Practical Manual
Paper: DSE III

Using Python

Dr Phonindra Nath Das

DEPARTMENT OF MATHEMATICS

RAMAKRISHNA MISSION VIVEKANANDA CENTENARY COLLEGE, RAHARA

Contents

1. Malthusian Model :

Let us compare the malthus model $\frac{dN}{dt} = rN$ with real data for example worlds population in every 10 years gap given by:

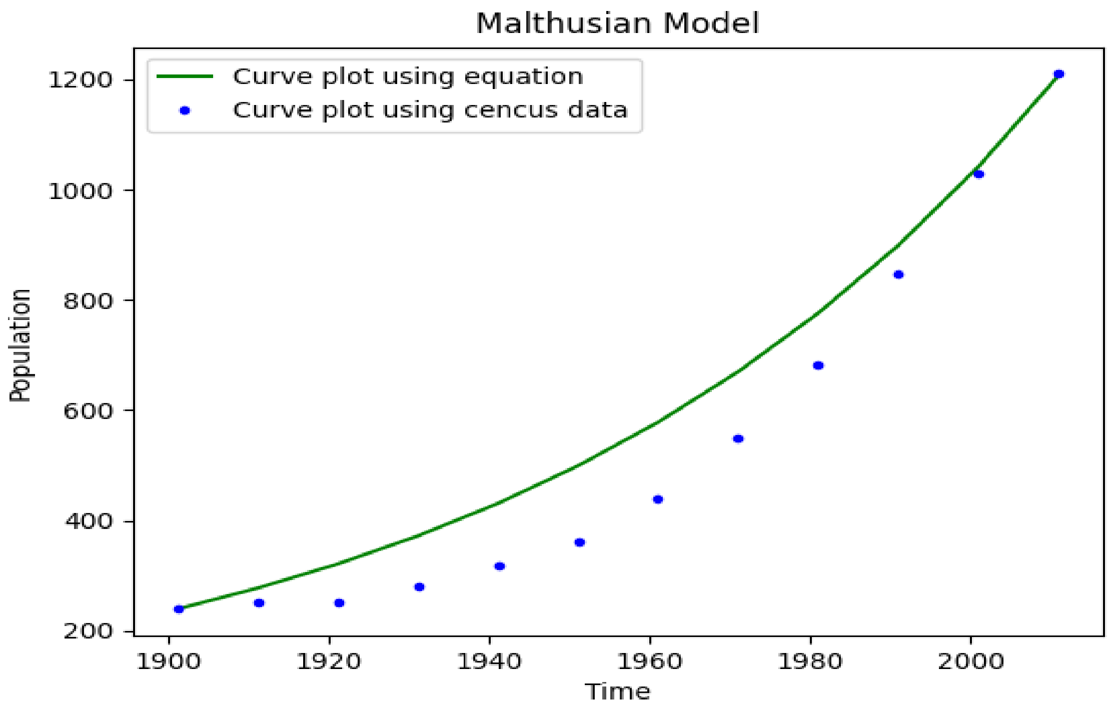
Year	1901	1911	1921	1931	1941	1951	1961	1971	1981	1991	2001	2011
Population in million	238.4	252.1	251.3	279.9	318.1	361.1	439.2	548.2	683.3	846.4	1028.7	1210.2

To do this exercise we required to import numpy, matplotlib.pyplot, math and then we just write corresponding code as follows:

Code:

```
import numpy as np
import matplotlib.pyplot as plt
import math
from math import *
population = [238.4, 252.1, 251.3, 279.9, 318.1, 361.1, 439.2, 548.2, 683.3, 846.4, 1028.7, 1210.2]
time = [1901, 1911, 1921, 1931, 1941, 1951, 1961, 1971, 1981, 1991, 2001, 2011]
def p(t):
    return  238.4*exp(0.01475*t)
mod_pop = []
for t in range (0,12):
    mod_pop.append(p(10*t))
plt.plot(time, mod_pop, 'g', label = 'Curve plot using equation')
plt.plot(time, population, '.b', label = 'Curve plot using cencus data')
plt.xlabel("Time")
plt.ylabel("Population")
plt.title('Malthusian Model')
plt.legend()
plt.show()
```

Output:



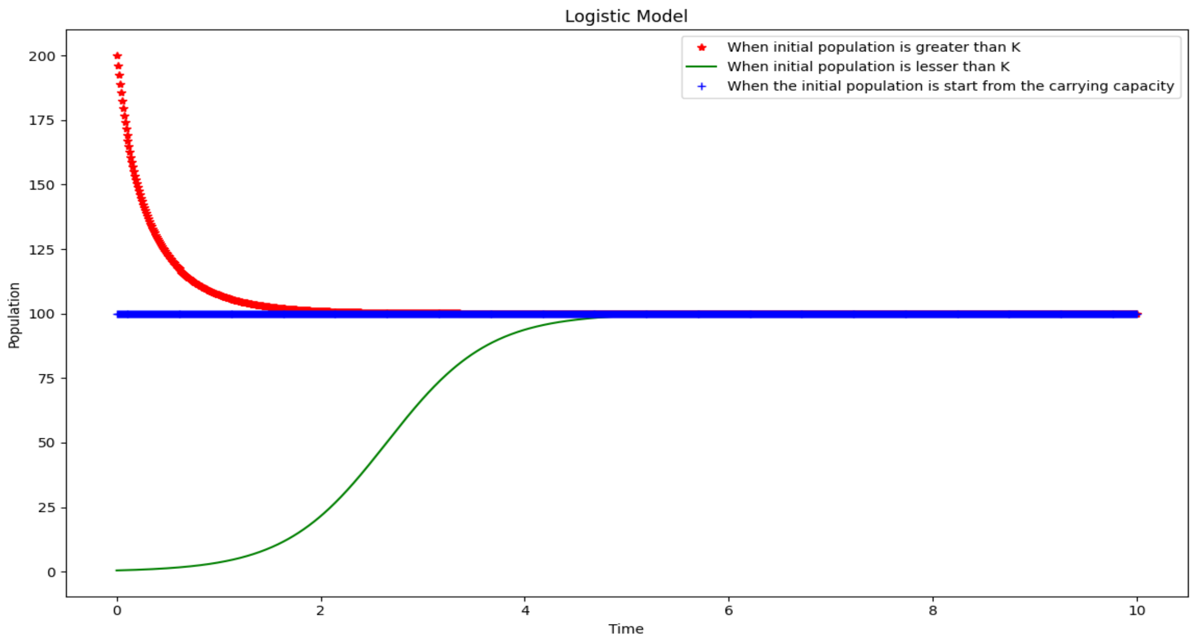
2. Logistic Model :

The logistic model is $\frac{dN}{dt} = rN(1 - \frac{N}{K})$. To solve this we need to import numpy, matplotlib.pyplot, math and then we just write corresponding code as follows:

Code:

```
import numpy
from numpy import *
import matplotlib.pyplot as plt
import math
from math import *
from scipy.integrate import odeint
k = 100
def log(x,t):
r = 2
dxdt = r*x[0]*(1 - (x[0])/k)
return [dxdt]
x0 = 200 ; x1 = 0.5
t = linspace(0,10,1000)
x = odeint(log, x0, t)
y = odeint(log, x1, t)
z = odeint(log, k, t)
plt.plot(t, x[:,0], '*r', label='When initial population is greater than K')
plt.plot(t, y[:,0], 'g', label='When initial population is lesser than K')
plt.plot(t, z[:,0], '+b', label='When the initial population is start from the carrying capacity')
plt.xlabel("Time")
plt.ylabel("Population")
plt.title('Logistic Model')
plt.legend()
plt.show()
```

Output:

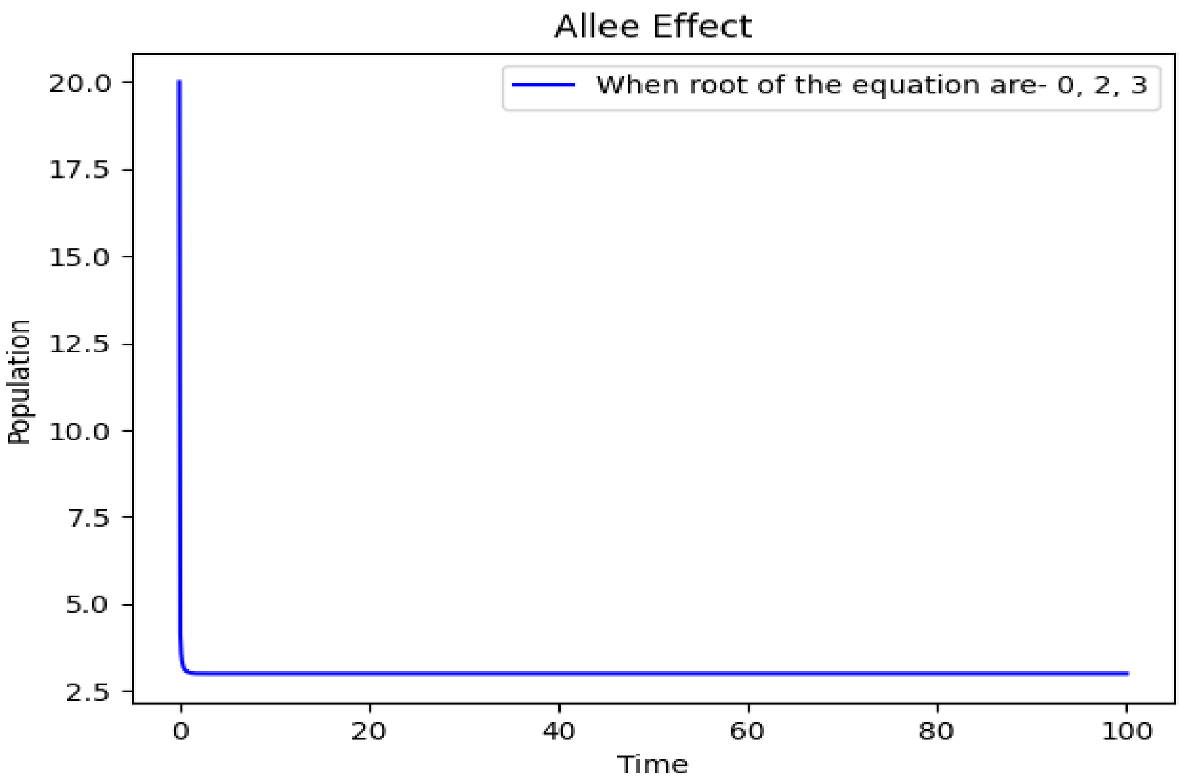


3. Allee Effect :

Code:

```
import numpy
from numpy import *
import matplotlib.pyplot as plt
import math
from math import *
from scipy.integrate import odeint
def allee(x,t):
a0 = -6
a1 = 5
a2 = -1
dxdt = x[0] * (a0 + a1*x[0] + a2*(x[0])**2)
return [dxdt]
x0 = 20
t = linspace(0,100, 1000)
x = odeint(allee, x0, t)
plt.plot(t, x[:,0], 'b', label = 'When root of the equation are- 0, 2, 3')
plt.xlabel("Time")
plt.ylabel("Population")
plt.title('Allee Effect')
plt.legend()
plt.show()
```

Output:

