734836 , I_n-I_a =  0.0073012522734836605 , (I_n-I_a)/
I_a =  0.007608860881586687
For n =  50 : I_n =  0.9738773239697982 , I_n-I_a =  0.014305005969798223 , (I_n-I_a)/I
_a =  0.014907689291843685
For n =  100 : I_n =  0.9753610199952111 , I_n-I_a =  0.015788701995211096 , (I_n-I_a)/
I_a =  0.016453894822767382
For n =  200 : I_n =  0.9759425783783413 , I_n-I_a =  0.016370260378341328 , (I_n-I_a)/
I_a =  0.017059954806179944
For n =  500 : I_n =  0.9762401789385105 , I_n-I_a =  0.01666786093851047 , (I_n-I_a)/I
_a =  0.017370093557149146
```

In [ ]:

In [ ]:

Problem 2

Modify the code in Problem 1 to use Simpson algorithm. Repeat the calculations in Problem 1. List the results in the similar table.

Solution Problem 2

Simpson's integration given by,

$$\int_a^b f(x)\,dx \approx \frac{h}{3}\left[f(a) + 2\sum_{i=1}^{\frac{N}{2}-1} f(x_{2i}) + 4\sum_{i=1}^{\frac{N}{2}} f(x_{2i-1}) + f(b)\right]$$

In [5]:

```
################################################################################
######################## simpson ###############################################
def simpson(func_math, a, b, n, I_a):

    """
    General info:
        This function returns a numerical integration by Simpson's rule of a mathematica
    Arguments:
        func_math : a mathematical function that we want to integrate
        a         : lower limit
        b         : higher limit
        n         : number of subintervals is n (i.e. n = 0,1,2,....,n)
        I_a       : analytical integration (for comparision)
    """

    if n <= 1:

        print("Not enough points for trapezoidal integration.")

    else:

        f_a = func(a)
        f_b = func(b)
        h   = (b-a) / n
        I_n = (f_a + f_b) / 3

        # Sum of even terms >>
        for i in range(1, int(n/2)):
            x_2i = a + 2*i*h
            I_n  = I_n + 2/3 * func(x_2i) * h

        # Sum of odd terms >>
        for i in range(1, int(n/2) + 1):
            x_2i_1 = a + (2*i-1)*h
            I_n    = I_n + 4/3 * func(x_2i_1) * h

        # Printing results >>
        print("For n = ", n, ": I_n = ", I_n, ", I_n-I_a = ", I_n-I_a, ", (I_n-I_a)/I_a

    return None
######################## simpson ###############################################
################################################################################


################################################################################
######################## func ##################################################
def func(x):
    return( x*np.exp(-x) )
######################## func ##################################################
```

```
##################################################################################
```

Final Solution Problem 2

In [6]:
```
simpson(func, 0, 5, 50, I_a)
simpson(func, 0, 5, 100, I_a)
simpson(func, 0, 5, 200, I_a)
simpson(func, 0, 5, 500, I_a)
```

```
For n =  50 : I_n =  0.9696775676528738 , I_n-I_a =  0.010105249652873849 , (I_n-I_a)/I
_a =  0.010530993301198846
For n =  100 : I_n =  0.9702406295044438 , I_n-I_a =  0.010668311504443806 , (I_n-I_a)/
I_a =  0.011117777476823593
For n =  200 : I_n =  0.9705214753401462 , I_n-I_a =  0.010949157340146232 , (I_n-I_a)/
I_a =  0.011410455611065504
For n =  500 : I_n =  0.9706899303865668 , I_n-I_a =  0.011117612386566833 , (I_n-I_a)/
I_a =  0.0115860078266314
```

In [ ]:
```

```

In [ ]:
```

```

Problem 3

Write a program to integrate Problem 1 using Gauss-Legendre algorithm. Just use 10 points. You may need to use the subroutine ‘gauleg’ from Numerical Recipies to generate {xi, wi}.
Hint: a transformation is needed to change [0,5] to [-1,1].

Solution Problem 3

$$I = \int_a^b f(x) \, dx$$

Change of variable

$$x = \frac{b-a}{2} \cdot t + \frac{a+b}{2}$$

$$\therefore \; dx = \frac{a+b}{2} \cdot dt \,, \quad \text{as } x \to 0, \; t \to -1 \quad \& \text{ as } x \to 5, \; t \to +1$$

$$I = \left(\frac{b-a}{2}\right) \int_{-1}^{+1} f\left(x_i(t)\right) dt$$

$$I \approx \left(\frac{b-a}{2}\right) \sum_{i=1}^{N} w_i \cdot f\left(x_i(t)\right),$$

where $x(t) = \frac{b-a}{2} \cdot t + \frac{a+b}{2}$

In [7]:
```
##################################################################################
######################## gau_leg #################################################
def gau_leg(func_math, a, b, I_a):

    """
    General info:
        This function returns a numerical integration by Gauss-Legendre's rule of a math
    Arguments:
        func_math : a mathematical function that we want to integrate
        a         : lower limit
        b         : higher limit
        I_a       : analytical integration (for comparision)
    """

    # Gauss-Legendre quadrature abscissas >>
    t = np.array([-.1488743389, .1488743389, -.4333953941, .4333953941, -.6794095682, .6
                  -.8650633666, .8650633666, -.9739065285, .9739065285])

    # Gauss-Legendre quadrature weights >>)
    w = np.array([.2955242247, .2955242247, .2692667193, .2692667193, .2190863625, .2190
                  .1494513491, .1494513491, .0666713443, .0666713443])

    # After change of variable: x = (b-a)t/2 + (a+b)/2 >>
    x = (b-a)*t/2 + (a+b)/2

    # Loop for n-terms of w_i.f(x_i) >>
    I_n = 0
    for x_i, w_i in zip(x, w):
        I_n = I_n + w_i * func(x_i)
    I_n = (b-a)/2 * I_n

    print("I_n = ", I_n, ", I_n-I_a = ", I_n-I_a, ", (I_n-I_a)/I_a = ", (I_n-I_a)/I_a)

    return None
######################## gau_leg #################################################
##################################################################################

##################################################################################
######################## func ####################################################
def func(x):
    return( x*np.exp(-x) )
######################## func ####################################################
##################################################################################
```

Final Solution Problem 3

In [8]:
```
gau_leg(func, 0, 5, I_a)
```

```
I_n =  0.9595723179855976 , I_n-I_a =  -1.4402368186949843e-11 , (I_n-I_a)/I_a =  -1.50
09153470546284e-11
```

In [ ]:

In [ ]:

Problem 4

Discuss the error and efficiency (# of points,) of the three algorithms.

Final Solution Problem 3

As you can see from Solution to Problem 1, 2 & 3:

1. Gauss-Legendre algorithm is **the most accurate** & **the most efficient** even with **just 10 points**.
2. Simpson's algorithm lie on 2nd spot in terms of accuracy.
3. Trapezoidal's algorithm is worst among the 3 algorithms.
4. In terms of efficiency Trapezoidal's & Simpson's algorithm are not even close to Gauss-Legendre algorithm.

In [ ]:

In [ ]: