In [1]:

```
import matplotlib.pyplot as plt
import numpy as np
import pylab as py
```

Lab1

11-06-2019

Aim: Explore basic python opertaion for scientefic competing

In [2]:

```
x=np.linspace(0,5,100)
#get 100 values from 0-5
print(x)
```

```
[0.         0.05050505 0.1010101  0.15151515 0.2020202  0.25252525
 0.3030303  0.35353535 0.4040404  0.45454545 0.50505051 0.55555556
 0.60606061 0.65656566 0.70707071 0.75757576 0.80808081 0.85858586
 0.90909091 0.95959596 1.01010101 1.06060606 1.11111111 1.16161616
 1.21212121 1.26262626 1.31313131 1.36363636 1.41414141 1.46464646
 1.51515152 1.56565657 1.61616162 1.66666667 1.71717172 1.76767677
 1.81818182 1.86868687 1.91919192 1.96969697 2.02020202 2.07070707
 2.12121212 2.17171717 2.22222222 2.27272727 2.32323232 2.37373737
 2.42424242 2.47474747 2.52525253 2.57575758 2.62626263 2.67676768
 2.72727273 2.77777778 2.82828283 2.87878788 2.92929293 2.97979798
 3.03030303 3.08080808 3.13131313 3.18181818 3.23232323 3.28282828
 3.33333333 3.38383838 3.43434343 3.48484848 3.53535354 3.58585859
 3.63636364 3.68686869 3.73737374 3.78787879 3.83838384 3.88888889
 3.93939394 3.98989899 4.04040404 4.09090909 4.14141414 4.19191919
 4.24242424 4.29292929 4.34343434 4.39393939 4.44444444 4.49494949
 4.54545455 4.5959596  4.64646465 4.6969697  4.74747475 4.7979798
 4.84848485 4.8989899  4.94949495 5.        ]
```

In [3]:

```
x=range(0,10,2)
# values in the range[0,10] with an increment of 2
#range can not produce floating point values
print(x)
```

```
range(0, 10, 2)
```

In [ ]:

```
x=np.arange(0,100,2.5)
# similar to range but can produce evenly spaced floating point values
print(x)
```

In [ ]:

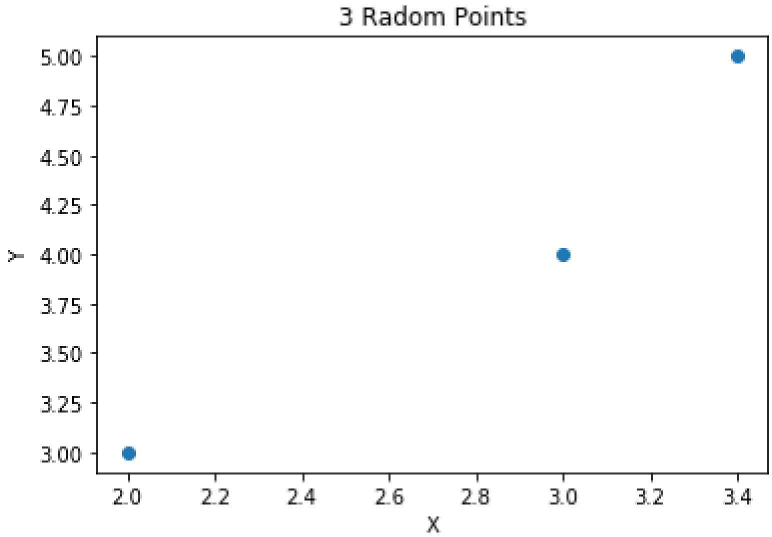
```
## Floor function
## it will return an integer not greater than the actual answer(quotient)
print(-12//5)
```

In [4]:

```
# How to plot:
## create two arrays seperately for x and y values
## now use plot command from matplotlib.pyplot

x=[2,3,3.4]
y=[3,4,5]
plt.plot(x,y,'o')
plt.title("3 Radom Points")
plt.xlabel("X")
plt.ylabel("Y")
```

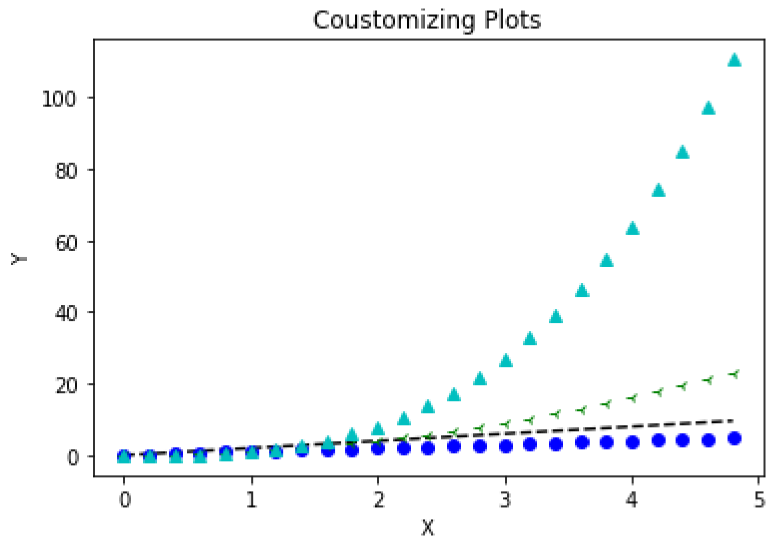
Out[4]:

```
Text(0, 0.5, 'Y')
```



In [6]:

```
#evenly spaced time at 200ms intervals
t=np.arange(0,5,0.2)

## red dashes , blue squares and green triangles
plt.plot(t,2*t,'k--',t,t,'bo',t,t**2,'g3',t,t**3,'c^')
plt.title("Coustomizing Plots")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```

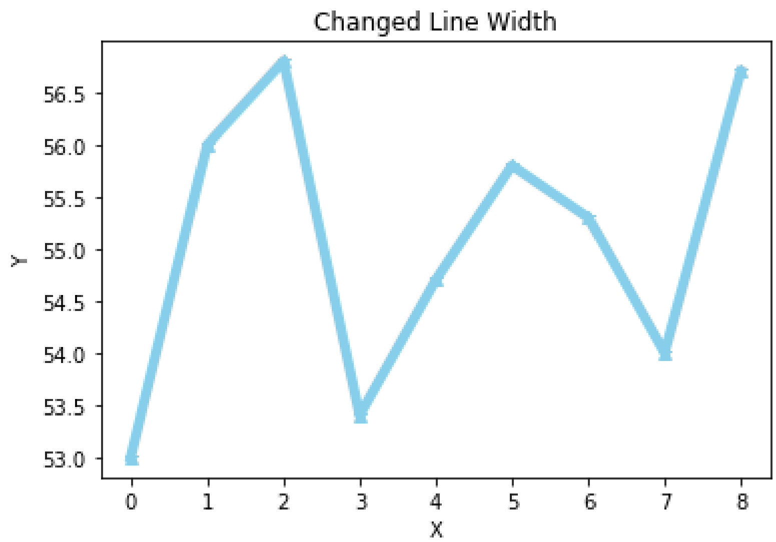


In [8]:

```
nyc_temp=[53,56,56.8,53.4,54.7,55.8,55.3,54,56.7]
py.plot(nyc_temp,'c^',color="skyblue")
plt.title("Changed Line Width")
plt.xlabel("X")
plt.ylabel("Y")
py.plot(nyc_temp,marker="*",color="skyblue",linewidth=5)
```

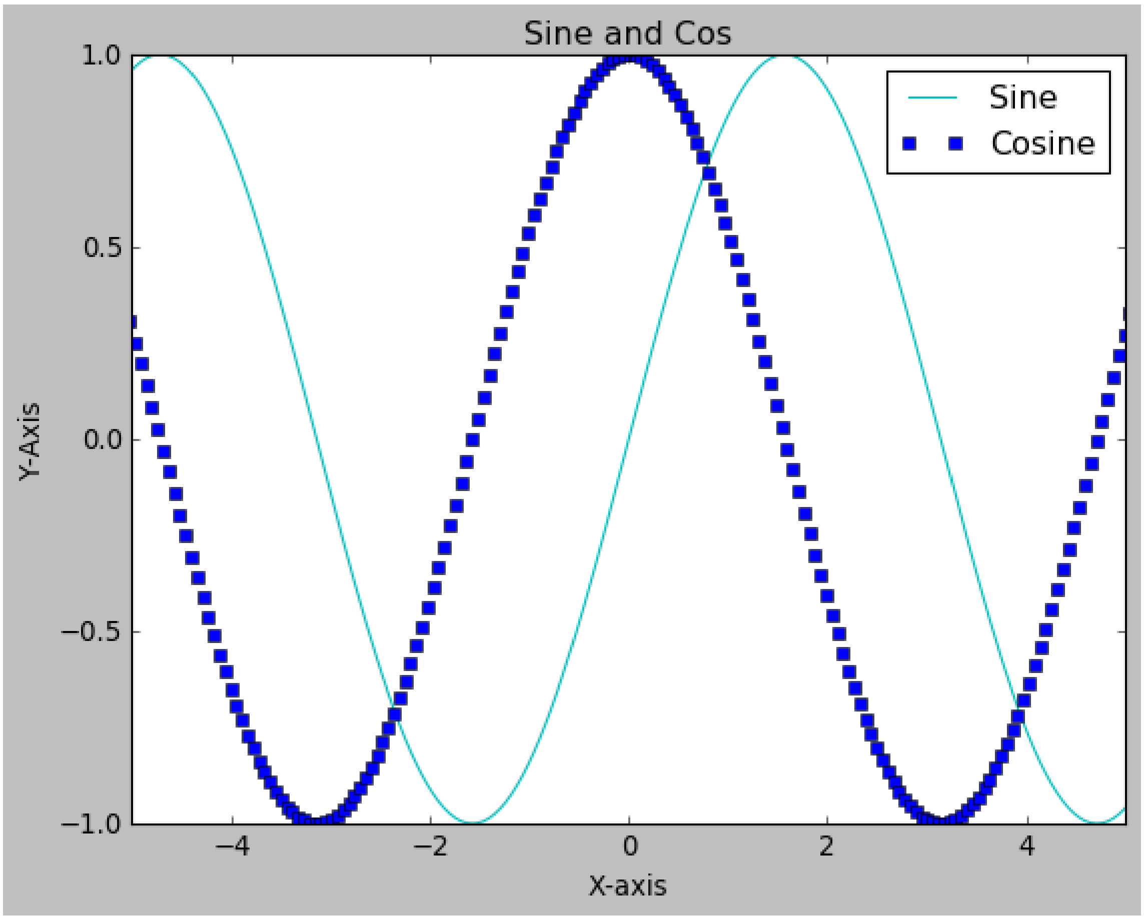
Out[8]:

```
[<matplotlib.lines.Line2D at 0x1becda5e208>]
```



In [10]:

```
plt.style.use("classic")
x=np.linspace(-10,4*np.pi,400)
plt.plot(x,np.sin(x),'c',x,np.cos(x),'s')
plt.xlabel("X-axis")
plt.xlim(-5,5)
plt.title("Sine and Cos")
plt.ylim(-1,1)
plt.ylabel("Y-Axis")
plt.legend(["Sine","Cosine"])

plt.show()
```

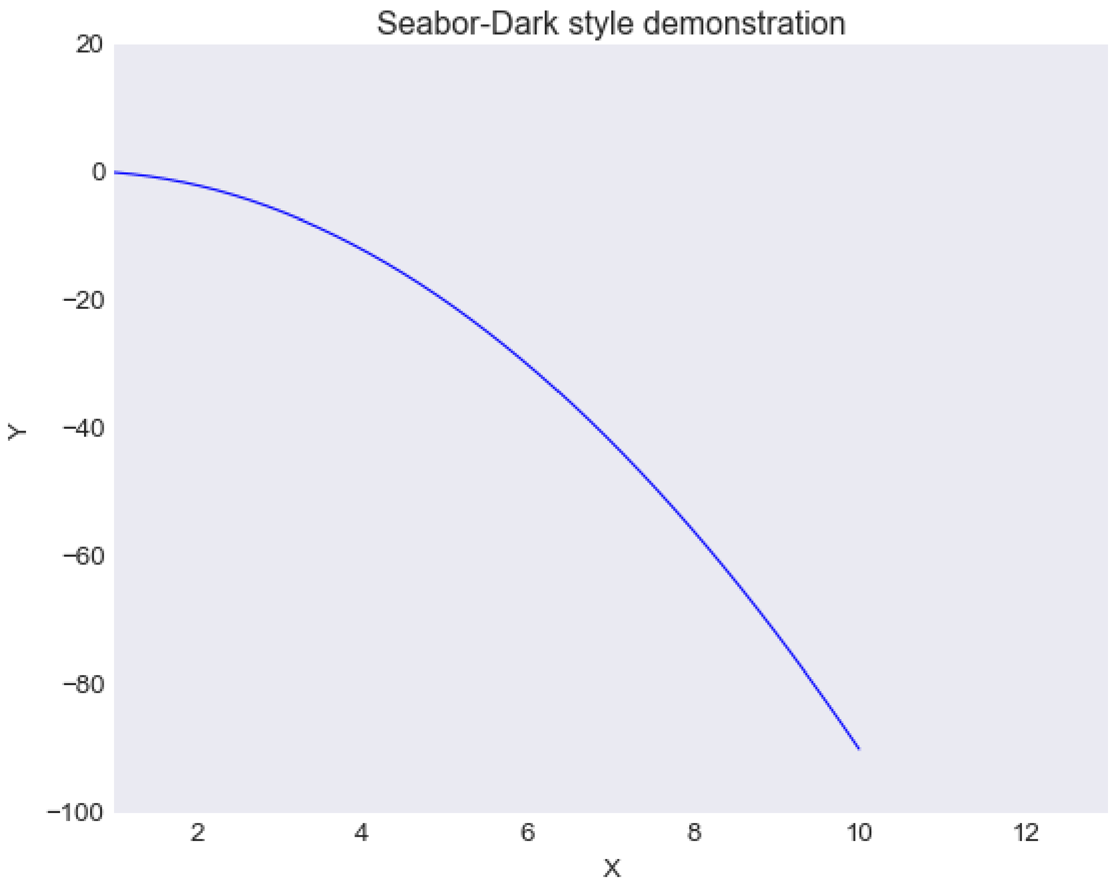


In [13]:

```
x=np.linspace(0,10,50)
plt.style.use('seaborn-dark')
y=x*(1-x)
plt.title("Seabor-Dark style demonstration")
plt.xlim(1,13)
plt.xlabel("X")
plt.ylabel("Y")
plt.plot(x,y)
```

Out[13]:

```
[<matplotlib.lines.Line2D at 0x1becdbac668>]
```



In [14]:

```
plt.style.available
```

Out[14]:

```
['bmh',
 'classic',
 'dark_background',
 'fast',
 'fivethirtyeight',
 'ggplot',
 'grayscale',
 'seaborn-bright',
 'seaborn-colorblind',
 'seaborn-dark-palette',
 'seaborn-dark',
 'seaborn-darkgrid',
 'seaborn-deep',
 'seaborn-muted',
 'seaborn-notebook',
 'seaborn-paper',
 'seaborn-pastel',
 'seaborn-poster',
 'seaborn-talk',
 'seaborn-ticks',
 'seaborn-white',
 'seaborn-whitegrid',
 'seaborn',
 'Solarize_Light2',
 'tableau-colorblind10',
 '_classic_test']
```

Conclusion:

```
Some of the basic opertaions such as linspace and floor function were applied.
Graph manipulation was done to some extent
```

Plot and Subplot

18.06.2019

Aim: To explore matplotlib.plot and its attributes

In [2]:

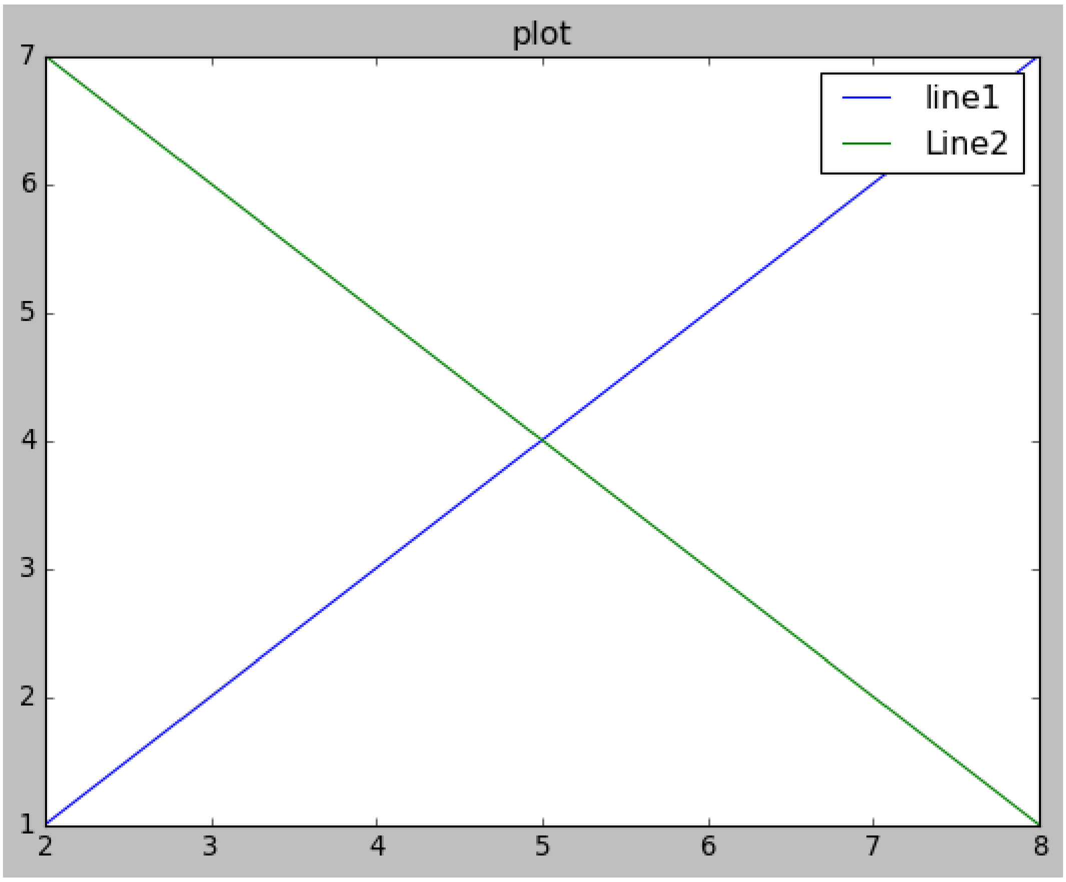
```
import matplotlib.pyplot as plt
import math
import numpy as np
import sympy as sp
from pylab import *
from scipy import *
```

In [78]:

```
x=[2,3,4,5,6,7,8]
y=[1,2,3,4,5,6,7]
y2=[7,6,5,4,3,2,1]
plt.plot(x,y,color="blue")
plt.plot(x,y2,color="green")
plt.title("plot")
plt.legend(["line1","Line2"])
```

Out[78]:

```
<matplotlib.legend.Legend at 0xe587ac8>
```



In [56]:

```
x=arange(-np.pi,np.pi,0.1)
```

In [49]:

```
x
```

Out[49]:

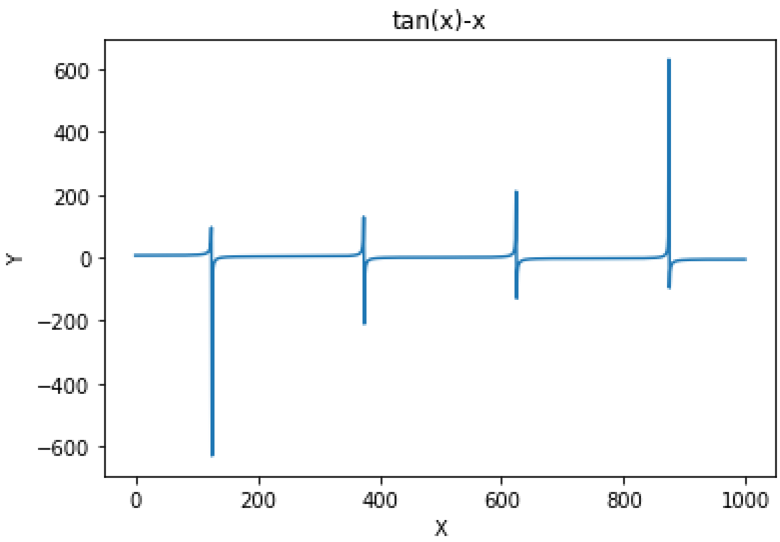
```
array([-3.14159265, -3.14059265, -3.13959265, ...,  3.13940735,
        3.14040735,  3.14140735])
```

In [6]:

```
plot(np.tan(x)-x)
plt.xlabel("X")
plt.ylabel("Y")
plt.title("tan(x)-x")
```

Out[6]:

```
Text(0.5, 1.0, 'tan(x)-x')
```

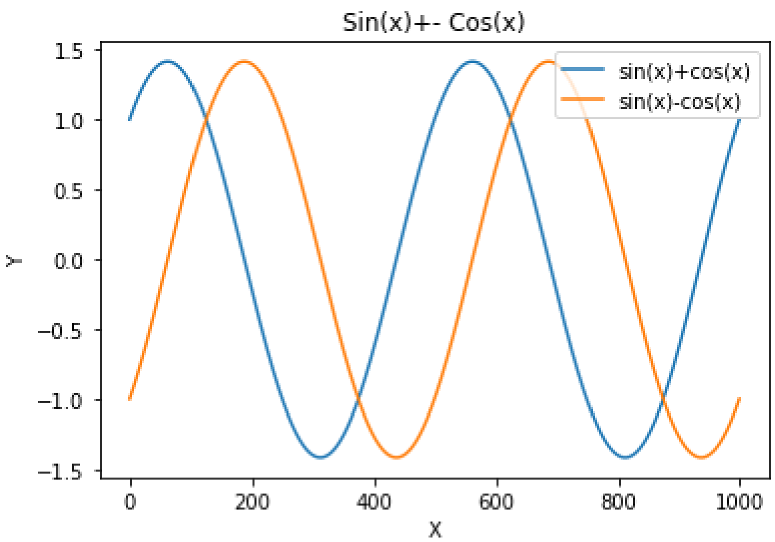


In [5]:

```
x=np.linspace(-2*np.pi,2*np.pi,1000)
plot(np.sin(x)+np.cos(x))
plot(np.sin(x)-np.cos(x))
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Sin(x)+- Cos(x)")
plt.legend(["sin(x)+cos(x)","sin(x)-cos(x)"])
```

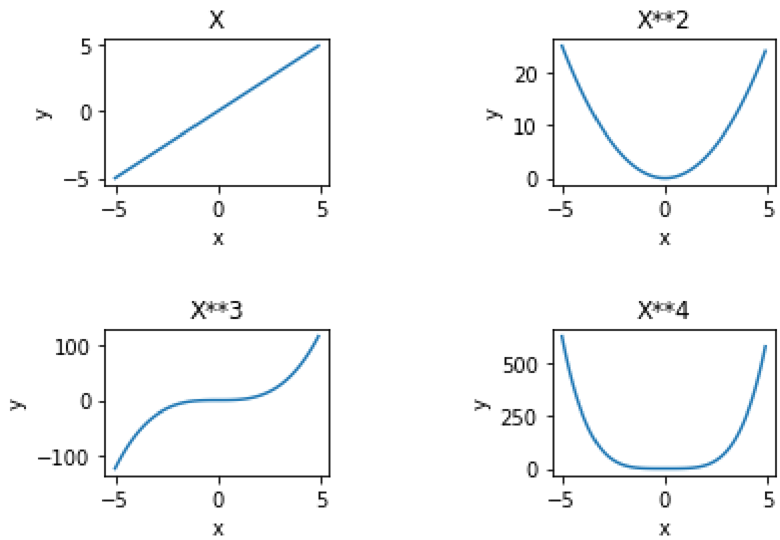
Out[5]:

```
<matplotlib.legend.Legend at 0x15c84cab550>
```



In [22]:

```
x=arange(-5,5,0.1)
plt.subplot(2,2,1)
plt.plot(x,x)
plt.title("X")
plt.xlabel("x")
plt.ylabel("y")
plt.subplot(2,2,2)
plt.plot(x,x**2)
plt.title("X**2")
plt.xlabel("x")
plt.ylabel("y")
plt.subplot(2,2,3)
plt.plot(x,x**3)
plt.title("X**3")
plt.xlabel("x")
plt.ylabel("y")
plt.subplot(2,2,4)
plt.plot(x,x**4)
plt.title("X**4")
plt.xlabel("x")
plt.ylabel("y")
plt.gcf().subplots_adjust(hspace=1)
plt.gcf().subplots_adjust(wspace=1)
```



Conclusion:

```
Attributes of matplotlib.pyplot were explored , their basic understanding was obta
ined.
```

Finding Roots

25-06-2019

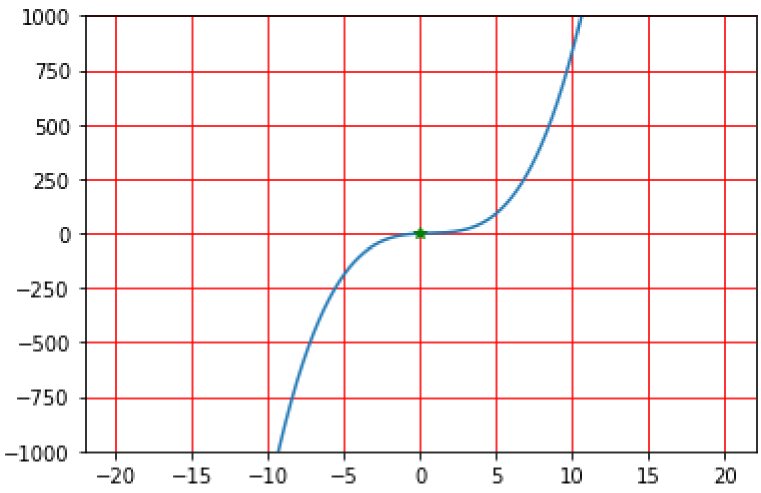
Aim: To solve and equation and plot the same along with its roots

In [17]:

```
x=np.linspace(-20,20,10000)
plt.plot(x,x**3-2*x**2+3*x-1)
plt.grid(color='r')
plt.plot(0,0,'g*')
plt.ylim(-1000,1000)
```

Out[17]:

```
(-1000, 1000)
```



In [16]:

```
from scipy import optimize
help(optimize)
```

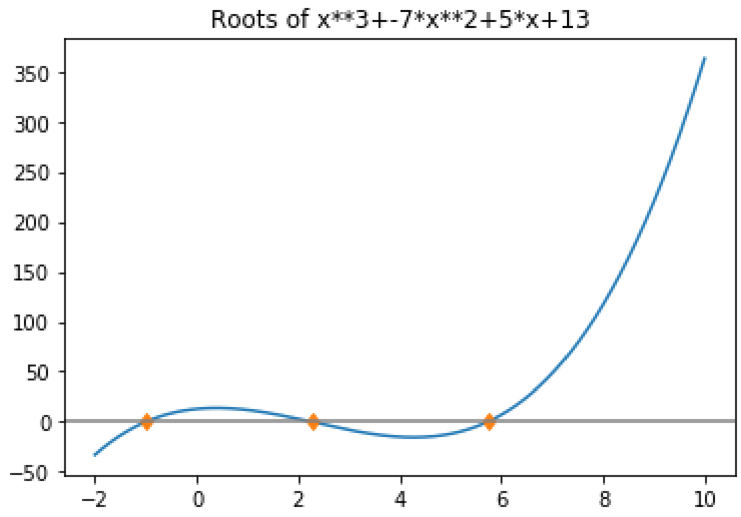
```
Help on package scipy.optimize in scipy:

NAME
    scipy.optimize

DESCRIPTION
    =====================================================
    Optimization and Root Finding (:mod:`scipy.optimize`)
    =====================================================

    .. currentmodule:: scipy.optimize

    SciPy ``optimize`` provides functions for minimizing (or maximizing)
    objective functions, possibly subject to constraints. It includes
    solvers for nonlinear problems (with support for both local and global
    optimization algorithms), linear programing,  constrained
    and nonlinear least-squares, root finding and curve fitting.

    Common functions and objects, shared across different solvers, are:
```

In [11]:

```
def func(x):
   # a=input("Enter coeff for x**3")
   # b=input("Enter coeff for x**2")
   # c=input("Enter coeff for x")
   # d=input("Enter the constant")
    a=1
    b=-7
    c=5
    d=13
    return a*x**3+b*x**2+c*x+d
```

In [21]:

```
x=np.linspace(-2,10,1000)
sol=optimize.root(func,[-2,2,10])
plt.plot(x,func(x))
plt.plot(sol.x,func(sol.x),'d')
plt.axhline(0,-2,2,color="gray")
plt.title("Roots of x**3+-7*x**2+5*x+13")
plt.show()
```



Conclusion:

```
Introduced to scipy.optimize to find roots of an equation
Plotted Roots along with its function
```

Lab 2: Solutions of algebraic and transcendental equations

Bisection Method

25-06-2019

AIM: To find an approximate root of an equtaion using Bisection Menthod

In [15]:

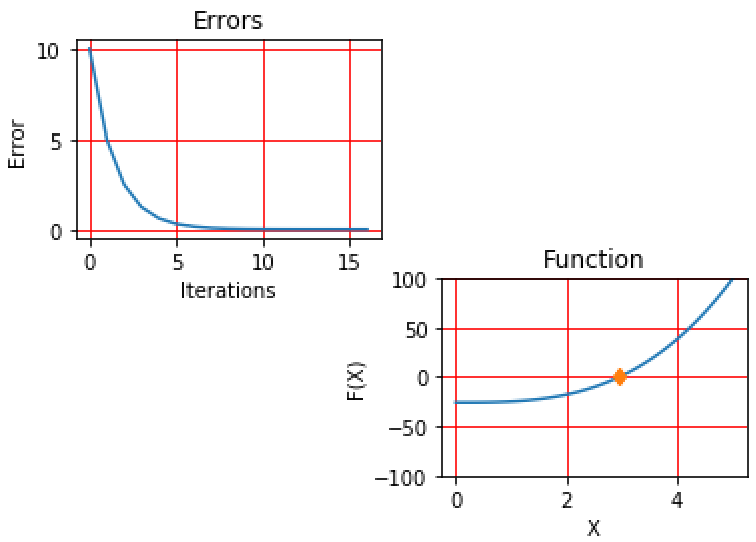
```
absol=[]
def func(x):
    return x**3-26

dash = '-' * 75

def bisection():
    a=int(input("Enter a"))
    b=int(input("Enter b"))
    if (func(a) * func(b) >= 0):
        print("You have not assumed right a and b\n")
        return

    c = a
    i=0
    if(i==0):
        #print("No\t\t  A\t\t\tB\t\tApproximation\t\t\t\tf(c)")
        print(dash)
        print('{:>12s}{:>12s}{:>12s}{:>12s}{:>12s}{:>12s}'.format('iteration','a','b','abs(
        print(dash)

    while ((b-a) >= 0.0001):
        #print(i+1,"\t\t",end=" ")
        #print("%.3f"%a,"\t\t\t",end=" ")

        #print("%.3f"%b,"\t\t",end=" ")
        # Find middle point
        c = (a+b)/2
        absol.append(abs(b-a))
        #print("%.6f"%c,"\t\t\t  ",end=" ")
        i=i+1
        #absol.append(abs(b-a))
        print('{:>12d}{:>12.6f}{:>12.6f}{:>12.6f}{:>12.6f}{:>12.6f}'.format(i+1,a,b,abs(b-a
        # Check if middle point is root
        if (func(c) == 0.0):
            break
        else:
            #print("%.6f"%func(c))
        # Decide the side  of the interval to repeat the  next steps
            if (func(c)*func(a) < 0):
                b = c
            else:
                a = c
    return c

