hw3b_ex22

March 6, 2020

[1]:
```
# -*- coding: utf-8 -*-
"""
Created on Thu Mar  5 19:23:19 2020

@author: akswa
"""

# python 3 version
import numpy as np
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d.axes3d as p3


def rk4(x,t,tau,derivsRK,param):
    """
    ##  Runge-Kutta integrator (4th order)
    ## Input arguments -
    ##   x = current value of dependent variable
    ##   t = independent variable (usually time)
    ##   tau = step size (usually timestep)
    ##   derivsRK = right hand side of the ODE; derivsRK is the
    ##              name of the function which returns dx/dt
    ##              Calling format derivsRK(x,t).
    ## Output arguments -
    ##   xout = new value of x after a step of size tau
    """
    half_tau = 0.5*tau
    F1 = derivsRK(x,t,param)
    t_half = t + half_tau
    xtemp = x + half_tau*F1
    F2 = derivsRK(xtemp,t_half,param)
    xtemp = x + half_tau*F2
    F3 = derivsRK(xtemp,t_half,param)
    t_full = t + tau
    xtemp = x + tau*F3
    F4 = derivsRK(xtemp,t_full,param)
    xout = x + tau/6.*(F1 + F4 + 2.*(F2+F3))
```

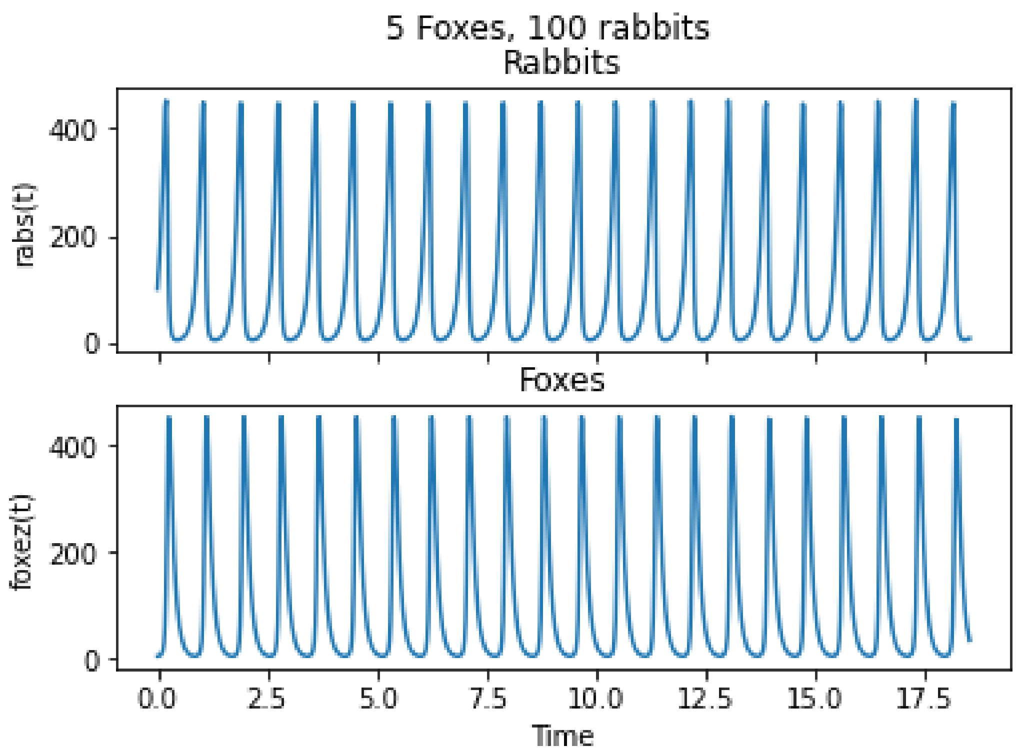
```python
    return xout

def rka(x,t,tau,err,derivsRK,param):
    """
    ## Adaptive Runge-Kutta routine
    ## Inputs
    ##   x          Current value of the dependent variable
    ##   t          Independent variable (usually time)
    ##   tau        Step size (usually time step)
    ##   err        Desired fractional local truncation error
    ##   derivsRK   Right hand side of the ODE; derivsRK is the
    ##              name of the function which returns dx/dt
    ##              Calling format derivsRK(x,t).
    ## Outputs
    ##   xSmall     New value of the dependent variable
    ##   t          New value of the independent variable
    ##   tau        Suggested step size for next call to rka
    """
    # Set initial variables
    tSave = t;  xSave = x    # Save initial values
    safe1 = .9;  safe2 = 4.  # Safety factors
    eps = np.spacing(1) # smallest value

    # Loop over maximum number of attempts to satisfy error bound
    maxTry = 100

    for iTry in range(1,maxTry):

        # Take the two small time steps
        half_tau = 0.5 * tau
        xTemp = rk4(xSave,tSave,half_tau,derivsRK,param)
        t = tSave + half_tau
        xSmall = rk4(xTemp,t,half_tau,derivsRK,param)

        # Take the single big time step
        t = tSave + tau
        xBig = rk4(xSave,tSave,tau,derivsRK,param)

        # Compute the estimated truncation error
        scale = err * (np.abs(xSmall) + np.abs(xBig))/2.
        xDiff = xSmall - xBig
        errorRatio = np.max( [np.abs(xDiff)/(scale + eps)] )

        #print safe1,tau,errorRatio

        # Estimate news tau value (including safety factors)
        tau_old = tau
```