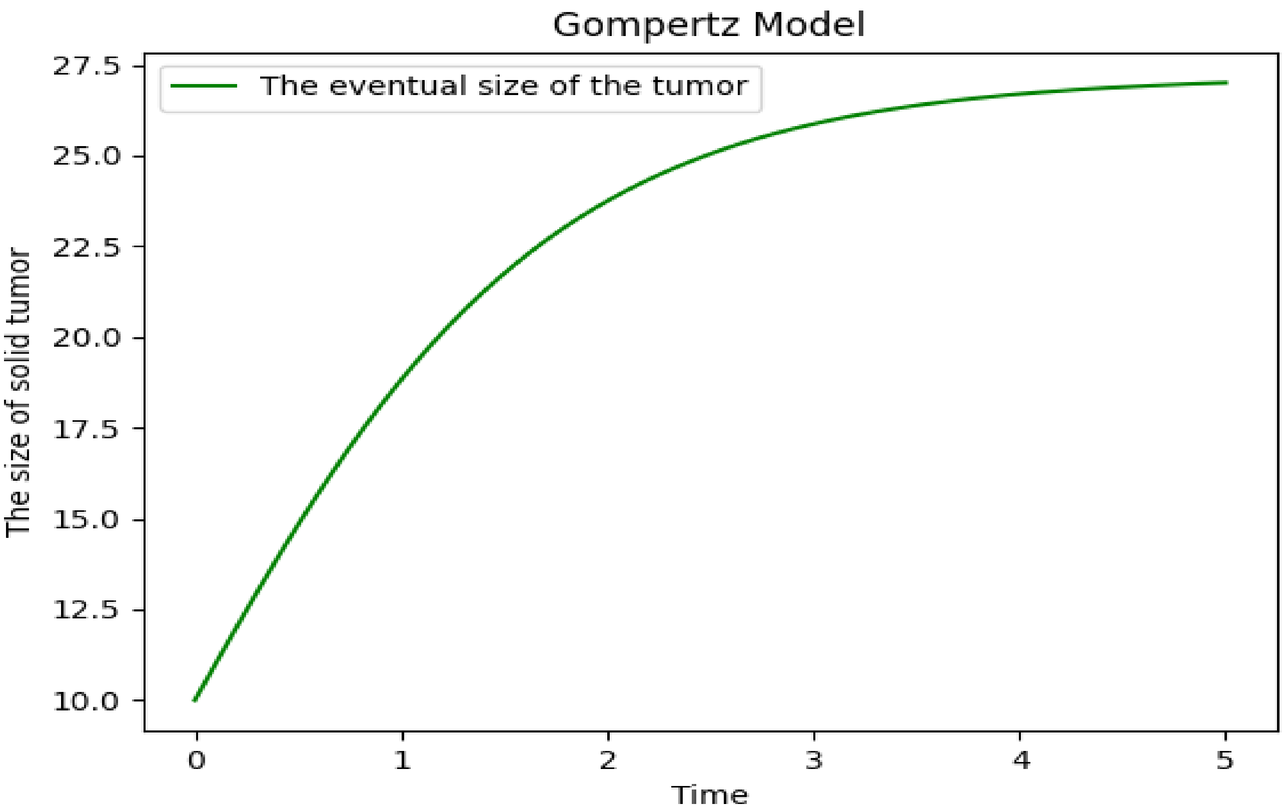


4. Gompertz Model:

Code:

```
import numpy
from numpy import *
import matplotlib.pyplot as plt
import math
from math import *
from scipy.integrate import odeint
t = linspace(0,5, 1000)
def gom(x,t):
r = 1
theta = 1
dxdt = x[0] * r * exp(- theta*t)
return [dxdt]
x0 = 10
x = odeint(gom, x0, t)
plt.plot(t, x[:,0], 'g', label = 'The eventual size of the tumor')
plt.xlabel("Time")
plt.ylabel("The size of solid tumor")
plt.title('Gompertz Model')
plt.legend()
plt.show()
```

Output:

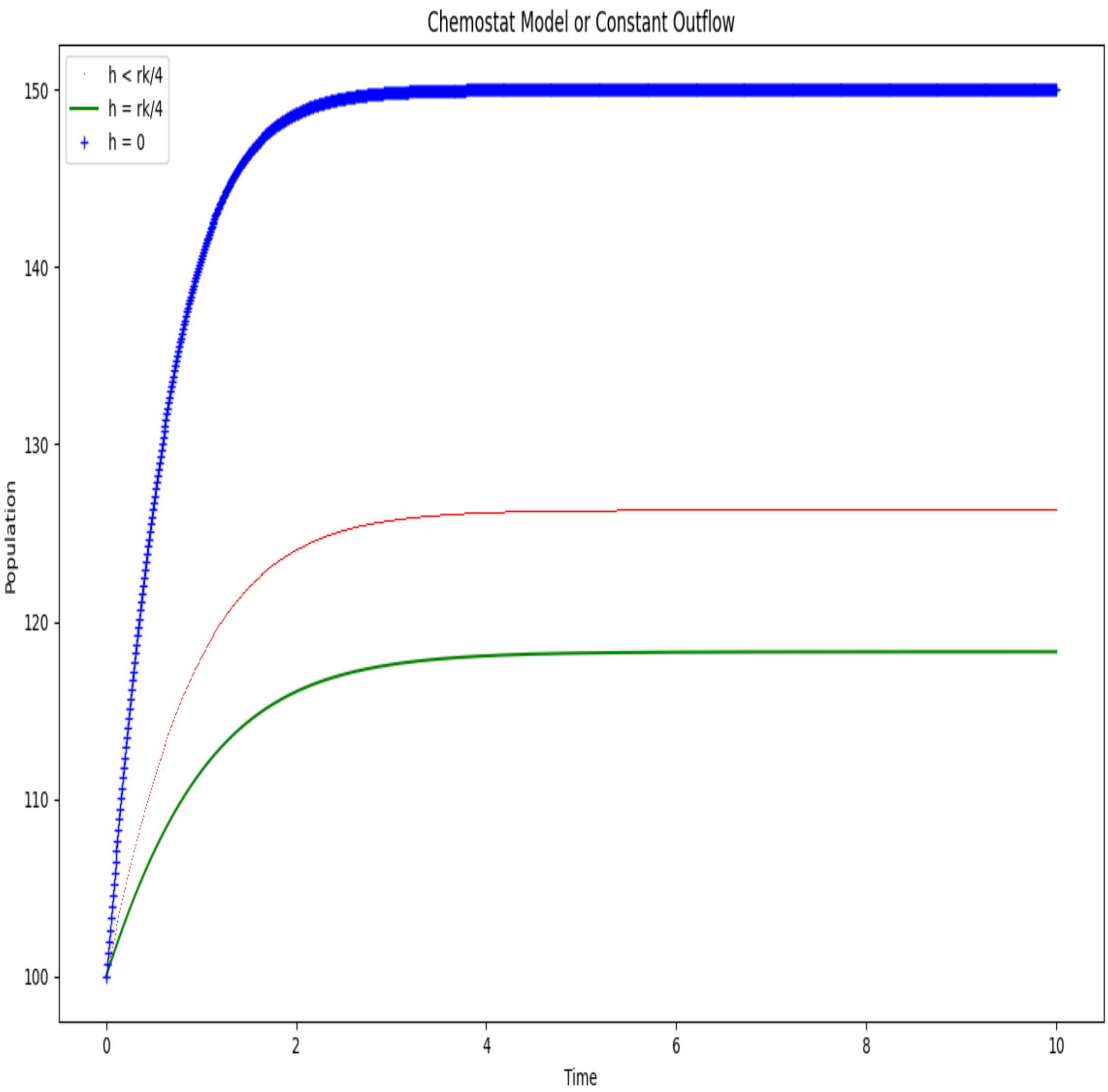


5. Chemostat Model With Constant Outflow :

Code:

```
import numpy
from numpy import *
import matplotlib.pyplot as plt
import math
from math import *
from scipy.integrate import odeint
k = 150
h = 40
h1 = 50
h2 = 0
r = 2
def chemo(x,t):
    dxdt = r*x[0]*(1 - (x[0])/k) - h
    return  [dxdt]
def chemo1(x,t):
    dxdt = r*x[0]*(1 - (x[0])/k) - h1
    return  [dxdt]
def chemo2(x,t):
    dxdt = r*x[0]*(1 - (x[0])/k) - h2
    return  [dxdt]
x0 = 100
t = linspace(0,10,1000)
x = odeint(chemo, x0, t)
x1 = odeint(chemo1, x0, t)
x2 = odeint(chemo2, x0, t)
plt.plot(t, x[:,0], ',r', label='h < rk/4')
plt.plot(t, x1[:,0], 'g', label='h = rk/4')
plt.plot(t, x2[:,0], '+b', label='h = 0')
plt.xlabel("Time")
plt.ylabel("Population")
plt.title('Chemostat Model or Constant Outflow')
plt.legend()
plt.show()
```

