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. ¶

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

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

$v(t) = math.sqrt$

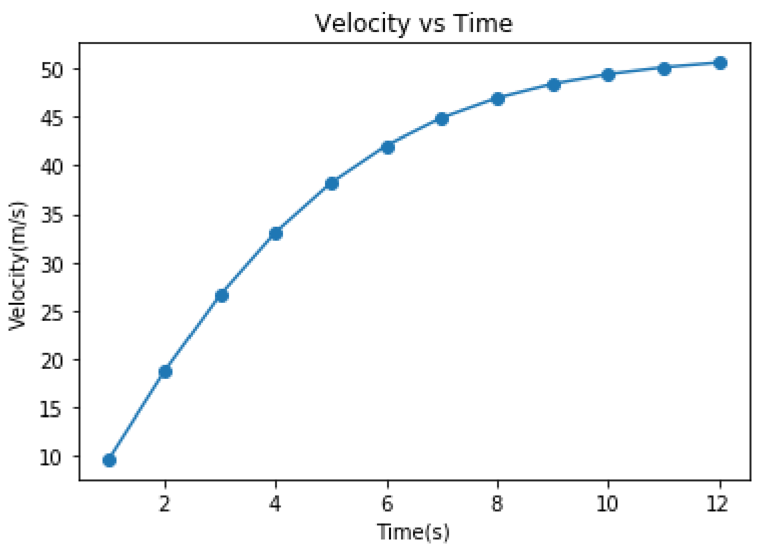
In [39]:

```
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%.

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

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

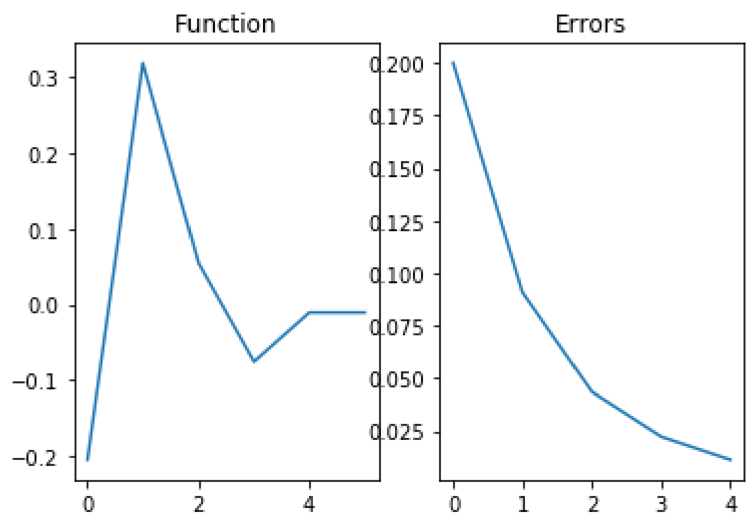
In [16]:

```
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>]
```


Function
0.3
0.2
0.1
0.0
−0.1
−0.2
0
2
4
Errors
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0
1
2
3
4

```
In [ ]:
```