A bungee jumper with a mass of 68.1 kgs leaps from a stationary hot air balloon. Use the equation to compute velocity for the first 12s of free fall. Also determine the terminal velocity that will be attained for an infinitely long cord. Use a drag coefficient of 0.25 kg/m.

$$\frac{d^2f}{dx^2} - 5f = 0$$

$$\frac{dv}{dt} = g - \frac{Cd}{m}v^2$$

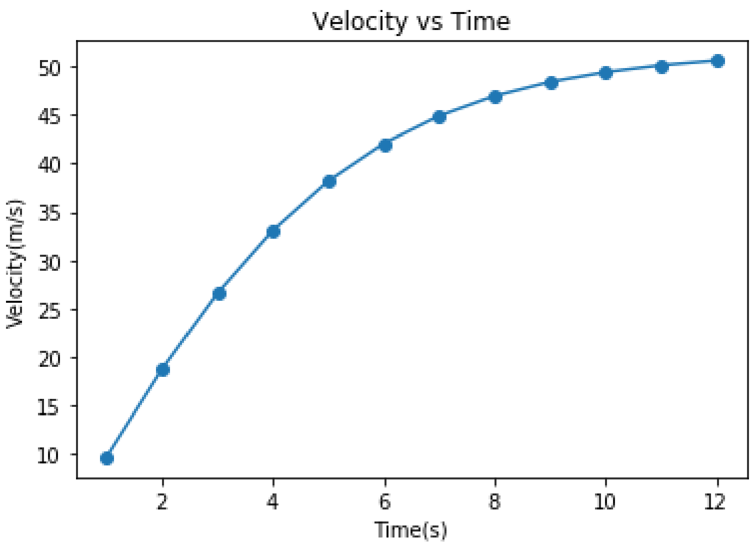
In [39]:

```
import math
import numpy as np
import matplotlib.pyplot as plt

print("------------------------------")
print("|T (s)      |      Velocity(m/s)|")
print("------------------------------")
def velocity(m,t,cd):
    g = 9.8
    return (math.sqrt(g*m/cd)*np.tanh(math.sqrt(g*cd/m)*t))

velist=[]
for i in range(1,13):
    vel=velocity(68.1,i,0.25)
    print("|  {0} \t  | \t  {1}  ".format(i,round(vel,4)))
    velist.append(vel)

t = range(1,13)
plt.plot(t,velist,marker="o")
plt.title("Velocity vs Time")
plt.xlabel('Time(s)')
plt.ylabel('Velocity(m/s)')
```

```
------------------------------
|T (s)      |      Velocity(m/s)|
------------------------------
|  1        |        9.6841
|  2        |        18.711
|  3        |        26.5902
|  4        |        33.0832
|  5        |        38.1846
|  6        |        42.0446
|  7        |        44.883
|  8        |        46.9266
|  9        |        48.3755
|  10       |        49.3919
|  11       |        50.0993
|  12       |        50.5892
```

Out[39]:

<matplotlib.text.Text at 0x14531c091d0>



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 and 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 [16]:

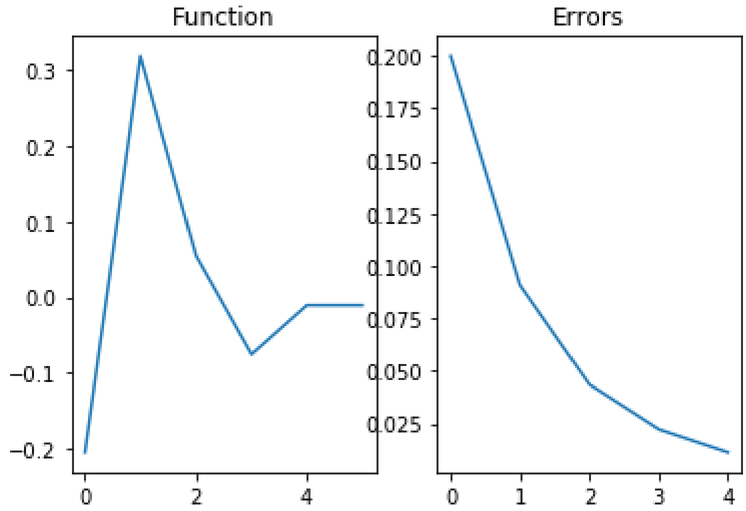
```
from scipy import optimize
import math
import numpy as np
import matplotlib.pyplot as plt

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")
plt.plot(funlist)
plt.subplot(1,2,2)
plt.title("Errors")
plt.plot(error)
```

```
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
```