<u>Output:</u>



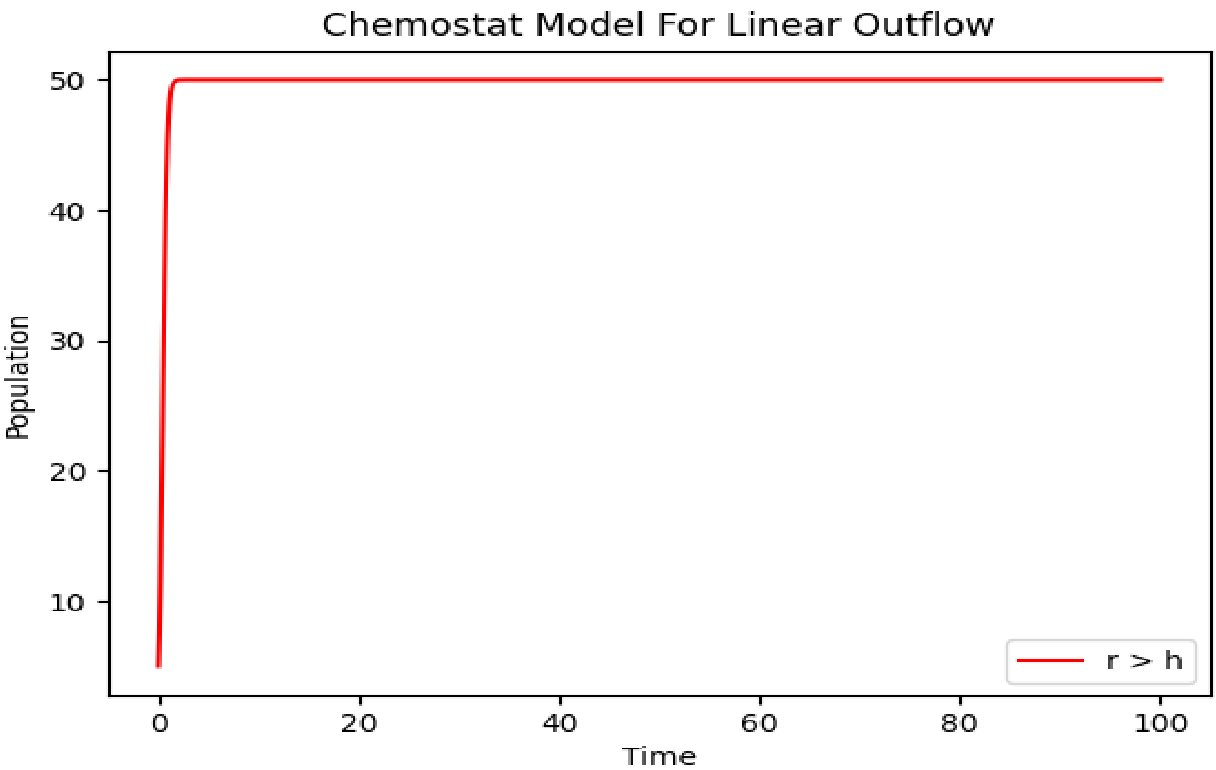
Chemostat Model or Constant Outflow
h < rk/4
h = rk/4
h = 0
150
140
130
120
110
100
Population
0
2
4
6
8
10
Time

6. Chemostat Model With Linear Outflow :

Code:

```
import numpy
from numpy import *
import matplotlib.pyplot as plt
import math
from math import *
from scipy.integrate import odeint
k = 100
h = 5
r = 10
def chemo(x,t):
    dxdt = (r*x[0]*(1 - (x[0])/k) - h*x[0])
    return  [dxdt]
x0 = 5
t = linspace(0,100,1000)
x = odeint(chemo, x0, t)
plt.plot(t, x[:,0], 'r', label='r > h')
plt.xlabel("Time")
plt.ylabel("Population")
plt.title('Chemostat Model For Linear Outflow')
plt.legend()
plt.show()
```

Output:

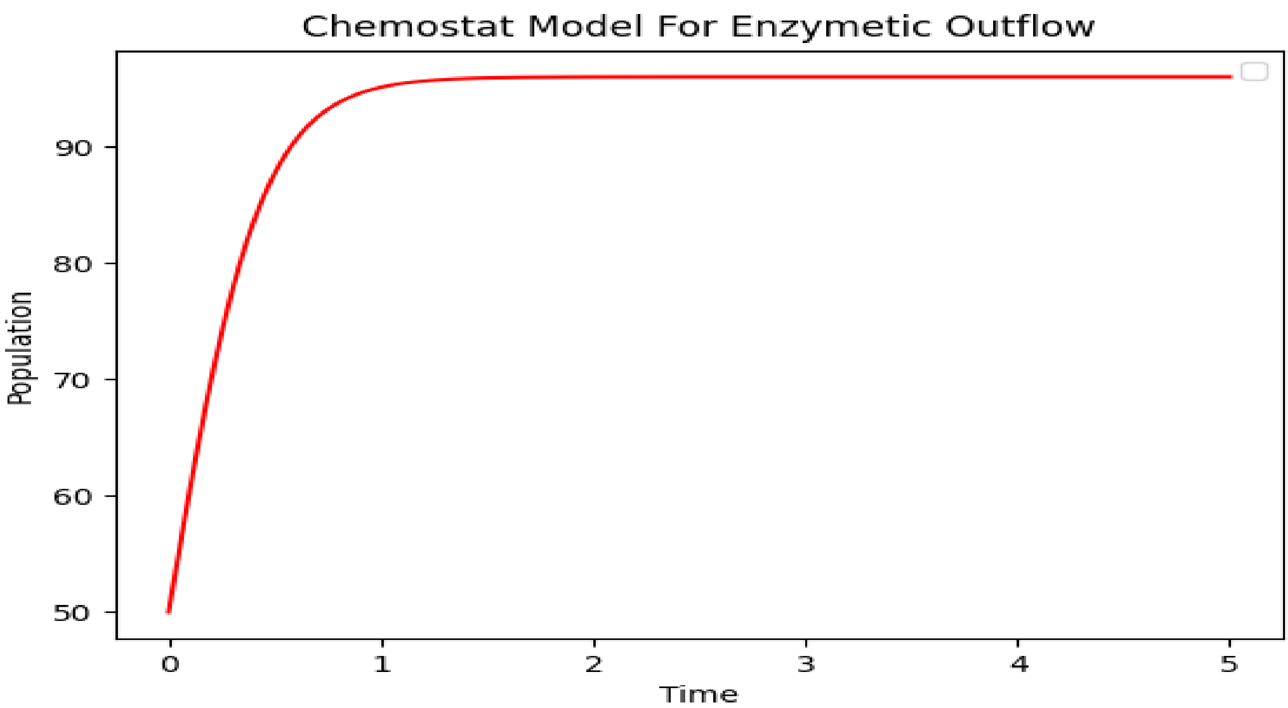


7. Chemostat Model With Enzymetic Outflow :

Code:

```
import numpy
from numpy import *
import matplotlib.pyplot as plt
import math
from math import *
from scipy.integrate import odeint
k = 100
h = 20
r = 5
c = 5
def chemo(x,t):
    dxdt = (r*x[0]*(1 - (x[0])/k) - (h*x[0]/(c + x[0])))
    return  [dxdt]
x0 = 50
t = linspace(0,5,1000)
x = odeint(chemo, x0, t)
plt.plot(t, x[:,0], 'r', label='')
plt.xlabel("Time")
plt.ylabel("Population")
plt.title('Chemostat Model For Enzymetic Outflow')
plt.legend()
plt.show()
```

Output:

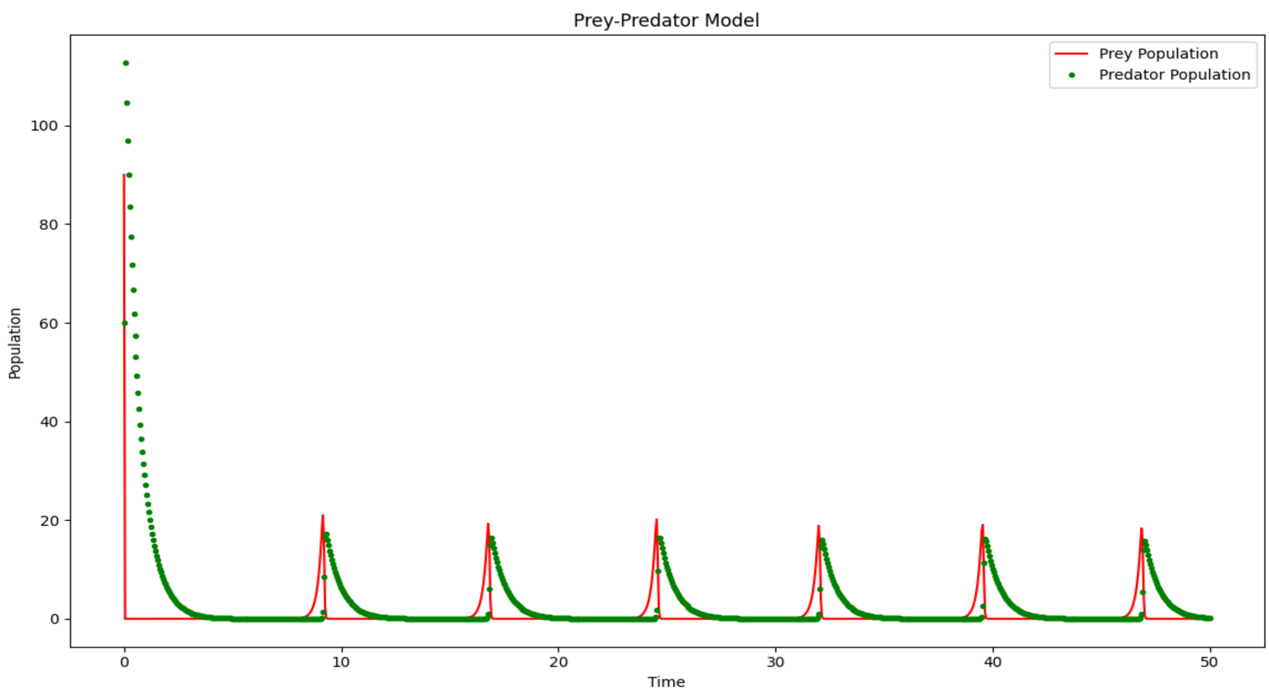


8. Lotka Voltera Model :

Code:

```
import numpy
from numpy import *
import matplotlib.pyplot as plt
import math
from math import *
from scipy.integrate import odeint
a = 5; b = 3; c = 2; d = 1.5
def lv(x,t):
   dxdt = a*x[0] - b*x[0]*x[1]
   dydt = c*x[0]*x[1] - d*x[1]
   return  [dxdt, dydt]
x0 = [90, 60]
t = linspace(0,50,1000)
x = odeint(lv, x0, t)
plt.plot(t, x[:,0], 'r', label='Prey Population')
plt.plot(t, x[:,1], '.g', label='Predator Population')
plt.xlabel("Time")
plt.ylabel("Population")
plt.title('Prey-Predator Model')
plt.legend()
plt.show()
```

Output:

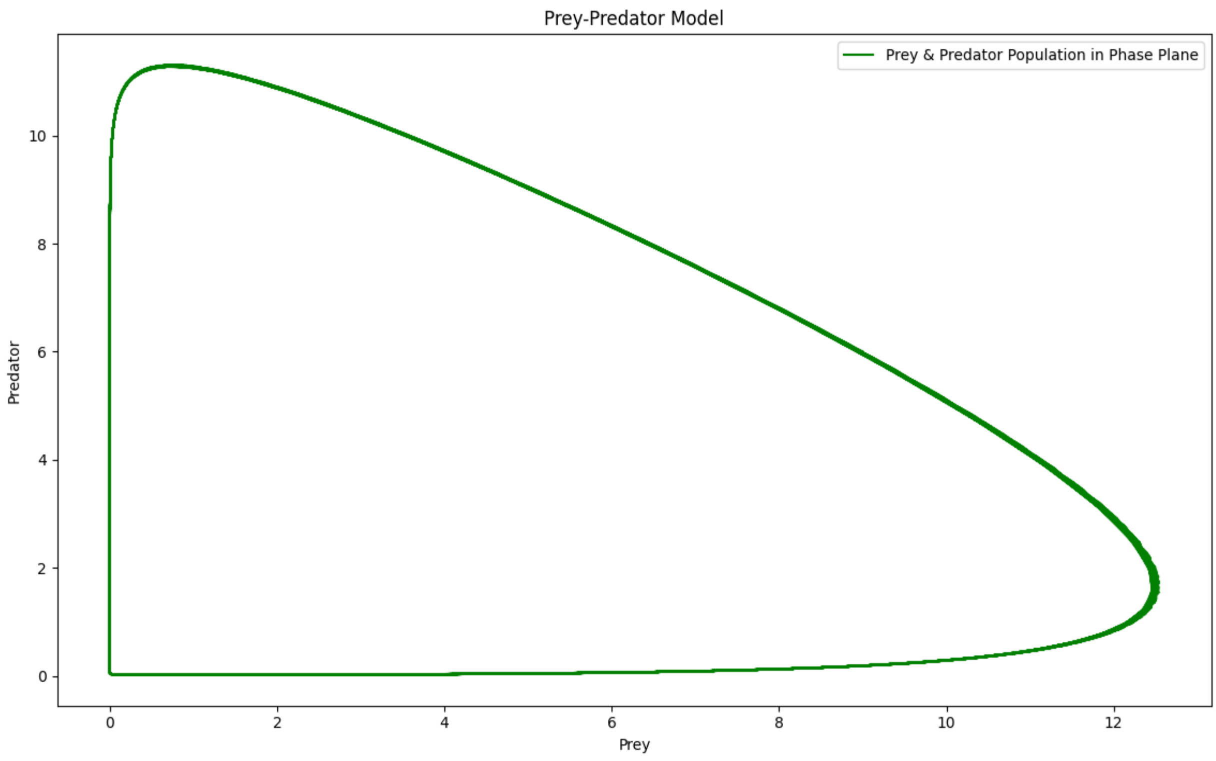


9. Lotka Voltera Model With Phase Plane :

Code:

```
import numpy
from numpy import *
import matplotlib.pyplot as plt
import math
from math import *
from scipy.integrate import odeint
a = 5; b = 3; c = 2; d = 1.5
def lv(x,t):
   dxdt = a*x[0] - b*x[0]*x[1]
   dydt = c*x[0]*x[1] - d*x[1]
   return  [dxdt, dydt]
x0 = [9, 6]
t = linspace(0,200,10000)
x = odeint(lv, x0, t)
plt.plot(x[:,0], x[:,1], 'g', label='Prey & Predator Population in Phase Plane')
plt.xlabel("Prey")
plt.ylabel("Predator")
plt.title('Prey-Predator Model')
plt.legend()
plt.show()
```

Output:

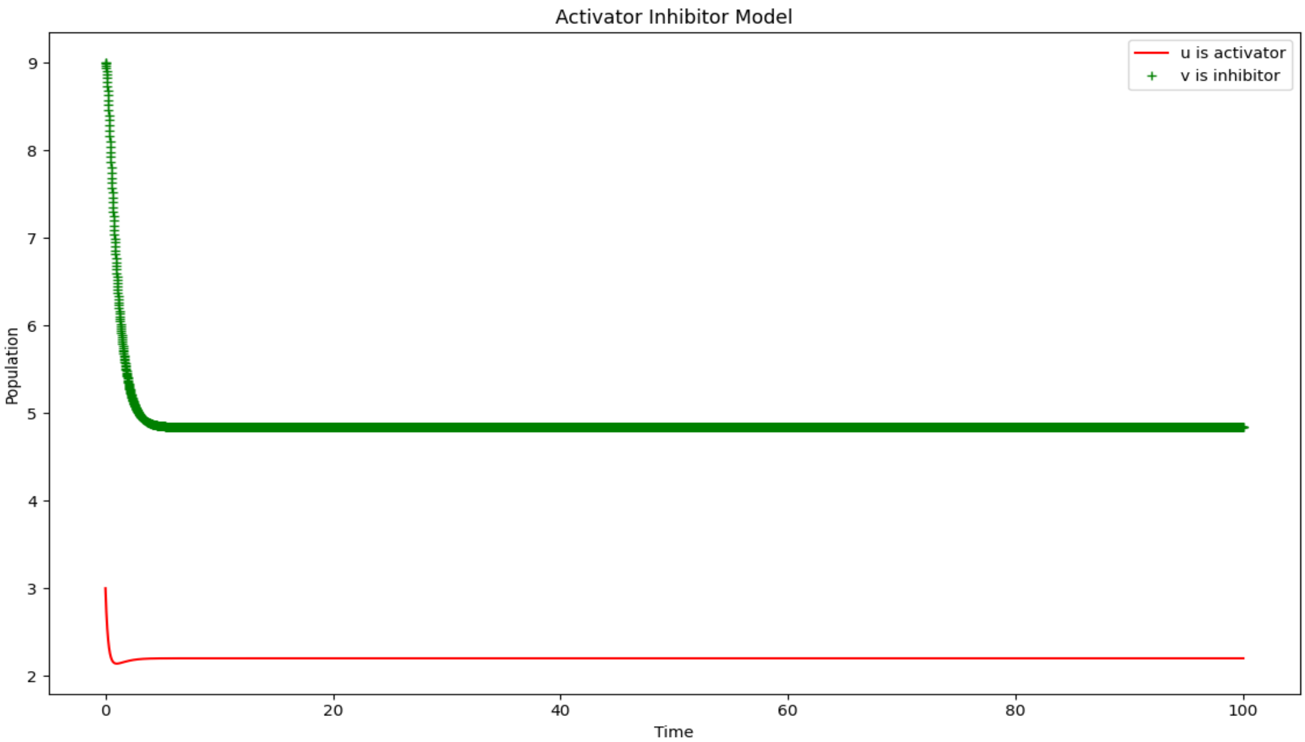


10. Activator Inhibitor Model :

Code:

```
import numpy
from numpy import *
import matplotlib.pyplot as plt
import math
from math import *
from scipy.integrate import odeint
a = 10; b = 5
def ai(u,t):
    dudt = a - b*u[0] + u[0]**2/u[1]
    dvdt = u[0]**2 - u[1]
    return  [dudt, dvdt]
u0 = [3, 9]
t = linspace(0,100,5000)
u = odeint(ai, u0, t)
plt.plot(t, u[:,0], 'r', label='u is activator')
plt.plot(t, u[:,1], '+g', label='v is inhibitor')
plt.xlabel("Time")
plt.ylabel("Population")
plt.title('Activator Inhibitor Model')
plt.legend()
plt.show()
```

Output:

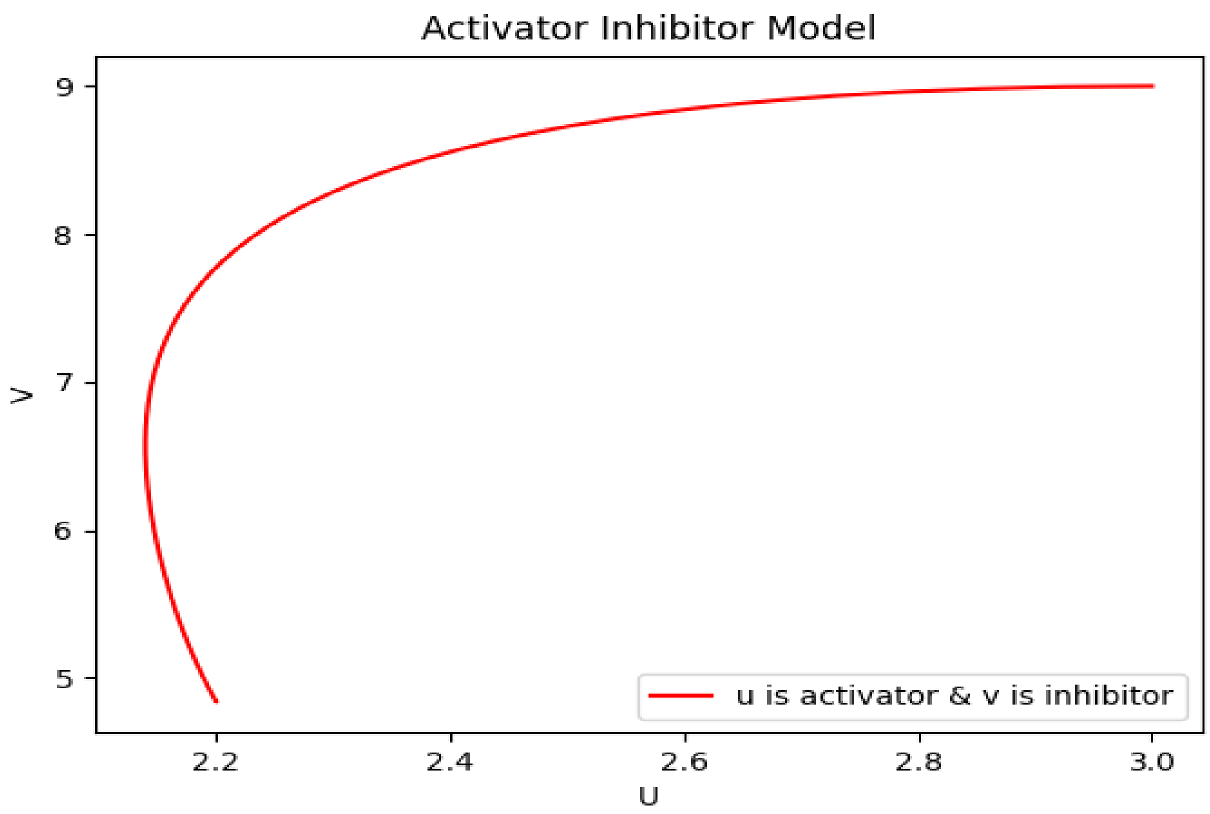


11. Activator Inhibitor Model With Phase Plane :

Code:

```
import numpy
from numpy import *
import matplotlib.pyplot as plt
import math
from math import *
from scipy.integrate import odeint
a = 10; b = 5
def ai(u,t):
    dudt = a - b*u[0] + u[0]**2/u[1]
    dvdt = u[0]**2 - u[1]
    return  [dudt, dvdt]
u0 = [3, 9]
t = linspace(0,100,5000)
u = odeint(ai, u0, t)
plt.plot(u[:,0], u[:,1], 'r', label='u is activator & v is inhibitor')
plt.xlabel("U")
plt.ylabel("V")
plt.title('Activator Inhibitor Model')
plt.legend()
plt.show()
```

Output:

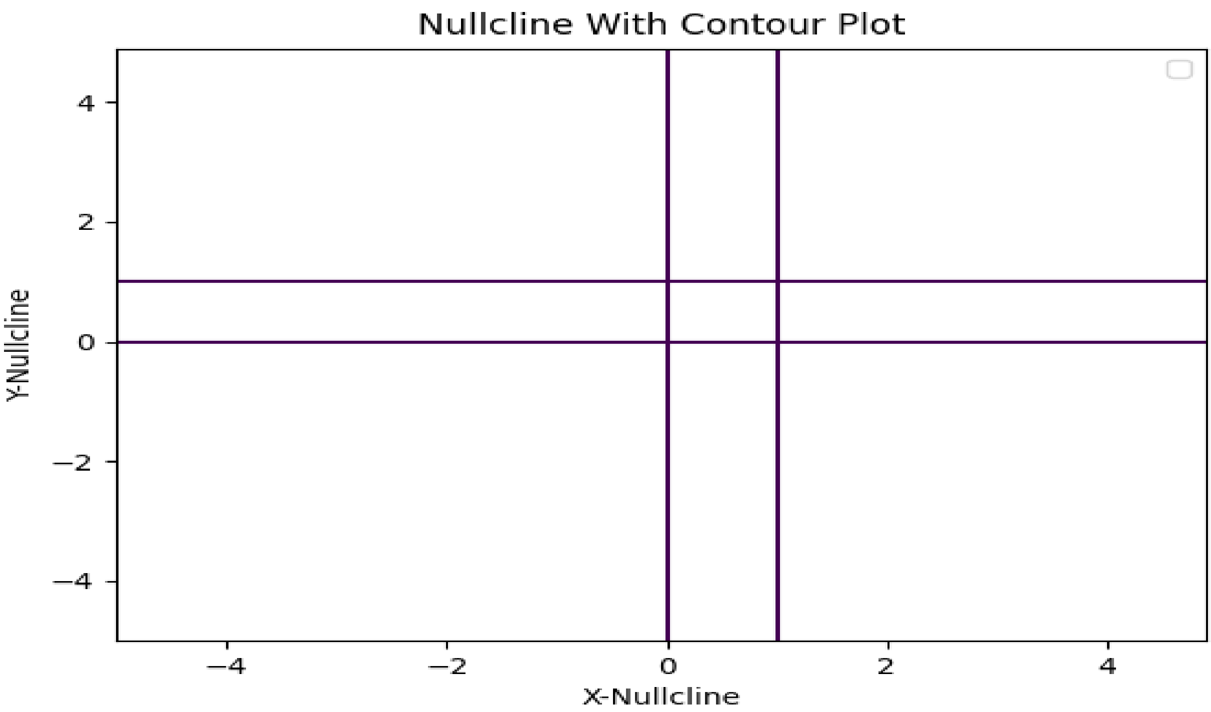


12. Nullcline :

Code:

```
import numpy
from numpy import *
import matplotlib.pyplot as plt
import math
from math import *
import array
from array import *
r = 1
m = 2
def f(x,y):
    return r*x*(1-y)
def g(x,y):
    return m*y*(1-x)
x = arange(-5, 5, 0.1)
y = arange(-5, 5, 0.1)
X,Y = meshgrid(x,y)
plt.contour(x, y, f(X,Y), [0])
plt.contour(x, y, g(X,Y), [0])
plt.xlabel("X-Nullcline")
plt.ylabel("Y-Nullcline")
plt.title('Nullcline With Contour Plot')
plt.legend()
plt.show()
```

Output:

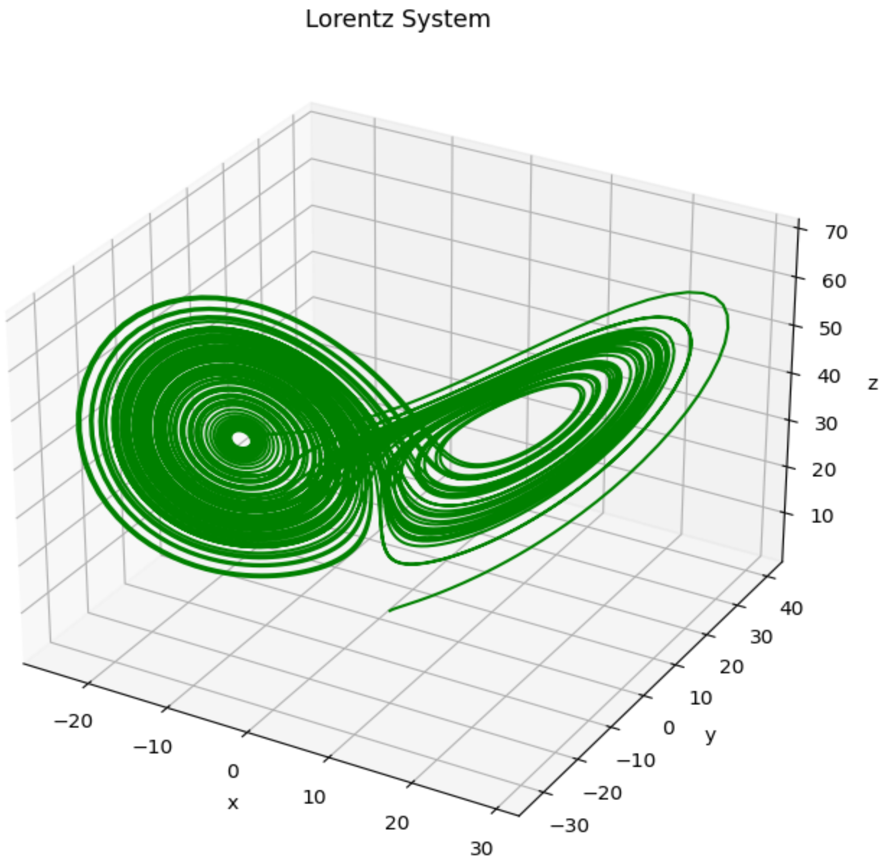


13. Lorentz System 3D :

Code:

```
import numpy
from numpy import *
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import math
from math import *
from scipy.integrate import odeint
a = 15; b = 18/5 ; r = 40
def ls(x,t):
    dxdt = a*(x[1] - x[0])
    dydt = r*x[0] - x[0]*x[2] - x[1]
    dzdt = x[0]*x[1] - b*x[2]
    return  [dxdt, dydt, dzdt]
x0 = [0.9, 1.1, 0.7]
t = linspace(0,50,1000)
x = odeint(ls, x0, t)
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.plot(x[:,0], x[:,1], x[:,2], 'g')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
ax.set_title('Lorentz System')
plt.show()
```

Output:

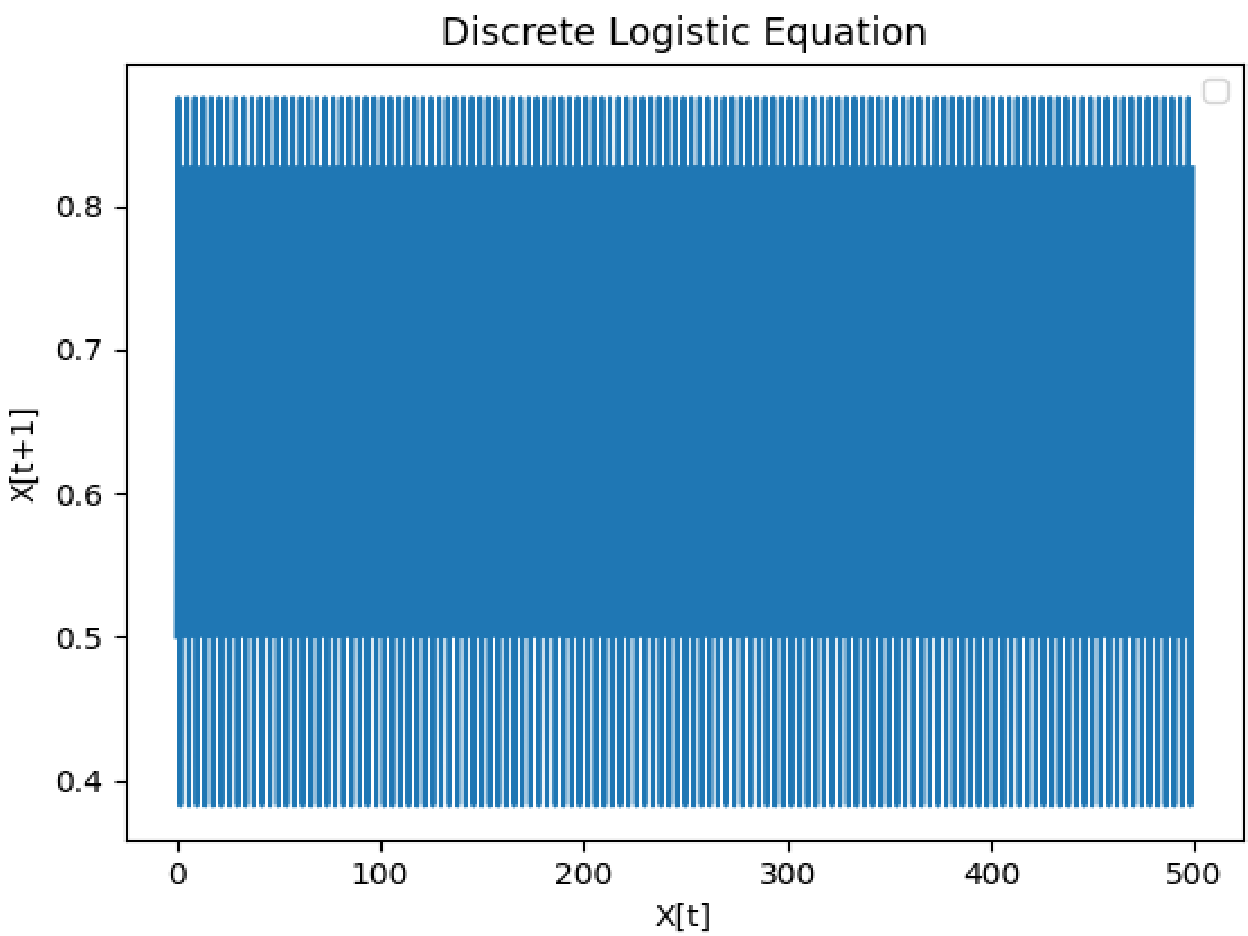


14. Discrete Logistic Model :

Code:

```
import numpy
from numpy import *
import matplotlib.pyplot as plt
import math
from math import *
r = 3.5
T = 500
x = 0.5 + zeros(T)
for t in range (T-1):
  x[t+1] = r*x[t]*(1 - x[t])
plt.plot(x)
plt.xlabel("X[t]")
plt.ylabel("X[t+1]")
plt.title('Discrete Logistic Equation')
plt.legend()
plt.show()
```

Output:



15. Discrete Logistic Chaos :

Code:

```
import numpy
from numpy import *
import matplotlib.pyplot as plt
import math
from math import *
R = linspace(1,4,5000)
T = 500
x = 0.5 + zeros(T)
xend = arange(round(T*0.9),T)
for i in range (len(R)):
    for t in range (T-1):
        x[t+1] = R[i]*x[t]*(1 - x[t])
    y = unique(x[xend])
    r = R[i]*ones(len(y))
    plt.plot(r,y,'k.', markersize=0.5)
plt.xlabel("X[t]")
plt.ylabel("X[t+1]")
plt.title('Discrete Logistic Chaos')
plt.legend()
plt.show()
```

Output: