Computational and Numerical Methods Lab - 1


```
Abhimanyu Karia - 202201435
Devarshi Patel - 202201447
```


In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import Image
import sympy as sym
import math as mt
```

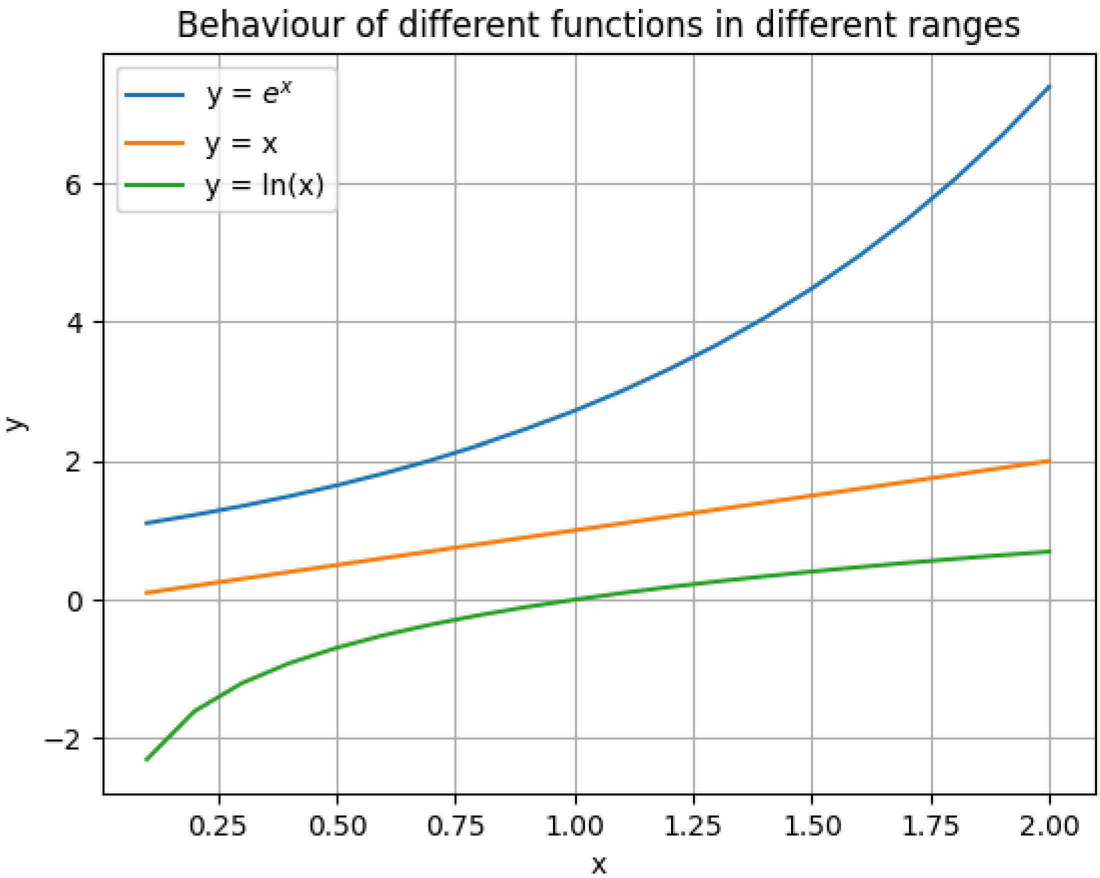
In [ ]:
```
h = 0.1
```

1. Basic Plotting Exercises

```
Q - 1
```

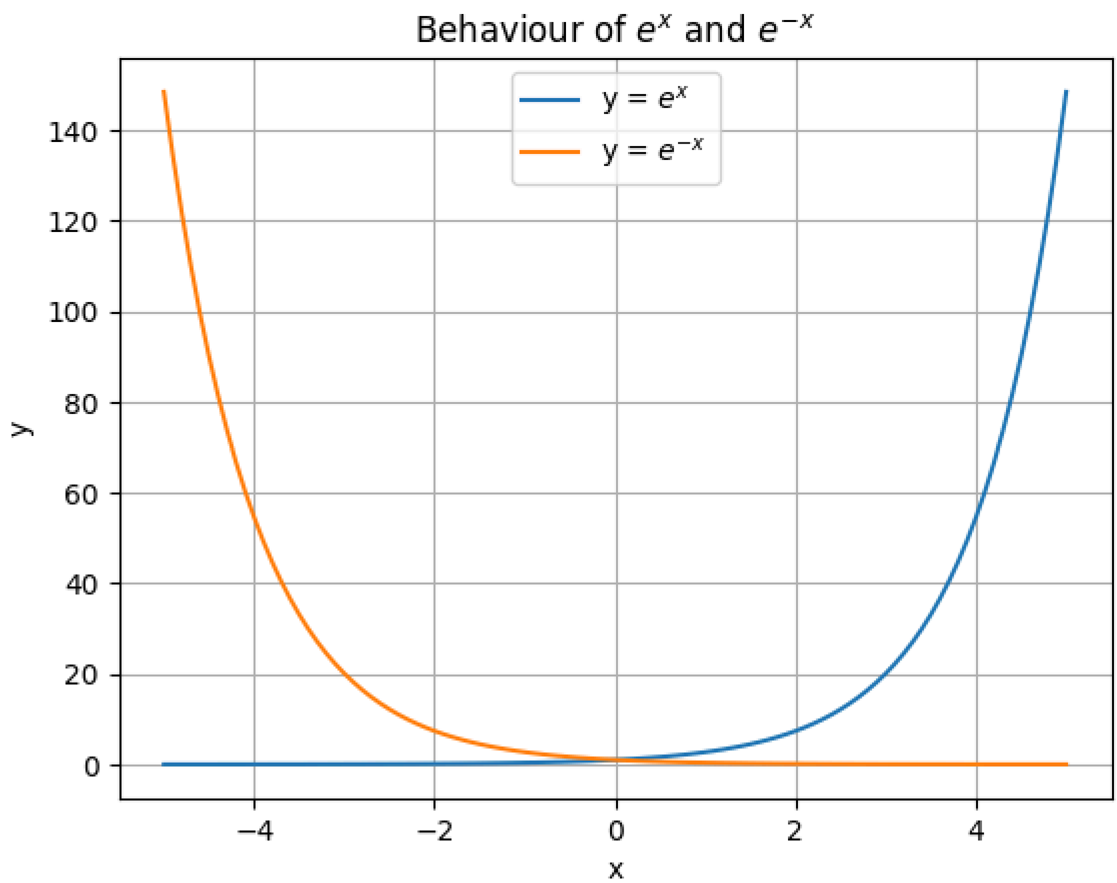
- Range of x = [0,2]
- Graphs of e^x, x, ln(x)
- From Graphs we see
  1. ln(x) lies below x and e^x lies above x.
  2. ln(x) and e^x are mirror images of each other with respect to y = x.

In [ ]:
```
x = np.arange(0+h,2+h,h)
y1 = np.exp(x)
y2 = x
y3 = np.log(x)


