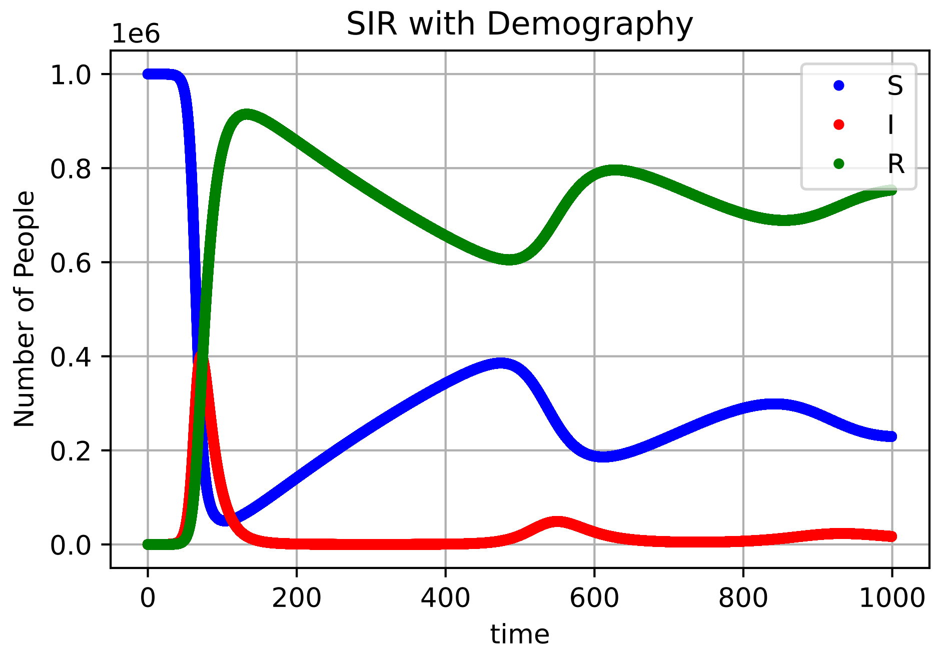
testing SIR Code

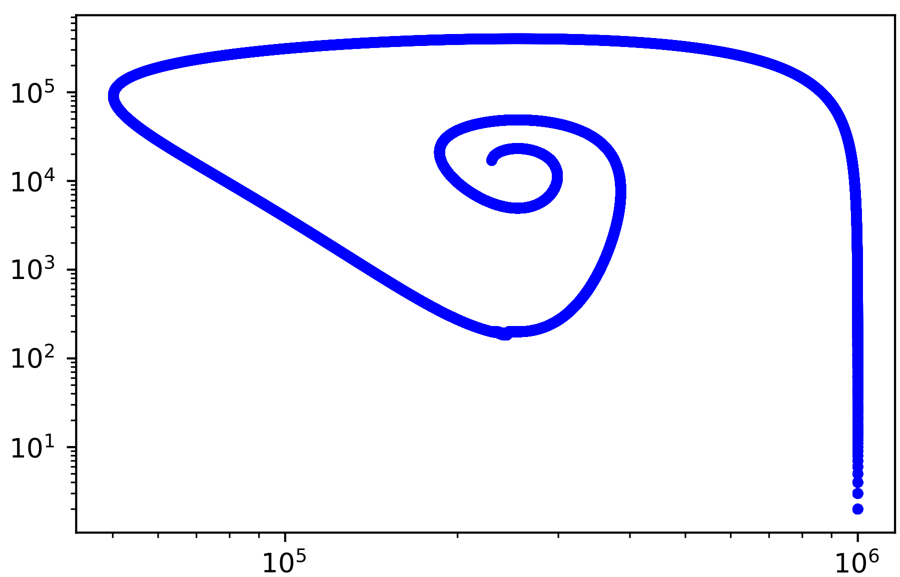
This code should execure the SIR example bellow: # # ———————————————— # | mWane$R$ / # v / # $mBirth$N —— mBeta$I$S/N —— mGamma$I$ —– # —> / S / —> / I / —> / R / # —– —– —– # / $mDeath$S | mDeath$I$ / $mDeath$R # V V V # V V

```
import numpy as np
import matplotlib.pyplot as plt
from JSF_Solver import JumpSwithFlowSimulator

# np.random.seed(3)

# These define the rates of the system
mBeta = 2/7  # Infect "___" people a week
mGamma = 0.5/7  # infecion for "___" weeks
mDeath = 1/(2*365)  # lifespan
mBirth = mDeath
mWane = 0/(2.0*365)

R_0 = mBeta/(mGamma+mDeath)

# These are the initial conditions
N0 = 10**6
I0 = 2
R0 = 0
S0 = N0-I0-R0

# How long to simulate for
tFinal = 1000

# These are solver options
dt = 10**-2
SwitchingThreshold = np.array([0.2, 200])
```

```
# kinetic rate parameters
X0 = np.array([S0, I0, R0])

# reactant stoichiometries
nuReactant = np.array([[1, 1, 0],
                       [0, 1, 0],
                       [1, 0, 0],
                       [0, 1, 0],
                       [0, 0, 1],
                       [1, 0, 0],
                       [0, 1, 0],
                       [0, 0, 1],
                       [0, 0, 1]])

# product stoichiometries
nuProduct = np.array([[0, 2, 0],
                      [0, 0, 1],
                      [2, 0, 0],
                      [1, 1, 0],
                      [1, 0, 1],
                      [0, 0, 0],
                      [0, 0, 0],
                      [0, 0, 0],
                      [1, 0, 0]])

# stoichiometric matrix
nu = nuProduct - nuReactant

# propensity function
k = np.array([[mBeta, mGamma, mBirth, mBirth, mBirth, mDeath, mDeath, mDeath, mWane]]).T

def rates(X, t):
    return k * np.array([[(X[0]*X[1])/(X[0]+X[1]+X[2]),
                          X[1],
                          X[0],
                          X[1],
                          X[2],
                          X[0],
                          X[1],
                          X[2],
                          X[2]]]).T
```

```python
# identify which reactions are discrete and which are continuous
# make sure that the shape of DoDisc is (3,1)
DoDisc = np.array([[0, 0, 0]]).T

# allow S and I to switch, but force R to be continuous
EnforceDo = np.array([[0, 0, 0]]).T

stoich = {'nu': nu, 'DoDisc': DoDisc, 'nuReactant': nuReactant, 'nuProduct': nuProduct}

solTimes = np.arange(0, tFinal+dt, dt)
myOpts = {'EnforceDo': EnforceDo, 'dt': dt, 'SwitchingThreshold': SwitchingThreshold}

X, TauArr = JumpSwithFlowSimulator(X0, rates, stoich, solTimes, myOpts)

plt.plot(TauArr, X[0], label='S', marker='.', linestyle='', color='blue')
plt.plot(TauArr, X[1], label='I', marker='.', linestyle='', color='red')
plt.plot(TauArr, X[2], label='R', marker='.', linestyle='', color='green')

plt.xlabel('time')
plt.ylabel('Number of People')
plt.title('SIR with Demography')
plt.grid(True)

plt.legend()  # Display legend
plt.show()

plt.yscale("log")
plt.xscale("log")
plt.plot(X[0],X[1], label='S', marker='.', linestyle='', color='blue')
plt.show()
```


SIR with Demography
1e6
1.0
0.8
0.6
0.4
0.2
0.0
Number of People
0
200
400
600
800
1000
time
S
I
R


10^5
10^4
10^3
10^2
10^1
10^5
10^6