    #print("The value of root is : ","%.4f"%c)
    #print(func(c))



root=bisection()
print("The root using bisection method is %.6f"%root)

from scipy import optimize
rangex=np.linspace(0,5,100)
plt.subplot(2,2,1)
plt.plot(absol)
plt.xlabel("Iterations")
plt.ylabel("Error")
```

```
plt.grid(color='r')
plt.title("Errors")
sol=optimize.root(func,[0,5])
plt.subplot(2,2,4)
plt.plot(rangex,func(rangex))
plt.grid(color='r')
plt.plot(sol.x,func(sol.x),'d')
plt.ylim(-100,100)
plt.xlabel("X")
plt.ylabel("F(X)")
plt.title("Function")
```

```
Enter a-5
Enter b5
---------------------------------------------------------------------------
   iteration          a           b     abs(b-a)          c         f(c)
---------------------------------------------------------------------------
           2  -5.000000    5.000000   10.000000    0.000000  -26.000000
           3   0.000000    5.000000    5.000000    2.500000  -10.375000
           4   2.500000    5.000000    2.500000    3.750000   26.734375
           5   2.500000    3.750000    1.250000    3.125000    4.517578
           6   2.500000    3.125000    0.625000    2.812500   -3.752686
           7   2.812500    3.125000    0.312500    2.968750    0.165009
           8   2.812500    2.968750    0.156250    2.890625   -1.846767
           9   2.890625    2.968750    0.078125    2.929688   -0.854290
          10   2.929688    2.968750    0.039062    2.949219   -0.348016
          11   2.949219    2.968750    0.019531    2.958984   -0.092350
          12   2.958984    2.968750    0.009766    2.963867    0.036117
          13   2.958984    2.963867    0.004883    2.961426   -0.028170
          14   2.961426    2.963867    0.002441    2.962646    0.003961
          15   2.961426    2.962646    0.001221    2.962036   -0.012108
          16   2.962036    2.962646    0.000610    2.962341   -0.004074
          17   2.962341    2.962646    0.000305    2.962494   -0.000057
          18   2.962494    2.962646    0.000153    2.962570    0.001952
The root using bisection method is 2.962570
```

Out[15]:

```
Text(0.5, 1.0, 'Function')
```



In [7]:

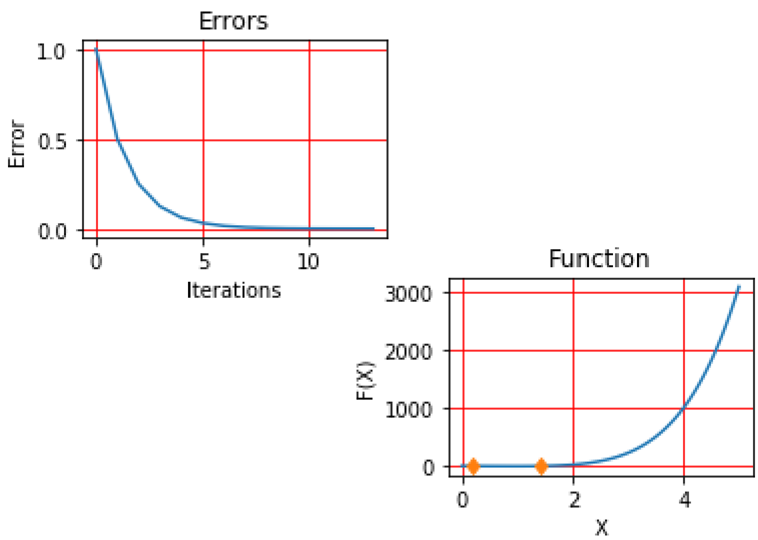
```
import numpy as np
import matplotlib.pyplot as plt
absol=[]
def func(x):
    return x**5-5*x+1

dash = '-' * 75

def bisection():
    a=int(input("Enter a"))
    b=int(input("Enter b"))
    if (func(a) * func(b) >= 0):
        print("You have not assumed right a and b\n")
        return

    c = a
    i=0
    if(i==0):
        #print("No\t\t  A\t\t\tB\t\tApproximation\t\t\t\tf(c)")
        print(dash)
        print('{:>12s}{:>12s}{:>12s}{:>12s}{:>12s}{:>12s}'.format('iteration','a','b','abs(
        print(dash)

    while ((b-a) >= 0.0001):
        #print(i+1,"\t\t",end=" ")
        #print("%.3f"%a,"\t\t\t",end=" ")

        #print("%.3f"%b,"\t\t",end=" ")
        # Find middle point
        c = (a+b)/2
        absol.append(abs(b-a))
        #print("%.6f"%c,"\t\t\t  ",end=" ")
        i=i+1
        #absol.append(abs(b-a))
        print('{:>12d}{:>12.6f}{:>12.6f}{:>12.6f}{:>12.6f}{:>12.6f}'.format(i+1,a,b,abs(b-a
        # Check if middle point is root
        if (func(c) == 0.0):
            break
        else:
            #print("%.6f"%func(c))
        # Decide the side  of the interval to repeat the  next steps
            if (func(c)*func(a) < 0):
                b = c
            else:
                a = c
    return c

    #print("The value of root is : ","%.4f"%c)
    #print(func(c))



root=bisection()
print("The root using bisection method is %.6f"%root)

from scipy import optimize
rangex=np.linspace(0,5,100)
plt.subplot(2,2,1)
plt.plot(absol)
```

```
plt.xlabel("Iterations")
plt.ylabel("Error")
plt.grid(color='r')
plt.title("Errors")
sol=optimize.root(func,[0,5])
plt.subplot(2,2,4)
plt.plot(rangex,func(rangex))
plt.grid(color='r')
plt.plot(sol.x,func(sol.x),'d')
plt.xlabel("X")
plt.ylabel("F(X)")
plt.title("Function")
```

```
Enter a0
Enter b1
------------------------------------------------------------------------------
   iteration          a           b      abs(b-a)           c        f(c)
------------------------------------------------------------------------------
           2   0.000000    1.000000    1.000000    0.500000   -1.468750
           3   0.000000    0.500000    0.500000    0.250000   -0.249023
           4   0.000000    0.250000    0.250000    0.125000    0.375031
           5   0.125000    0.250000    0.125000    0.187500    0.062732
           6   0.187500    0.250000    0.062500    0.218750   -0.093249
           7   0.187500    0.218750    0.031250    0.203125   -0.015279
           8   0.187500    0.203125    0.015625    0.195312    0.023722
           9   0.195312    0.203125    0.007812    0.199219    0.004220
          10   0.199219    0.203125    0.003906    0.201172   -0.005530
          11   0.199219    0.201172    0.001953    0.200195   -0.000655
          12   0.199219    0.200195    0.000977    0.199707    0.001783
          13   0.199707    0.200195    0.000488    0.199951    0.000564
          14   0.199951    0.200195    0.000244    0.200073   -0.000046
          15   0.199951    0.200073    0.000122    0.200012    0.000259
The root using bisection method is 0.200012
```

Out[7]:

```
Text(0.5, 1.0, 'Function')
```



Cocnlusion:

The Bisection method was used to find the approximate roots of the given equtaions

Newton Raphson Method

069-07-2109

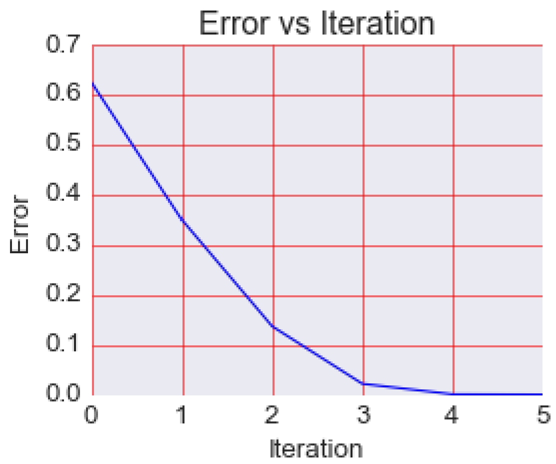
In [20]:

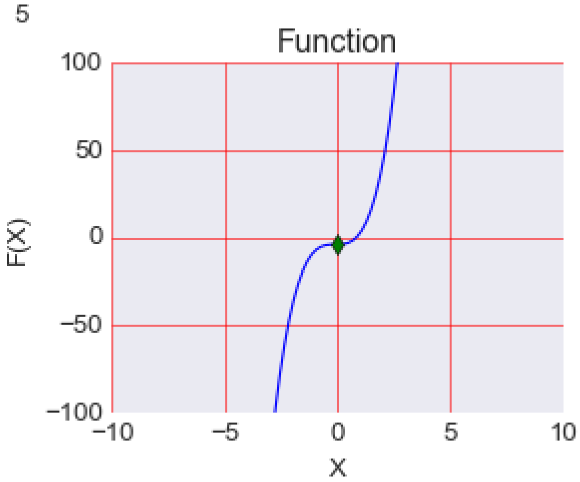
```
import math
import numpy as np
def func( x ):
    return (5*x**3+x**2-4)
    #return x*math.sin(x)+math.cos(x)
def derreturn(x):
    #return x*math.sin(x)
    return (15*x**2+2*x)
hlist=[]
def newtonR(x):
    h= func(x)/derreturn(x)
    i=0
    while abs(h) >= 0.0001:
        i+=1
        #print("X: ",x)
        #print('F(x) : ',func(x))
        #print("f'(x)",derreturn(x))
        h= (func(x)/derreturn(x))
        hlist.append(h)
        print('{:>12d}{:>12.6f}{:>12.6f}{:>12.6f}'.format(i,x,func(x),derreturn(x)),end=" "
        x = x - h
        print('{:>12.6f}'.format(x))

    print("Value of Root is : ",x)
X=float(input("Enter Approximate Root "))
print('{:>12s}{:>12s}{:>12s}{:>12s}{:>12s}'.format('iteration','x','f(x)',"f'(x)"," X calcu
newtonR(X)
rangex=np.linspace(-10,10,100)
sol=optimize.root(func,[0,5])
plt.subplot(2,2,4)
plt.plot(rangex,func(rangex))
plt.grid(color='r')
plt.plot(sol.x,func(sol.x),'d')
plt.ylim(-100,100)
plt.xlabel("X")
plt.ylabel("F(X)")
plt.title("Function")

plt.subplot(2,2,1)
plt.plot(hlist)
plt.grid(color='r')
plt.xlabel("Iteration")
plt.ylabel("Error")
plt.title("Error vs Iteration")
```

```
Enter Approximate Root 2
   iteration           x        f(x)       f'(x) X calculated
           1    2.000000   40.000000   64.000000    1.375000
           2    1.375000   10.888672   31.109375    1.024987
           3    1.024987    2.434855   17.808964    0.888267
           4    0.888267    0.293309   13.611800    0.866719
           5    0.866719    0.006601   13.001454    0.866211
           6    0.866211    0.000004   12.987241    0.866211
Value of Root is :  0.866210602253048
```

Out[20]:

```
Text(0.5, 1.0, 'Error vs Iteration')
```


Error vs Iteration
Error
Iteration
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
1
2
3
4
5


Function
F(X)
X
100
50
0
−50
−100
−10
−5
0
5
10

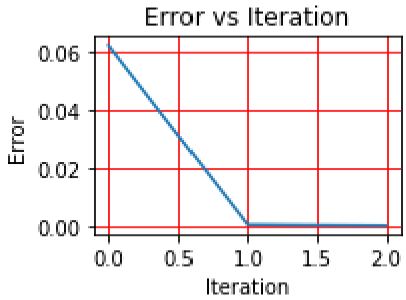
In [8]:

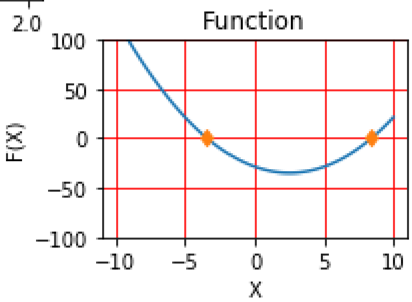
```
import math
import numpy as np
def func( x ):
    return (x**2-5*x-29)
    #return x*math.sin(x)+math.cos(x)
def derreturn(x):
    #return x*math.sin(x)
    return (2*x-5)
hlist=[]
def newtonR(x):
    h= func(x)/derreturn(x)
    i=0
    while abs(h) >= 0.0001:
        i+=1
        #print("X: ",x)
        #print('F(x) : ',func(x))
        #print("f'(x)",derreturn(x))
        h= (func(x)/derreturn(x))
        hlist.append(h)
        print('{:>12d}{:>12.6f}{:>12.6f}{:>12.6f}'.format(i,x,func(x),derreturn(x)),end=" "
        x = x - h
        print('{:>12.6f}'.format(x))

    print("Value of Root is : ",x)
X=float(input("Enter Approximate Root "))
print('{:>12s}{:>12s}{:>12s}{:>12s}{:>12s}'.format('iteration','x','f(x)',"f'(x)"," X calcu
newtonR(X)
rangex=np.linspace(-10,10,100)
sol=optimize.root(func,[0,5])
plt.subplot(2,2,4)
plt.plot(rangex,func(rangex))
plt.grid(color='r')
plt.plot(sol.x,func(sol.x),'d')
plt.ylim(-100,100)
plt.xlabel("X")
plt.ylabel("F(X)")
plt.title("Function")

plt.subplot(2,2,1)
plt.plot(hlist)
plt.grid(color='r')
plt.xlabel("Iteration")
plt.ylabel("Error")
plt.title("Error vs Iteration")
```

```
Enter Approximate Root 8.5
   iteration           x        f(x)       f'(x) X calculated
           1    8.500000    0.750000   12.000000     8.437500
           2    8.437500    0.003906   11.875000     8.437171
           3    8.437171    0.000000   11.874342     8.437171
Value of Root is :  8.437171043518958
```

Out[8]:

```
Text(0.5, 1.0, 'Error vs Iteration')
```





Cocnlusion:

```
The Newton Raphson method was used to find the approximate roots of the given equt
aions
```

16-07-2019

A bungee jumper with the mass og 68.1 Kg leaps froma stationary hot air balloon. Use equation to compute velocity for the first 12s of free fall. Also detrmine the terminal velocity that will be attained for an infinitely long cord. Use drag coefficient of .25Kg/m

In [2]:

```
time=[1,2,3,4,5,6,7,8,9,10,11,12]
import math
import numpy as np
def velocity(m,t,cd):

    vel=[]
    for i in range(1,t+1):
            g=9.8
            ans=math.sqrt(g*m/cd)*np.tanh(math.sqrt(g*cd/m)*i)
            vel.append(ans)
    return vel


m=float(input("Enter Mass: "))
t=int(input("Enter How many seconds we take in consideration "))
cd=float(input("Enter drag coefficient"))
veloc=velocity(m,t,cd)

print("Time (S)\t Velocity(m/s)")
print(" ")
for i  in range(t):

    print(time[i],end=" ")
    print("\t\t",veloc[i])
```

```
Enter Mass: 68.1
Enter How many seconds we take in consideration 12
Enter drag coefficient.25
Time (S)         Velocity(m/s)

1                9.684143706522294
2                18.71095473908489
3                26.59022791243773
4                33.08315003389213
5                38.18457821708906
6                42.04464933704515
7                44.88304249226504
8                46.92655847112579
9                48.37553317803714
10               49.39186909226721
11               50.09933795182978
12               50.58919926157048
```

Out[2]:

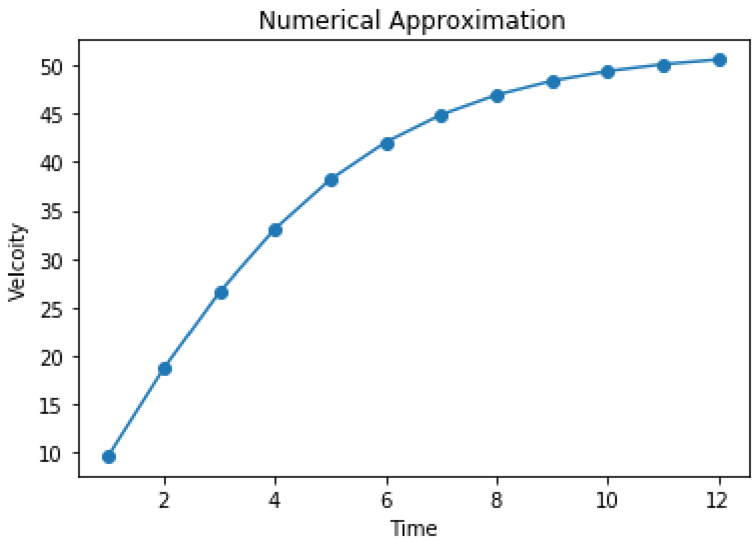
```
Text(0, 0.5, 'Velcoity')
```

In [3]:

```
import matplotlib.pyplot as plt
plt.plot(time,veloc,marker="o")
plt.title("Numerical Approximation")
plt.xlabel("Time")
plt.ylabel("Velcoity")
```

Out[3]:

```
Text(0, 0.5, 'Velcoity')
```



Use bisection method to determine the drag coefficient needed so that an 80 - kg bungee jumper has a velocity of 36 m/s after 4s of free fall.Note: The acceleration of gravity is 9.81 m/s^2. Start with an initial guesses of x(l) = 0.1 and x(u) = 0.2 iterate until the approximate relative error falls below 2%.

$$v(t) = \sqrt{\frac{gm}{cd}} tanh \sqrt{\frac{gcd}{m}} t$$

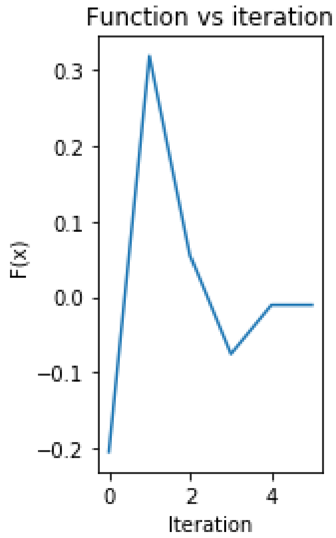
$$f(cd) = \sqrt{\frac{9.81*80}{cd}} tanh(\sqrt{\frac{9.81cd}{80}} 4) - 36$$

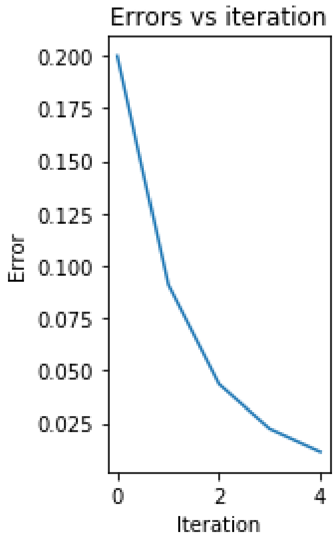
In [3]:

```
from scipy import optimize
import math
import numpy as np
import matplotlib.pyplot as plt

def fun(x)->float:
    return math.sqrt(9.81*80/x)*np.tanh(math.sqrt(9.81*x/80)*4)-36

def nextapprox(a, b)->float:
    return (a+b)/2

if __name__=="__main__":
    a=float(input("Enter lower limit: "))
    b=float(input("Enter upper limit: "))
    X=np.linspace(a-1,b+1,1000)

    print()
    print("a={0}".format(a))
    print("b={0}".format(b))
    neg=0.0
    pos=0.0
    count=0
    print("f(a)={0}\nf(b)={1}\n\n".format(round(fun(a),6),round(fun(b),6)))
    error=[]
    funlist=[]
    if fun(a)*fun(b)<0.0:
        dash = '-' * 113
        print(dash)
        print("x\t\t   a\t\t       b\t\t     Aprroximation\t\t f(approx)\tRel err")
        print(dash)
        print()
        if fun(a)<0.0:
            neg=a
            pos=b
        else:
            neg=b
            pos=a
        print("{0}\t\t{1:.6f}\t\t{2:.6f}\t\t{3:.6f}\t\t{4:.6f}\t{5:.6f}".format(count+1,rou
        while True:
            count=count+1
            if fun(neg)*fun(pos)<0.0:
                print()
                x0=nextapprox(neg, pos)
                funlist.append(fun(x0))
                if fun(x0)<0:
                    neg=round(x0, 6)
                else:
                    pos=round(x0, 6)
                x1=nextapprox(neg, pos)
                error.append(abs(pos-neg)/abs(pos+neg))
                #sol=optimize.root(fun,[1,4])
                print("{0}\t\t{1:.6f}\t\t{2:.6f}\t\t{3:.6f}\t\t{4:.6f}\t{5:.6f}".format(cou
            #if math.trunc(10**3 * x0) / 10**3==math.trunc(10**3 * x1) / 10**3:
            if(abs(pos-neg)/abs(pos+neg) < 0.02) :
                print()
                print("Approximate root is {0}".format(round(x1,6)))
                funlist.append(fun(x0))
                break
    else:
```

```
        print()
        print("Invalid interval entered")

plt.subplot(1,2,1)
plt.title("Function vs iteration")
plt.plot(funlist)
plt.xlabel("Iteration")
plt.ylabel("F(x)")
plt.subplot(1,2,2)
plt.title("Errors vs iteration")
plt.xlabel("Iteration")
plt.ylabel("Error")
plt.plot(error)
plt.gcf().subplots_adjust(hspace=1)
plt.gcf().subplots_adjust(wspace=1)
```

```
Enter lower limit: 0.1
Enter upper limit: 0.2

a=0.1
b=0.2
f(a)=0.860291
f(b)=-1.19738


---------------------------------------------------------------------------
------------------------------------
x                 a                   b                     Aprroximation
f(approx)       Rel err
---------------------------------------------------------------------------
------------------------------------

1               0.200000              0.100000                 0.150000
-0.204516       -0.100000

2               0.150000              0.100000                 0.125000
0.318407        0.050000

3               0.150000              0.125000                 0.137500
0.054639        0.025000

4               0.150000              0.137500                 0.143750
-0.075508       0.012500

5               0.143750              0.137500                 0.140625
-0.010578       0.006250

6               0.140625              0.137500                 0.139063
0.021995        0.003125

Approximate root is 0.139063
```





The volume of liquid V in a hollow horizontal cylinder of radius r and length L is related to the depth of the liquid h by

$V = [r^2 cos^{-1}(\frac{r-h}{r}) - (r-h)\sqrt{2rh - h^2}]L$

Determine h given r=2m,L=5m and V=8

In [28]:

```
import math
import matplotlib.pyplot as plt
import numpy as np
def func( x ):
    #if ((2-x)/x)>1 or (2-x)/x<-1:
    #    return 0
    #else:
    # return ((4*math.acos((2-x)/x)-(2-x)*math.sqrt(4*x-x**2))*5-8)
        return ((2**2)*math.acos((2-x)/2)-(2-x)*math.sqrt(2*2*x-x*x))*5-8
def derreturn(x):
    #return 5*((4/math.sqrt(-(x**2)+4*x))-(2*(x**2)-8*x+4)/math.sqrt(4*x-x**2))
    return (((2**2)*(1/np.sqrt(1-((2-x)/2)**2))+np.sqrt(2*2*x-x**2)-(2-x)*(2*2-2*x)/2*np.sc
hlist=[]
def newtonR(x):
    h= func(x)/derreturn(x)
    i=0
    while abs(h) >= 0.0001:
        i+=1
        h= (func(x)/derreturn(x))
        hlist.append(h)
        print('{:>12d}{:>12.6f}{:>12.6f}{:>12.6f}'.format(i,x,func(x),derreturn(x)),end=" "
        x = x - h
        print('{:>12.6f}'.format(x))

    print("Value of Root is :{:>12.6f}".format(x))
X=float(input("Enter Approximate Root "))
print('{:>12s}{:>12s}{:>12s}{:>12s}{:>12s}'.format('iteration','x','f(x)',"f'(x)"," X calcu
newtonR(X)
```