plt.plot(x,y1,label = 'y = $e^x$')
plt.plot(x,y2,label = 'y = x')
plt.plot(x,y3,label = 'y = ln(x)')
plt.legend()
plt.xlabel('x')
plt.ylabel('y')
plt.title('Behaviour of different functions in different ranges')
plt.grid(True)
plt.show()
```



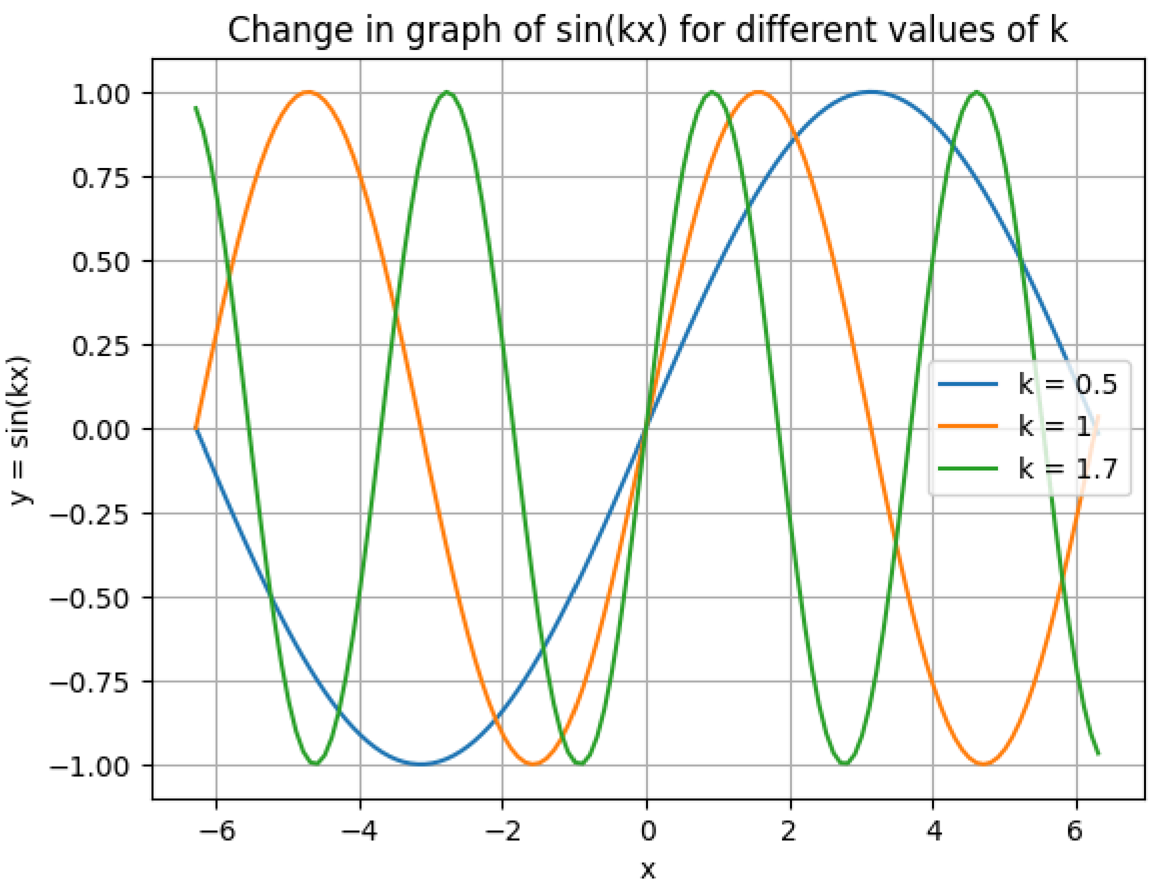
- Range of x = (-5,5)
- y = e^x and y = e^{-x} are mirror images or each other with respect to y-axis(x = 0)

In [ ]:
```
x = np.arange(-5,5+h,h)
y1 = np.exp(x)
y2 = np.exp(-x)
plt.plot(x,y1,label = 'y = $e^x$')
plt.plot(x,y2,label = 'y = $e^{-x}$')
plt.legend()
plt.xlabel('x')
plt.ylabel('y')
plt.title('Behaviour of $e^x$ and $e^{-x}$')
plt.grid(True)
plt.show()
```



Q - 2

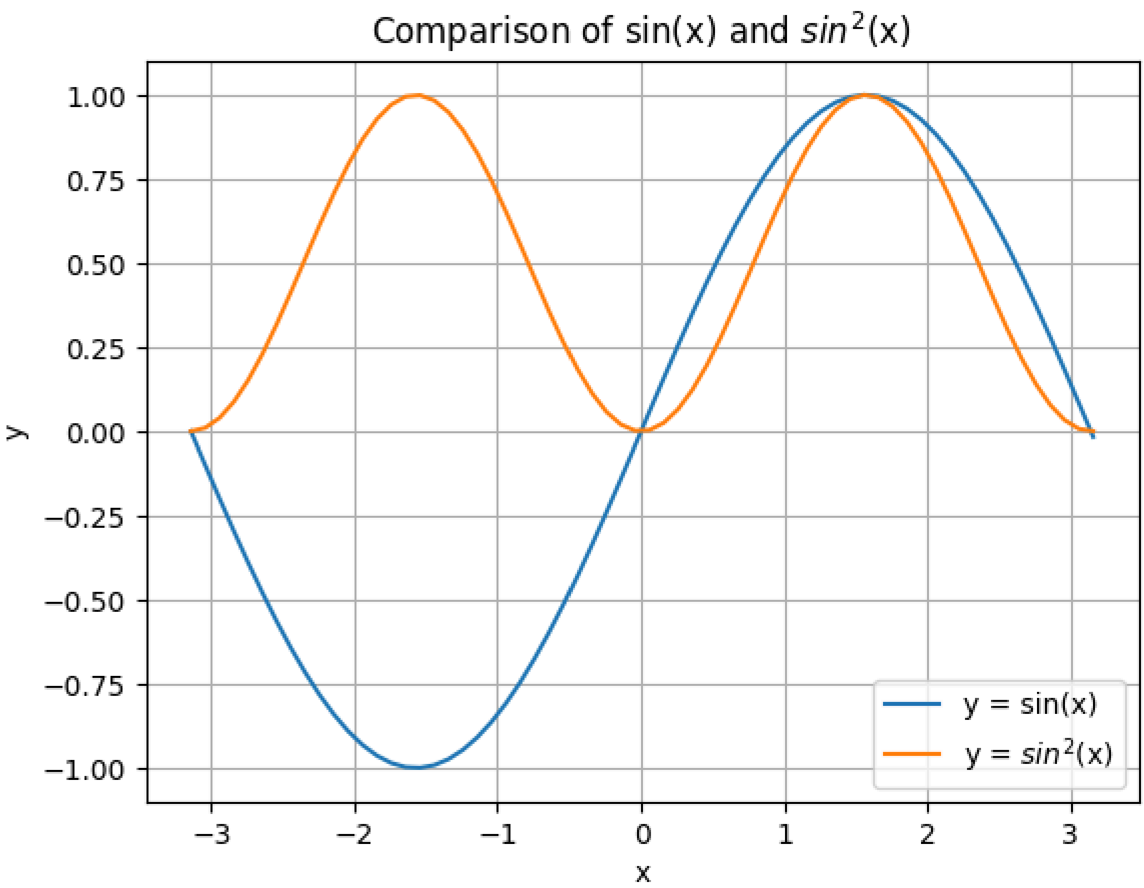
```
In [ ]: k_val = [0.5,1,1.7]
x = np.arange(-2*np.pi,2*np.pi + h,h)
for k in k_val:
    plt.plot(x,np.sin(k*x),label = 'k = ' + str(k))
plt.legend()
plt.grid(True)
plt.xlabel('x')
plt.ylabel('y = sin(kx)')
plt.title('Change in graph of sin(kx) for different values of k')
plt.show()
```



From the Graph we can conclude following points:

- sin(kx) graph expands along x axis for k < 1.
- sin(kx) graph contracts along x-axis for k > 1.
- The Frequency of graph changes by a factor of 1/k.

In [ ]:
```
x = np.arange(-np.pi,np.pi + h,h)
y1 = np.sin(x)
y2 = np.sin(x)**2

plt.plot(x,y1,label = 'y = sin(x)')
plt.plot(x,y2,label = 'y = $sin^2$(x)')
plt.legend()
plt.grid(True)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Comparison of sin(x) and $sin^2$(x)')
plt.show()
```



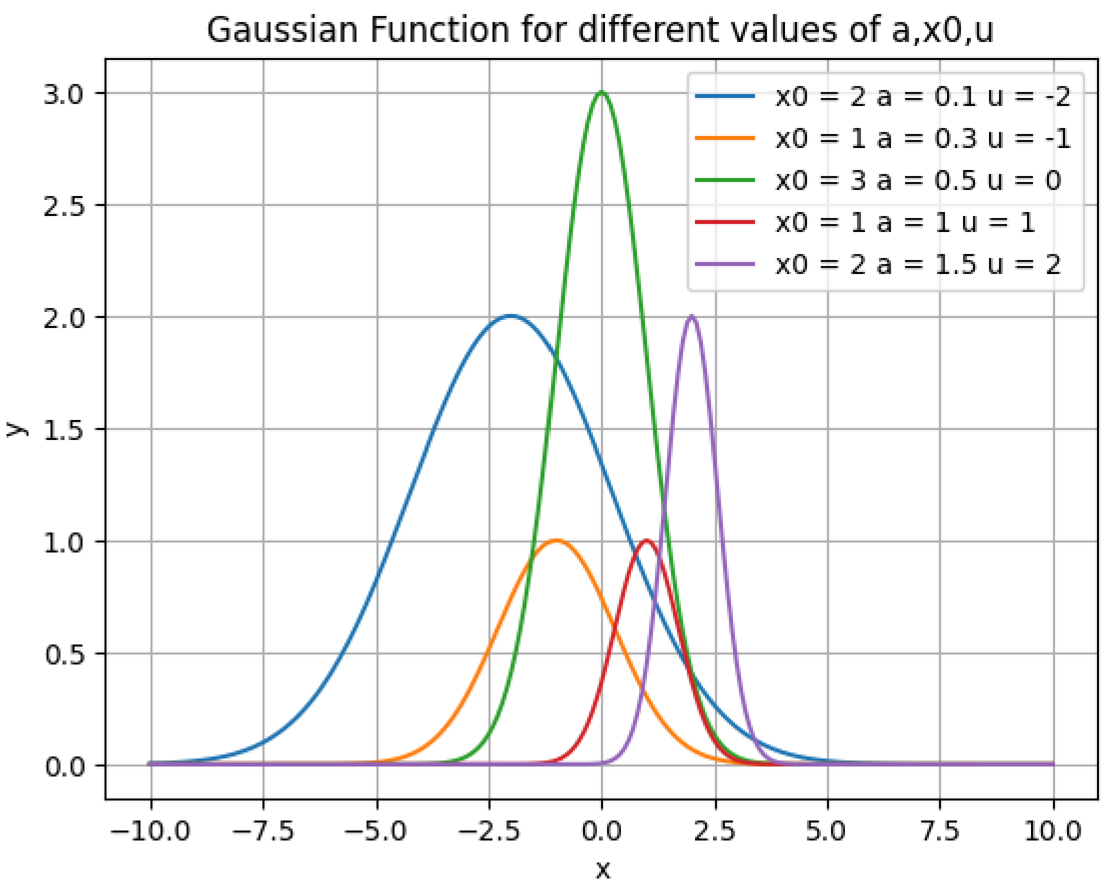
- In range [-pi,0] sin^2(x) is positive while sin(x) is negative.
- In range [0,pi] both graphs are positive.
- Both graph are periodic with period of sin^2(x) double the period of sin(x).

Q - 3

In [ ]:

```
def fun(a,u,x0,x):
    return  x0*np.exp(-a*((x-u)**2))

a = [0.1,0.3,0.5,1,1.5]
u = [-2,-1,0,1,2]
x0 = [2,1,3,1,2]
x = np.arange(-10,10+h,h)

for i in range(0,len(a)):
    plt.plot(x,fun(a[i],u[i],x0[i],x),label = 'x0 = ' + str(x0[i]) + ' a = ' + s

plt.legend()
plt.grid(True)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Gaussian Function for different values of a,x0,u')
plt.plot()
```

Out[ ]: []

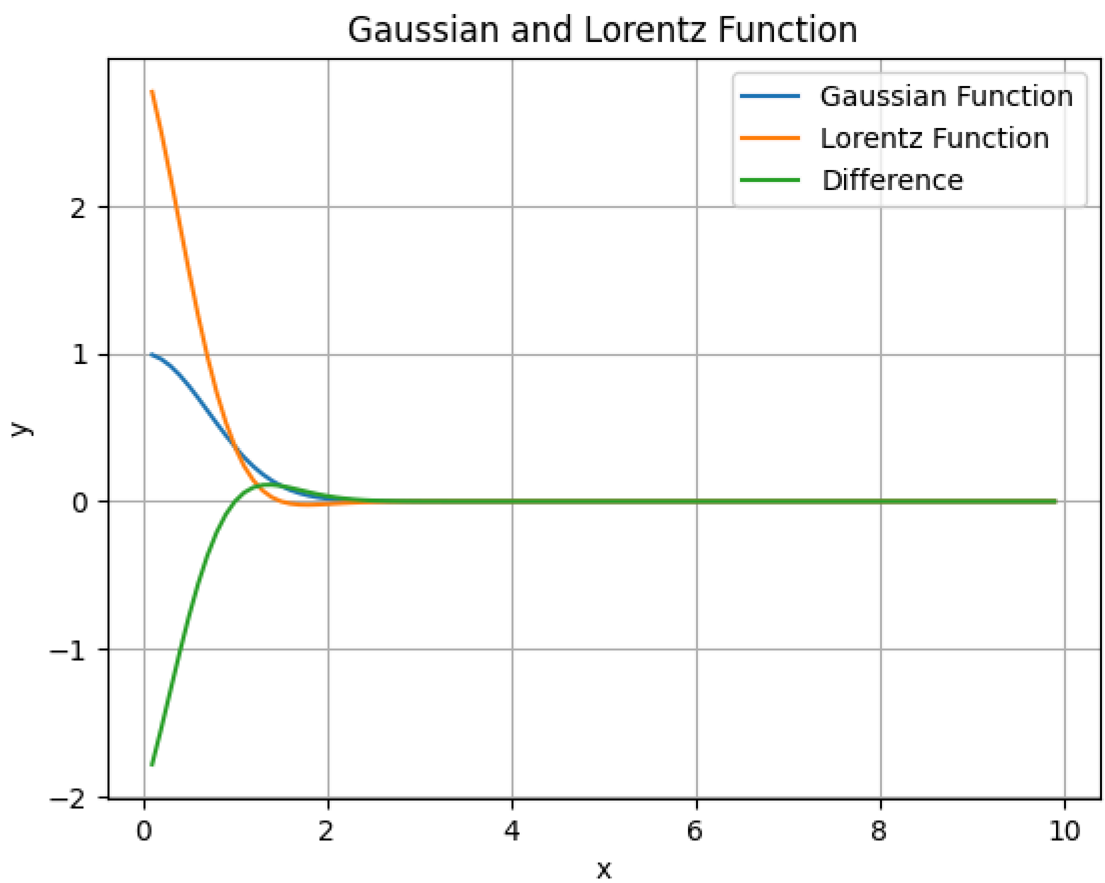


Plot the Gaussian function for different values of a,u,x0. The values we have considered are: a = [0.1,0.3,0.5,1,1.5] u = [-2,-1,0,1,2] x0 = [2,1,3,1,2]

In [ ]:

```
a = 1
u = 0
x0 = 1

def fun(a,u,x0,x):
    return  x0*np.exp(-a*((x-u)**2))

x = np.arange(0+h,10,h)
y_gauss = fun(a,u,x0,x)

y_lorentz = (1 - (2*(x-1)))*(np.exp(-x**2))
plt.plot(x,y_gauss,label = 'Gaussian Function')
plt.plot(x,y_lorentz,label = 'Lorentz Function')
plt.plot(x,y_gauss - y_lorentz,label = 'Difference')
plt.legend()
plt.grid(True)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Gaussian and Lorentz Function')
plt.plot()
```

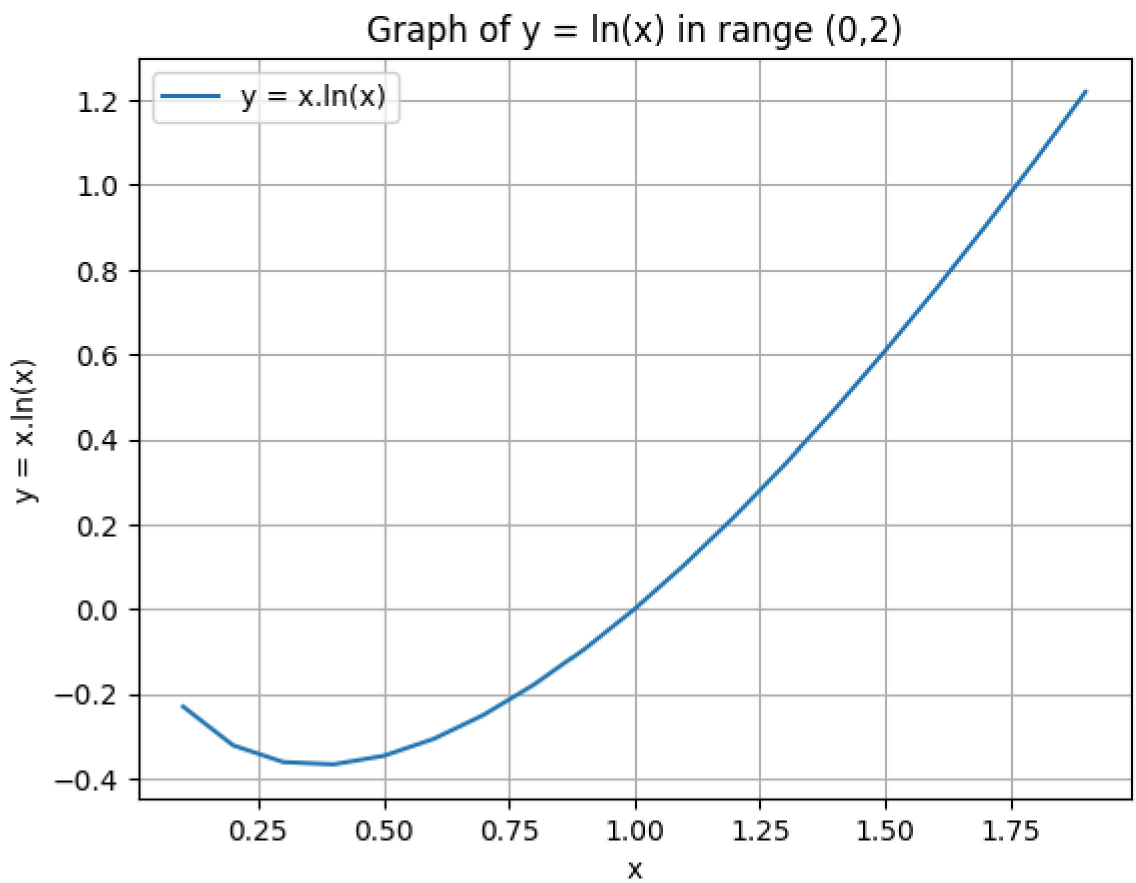
Out[ ]: []



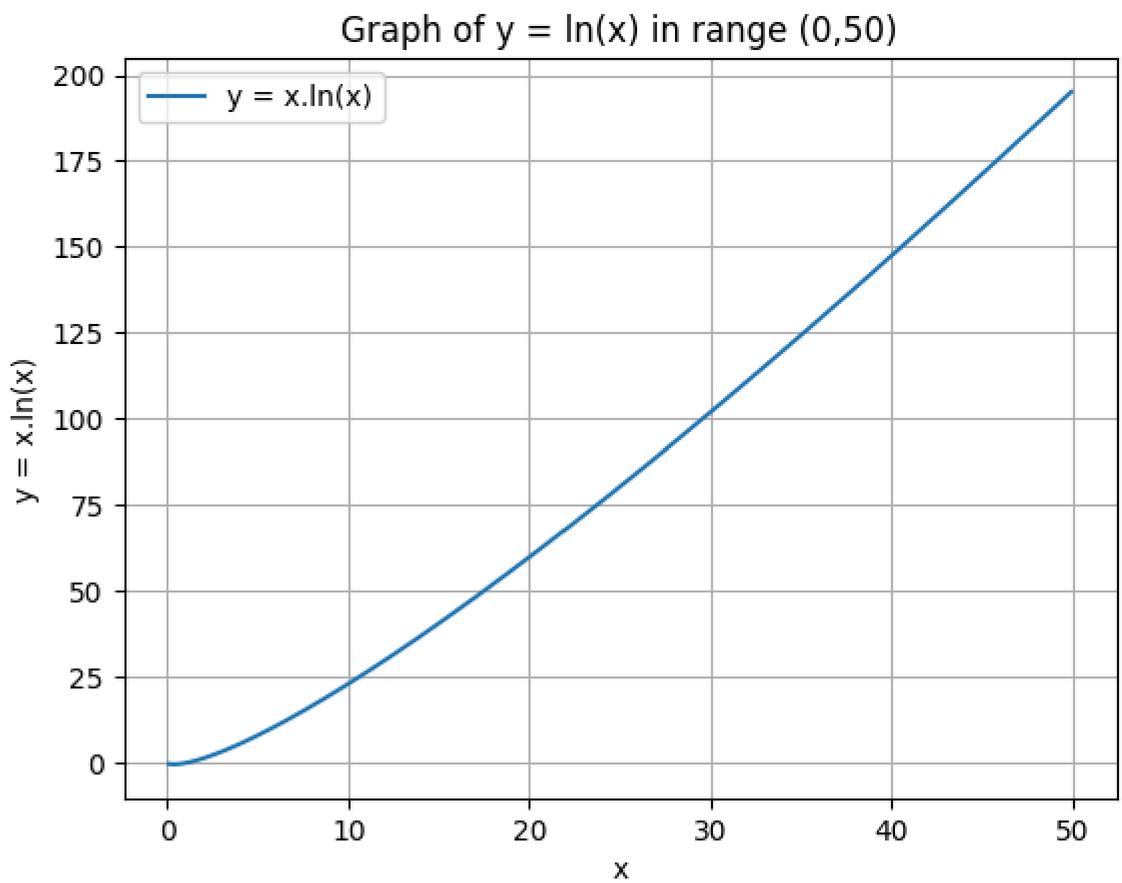
- Lorentz function in this case is first order expansion of the Gaussian expansion.
- Lorentz function= $(1-2*(x-1))*e^{-x^2}$;
- Plotting graphs for both functions in the range 0<x<10 and also plotting the absolute difference between both functions in the range of x.

Q - 4

In [ ]:
```
# y = x ln(x)
# for range(0,2)

x = np.arange(0+h,2,h)
y = x * np.log(x)
plt.plot(x,y,label = 'y = x.ln(x)')
plt.legend()
plt.grid(True)
plt.xlabel('x')
plt.ylabel('y = x.ln(x)')
plt.title('Graph of y = ln(x) in range (0,2)')
plt.show()
```



In [ ]:
```
# For large x
x = np.arange(0+h,50,h)
y = x * np.log(x)
plt.plot(x,y,label = 'y = x.ln(x)')
plt.legend()
plt.grid(True)
plt.xlabel('x')
plt.ylabel('y = x.ln(x)')
plt.title('Graph of y = ln(x) in range (0,50)')
plt.show()
```



The behaviour of y = x.ln(x)

- For small values of x, the graph is decreasing.
- At a particular x, graph hits minimum.
- After that point the graph is monotonically increasing.

In [ ]:
```
print('Analytical Justification of graph of y = x.ln(x)')
Image(filename='xln(x).png')
```

