Lotka-Volterra program revisions

Ryan Ellison

June 05 2023

Email

Hi! This is Hatfield again. I spent a little bit this weekend working on the attached; I commented it to hell and back (I'm usually not this gratuitous with notation- I'm not 100% sure this is how you would have ideally wanted notation done, but at least this way nothing is unexplained), the code is broadly modular and can be broken up into chunks roughly as you described, it's user-friendly (as long as the user is able to edit and run a python script without a GUI), it spits out a pretty graph, and it relies on libraries only for extremely basic math stuff (math and NumPy) and graphical stuff (Matplotlib, because I really did not see point in building out my own graphical stuff here). The "integrator", if you want to call it that, is forward-euler and entirely of my own hand. It's odd to manipulate because it breaks pretty easily if the values given to it are weird, but I think that is probably more as a consequence of Lotka-Volterra itself really needing it's parameters to pass the "does this make sense test", given it's for modeling real and sensical values, than anything I did. I found this to be fairly simple and well within (admittedly fairly limited) abilities.

I welcome thoughts, questions, comments, criticisms, corrections, all that sort of thing. And as always, thank you for your time.

Model Output Comparisons

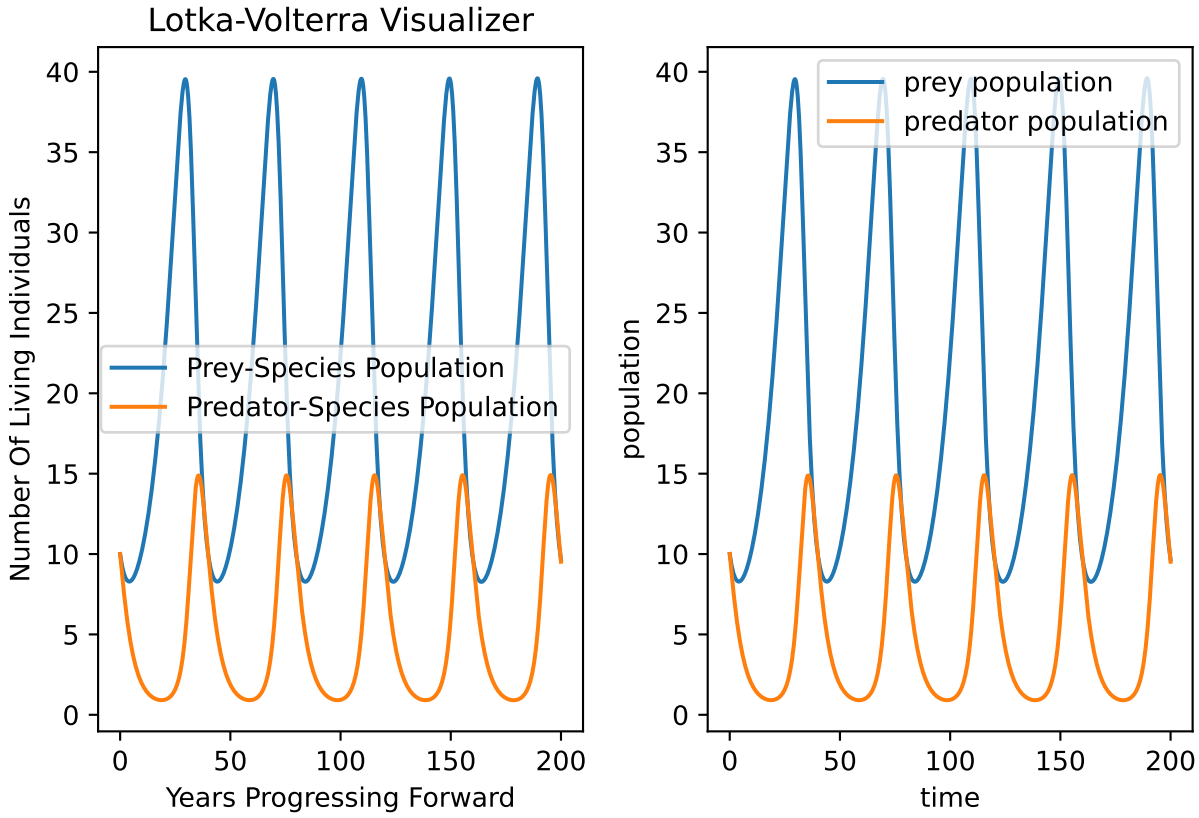

Figure 1: Comparison of results.

Hatfield L-V Model

```
+*In[78]:*+
[source, ipython3]
----
######
#Hello! Input your
alpha=.1 #(Reproduction rate of prey),
beta=.02 #(Rate at which predators kill prey)
gamma=.3 #(Mortality rate of predators)
delta=.015 #(Benifit to reproduction rate of predators per prey consumed)
n=200 #(Length of time in years that you want to model)
tau=.001 #(Timestep in years you want for your graph)
nPrey=10 #(Initial number of prey-species individuals)
nPredator=10 #(Initial number of predator-species individuals)
#Note that setting ?? too high or low for your dataset might very easily break the visualiser
######

#Library Block
#Here I'm calling all necessary libraries
import math
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
#Graphing/Functional Block
#This block does a few things- it pulls from the above greek-letter objects that we previously assigned
plt.title('Lotka-Volterra Visualizer')
plt.xlabel ('Years Progressing Forward')
plt.ylabel ('Number Of Living Individuals')
#the above makes the graph human-readable
x = []
y = []
y2= []
#defining that what we're graphing and what we're graphing it against is an array
for i in range(0, int(n/??)):#because we want this in years, not arbitrary time steps
    nPreyPrevious = nPrey #"nPreyPrevious is the "value previous" to the next number we're going to app
    nPredatorPrevious = nPredator
    nPrey = nPreyPrevious+tau*(alpha*nPreyPrevious-beta*nPreyPrevious*nPredatorPrevious)
    nPredator = nPredatorPrevious+tau*(delta*nPreyPrevious*nPredatorPrevious-gamma*nPredatorPrevious)
    y.append(nPrey)
    y2.append(nPredator)
    x.append(??*i)
    #It then finds dx/dt and dy/dt and goes ham looping over itself using forward euler.

    #Finishing Block
    #here we spit out our graph from the array we just made-
plt.plot(x,y,label='Prey-Species Population')
plt.plot(x,y2,label='Predator-Species Population')
plt.legend()
plt.show()
#-Hatfield
----
```

```
++*Out[78]:*++
----
![png](output_0_0.png)
----


++*In[ ]:*++
[source, ipython3]
----

----
```

Ellison L-V Model

```
###Sys block
import warnings
warnings.simplefilter(action = 'ignore',
                      category = FutureWarning)

###Lib block
import numpy as np
import sys
import pandas as pd
from matplotlib import pyplot as plt

###Fxn block
def mod(alpha, beta, delta, gamma, time, dt, prey0, pred0):
    '''
    Lokta-Voltera model construction:
        -alpha = reproduction rate of prey
        -beta = mortality rate of prey
        -delta = reproduction rate of predator
        -gamma = mortality rate of predators
    '''

    #Modl params
    alpha = alpha
    beta = beta
    delta = delta
    gamma = gamma

    #Run controls
    time = time #arbitrary
    dt = dt
    n = time/dt
    t = np.arange(0 + dt, time, dt) #0 + dt to account for initialization

    #Anti-bug run controls
    if len(t) == (n - 1):
        print('\nRun controls set\n')

    else:
        print('\nRun control exception\n')
        sys.exit(1)


    #DAQ vectors
    x = []
    y = []

    #Initialization
    x.append(prey0)
    y.append(pred0)

    #Nested helper fxns
    def dxdt(x, y, alpha, beta):
```

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

    def dydt(x, y, delta, gamma):
        return(delta*x*y - gamma*y)

    #Integrate using forward-Euler formalism
    for i,_ in enumerate(t):
        x.append(x[i] + dt*dxdt(x[i], y[i], alpha, beta))
        y.append(y[i] + dt*dydt(x[i], y[i], delta, gamma))

    #Package vectors in 2d heterogenous df
    data = pd.DataFrame({'t': np.insert(t, 0, 0),
                         'x': x,
                         'y': y})

    #Fxn I/O
    return(data)

def dataViz(df):
    '''
    Simple visualizer
    '''

    #Vizualize
    plt.figure()
    plt.plot(df['t'], df['x'],
             label = 'prey population')
    plt.plot(df['t'], df['y'],
             label = 'predator population')
    plt.xlabel('time')
    plt.ylabel('population')
    plt.legend(loc = 'upper right')
    plt.show()

def main(sim, viz):
    #Boolean flag for simulation
    if sim:
        #Simulate model; capture as object
        modl = mod(alpha = 0.1,
                   beta = 0.02,
                   delta = 0.015,
                   gamma = 0.3,
                   time = 200,
                   dt = 0.001,
                   prey0 = 10,
                   pred0 = 10)

    #Boolean flag for visualization
    if viz:
        #Visualize
        dataViz(df = modl)

###Main block
```

```
main(sim = True,
     viz = True)
```