Tutorial 4

Part 1- Write down the differential equation describing the system of chemical equations (assuming a volume of 1)

The system of chemical reactions is described by equation 1.

$$\begin{gathered} \emptyset \rightarrow X \; at \; a \; rate \; k_1 = 1 \\ X \rightarrow Y \; at \; a \; rate \; k_2 = 2 \\ 2X + Y \rightarrow 3X \; at \; a \; rate \; k_3 = 0.02 \\ X \rightarrow \emptyset \; at \; a \; rate \; k_4 = 0.04 \end{gathered}$$

Equation 1: System of chemical reactions

The species on the left-hand side of the arrow are the reactants whereas the species on the right-hand side of the arrow are the products. The variables, X and Y, represent the concentration of molecules, of the respective species, at any given time. Thus, the concentration of molecules varies over time as the reactions occur. Hence, a deterministic system of differential equations can be derived for expressing the change in concentration of molecules over time, these are commonly referred to as the rate equations and are shown by equation 2.

$$\frac{dX}{dt} = k_1 - k_2X + k_3YX^2 - k_4X$$

$$\frac{dY}{dt} = k_2X - k_3YX^2$$

Equation 2: Rate equations

With reference to equation 1, the first reaction has no reactants involved and species X is produced. The concentration of molecules increases at a rate k_1; this is represented in the rate equation of X by the first term. The second reaction shows the molecules of species X, reacting at a rate k_2, to produce molecules of species Y. This is denoted by the second term in the rate equation of X and the first term in the rate equation of Y. This shows the concentration of X and Y, due to this reaction, are respectively increasing and decreasing by the same amount. The amount they change depends on the concentration of X. The third reaction requires two molecules of X and one molecule of Y to produce three molecules of X. This is represented by the third term in the rate equation of X and the second term in the rate equation of Y. This reaction causes X to increase and Y to decrease. The fourth reaction was represented in the rate equation of X by the fourth term.

Part 2- Use a package to solve the differential equation for 500 time units starting from X(0)=Y(0)=0

Solutions to the rate equations were found using the *odeint* solver of the Python library- *Scipy*. Solutions were computed at 1000 discrete time steps over the 500 second interval. Figure 1 shows the time domain solutions for the concentration of molecules of species X and Y. Figure 2 shows the concentration of molecules plotted in the phase plane.

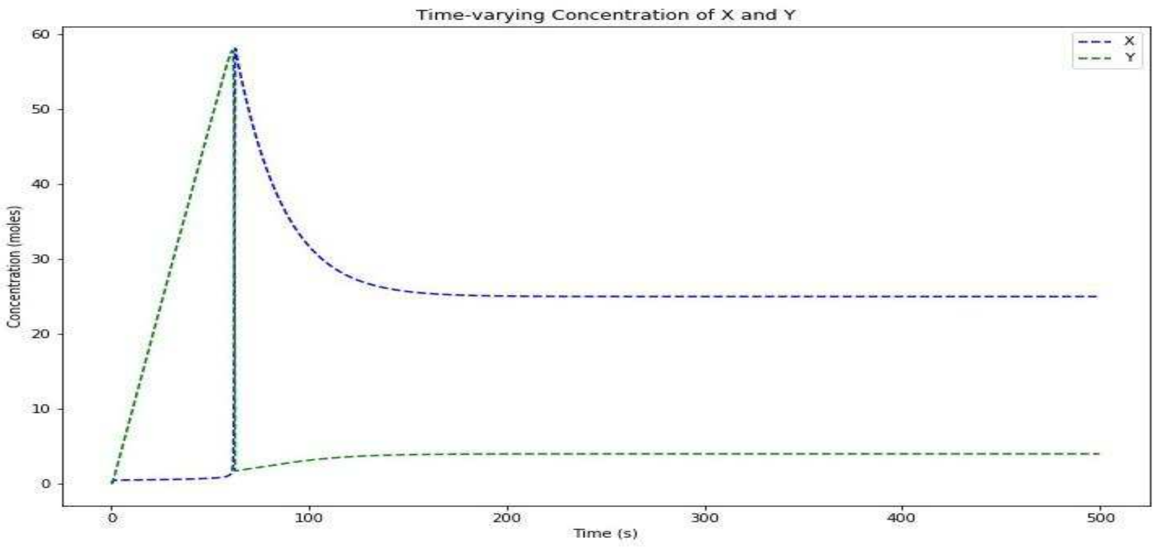


Figure 1: Time-varying Concentration of Species X and Y

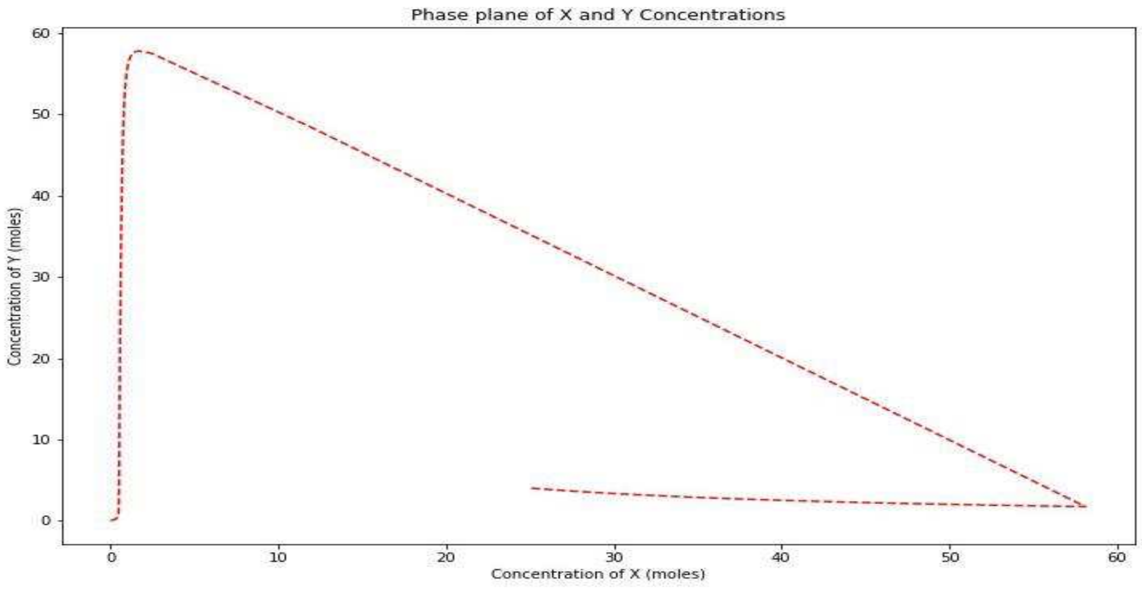


Figure 2: Phase Plane of the Concentration of Species X and Y

As shown by figure 1, the concentration of X was approximately constant, followed by a rapid increase and then exponential decay towards the 25 moles steady state. The concentration of Y increased linearly, rapidly dropped, then tended towards its steady state of 4 moles. Figure 2 shows when the concentration of X reached 0.5 moles, it acted like a switch and caused the concentration of Y to increase. Once the concentration of Y reached 58 moles, it acted like a switch to initiate the increase in the concentration of X and decrease in its own concentration. When the concentration of X was 58 moles and the concentration of Y was 2 moles, the states started to stabilise. This suggests the species regulate each other.

Task 3- Write a Gillespie algorithm to simulate the same four chemical equation and plot the results for 500 time units

The Gillespie algorithm is a stochastic formulation for the temporal modelling of dynamic systems. Where the deterministic rate equations describe the average change to concentrations over time, the stochastic Gillespie algorithm simulates exactly how the concentrations change over time. The algorithm is based on the principle that one event occurs during each discrete time step. The length of time between events is randomly generated using an exponential distribution. The event that occurs is randomly sampled from the discrete probability distribution of the events. In this case, the events are the reactions.

The algorithm recursively computes the probability of each reaction occurring (these are dependent on the concentrations at that time); then it generates the time step and selects a reaction to occur by sampling from the updated probability distribution of the reactions; finally, the concentrations are updated based on the reaction that occurred [1]. Figure 3 shows the time varying concentrations of X and Y computed using the Gillespie algorithm. Figure 4 shows the concentrations of X and Y in the phase plane.

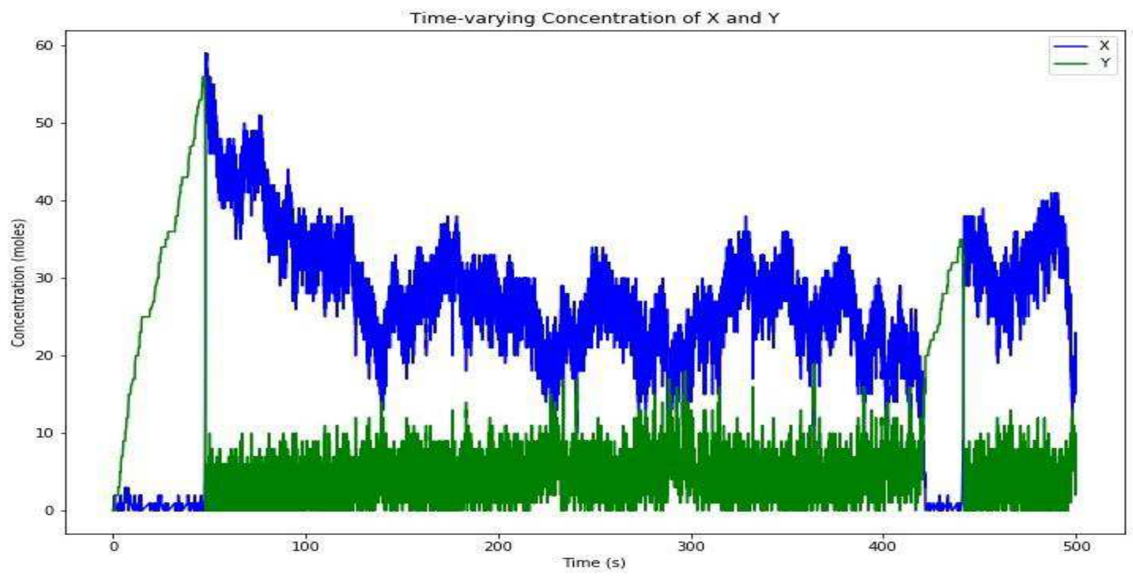


Figure 3: Time-varying Concentration of Species X and Y Computed Using the Gillespie Algorithm

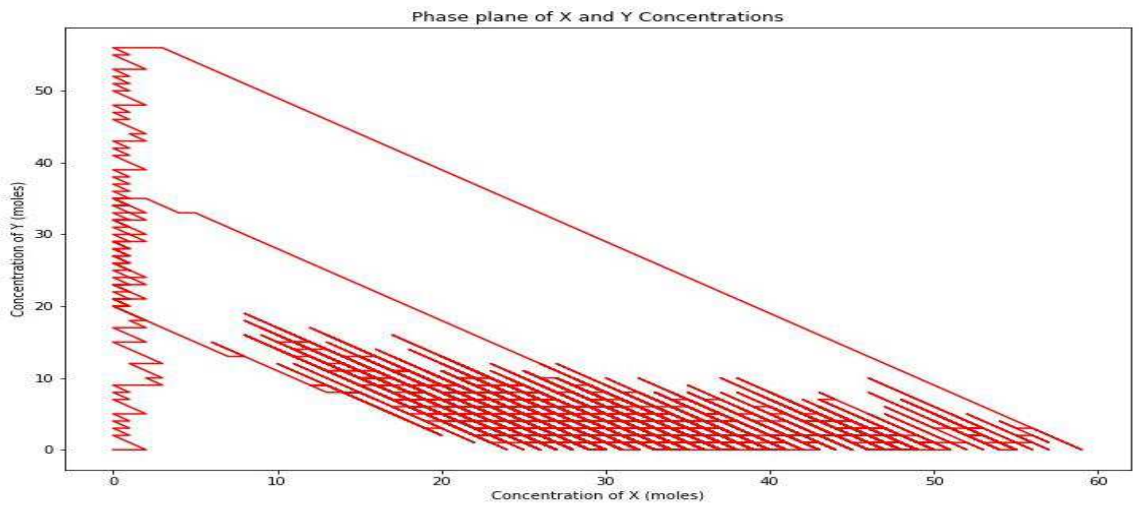


Figure 4: Phase Plane of the Concentration of Species X and Y Computed Using the Gillespie Algorithm

Figures 5 and 6 show the comparison of the solutions in the time domain and phase plane computed by the deterministic and stochastic methods. They show the deterministic solution of the rate equations described the mean concentration of the stochastic solutions. This is illustrated by the deterministic solutions looking like the stochastic solutions with their high frequency noise removed.

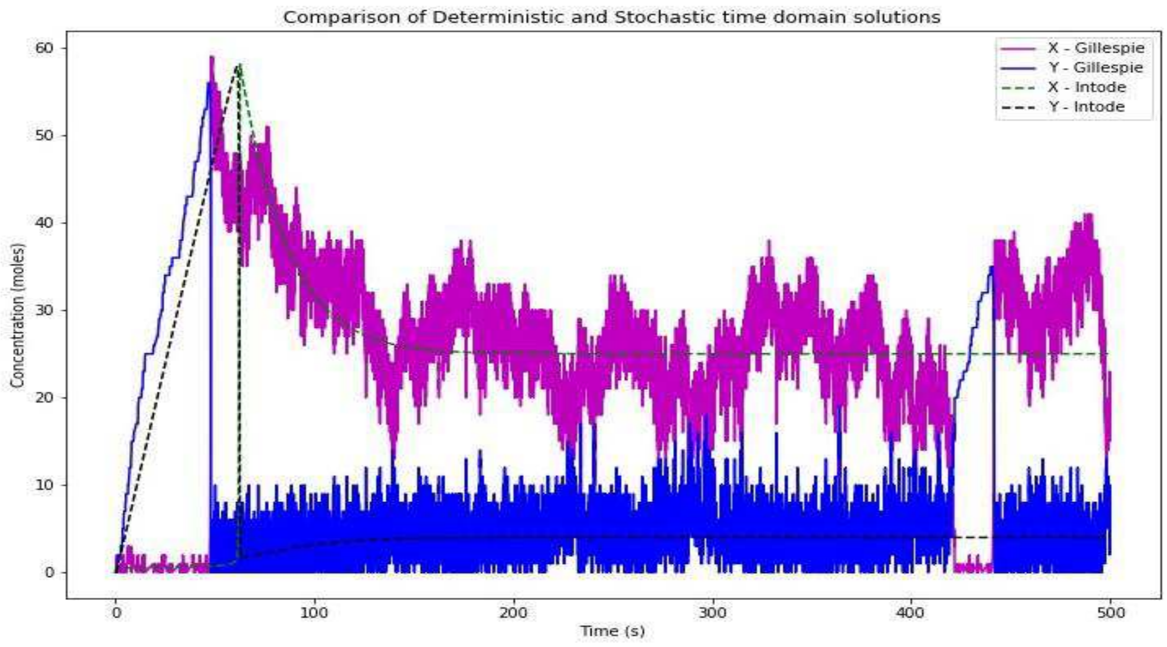


Figure 5: Deterministic and Stochastic Time Domain Solutions

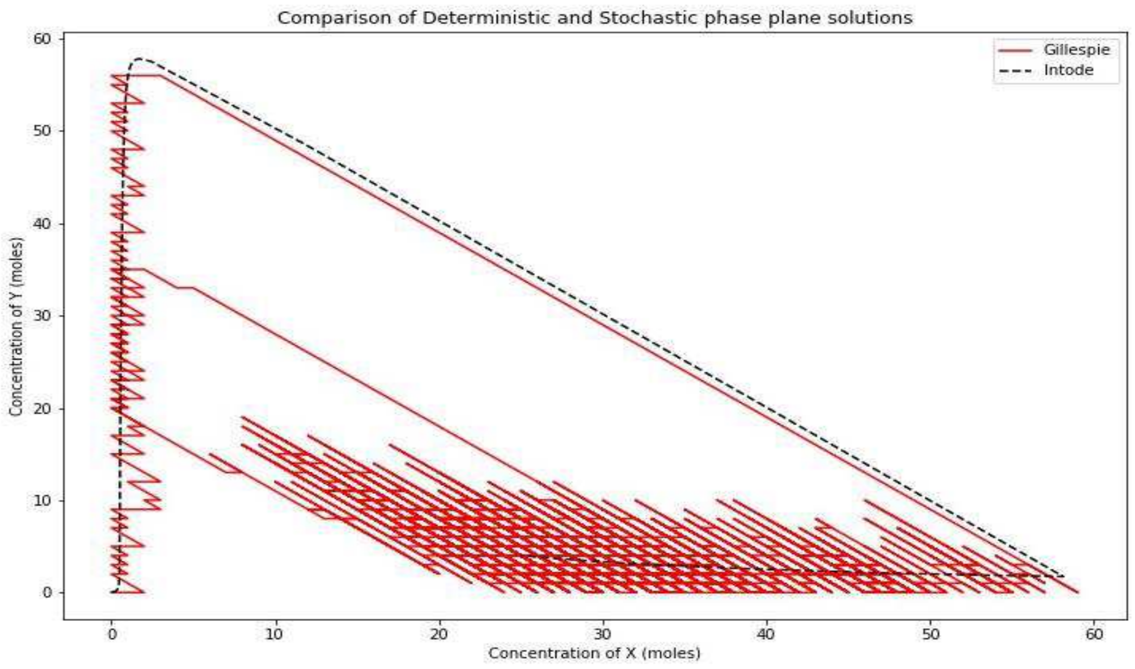


Figure 6: Deterministic and Stochastic Phase Plane Solutions

References

[1] K. Sasaki. "Understand the Gillespie algorithm and build it yourself in Python." https://github.com/karinsasaki/gillespie-algorithm-python/blob/master/.ipynb_checkpoints/build_your_own_gillespie_solutions-checkpoint.ipynb?fbclid=IwAR0sJTE3uYvjpEvioT15kcBvZW6MRUT6hxv9OAVFQ1nVZBYHj8BF3xTd4UI (accessed.

Appendix- Code

```python
# To support both python 2 and python 3
from __future__ import division, print_function, unicode_literals
%matplotlib inline

import numpy as np # numerical computation packages in python
import matplotlib.pyplot as plt # plotting routines
from scipy.integrate import odeint # solving the system of differential equations
from scipy.special import comb #number of combinations of N things taken k at a time
from scipy.special import factorial
```

```python
#1. Write down the differential equation describing the system of chemical equations (assuming a volume of 1)
def gene_reg(y, t):

    #reaction rates
    k1 = 1
    k2 = 2
    k3 = 0.02
    k4 = 0.04

    X, Y = y #unpacking state vector

    dX = k1 - k2*X + k3*Y*(X**2) - k4*X
    dY = k2*X - k3*Y*(X**2)
    dydt = [dX, dY]
    return dydt
```

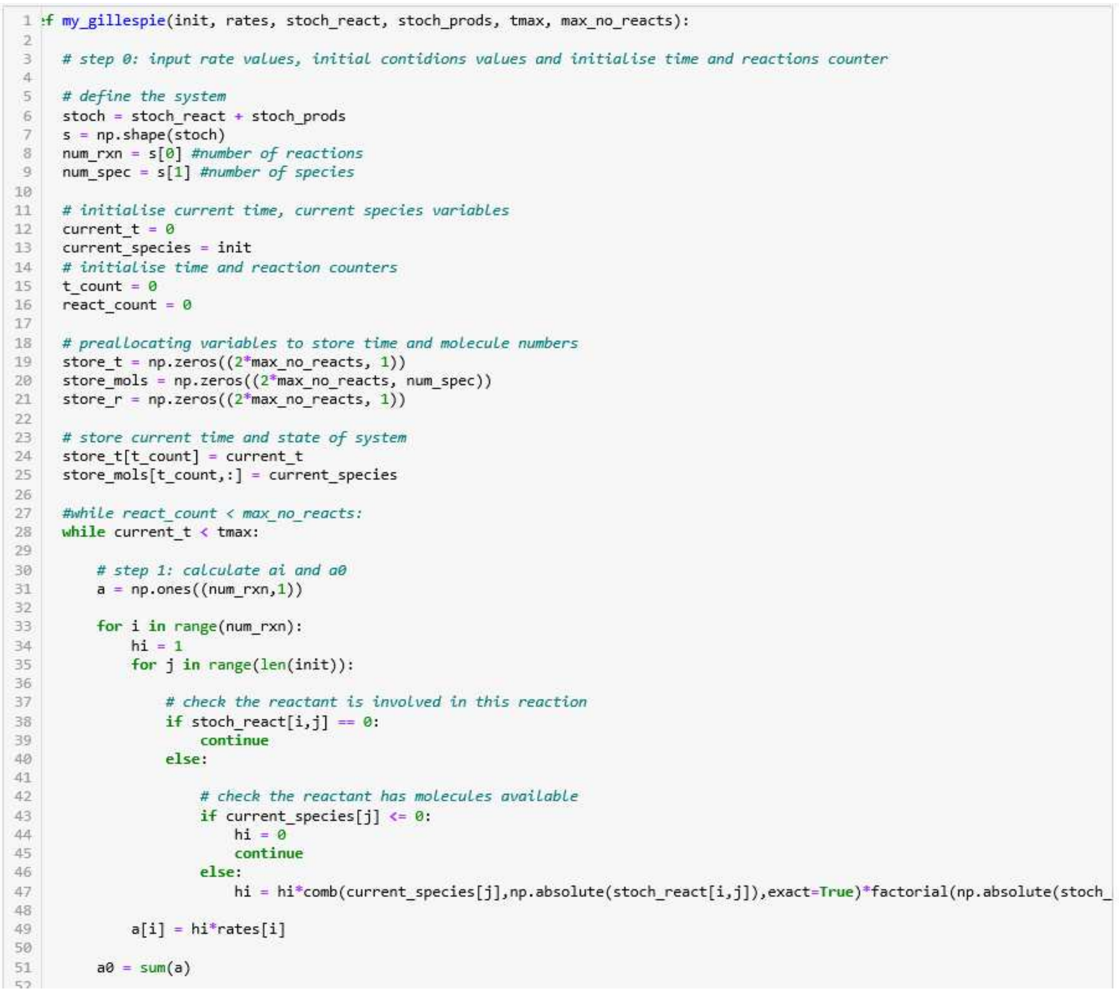
```python
#2. Use a package to solve the differential equation for 500 time units starting from X(0)=Y(0)=0
y0 = [0, 0] #initial conditions y0=[X Y]
t = np.linspace(0, 500, 1000) #time span
sol = odeint(gene_reg, y0, t) #solver
```

```python
#concentrations in the time domain
plt.figure(figsize=(12,8))
plt.plot(t, sol[:, 0], '--', color='b', label='X')
plt.plot(t, sol[:, 1], '--', color='g', label='Y')
plt.legend(loc='best')
plt.xlabel('Time (s)')
plt.ylabel('Concentration (moles)')
plt.title('Time-varying Concentration of X and Y')
plt.savefig('D_PvT.jpg')
plt.show()
```

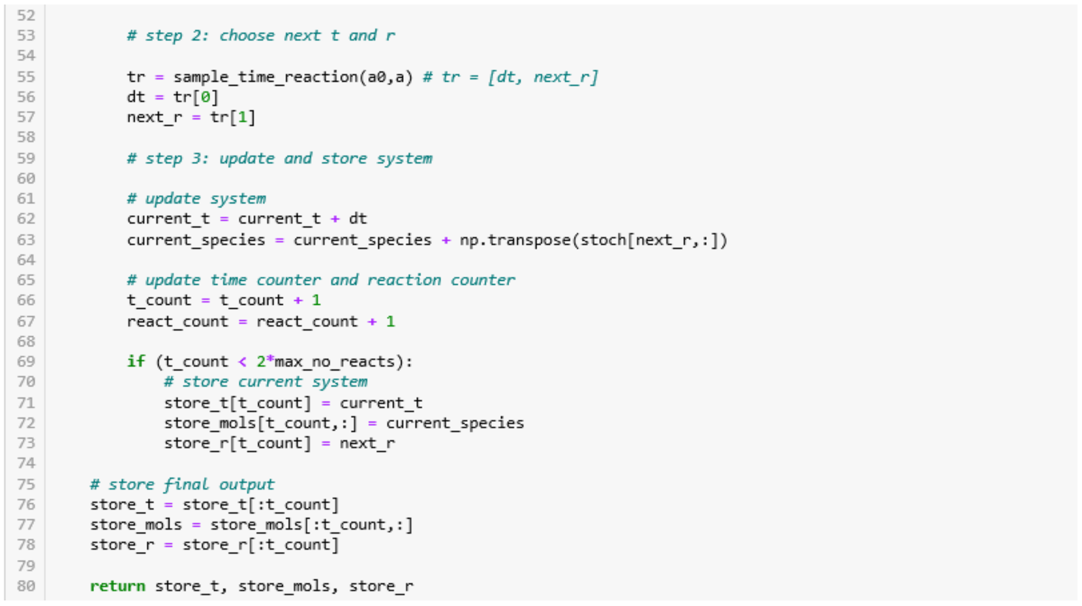
```python
#phase plane plot of the concentrations
plt.figure(figsize=(12,8))
plt.plot(sol[:, 0], sol[:, 1], '--', color='r')
plt.xlabel('Concentration of X (moles)')
plt.ylabel('Concentration of Y (moles)')
plt.title('Phase plane of X and Y Concentrations')
plt.savefig('D_PP.jpg')
plt.show()
```

```python
#3. Write a Gillespie algorithm to simulate the same four chemical equation and plot the results for 500 time units
```

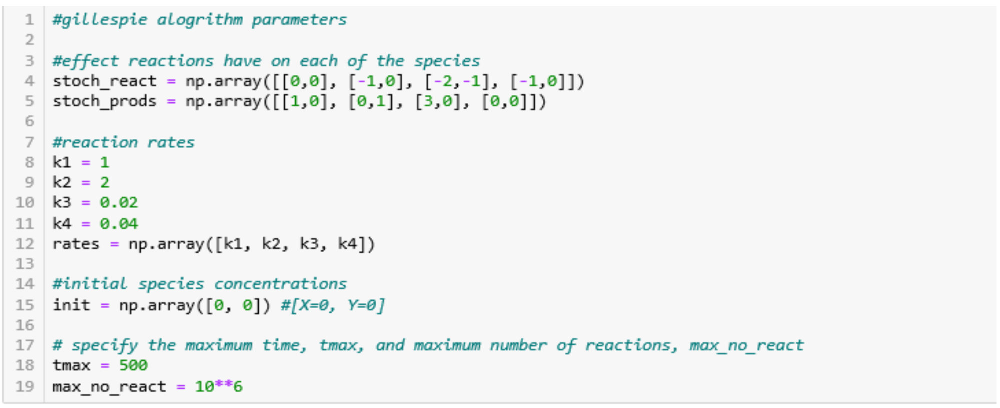
```python
#define a function to choose the next time and reaction
def sample_time_reaction(a0,a):

    #a = set of reactions ki*hi where hi= total number of all possible combinations of molecules in reaction i
    #a0 = sum of reactions in a

    #generate random numbers to be used in next time and next reaction calculations
    rand = np.random.random(2)
    rand1 = rand[0]
    rand2 = rand[1]

    # choose next time increment dt
    dt = (1/a0)*np.log(1/rand1)

    # choose next reaction, mu
    num = 0
    N = rand2*a0 - a[num] #once N=<0- have found the next rection

    while N > 0:
        num = num + 1
        N = N - a[num]

    next_react = num

    return dt, next_react
```



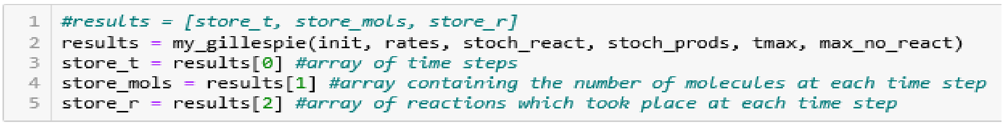
```
:f my_gillespie(init, rates, stoch_react, stoch_prods, tmax, max_no_reacts):

    # step 0: input rate values, initial contidions values and initialise time and reactions counter

    # define the system
    stoch = stoch_react + stoch_prods
    s = np.shape(stoch)
    num_rxn = s[0] #number of reactions
    num_spec = s[1] #number of species

    # initialise current time, current species variables
    current_t = 0
    current_species = init
    # initialise time and reaction counters
    t_count = 0
    react_count = 0

    # preallocating variables to store time and molecule numbers
    store_t = np.zeros((2*max_no_reacts, 1))
    store_mols = np.zeros((2*max_no_reacts, num_spec))
    store_r = np.zeros((2*max_no_reacts, 1))

    # store current time and state of system
    store_t[t_count] = current_t
    store_mols[t_count,:] = current_species

    #while react_count < max_no_reacts:
    while current_t < tmax:

        # step 1: calculate ai and a0
        a = np.ones((num_rxn,1))

        for i in range(num_rxn):
            hi = 1
            for j in range(len(init)):

                # check the reactant is involved in this reaction
                if stoch_react[i,j] == 0:
                    continue
                else:

                    # check the reactant has molecules available
                    if current_species[j] <= 0:
                        hi = 0
                        continue
                    else:
                        hi = hi*comb(current_species[j],np.absolute(stoch_react[i,j]),exact=True)*factorial(np.absolute(stoch_

            a[i] = hi*rates[i]

        a0 = sum(a)
```



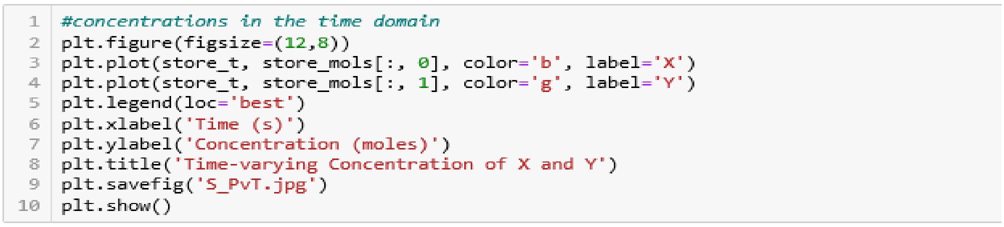
```

        # step 2: choose next t and r

        tr = sample_time_reaction(a0,a) # tr = [dt, next_r]
        dt = tr[0]
        next_r = tr[1]

        # step 3: update and store system

        # update system
        current_t = current_t + dt
        current_species = current_species + np.transpose(stoch[next_r,:])

        # update time counter and reaction counter
        t_count = t_count + 1
        react_count = react_count + 1

        if (t_count < 2*max_no_reacts):
            # store current system
            store_t[t_count] = current_t
            store_mols[t_count,:] = current_species
            store_r[t_count] = next_r

    # store final output
    store_t = store_t[:t_count]
    store_mols = store_mols[:t_count,:]
    store_r = store_r[:t_count]

    return store_t, store_mols, store_r
```



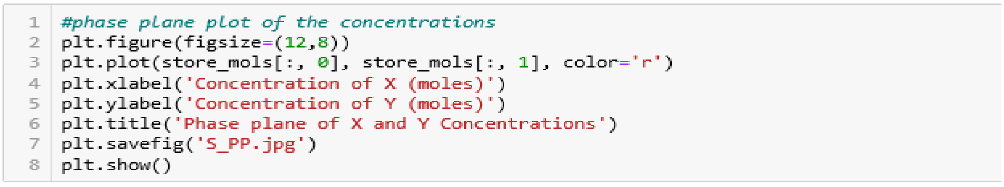
```
#gillespie alogrithm parameters

#effect reactions have on each of the species
stoch_react = np.array([[0,0], [-1,0], [-2,-1], [-1,0]])
stoch_prods = np.array([[1,0], [0,1], [3,0], [0,0]])

#reaction rates
k1 = 1
k2 = 2
k3 = 0.02
k4 = 0.04
rates = np.array([k1, k2, k3, k4])

#initial species concentrations
init = np.array([0, 0]) #[X=0, Y=0]

# specify the maximum time, tmax, and maximum number of reactions, max_no_react
tmax = 500
max_no_react = 10**6
```