```
Analytical Justification of graph of y = x.ln(x)
```

Out[ ]:

$y = x . \ln(x)$

Differentiating $y = x \ln x$ w.r.t. x

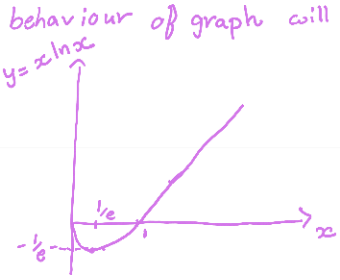
$$\frac{dy}{dx} = \ln x + 1 = 0$$

$$\ln x = -1$$

$$x = \frac{1}{e} \approx 0.37$$

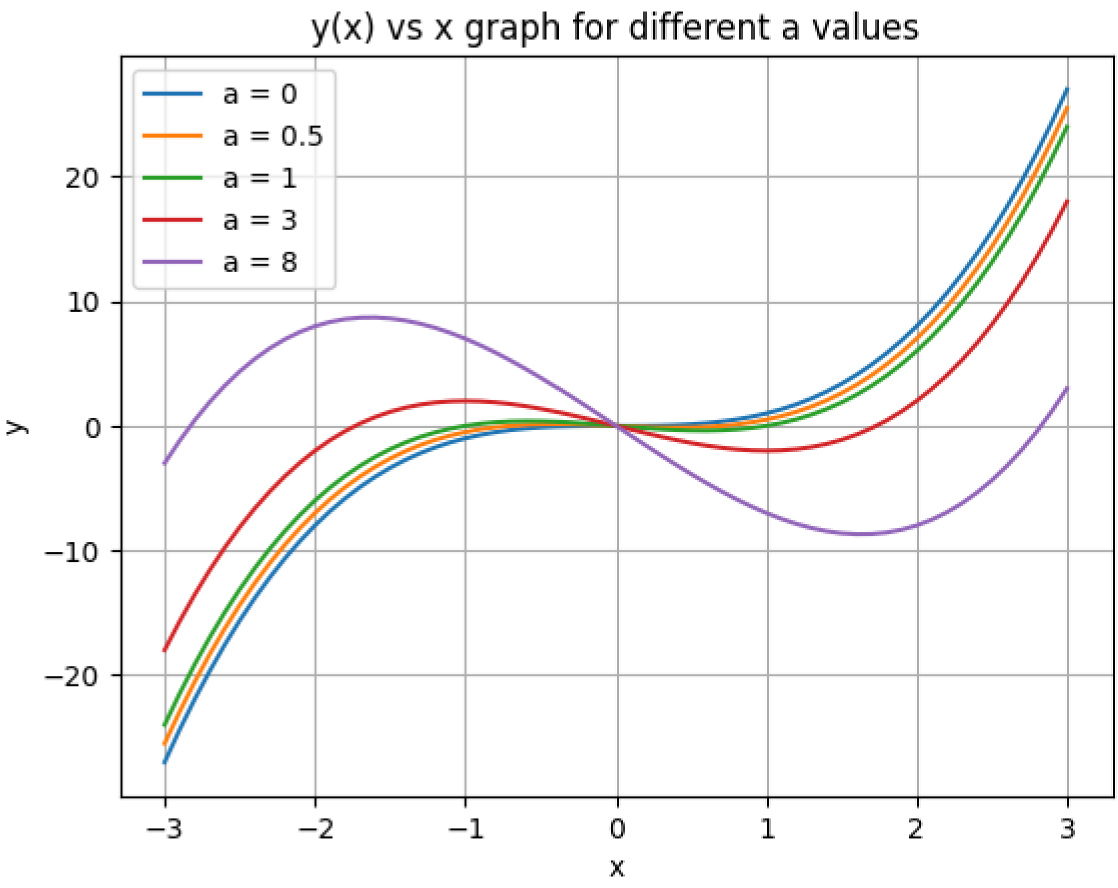
$$\frac{d^2y}{dx^2} = \frac{1}{x}\Big|_{x=0.37} = e > 0 \rightarrow \text{Minima}$$

Thus, behaviour of graph will be like



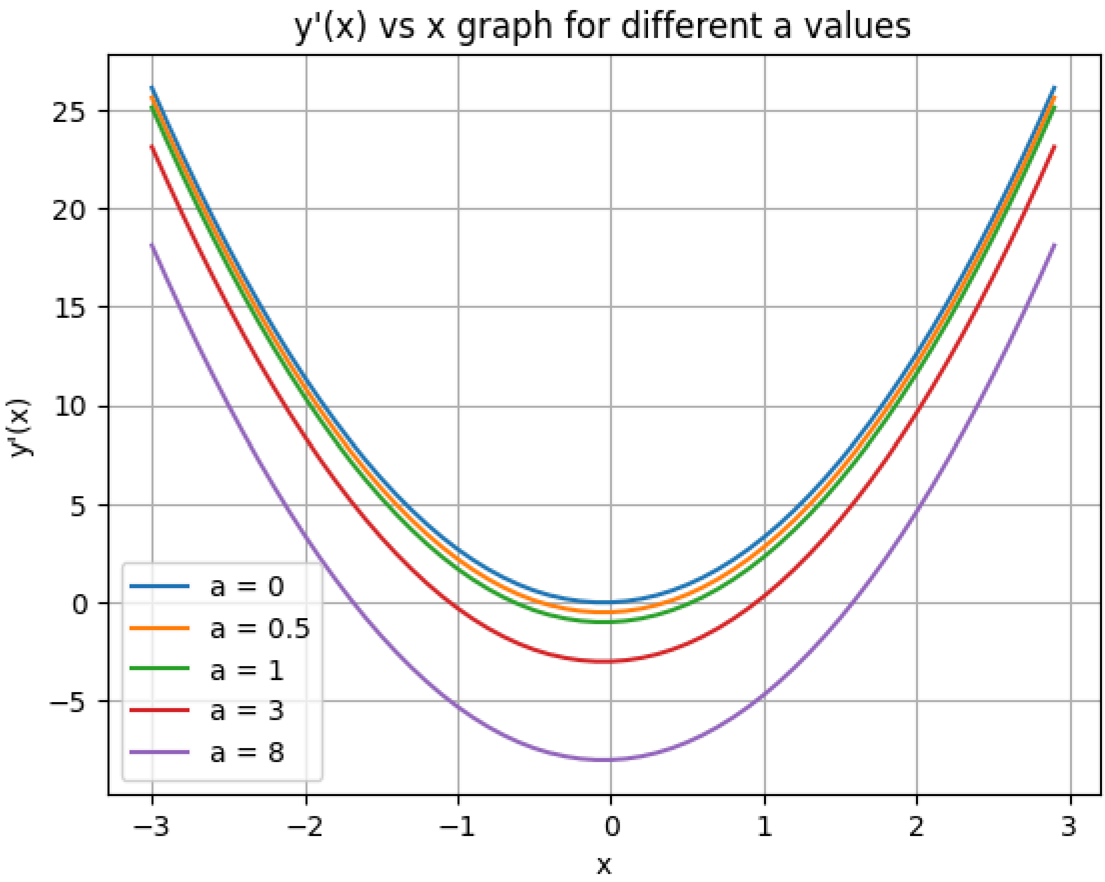
Q - 5A

In [ ]:
```
#y = -ax + x^3
def fun(a,x):
    return -a*x + x**3

a_val = [0,0.5,1,3,8]
x = np.arange(-3,3+h,h)
y_a = []
for a in a_val:
    y = fun(a,x)
    y_a.append(y)
    plt.plot(x,y,label = 'a = ' + str(a))
plt.legend()
plt.grid(True)
plt.xlabel('x')
plt.ylabel('y')
plt.title('y(x) vs x graph for different a values')
plt.show()
```



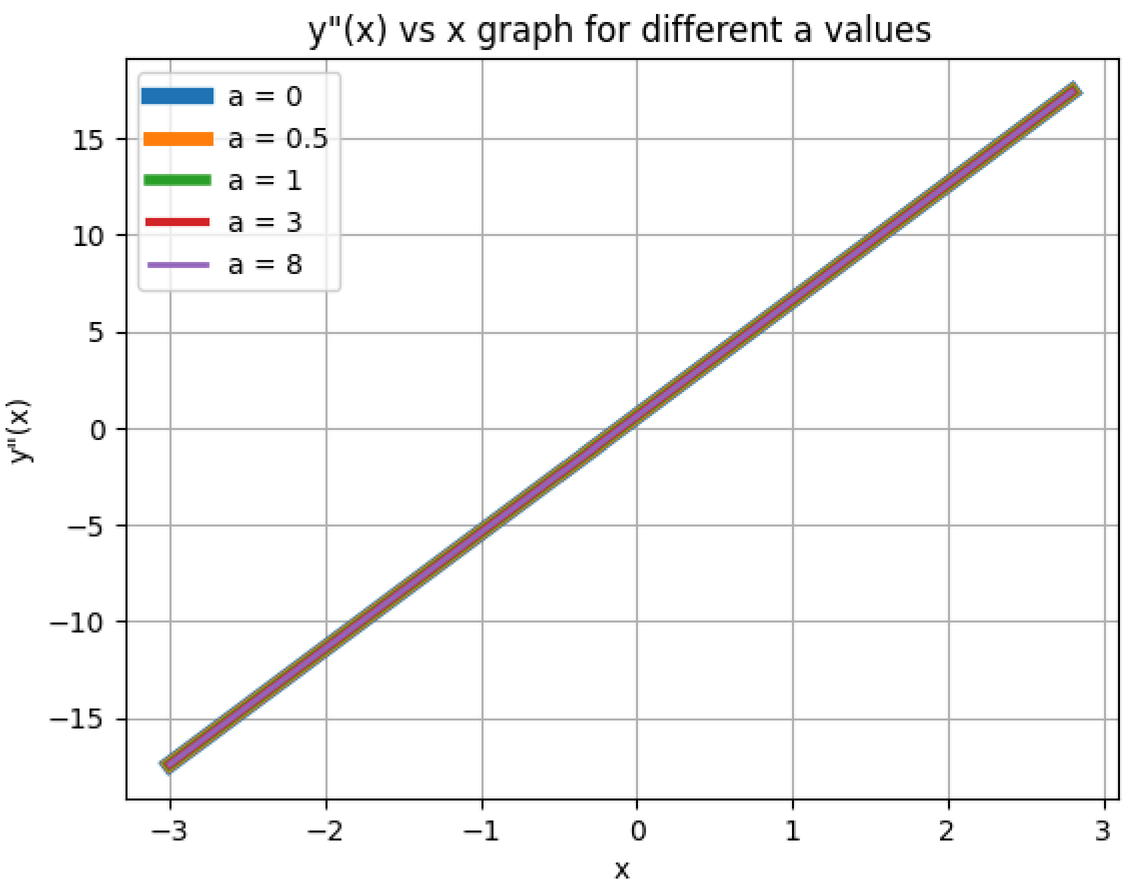
In [ ]:
```
def differentiate(f):
    dy = []
    for i in range(1,len(f)):
        dy.append((f[i] - f[i-1])/h)
    return dy

dy_a = []
for i in range(0,len(y_a)):
    dy = differentiate(y_a[i])
    dy_a.append(dy)
```

```
    x_new = x[:len(x)-1]
    plt.plot(x_new,dy,label = 'a = ' + str(a_val[i]))

plt.legend()
plt.grid(True)
plt.xlabel('x')
plt.ylabel('y\'(x)')
plt.title('y\'(x) vs x graph for different a values')
plt.show()
```



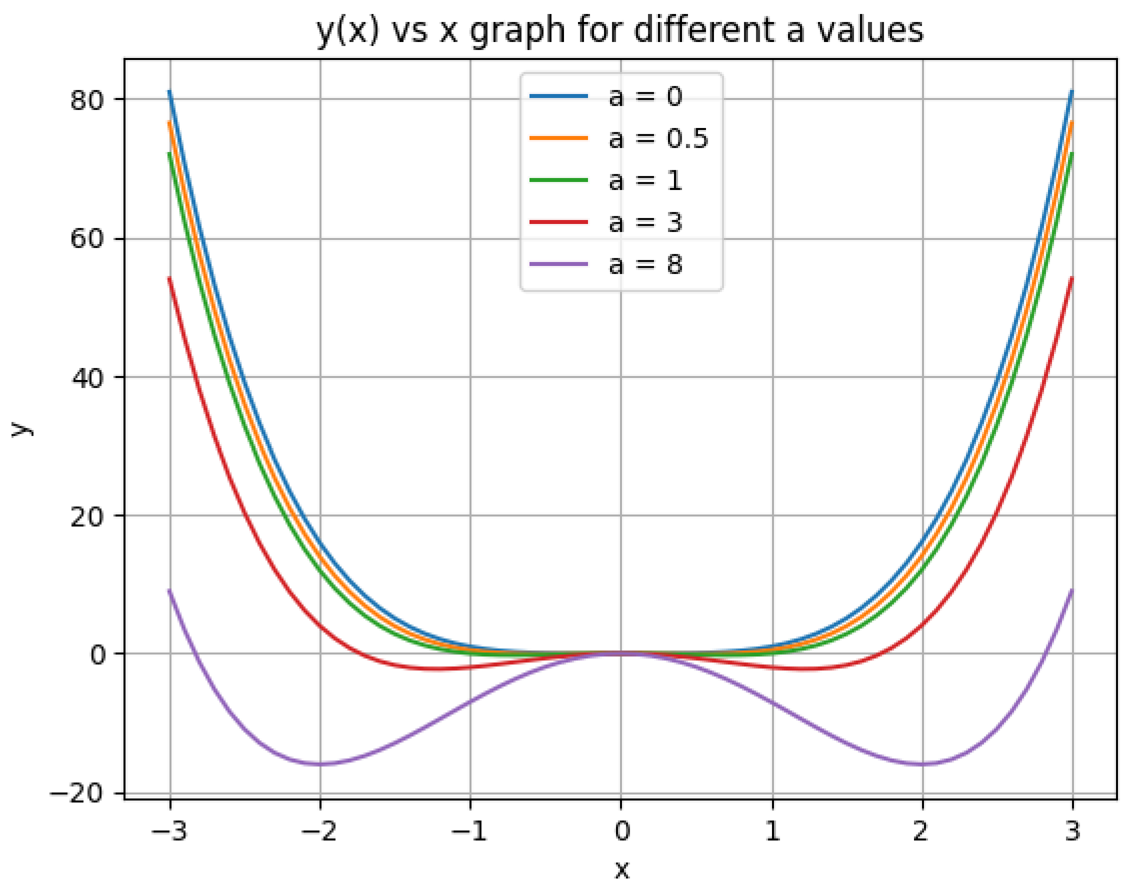
In [ ]:

```
d2y_a = []
for i in range(0,len(y_a)):
    d2y = differentiate(dy_a[i])
    d2y_a.append(d2y)
    x_new = x[:len(x)-2]
    plt.plot(x_new,d2y,label = 'a = ' + str(a_val[i]),linewidth = 6-i)

plt.legend()
plt.grid(True)
plt.xlabel('x')
plt.ylabel('y"(x)')
plt.title('y"(x) vs x graph for different a values')
plt.show()
```

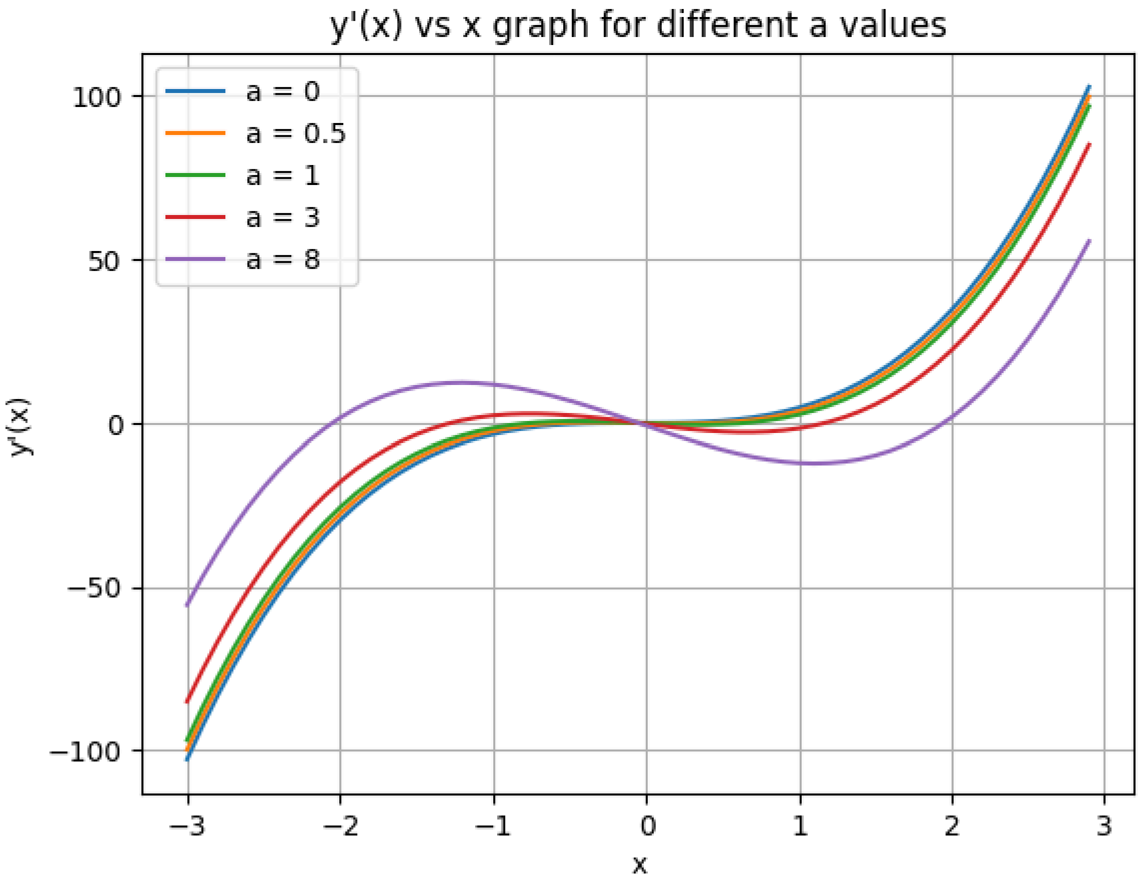


Q - 5B

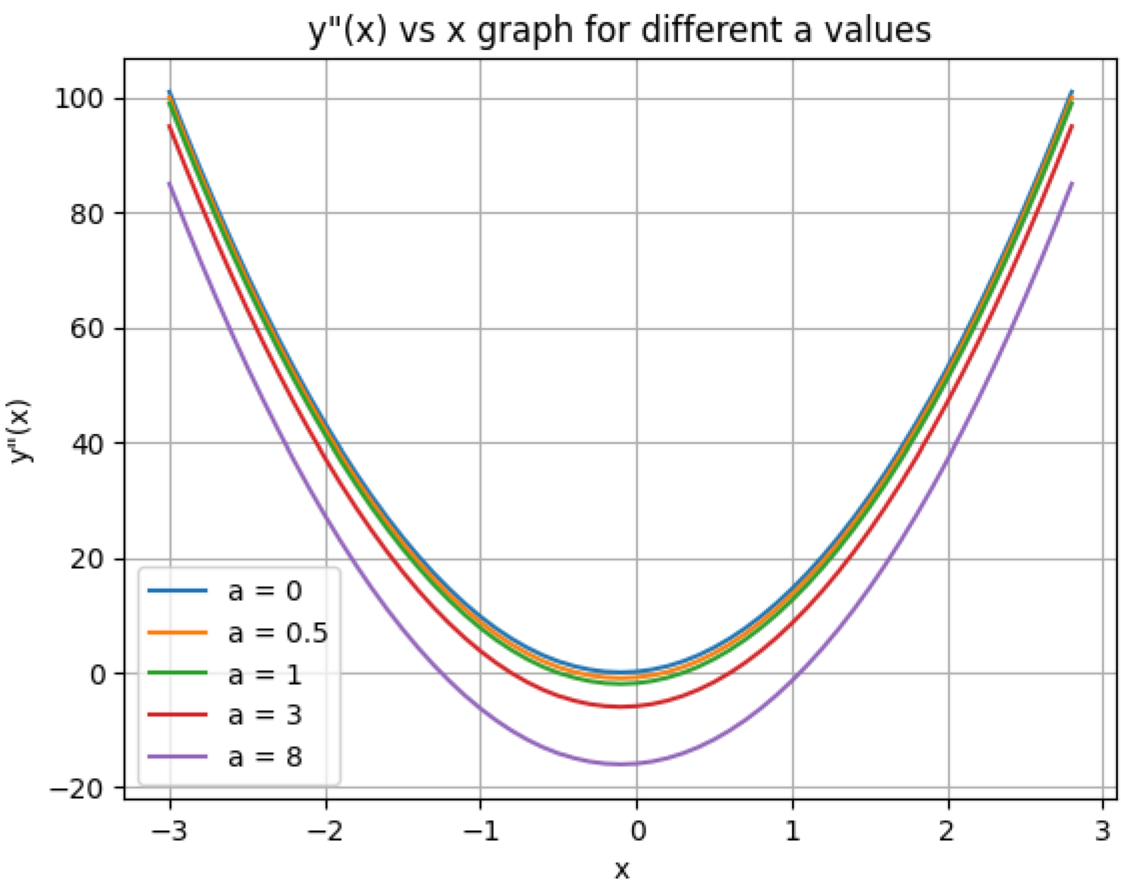
In [ ]:
```
#y = -ax^2 + x^4
def fun(a,x):
    return -a*x*x + x**4