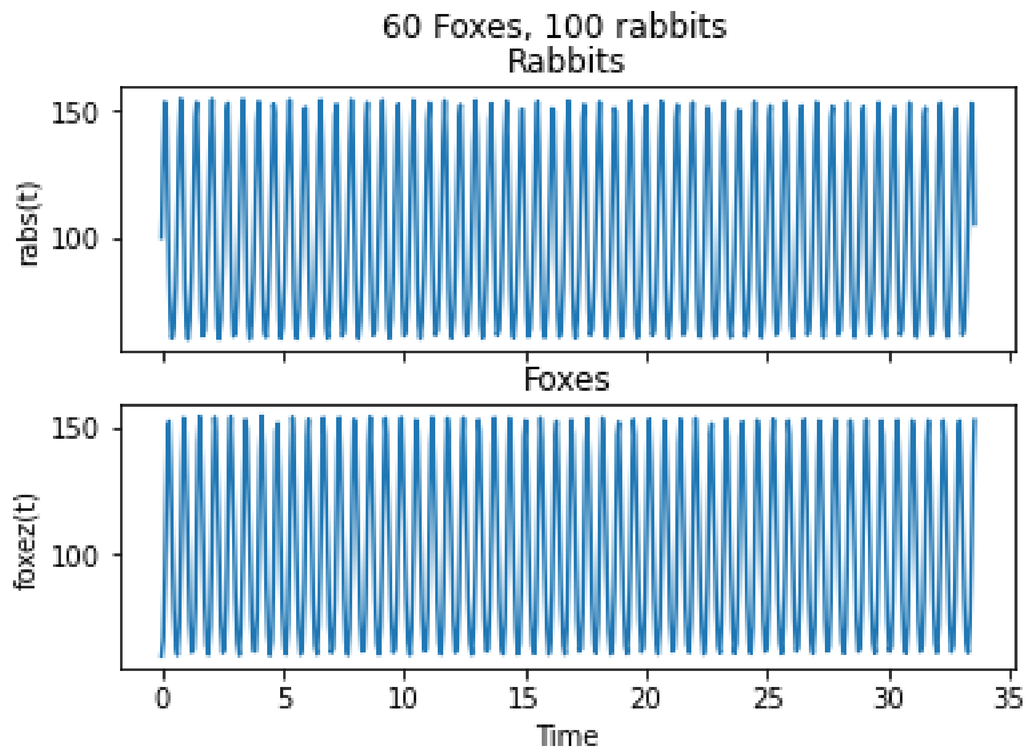
```
        tau = safe1*tau_old*errorRatio**(-0.20)
        tau = np.max([tau,tau_old/safe2])
        tau = np.min([tau,safe2*tau_old])

        # If error is acceptable, return computed values
        if errorRatio < 1 :
            xSmall = xSmall
            return xSmall, t, tau


def lotka_volterra(s,t,param):
    a = 10
    b = 10**6
    c = .1

    r = s[0]
    f = s[1]

    deriv = np.zeros(2)
    deriv[0] = a*(1-(r/b))*r - c*r*f
    deriv[1] = -a*f + c*r*f
    return deriv




def lorenz_data_gen(init_r,init_f,param):
    """
    Generates data needed to plot the results
    of lorentz.py using rk4 as in Ch3 ex 25

    Parameters
    ----------
    init_r : Int
        Inital rabbits value.
    init_y : Int
        Initial fox value.
    param : list
        List of model parameter (a,b,c).


    Returns
    -------
    rplot : Numpy array
        Array of r-values used to plot.
    fplot : Numpy array
```