Out[16]:

```
[<matplotlib.lines.Line2D at 0x1f5d7370ba8>]
```



The volume of liquid V in a hollow horizontal cylinder of radious 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 = $5m$; V = 8m^3 using Newton Raphson Method

$$8 = [4\cos^{-1}(\frac{2-h}{2}) - (2-h)\sqrt{4h-h^2}]5$$

$$f(h) = [4\cos^{-1}(\frac{2-h}{2}) - (2-h)\sqrt{4h-h^2}]5 - 8$$

In [6]:

```
import math
import numpy as np

X=list()
X1=list()
r=2
l=5
v=8

pres=float(input('enter precision limit: '))
def fun(h):
    return ((r**2)*math.acos((r-h)/r)-(r-h)*math.sqrt(2*r*h-h*h))*l-v
def derv(h):
    return (((r**2)*(1/np.sqrt(1-((2-h)/2)**2))+np.sqrt(2*r*h-h**2)-(r-h)*(2*r-2*h)/2*np.sc

x0=float(input("enter the initial guess: "))
dash = '-' * 66
X.append(0)
X.append(x0)
X1.append(abs(X[-2]-X[-1]))
print(dash)
print('|{:^12s}|{:^12s}|{:^12s}|{:^12s}|{:^12s}|'.format('iteration','h','f(h)',"f'(h)",'er
print(dash)
i=0
print('|{:^12d}|{:^12.6f}|{:^12.6f}|{:^12.6f}|{:^12.6f}|'.format(i,X[-1],fun(X[-1]),derv(X[
while(True):

    X.append(X[-1]-(fun(X[-1])/derv( X[-1])))

    X1.append(abs(X[-2]-X[-1]))
    i=i+1
    print('|{:^12d}|{:^12.6f}|{:^12.6f}|{:^12.6f}|{:^12.6f}|'.format(i,X[-1],fun(X[-1]),der
    if fun(X[-1])==0:
        break
    if X1[-1]<pres:
        break
print(dash)
print('By Newton Raphson method the root is {:.6f} at the {}th iteration'.format(X[-1],i))
```

```
enter precision limit: 0.001
enter the initial guess: 3
------------------------------------------------------------------
| iteration  |     h      |    f(h)    |   f'(h)    |   error    |
------------------------------------------------------------------
|     0      |  3.000000  | 42.548156  | 23.094011  |  3.000000  |
|     1      |  1.157611  |  7.080466  | 24.685103  |  1.842389  |
|     2      |  0.870779  |  2.096596  | 21.961037  |  0.286832  |
|     3      |  0.775310  |  0.553181  | 21.345652  |  0.095469  |
|     4      |  0.749395  |  0.146042  | 21.227002  |  0.025915  |
|     5      |  0.742515  |  0.038851  | 21.199693  |  0.006880  |
|     6      |  0.740682  |  0.010363  | 21.192730  |  0.001833  |
|     7      |  0.740193  |  0.002766  | 21.190895  |  0.000489  |
------------------------------------------------------------------
By Newton Raphson method the root is 0.740193 at the 7th iteration
```

You buy a 35,000 vehicle for nothing down at 8,500 per year for 7 years. Use the bisect function to determine the interest rate that

you are paying. Employ initial guesses for the interest rate of 0.01 and 0.3 and a stopping criterion of 0.00005. The formula relating present worth P, annual payments A, number of years n, and interest rate i is