a_val = [0,0.5,1,3,8]
x = np.arange(-3,3+h,h)
y_a = []
for a in a_val:
    y = fun(a,x)
    y_a.append(y)
    plt.plot(x,y,label = 'a = ' + str(a))
plt.legend()
plt.grid(True)
plt.xlabel('x')
plt.ylabel('y')
plt.title('y(x) vs x graph for different a values')
plt.show()
```



In [ ]:
```
dy_a = []
for i in range(0,len(y_a)):
    dy = differentiate(y_a[i])
    dy_a.append(dy)
    x_new = x[:len(x)-1]
    plt.plot(x_new,dy,label = 'a = ' + str(a_val[i]))

plt.legend()
plt.grid(True)
plt.xlabel('x')
plt.ylabel('y\'(x)')
plt.title('y\'(x) vs x graph for different a values')
plt.show()
```



In [ ]:
```
d2y_a = []
for i in range(0,len(y_a)):
    d2y = differentiate(dy_a[i])
    d2y_a.append(d2y)
    x_new = x[:len(x)-2]
    plt.plot(x_new,d2y,label = 'a = ' + str(a_val[i]))

plt.legend()
plt.grid(True)
plt.xlabel('x')
plt.ylabel('y"(x)')
plt.title('y"(x) vs x graph for different a values')
plt.show()
```



2. Taylor Polynomials

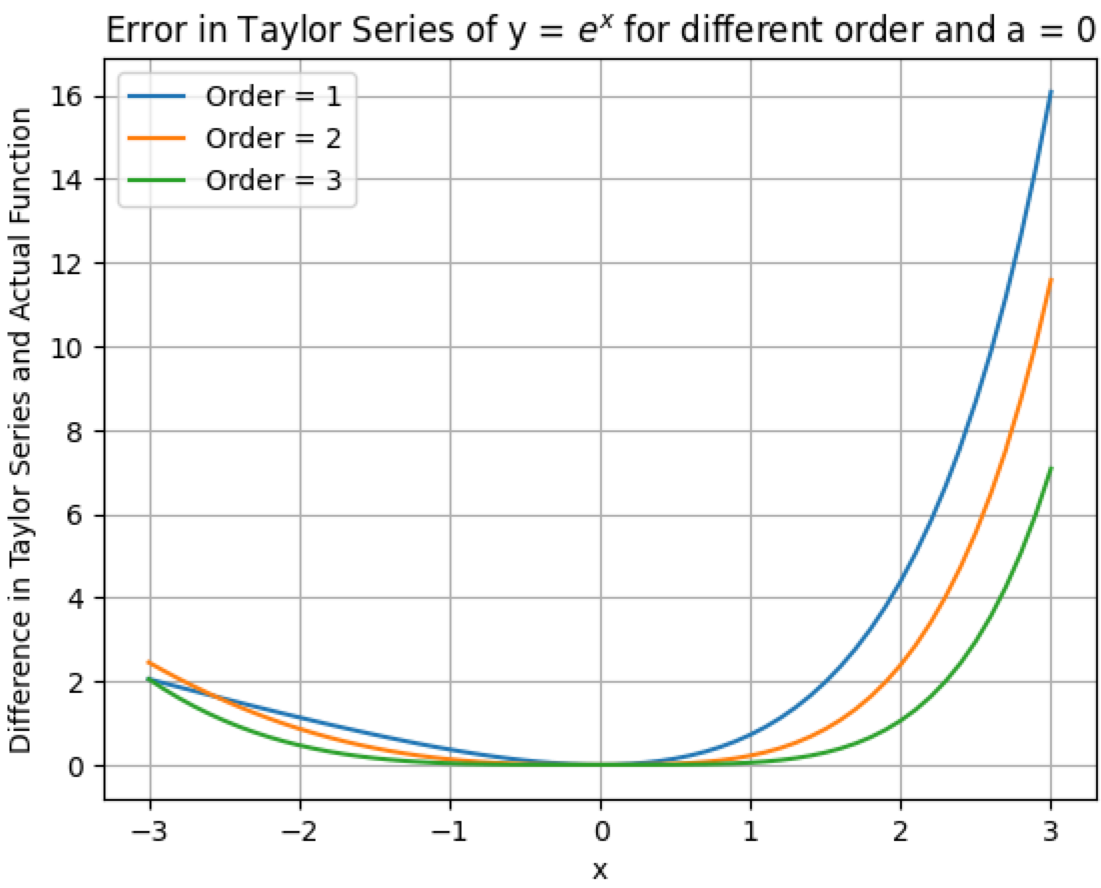
Q - 1A

- y = e^x
- Range of x = (-3,3)
- a = 0
- Degree of Taylor Series [1,2,3]

In [ ]:
```
#y = e^x
def fun(x):
    return np.exp(x)

def taylor(x,n):
    y = [0]*len(x)
    for i in range(0,n+1):
        y += ((x**i)/mt.factorial(i))
    return y

def error(x,n):
    return abs(fun(x) - taylor(x,n))


x = np.arange(-3,3+h,h)
order_n = 3
#remain = []
for n in range(1,order_n+1):
    err_val = error(x,n)
```

```
    plt.plot(x,err_val,label = 'Order = ' + str(n))

plt.legend()
plt.grid(True)
plt.xlabel('x')
plt.ylabel('Difference in Taylor Series and Actual Function')
plt.title('Error in Taylor Series of y = $e^x$ for different order and a = 0')
plt.plot()
```

Out[ ]: []



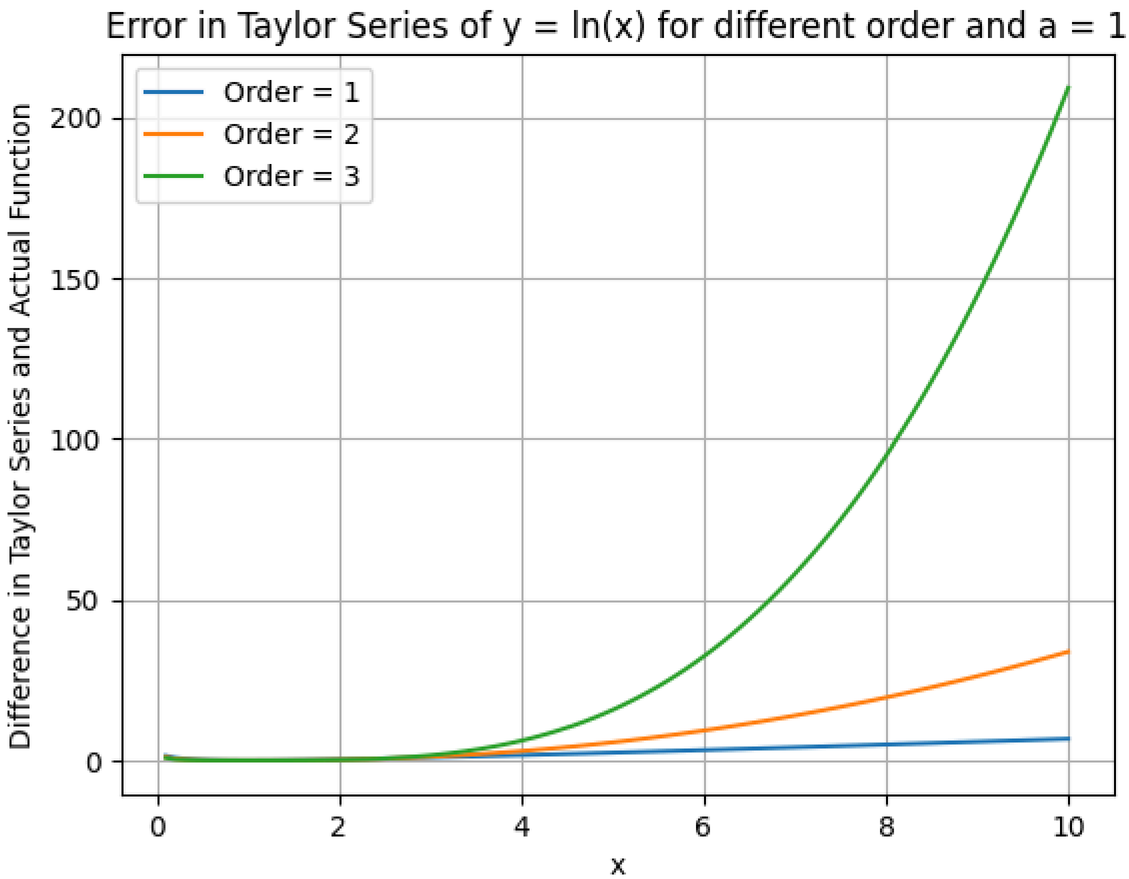
Q - 1B

- y = ln(x)
- Range of x = (0,10)
- a = 1
- Degree of Taylor Series [1,2,3]

In [ ]:
```
def fun(x):
    return np.log(x)

def taylor(x,n):
    y = [0]*len(x)
    for i in range(1,n+1):
        y += ((-1)**(i+1))*((((x-1)**(i))*mt.factorial(i-1))/mt.factorial(i))
    return y

x = np.arange(0+h,10+h,h)
order_n = 3
for n in range(1,order_n+1):
```

```
    err_val = error(x,n)
    plt.plot(x,err_val,label = 'Order = ' + str(n))

plt.legend()
plt.grid(True)
plt.xlabel('x')
plt.ylabel('Difference in Taylor Series and Actual Function')
plt.title('Error in Taylor Series of y = ln(x) for different order and a = 1')
plt.plot()
```

Out[ ]: []



Q - 1C

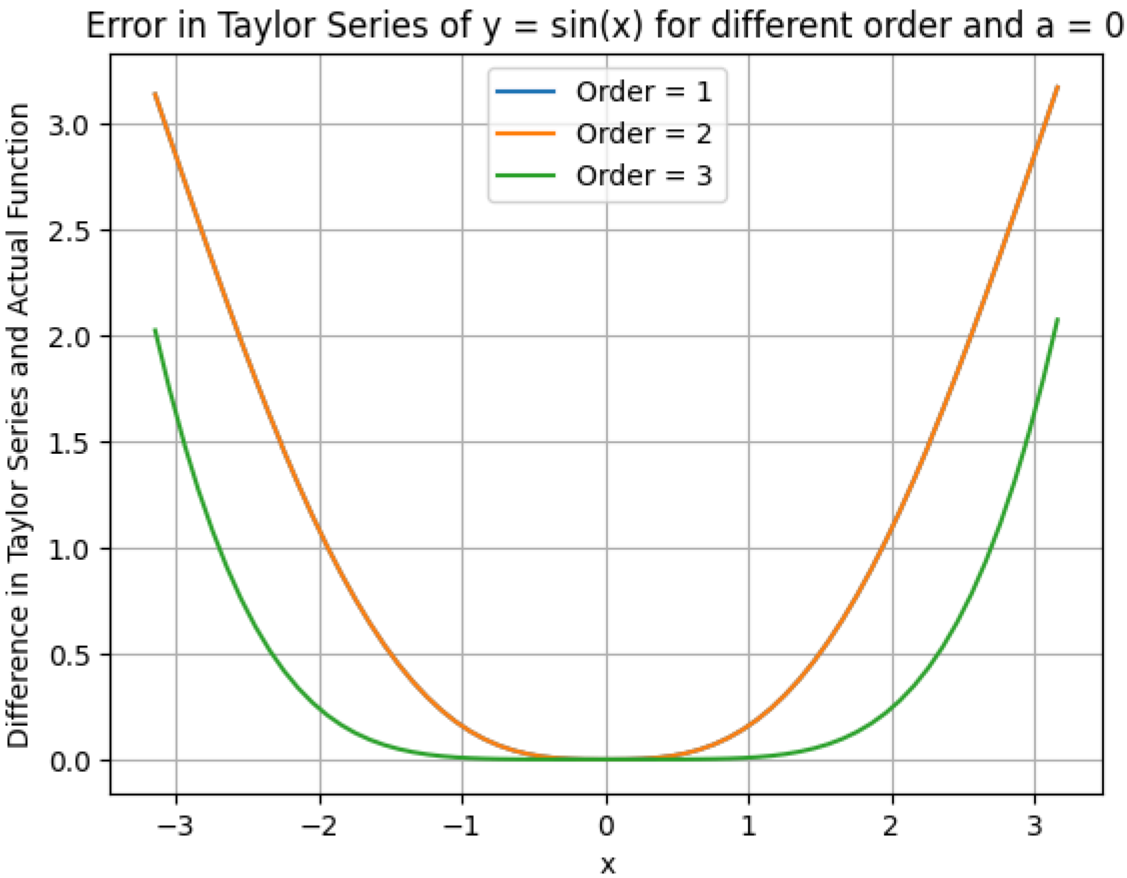
- y = sin(x)
- Range of x = (-pi,pi)
- a = 0
- Degree of Taylor Series [1,2,3]

In [ ]:

```
order_n = 3

def fun(x):
    return np.sin(x)

def taylor(x,n):
    y = [0]*len(x)
    for i in range(0,int((n+1)/2)):
        y += ((-1)**(i))*((x**(2*i+1))/mt.factorial(2*i+1))
    return y
```

```
x = np.arange(-np.pi,np.pi+h,h)
for n in range(1,order_n+1):
    err_val = error(x,n)
    plt.plot(x,err_val,label = 'Order = ' + str(n))

plt.legend()
plt.grid(True)
plt.xlabel('x')
plt.ylabel('Difference in Taylor Series and Actual Function')
plt.title('Error in Taylor Series of y = sin(x) for different order and a = 0')
plt.plot()
```

Out[ ]: []



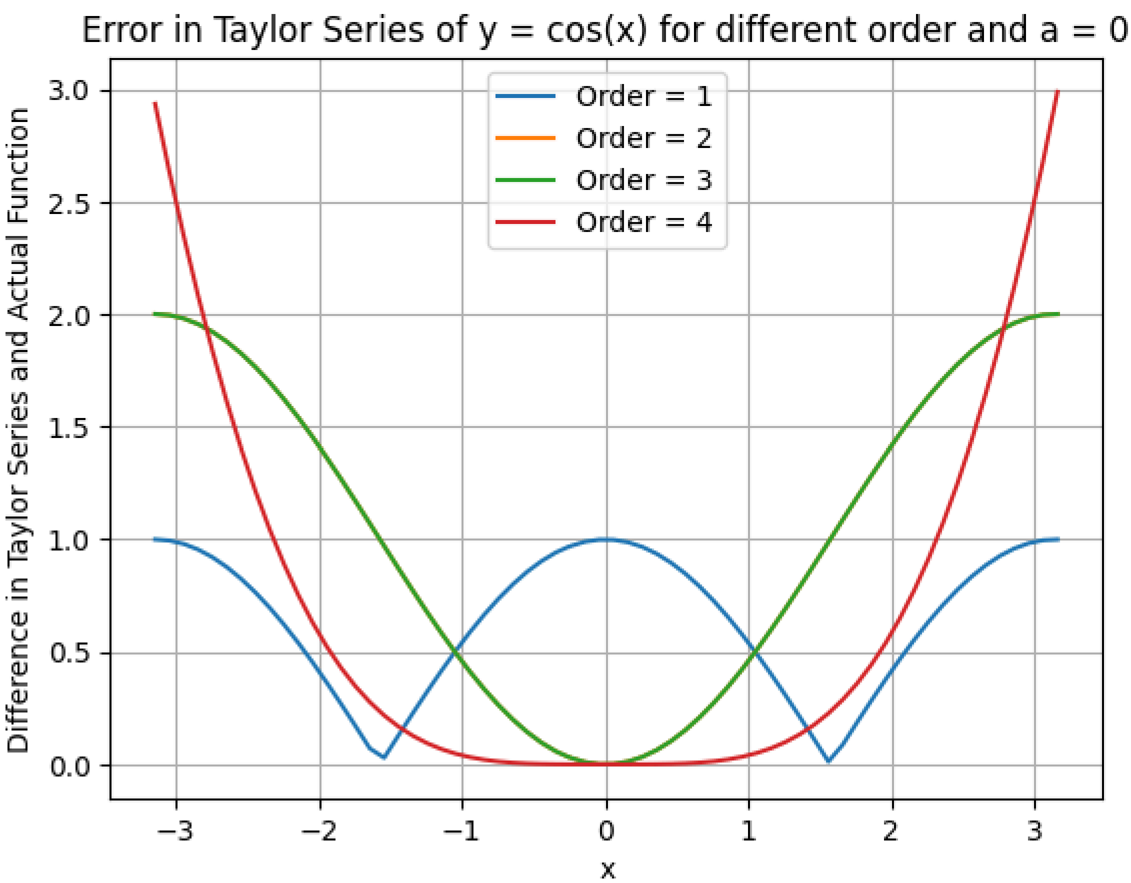
Q - 1D

- y = sin(x)
- Range of x = (-pi,pi)
- a = 0
- Degree of Taylor Series [1,2,3,4]

In [ ]:
```
order_n = 4

def fun(x):
    return np.cos(x)

def taylor(x,n):
    y = [0]*len(x)
    for i in range(0,int((n)/2)):
        y += ((-1)**(i))*((x**(2*i))/mt.factorial(2*i))
```

```python
    return y

x = np.arange(-np.pi,np.pi+h,h)
for n in range(1,order_n+1):
    err_val = error(x,n)
    plt.plot(x,err_val,label = 'Order = ' + str(n))

plt.legend()
plt.grid(True)
plt.xlabel('x')
plt.ylabel('Difference in Taylor Series and Actual Function')
plt.title('Error in Taylor Series of y = cos(x) for different order and a = 0')
plt.plot()
```

Out[ ]: []



Q - 2

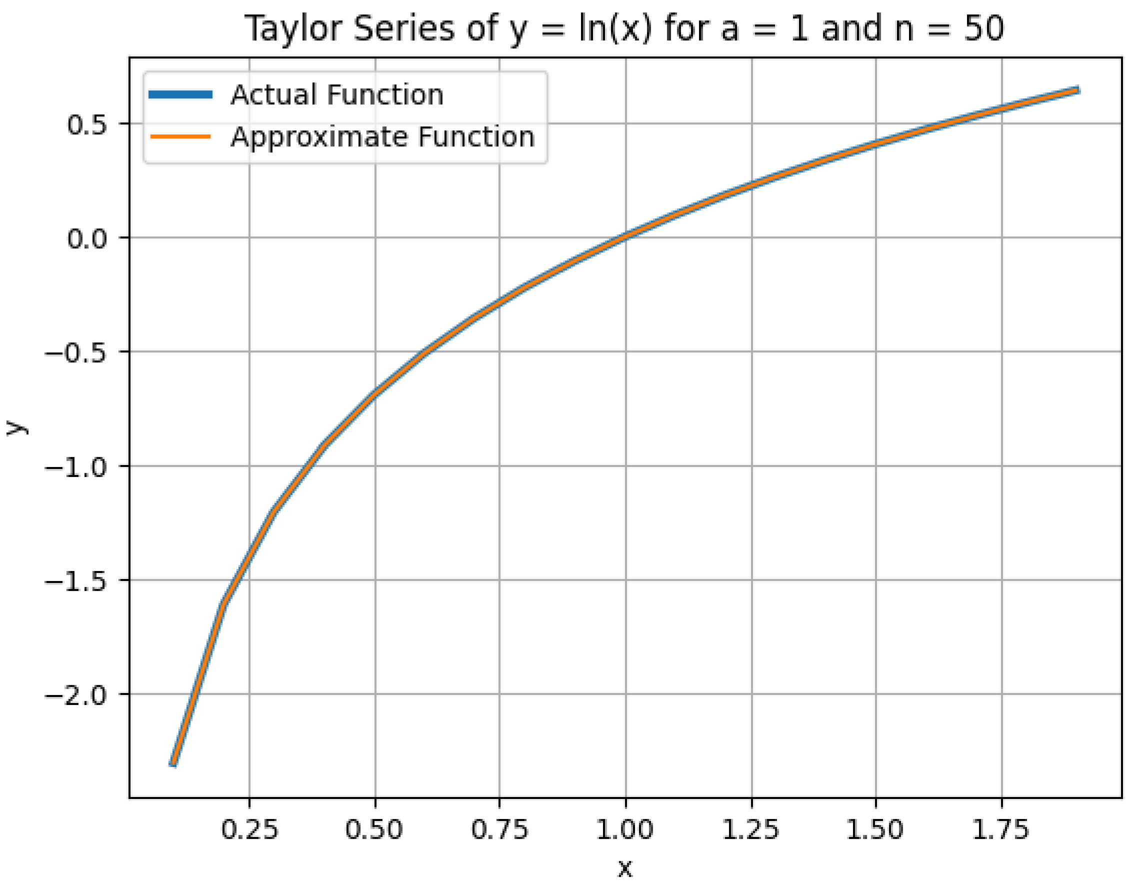
- y = ln(x)
- Range of x = (0,2)
- Maximum Error at x = 2 must be 0.01
- Taylor Series Expansion will be used around a = 1 for increasing order until error around x = 2 is greater than 0.01

In [ ]:
```python
def fun(x):
    return np.log(x)

def taylor(x,n):
    y = 0
```

```python
    for i in range(1,n+1):
        y += ((-1)**(i+1))*((((x-1)**(i))*mt.factorial(i-1))/mt.factorial(i))
    return y

n = 1
max_error = 0.01
point_x = 2
err_val = 1000000
while err_val > max_error :
    err_val = error(point_x,n)
    n += 1
n -= 1
print('Order/Degree of Approxmate Taylor Polynomial must be atleast = ' +str(n))

x = np.arange(0+h,2,h)
y_true = fun(x)
y_taylor = taylor(x,n)


plt.plot(x,y_true,label = 'Actual Function',linewidth = 3)
plt.plot(x,y_taylor,label = 'Approximate Function')
plt.legend()
plt.grid(True)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Taylor Series of y = ln(x) for a = 1 and n = ' + str(n))
plt.plot()
```

```
Order/Degree of Approxmate Taylor Polynomial must be atleast = 50
```

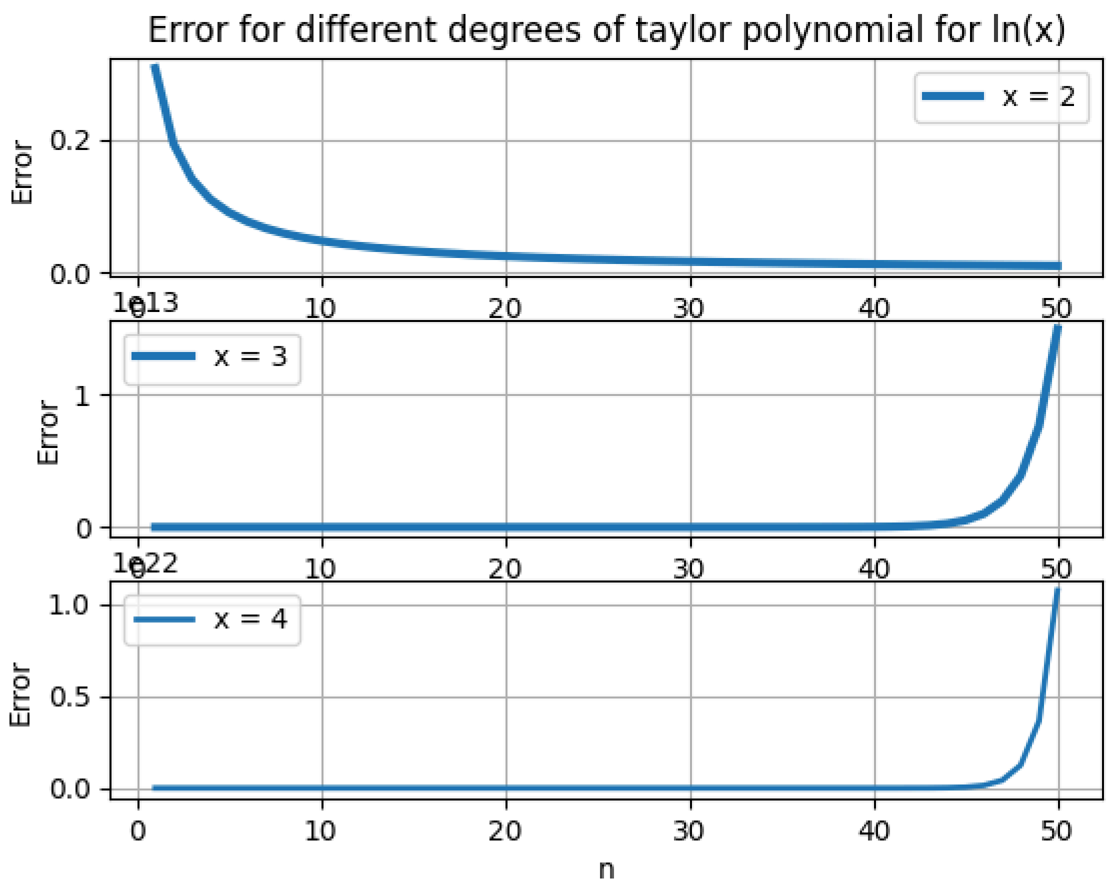
Out[ ]: []



Q - 3

We will find errors in taylor expansion for orders i.e in range [1,50] for x = [2,3,4]

In [ ]:
```
def fun(x):
    return np.log(x)

def taylor(x,n):
    y = 0
    for i in range(1,n+1):
        y += ((-1)**(i+1))*((((x-1)**(i))*mt.factorial(i-1))/mt.factorial(i))
    return y

def error(x,n):
    return abs(fun(x) - taylor(x,n))
n = np.arange(1,51,1)
x_val = [2,3,4]

err_lst = []
for x in x_val:
    lst = []
    for n_val in n:
        lst.append(error(x,n_val))
    err_lst.append(lst)
plt.subplot(3,1,1)
plt.plot(n,err_lst[0],label = 'x = ' + str(x_val[0]),linewidth = 3)
plt.legend()
plt.grid(True)
plt.xlabel('n')
plt.ylabel('Error')
plt.title('Error for different degrees of taylor polynomial for ln(x)')
plt.subplot(3,1,2)
plt.plot(n,err_lst[1],label = 'x = ' + str(x_val[1]),linewidth = 3)
plt.legend()
plt.grid(True)
plt.xlabel('n')
plt.ylabel('Error')
plt.subplot(3,1,3)
plt.plot(n,err_lst[2],label = 'x = ' + str(x_val[2]),linewidth = 2)
plt.legend()
plt.grid(True)
plt.xlabel('n')
plt.ylabel('Error')
plt.show()
```



Analysis for the graph

- The taylor series expansion for y = ln(x) will work only if $|x$-$x_0|$ <= 1.
- $|x$-$x_0|$ > 1 the seroes will diverge so error would increase.
- Thus for x = 2 error graph will decrease while for x =3,4 error will increase around x_0 = 1.