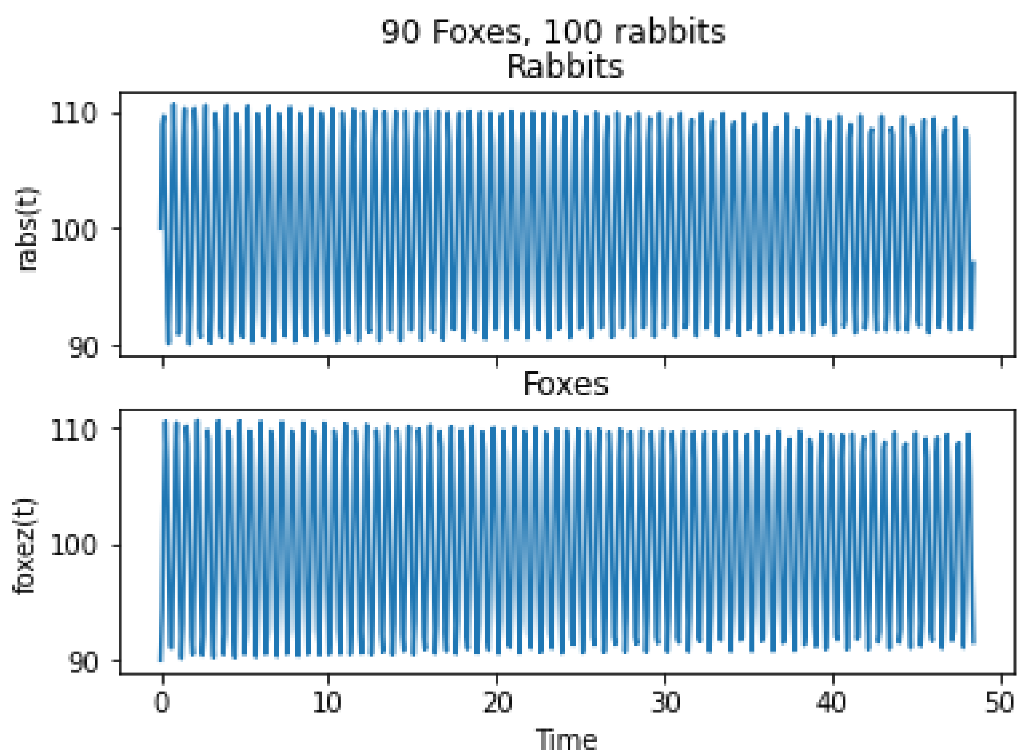
```
        Array of f-values used to plot.
    tplot : Numpy array
        Array of time-values used to plot.

    """
    # Set initial state x,y,z and parameters r,sigma,b
    srin,sfin = init_r,init_f
    state = np.zeros(2)
    state[0] = float(srin)
    state[1] = float(sfin)

    # Model paramaters
    a = param[0]
    b = param[1]
    c = param[2]
    tau = 1       # Timestep from lorenz with n=500
    err = 1.e-3   # Error tolerance

    # Loop over the desired number of steps
    time = 0
    nstep = 500
    # initialize arrays
    tplot = np.array([])
    tauplot = np.array([])
    rplot = np.array([])
    fplot = np.array([])

    for istep in range(0,nstep):
        # Record values for plotting
        r = state[0]
        f = state[1]
        tplot = np.append(tplot,time)
        tauplot = np.append(tauplot,tau)
        rplot = np.append(rplot,r)
        fplot = np.append(fplot,f)
        #if( istep%50 ==0 ):
          #print('Finished %d steps out of %d '%(istep,nstep))
        # Find new state using Runge-Kutta4
        #state = rk4(state,time,tau,lotka_volterra,param)
        #time += tau
        [state, time, tau] = rka(state,time,tau,err,lotka_volterra,param)

    # Print max and min time step returned by rka
    #print('Adaptive time step: Max = %f,  Min = %f '%(max(tauplot[1:]),␣
↪min(tauplot[1:])));
```

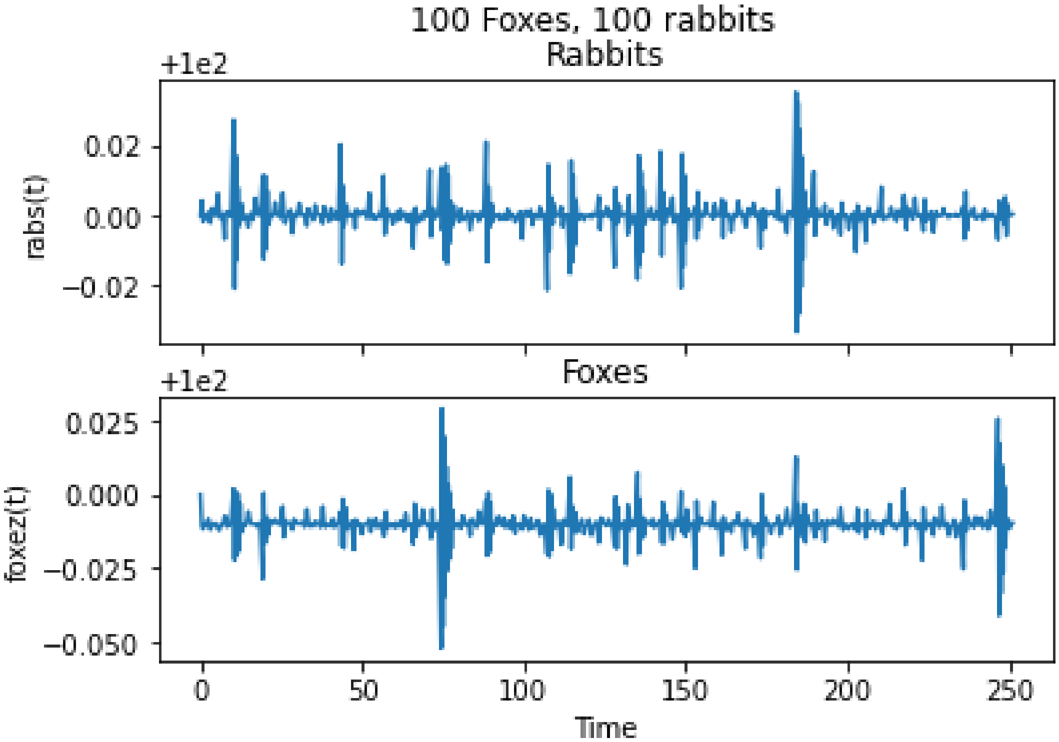
```python
    plotting = True
    if plotting:
        # Graph the time series x(t)
        fig,ax = plt.subplots(2,1,sharex = True)
        ax[0].plot(tplot,rplot,'-')
        ax[0].set_ylabel('rabs(t)')
        ax[0].set_title('Rabbits')

        ax[1].plot(tplot,fplot,'-')
        ax[1].set_xlabel('Time');
        ax[1].set_ylabel('foxez(t)')
        ax[1].set_title('Foxes')
        fig.suptitle('%s Foxes, %s rabbits' %(init_f,init_r))

    return rplot,fplot,tplot
```

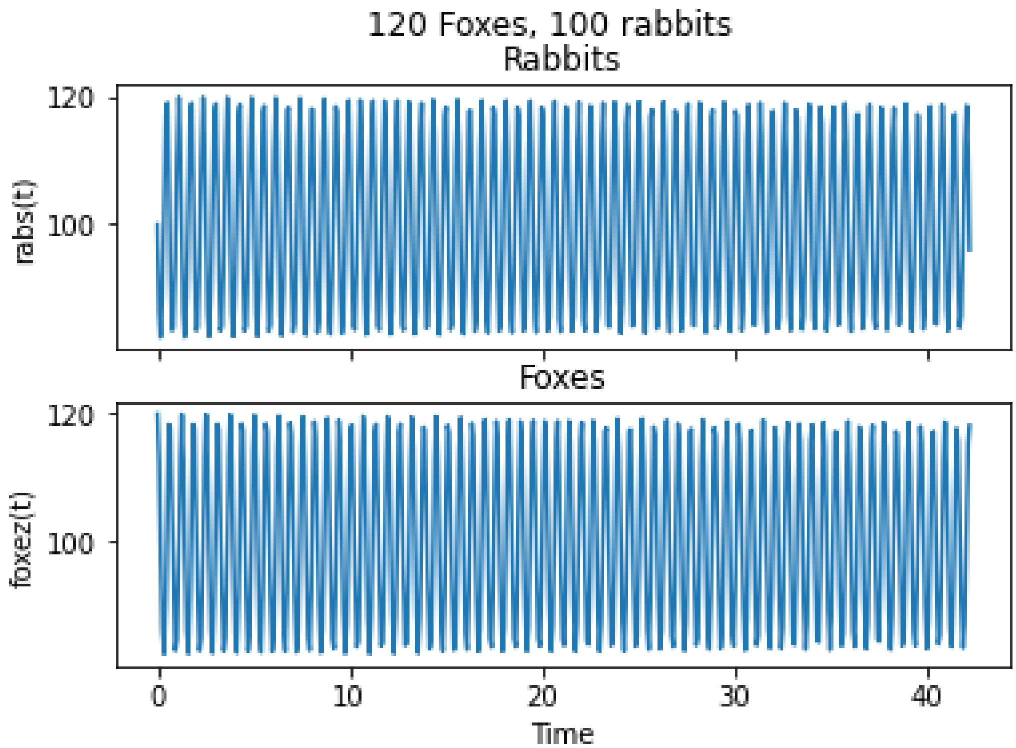
[2]:
```python
for inital_foxes in [5,60,90,100,120,140]:
    initial_cond_list = [(100,inital_foxes,(10,10**6,.1) )]
    ic_1 = initial_cond_list[0]
    rplot1,fplot1,tplot1 = lorenz_data_gen(ic_1[0],ic_1[1],ic_1[2])

#lorenz_plot(initial_cond_list)
```

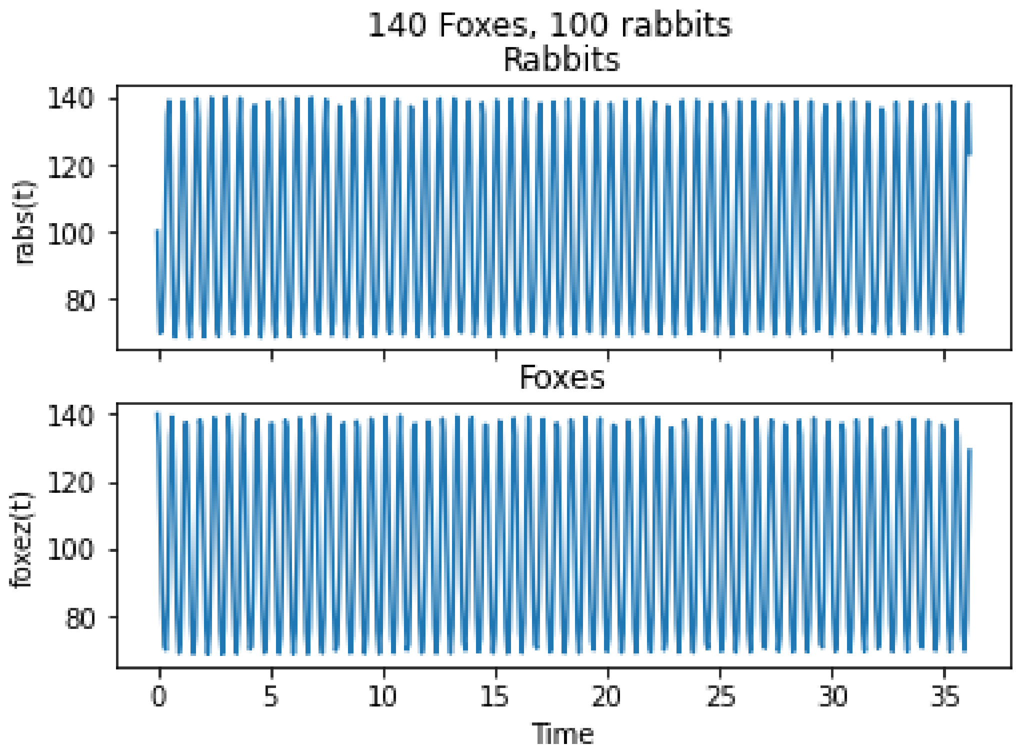



60 Foxes, 100 rabbits
Rabbits
rabs(t)
150
100
Foxes
foxez(t)
150
100
0
5
10
15
20
25
30
35
Time


90 Foxes, 100 rabbits
Rabbits
rabs(t)
110
100
90
Foxes
foxez(t)
110
100
90
0
10
20
30
40
50
Time


100 Foxes, 100 rabbits
Rabbits
+1e2
0.02
0.00
−0.02
rabs(t)
Foxes
+1e2
0.025
0.000
−0.025
−0.050
foxez(t)
0
50
100
150
200
250
Time


120 Foxes, 100 rabbits
Rabbits
rabs(t)
120
100
Foxes
foxez(t)
120
100
0
10
20
30
40
Time


140 Foxes, 100 rabbits
Rabbits
rabs(t)
140
120
100
80
Foxes
foxez(t)
140
120
100
80
0
5
10
15
20
25
30
35
Time

```
[ ]:
```