Lotka Volterra

Yicheng Hong

May 2024

1 Introduction

Lotka Volterra , aka Predator Prey, is a model used to predict the population between 2 competitive species, over time, given some initial conditions
A **key point** to notice is, the change in the population of one specie, is a function not only of itself, but also as a function including the population of the other specie.

This This article is going to introduce the simplest simulation of this model using the Euler forward method.

2 Euler forward

The euler forward method is defined as:

$$y_{n+1} = y_n + hF(x_n, y_n) \tag{1}$$

Where $F(x, y)$ is defined as $\frac{dx}{dy}_{(x,y)}$, which is the derivative of y about x, in the form of a function of x, and y.
h is the size of the step, and like that, an euler forward loop is formed.

3 Codes

3.1 Differential functions setup

In this section, I am going to explain my code part by part.
Let X represent the population of the prey, and Y represent the population of the predator.

```
def f_x(x, y, params):
  a = params[0]
  b = params[1]
  dx_dt = a * x - ( b * x * y )
  return dx_dt

def f_y(x, y, params):
  c = params[2]
  d = params[3]
  dy_dt = c * x * y - ( d * y )
  return dy_dt
```

Figure 1: Creating differential equations for X and Y

In Figure 1, I included 4 parameters, which are:

a: the birthrate of the prey

b: the eating rate of the predator

c: the birthrate of the predator

d: the death rate of predator

We assume that: the birthrate of the prey and the death rate of the predator are only associated with their own population.
While the eating rate of the predator and the birthrate of the predator are associated with both populations.
The functions would return the derivatives of X and Y, respectively.

3.2 Subscripting initial conditions

```
def lotka_volterra(init_cond, params, h, t_fin):
    t = init_cond[0]
    x = init_cond[1]
    y = init_cond[2]
    T = []
    T.append(t)
    X = []
    X.append(x)
    Y = []
    Y.append(y)
```

Figure 2: Subscripting initial conditions and parameters

I will start off by introducing the inputs:

init_cond: a list containing the initial conditions (x_0, y_0, t_0)

params: a list containing the parameters needed(a, b, c, d) to call the differential functions

h: the size of the step

t_fin: the time that the simulation ends

In Fig 2, all I am doing is subscripting the initial conditions from the list init_cond, and create empty lists for X,Y,and T, then append the initial conditions into the empty lists, respectively, so they are prepared for a forward loop.

3.3 Creating the forward loop

```
for i in range(t, int(t_fin/h)):
  x_next = x + h * f_x(x, y, params)
  y_next = y + h * f_y(x, y, params)
  t = t + h
  x = x_next
  y = y_next
  T.append(t)
  X.append(x)
  Y.append(y)
```

Figure 3: The forward loop

I created a forward loop, using the "for" function in python, by applying the formula illustrated in (1) on X and Y, respectively, however, to use the method illustrated in (1), we need the derivatives $\frac{dx}{dt}$ and $\frac{dy}{dt}$, which we defined at the beginning of the code, hence we just have to call the functions that we pre-defined shown in Figure 2, inside our body function, and then establish the forward loop.

The function is defined in a way such that X and Y will be monitored for every h time units, until time reaches the final time(t_fin).

3.4 Graphing

```
plt.plot(T, X, 'r')
plt.plot(T, Y, 'b')
plt.xlabel('Time')
plt.ylabel('Amount of Population')
plt.title('Predator Prey Model')
plt.show()
print('At time', t, ', the population of prey is', x, ', the population of predator is', y)
return None
```

Figure 4: code to create the graph

Using the functions from the library *matplotlib.pyplot*, I will be able to actually make the graphs of the population of X against time, and population of Y against time, on the same graph, that's basically what the last part of the code is doing.

4 Graphs