$$A = P\frac{i(1+i)^n}{(1+i)^n - 1}$$

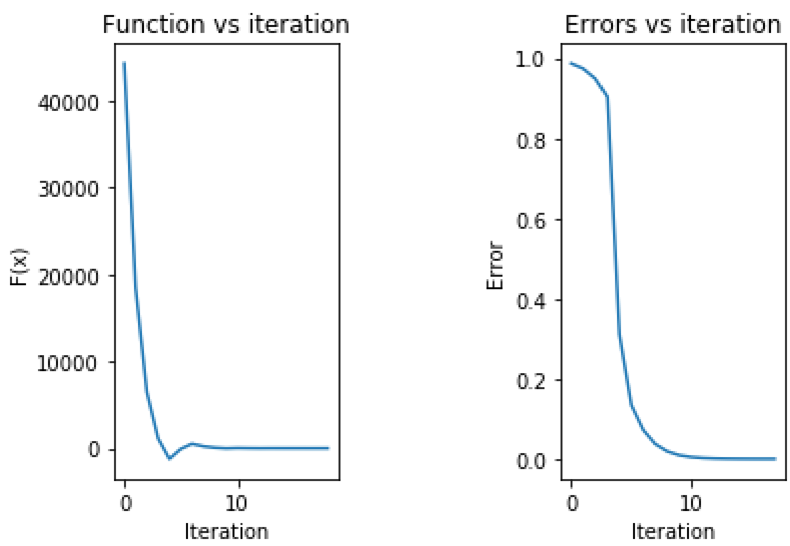
In [10]:

```
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: 3

a=0.01
b=3.0
f(a)=-3298.010098
f(b)=96506.409083


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

1               0.010000            3.000000               1.505000
44260.241811    2.990000

2               0.010000            1.505000               0.757500
18534.476105    1.495000

3               0.010000            0.757500               0.383750
6472.570400     0.747500

4               0.010000            0.383750               0.196875
1126.769275     0.373750

5               0.010000            0.196875               0.103438
-1229.247973    0.186875

6               0.103438            0.196875               0.150156
-83.437363      0.093437

7               0.150156            0.196875               0.173515
514.070296      0.046719

8               0.150156            0.173515               0.161836
213.349214      0.023359

9               0.150156            0.161836               0.155996
64.460144       0.011680

10              0.150156            0.155996               0.153076
```

```
-9.619790       0.005840

11              0.153076                0.155996                0.154536
27.389073       0.002920

12              0.153076                0.154536                0.153806
8.876851        0.001460

13              0.153076                0.153806                0.153441
-0.373419       0.000730

14              0.153441                0.153806                0.153623
4.251229        0.000365

15              0.153441                0.153623                0.153532
1.932448        0.000182

16              0.153441                0.153532                0.153486
0.779484        0.000091

17              0.153441                0.153486                0.153464
0.196690        0.000045

18              0.153441                0.153464                0.153452
-0.082032       0.000023

19              0.153452                0.153464                0.153458
0.057329        0.000012

Approximate root is 0.153458
```



Regular Falsi Method

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

In [8]:

```
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: 2

x1=0.0
x2=2.0

Approximation 1 is x3=0.406006
f(0.406006) is -2.390662
Now root lies in (0.406006, 2.0)
Next approximation will be x=0.674957

Approximation 2 is x4=0.674957
f(0.674957) is -1.67442
```

```
Now root lies in (0.674957, 2.0)
Next approximation will be x=0.839883

Approximation 3 is x5=0.839883
f(0.839883) is -1.054749
Now root lies in (0.839883, 2.0)
Next approximation will be x=0.935235

Approximation 4 is x6=0.935235
f(0.935235) is -0.617199
Now root lies in (0.935235, 2.0)
Next approximation will be x=0.988253

Approximation 5 is x7=0.988253
f(0.988253) is -0.345024
Now root lies in (0.988253, 2.0)
Next approximation will be x=1.017047

Approximation 6 is x8=1.017047
f(1.017047) is -0.187846
Now root lies in (1.017047, 2.0)
Next approximation will be x=1.032478

Approximation 7 is x9=1.032478
f(1.032478) is -0.100787
Now root lies in (1.032478, 2.0)
Next approximation will be x=1.040687

Approximation 8 is x10=1.040687
f(1.040687) is -0.053648
Now root lies in (1.040687, 2.0)
Next approximation will be x=1.045037

Approximation 9 is x11=1.045037
f(1.045037) is -0.028435
Now root lies in (1.045037, 2.0)
Next approximation will be x=1.047337

Approximation 10 is x12=1.047337
f(1.047337) is -0.015037
Now root lies in (1.047337, 2.0)
Next approximation will be x=1.048552

Approximation 11 is x13=1.048552
f(1.048552) is -0.007942
Now root lies in (1.048552, 2.0)
Next approximation will be x=1.049193

Approximation 12 is x14=1.049193
f(1.049193) is -0.004191
Now root lies in (1.049193, 2.0)
Next approximation will be x=1.049531

Approximate root is 1.049
```

In [ ]: