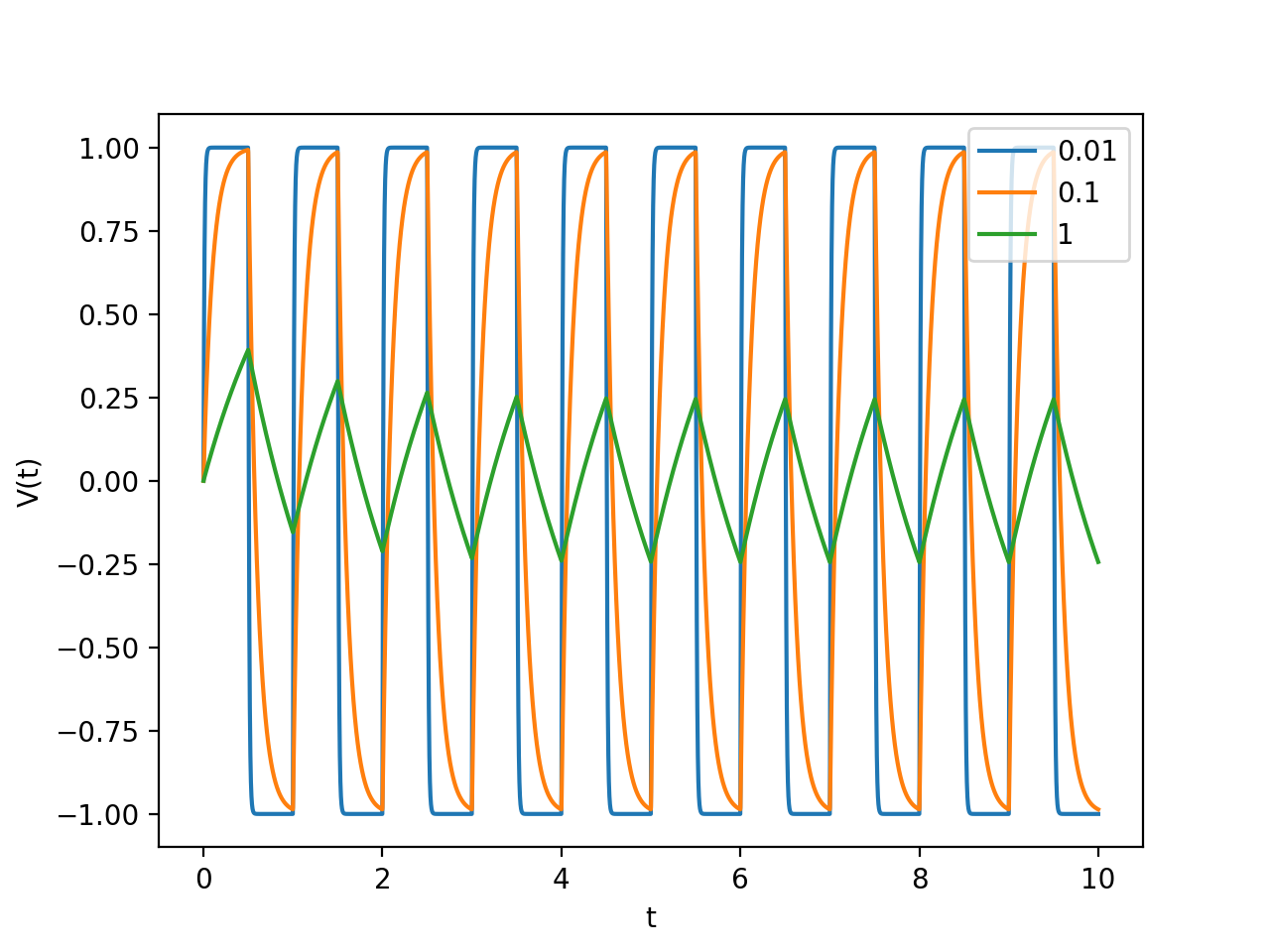
**Q1)**

**a)**



The resulting output is shown in the plot above. For different RC values, the resulting shape of the signal was changed, along with the amplitude. The general trend was as the value of RC increased, the output wave became more triangular in shape and decreased in amplitude. Value of N was chosen to be 5000, because it still ran fast, while outputting accurate results. Larger values of N would also be considered, but since the plot is very accurate with 5000 points, there seems to be no reason to further increase the step size unless absolutely high accuracy is needed.

import numpy as np

import matplotlib.pyplot as plt

# RC = 0.01 #0.1 #1

def V\_in(t):

# if even

if np.floor(2\*t) % 2 == 0:

return 1

#if odd

if np.floor(2\*t) % 2 == 1:

return (-1)

V\_out = 0

def f(V,t,RC):

return 1/RC \*(V\_in(t) - V)

t\_start = 0.0

t\_final = 10.0

N = 5000

h = (t\_final-t\_start)/N

tpoints = np.arange(t\_start,t\_final,h)

def vary\_RC(RC):

Vpoints = []

#initial V = V\_out = 0

V = V\_out

for t in tpoints:

Vpoints.append(V)

k1 = h\*f(V,t, RC)

k2 = h\*f(V+0.5\*k1,t+0.5\*h,RC)

k3 = h\*f(V+0.5\*k2,t+0.5\*h,RC)

k4 = h\*f(V+k3,t+h,RC)

V += (k1+2\*k2+2\*k3+k4)/6

return Vpoints

plt.figure()

plt.plot(tpoints,vary\_RC(0.01),label="0.01")

plt.plot(tpoints,vary\_RC(0.1),label="0.1")

plt.plot(tpoints,vary\_RC(1),label="1")

plt.xlabel("t")

plt.ylabel("V(t)")

plt.legend()

plt.show()

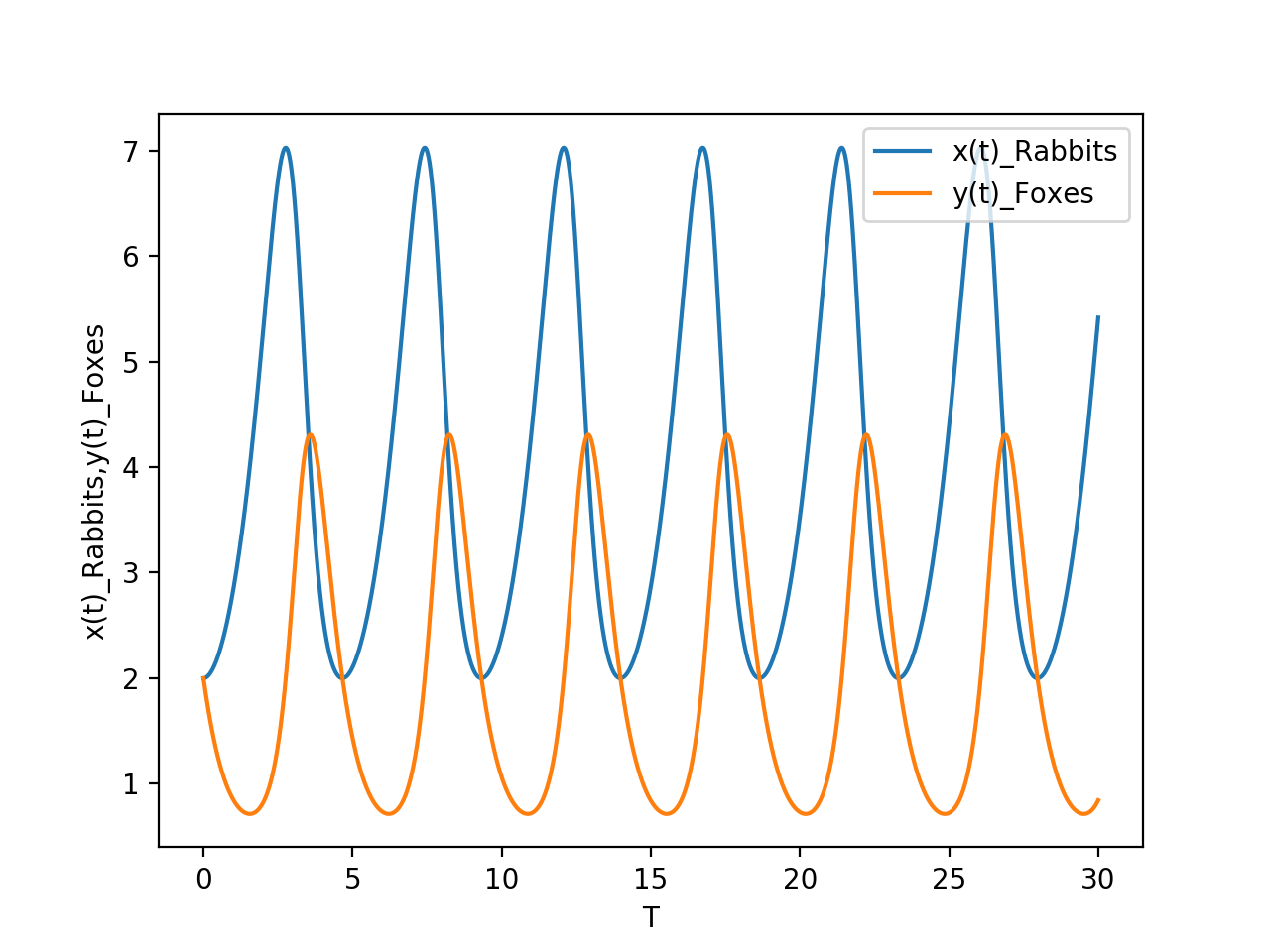
**b)**

A low pass filter passes signals with a frequency lower than the cutoff frequency and attenuates signals with higher frequencies than the cutoff frequency. The cutoff frequency is given by:

And the graphs with different RC values reflect different cutoff frequencies and hence shows different amplitudes and shapes. At low RC values, the output wave is a square wave. At medium RC values, the output wave is shaped like a shark fin. At high RC values, the output wave becomes a triangle wave. The amplitudes also decrease in value as RC increases as shown in the graphs. The RC integrator circuit converts an input signal to an output signal of different amplitude and shape, which is used in music players to equalize the music.

**Q2)**

**a)**



The output is shown in the plot above. The number of rabbits and foxes seem to stay constant over the time period and follows the same pattern as expected, because they are dependent on each other. Even altering the value of T over a longer period and altering the value of h resulted in the same general plot, which is a bit strange.

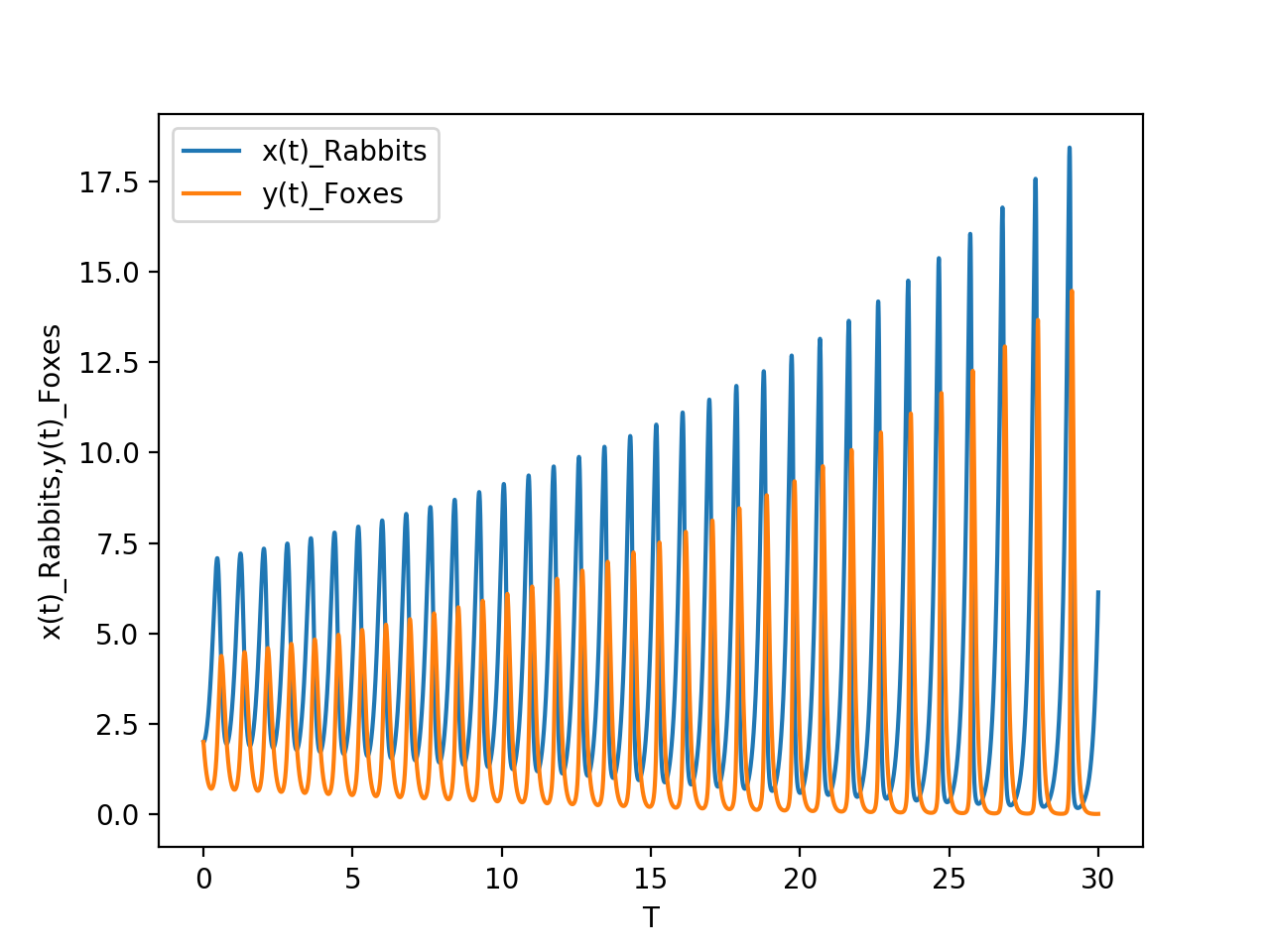
After testing for different values of variables, one change that resulted in an interesting plot was when the formula:

r += (k1+2\*k2+2\*k3+k4)/6

had been changed to:

r += (k1+2\*k2+2\*k3+k4)

the resulting output is shown below:



This plot seems much more likely from a common sense perspective, because while the number of rabbits and foxes may follow a pattern, it is expected and likely for the total population to increase with time because there will not be any external factors involved. But since this plot uses a formula different from the calculated/given formula, it is obviously expected to give different results, which may or may not be correct.

import numpy as np

import matplotlib.pyplot as plt

#initial setup

alpha = 1

beta = 0.5

gamma = 0.5

delta = 2

x\_initial = 2

y\_initial = 2

t\_start = 0.0

t\_final = 30.0

N = 20000

h = (t\_final-t\_start)/N

tpoints = np.arange(t\_start,t\_final,h)

def f\_x(x,y):

return (alpha\*x) - (beta\*x\*y)

def f\_y(x,y):

return (gamma\*x\*y) - (delta\*y)

def f(r,t):

x = r[0]

y = r[1]

# return [f\_x(x,y),f\_y(x,y)]

return np.array([f\_x(x,y),f\_y(x,y)],float)

xpoints = []

ypoints = []

# r = [x\_initial,y\_initial]

r = np.array([x\_initial,y\_initial],float)

for t in tpoints:

xpoints.append(r[0])

ypoints.append(r[1])

k1 = h\*f(r,t)

k2 = h\*f(r+0.5\*k1,t)

k3 = h\*f(r+0.5\*k2,t)

k4 = h\*f(r+k3,t)

r += (k1+2\*k2+2\*k3+k4)/6

plt.figure()

plt.plot(tpoints,xpoints, label = "x(t)\_Rabbits")

plt.plot(tpoints,ypoints, label = "y(t)\_Foxes")

plt.xlabel("T")

plt.ylabel("x(t)\_Rabbits,y(t)\_Foxes")

plt.legend()

plt.show()

**b)**

Using the first plot in part a), the number of rabbits and foxes fluctuate with dependence on each other. The number of rabbits is proportional to the number of foxes and vice versa. In addition, the total number of the population (maximum number of population) seem to be constant over any time period, even when the values of ‘t’ and ‘h’ had been changed. Therefore, it may not reflect the ‘true’ phenomenon in nature, but the general dependence on the number of rabbits and foxes seem to be correct.

On the other hand, the second plot in part a) represents a more likely scenario (at least in my expectations), because the total number of rabbits and foxes increase with time while maintaining the proportionality. As ‘t’ increases, the total number of population of rabbits and foxes should also increase, and this plot shows this behavior correctly.