```
Enter Approximate Root 2
   iteration           x        f(x)       f'(x) X calculated
           1    2.000000   23.415927   30.000000     1.219469
           2    1.219469    8.211162   25.320389     0.895179
           3    0.895179    2.501374   22.154013     0.782270
           4    0.782270    0.663416   21.381523     0.751243
           5    0.751243    0.174894   21.234646     0.743006
           6    0.743006    0.046496   21.201582     0.740813
           7    0.740813    0.012400   21.193224     0.740228
           8    0.740228    0.003310   21.191026     0.740072
           9    0.740072    0.000884   21.190441     0.740030
Value of Root is :    0.740030
```

You buy a 35000 vehicle for no down payment and 8500 per year for 7 years. Use the bisection function to determine the interest rate that you pay . employ initial guesses for the interest rate of 0.01 and 0.3, the stoppin criteria of 0.00005 the formula relating the present worth(P), annual payments(A), no. of years(n) and rate(i) is given by $A = \frac{P(i(i+1)^n)}{(1+i)^n - 1}$

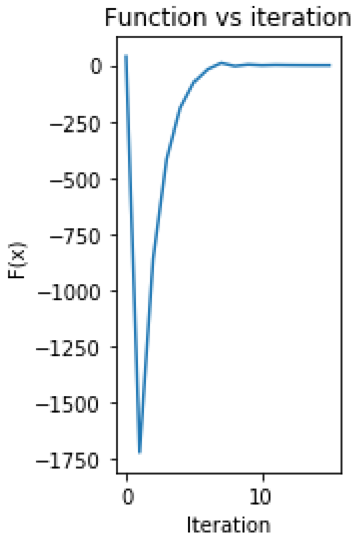
In [52]:

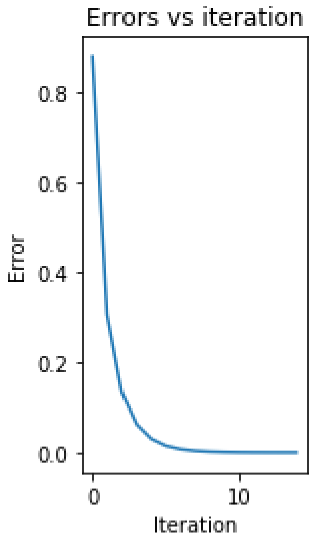
```
from scipy import optimize
import math
import numpy as np
import matplotlib.pyplot as plt

def fun(x)->float:
    return ((35000*(x*(1+x)**7))/(((1+x)**7)-1))-8500

def nextapprox(a, b)->float:
    return (a+b)/2

if __name__=="__main__":
    a=float(input("Enter lower limit: "))
    b=float(input("Enter upper limit: "))
    X=np.linspace(a-1,b+1,1000)

    print()
    print("a={0}".format(a))
    print("b={0}".format(b))
    neg=0.0
    pos=0.0
    count=0
    print("f(a)={0}\nf(b)={1}\n\n".format(round(fun(a),6),round(fun(b),6)))
    error=[]
    funlist=[]
    if fun(a)*fun(b)<0.0:
        dash = '-' * 113
        print(dash)
        print("x\t\t   a\t\t       b\t\t     Aprroximation\t\t f(approx)\tRel err")
        print(dash)
        print()
        if fun(a)<0.0:
            neg=a
            pos=b
        else:
            neg=b
            pos=a
        print("{0}\t\t{1:.6f}\t\t{2:.6f}\t\t{3:.6f}\t\t{4:.6f}\t{5:.6f}".format(count+1,rou
        while True:
            count=count+1
            if fun(neg)*fun(pos)<0.0:
                print()
                x0=nextapprox(neg, pos)
                funlist.append(fun(x0))
                if fun(x0)<0:
                    neg=round(x0, 6)
                else:
                    pos=round(x0, 6)
                x1=nextapprox(neg, pos)
                error.append(abs(pos-neg)/abs(pos+neg))
                #sol=optimize.root(fun,[1,4])
                print("{0}\t\t{1:.6f}\t\t{2:.6f}\t\t{3:.6f}\t\t{4:.6f}\t{5:.6f}".format(cou
            #if math.trunc(10**3 * x0) / 10**3==math.trunc(10**3 * x1) / 10**3:
            if(abs(pos-neg)/abs(pos+neg) < 0.00005) :
                print()
                print("Approximate root is {0}".format(round(x1,6)))
                funlist.append(fun(x0))
                break
    else:
```

```
        print()
        print("Invalid interval entered")
plt.subplot(1,2,1)
plt.title("Function vs iteration")
plt.plot(funlist)
plt.xlabel("Iteration")
plt.ylabel("F(x)")
plt.subplot(1,2,2)
plt.title("Errors vs iteration")
plt.xlabel("Iteration")
plt.ylabel("Error")
plt.plot(error)
plt.gcf().subplots_adjust(hspace=1)
plt.gcf().subplots_adjust(wspace=1)
```

```
Enter lower limit: 0.01
Enter upper limit: 0.3

a=0.01
b=0.3
f(a)=-3298.010098
f(b)=3990.57729


----------------------------------------------------------------------------
-------------------------------------
x                  a                   b                      Aprroximation
f(approx)       Rel err
----------------------------------------------------------------------------
-------------------------------------

1               0.010000               0.300000               0.155000
39.163831       0.290000

2               0.010000               0.155000               0.082500
-1719.879252    0.145000

3               0.082500               0.155000               0.118750
-861.252456     0.072500

4               0.118750               0.155000               0.136875
-416.049835     0.036250

5               0.136875               0.155000               0.145937
-189.667160     0.018125

6               0.145937               0.155000               0.150469
-75.560624      0.009063

7               0.150469               0.155000               0.152734
-18.267314      0.004531

8               0.152734               0.155000               0.153867
10.423166       0.002266

9               0.152734               0.153867               0.153301
-3.933100       0.001133

10              0.153301               0.153867               0.153584
3.250195        0.000566
```

```
11            0.153301        0.153584        0.153442
-0.335412     0.000283

12            0.153442        0.153584        0.153513
1.450983      0.000142

13            0.153442        0.153513        0.153477
0.551433      0.000071

14            0.153442        0.153477        0.153459
0.095337      0.000035

15            0.153442        0.153459        0.153450
-0.132708     0.000017

16            0.153450        0.153459        0.153454
-0.031355     0.000009

Approximate root is 0.153454
```





Regular Falsi Method

To find the approximate root of the given equation using Method of False Position/ Regula-Falsi Method

In [2]:

```
import math

def fun(x)->float:
    return round((x*math.exp(x)-3), 6)

def nextapprox(a, b)->float:
    return (a*fun(b)-b*fun(a))/(fun(b)-fun(a))

if __name__=="__main__":
    a=float(input("Enter lower limit: "))
    b=float(input("Enter upper limit: "))
    print()
    print("x1={0}".format(a))
    print("x2={0}".format(b))
    neg=0.0
    pos=0.0
    count=0
    if fun(a)*fun(b)<0.0:
        if fun(a)<0.0:
            neg=a
            pos=b
        else:
            neg=b
            pos=a
        while True:
            count=count+1
            if fun(neg)*fun(pos)<0.0:
                print()
                x0=nextapprox(neg, pos)
                print("Approximation {1} is x{2}={0}".format(round(x0,6), count, count+2))
                print("f({0}) is {1}".format(round(x0, 6), round(fun(x0), 6)))
                if fun(x0)<0:
                    neg=round(x0, 6)
                else:
                    pos=round(x0, 6)
                print("Now root lies in ({0}, {1})".format(neg, pos))
                x1=nextapprox(neg, pos)
                print("Next approximation will be x={0}".format(round(x1, 6)))
            if math.trunc(10**3 * x0) / 10**3==math.trunc(10**3 * x1) / 10**3:
                print()
                print("Approximate root is {0}".format(math.trunc(10**3 * x1) / 10**3))
                break
    else:
        print()
        print("Invalid interval entered")
```

```
Enter lower limit: 0
Enter upper limit: 5

x1=0.0
x2=5.0

Approximation 1 is x3=0.020214
f(0.020214) is -2.979373
Now root lies in (0.020214, 5.0)
Next approximation will be x=0.040208

Approximation 2 is x4=0.040208
f(0.040208) is -2.958142
```

```
Now root lies in (0.040208, 5.0)
Next approximation will be x=0.059981

Approximation 3 is x5=0.059981
f(0.059981) is -2.936312
Now root lies in (0.059981, 5.0)
```

In [ ]:

```

```

In [ ]:

```

```