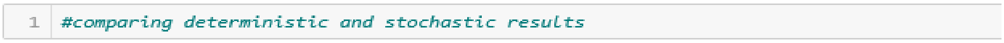
```
#results = [store_t, store_mols, store_r]
results = my_gillespie(init, rates, stoch_react, stoch_prods, tmax, max_no_react)
store_t = results[0] #array of time steps
store_mols = results[1] #array containing the number of molecules at each time step
store_r = results[2] #array of reactions which took place at each time step
```



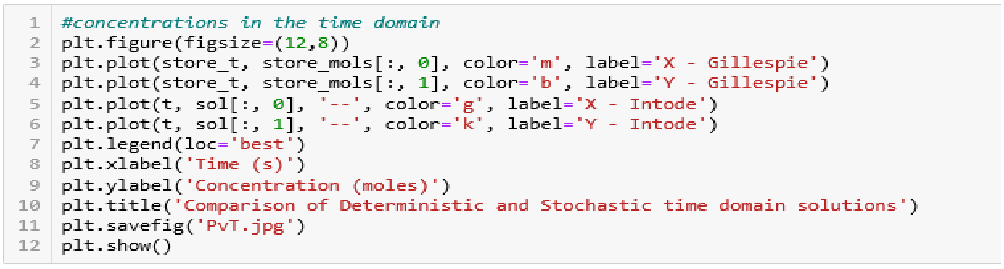
```
#concentrations in the time domain
plt.figure(figsize=(12,8))
plt.plot(store_t, store_mols[:, 0], color='b', label='X')
plt.plot(store_t, store_mols[:, 1], color='g', label='Y')
plt.legend(loc='best')
plt.xlabel('Time (s)')
plt.ylabel('Concentration (moles)')
plt.title('Time-varying Concentration of X and Y')
plt.savefig('S_PvT.jpg')
plt.show()
```



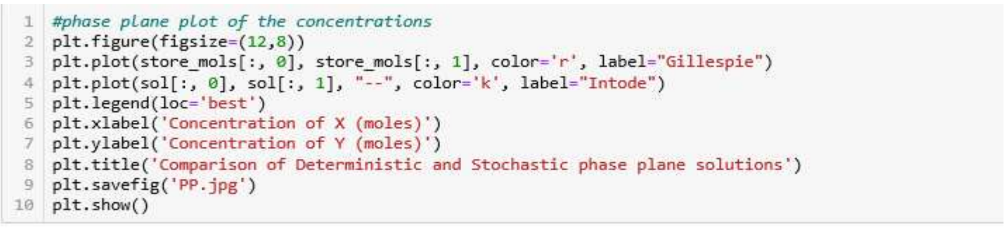
```
#phase plane plot of the concentrations
plt.figure(figsize=(12,8))
plt.plot(store_mols[:, 0], store_mols[:, 1], color='r')
plt.xlabel('Concentration of X (moles)')
plt.ylabel('Concentration of Y (moles)')
plt.title('Phase plane of X and Y Concentrations')
plt.savefig('S_PP.jpg')
plt.show()
```



```
#comparing deterministic and stochastic results
```



```
#concentrations in the time domain
plt.figure(figsize=(12,8))
plt.plot(store_t, store_mols[:, 0], color='m', label='X - Gillespie')
plt.plot(store_t, store_mols[:, 1], color='b', label='Y - Gillespie')
plt.plot(t, sol[:, 0], '--', color='g', label='X - Intode')
plt.plot(t, sol[:, 1], '--', color='k', label='Y - Intode')
plt.legend(loc='best')
plt.xlabel('Time (s)')
plt.ylabel('Concentration (moles)')
plt.title('Comparison of Deterministic and Stochastic time domain solutions')
plt.savefig('PvT.jpg')
plt.show()
```



```
#phase plane plot of the concentrations
plt.figure(figsize=(12,8))
plt.plot(store_mols[:, 0], store_mols[:, 1], color='r', label="Gillespie")
plt.plot(sol[:, 0], sol[:, 1], "--", color='k', label="Intode")
plt.legend(loc='best')
plt.xlabel('Concentration of X (moles)')
plt.ylabel('Concentration of Y (moles)')
plt.title('Comparison of Deterministic and Stochastic phase plane solutions')
plt.savefig('PP.jpg')
plt.show()
```