trabajo_01_millan

August 20, 2024

1 Modelo Lotka - Volterra

Trabajo por: Jessica Naomi Millan Sánchez

1.1 ¿Qué es el modelo?

Es un modelo que representa la interacción entre dos especies, presa y depredador, en un ambiente controlado. Para buscar un equilibrio entre ambas especies.

1.1.1 Implementación del modelo en python

[ ]:
```
# Importamos las librerias necesarias
```

[1]:
```
import matplotlib.pyplot as plt
from random import *
from numpy import *
import sys
```

Estado original En este ejemplo se inicia con una población de depredadores mayor (0.5) a comparación de la población inicial de presas (0.1), sin embargo, el factor de aumento natural de las presas es mucho mayor, es decir, que se reproducen más rápido. Por lo que podemos sugerir que la gráfica comenzará con un valor mayor para los depredadores, pero pronto será alcanzado y superado por el número de presas. Los parámetros de muerte para ambas especies se encuentran equilibrados, por lo que no veremos descensos o ascensos de poblaciones tan extremos.

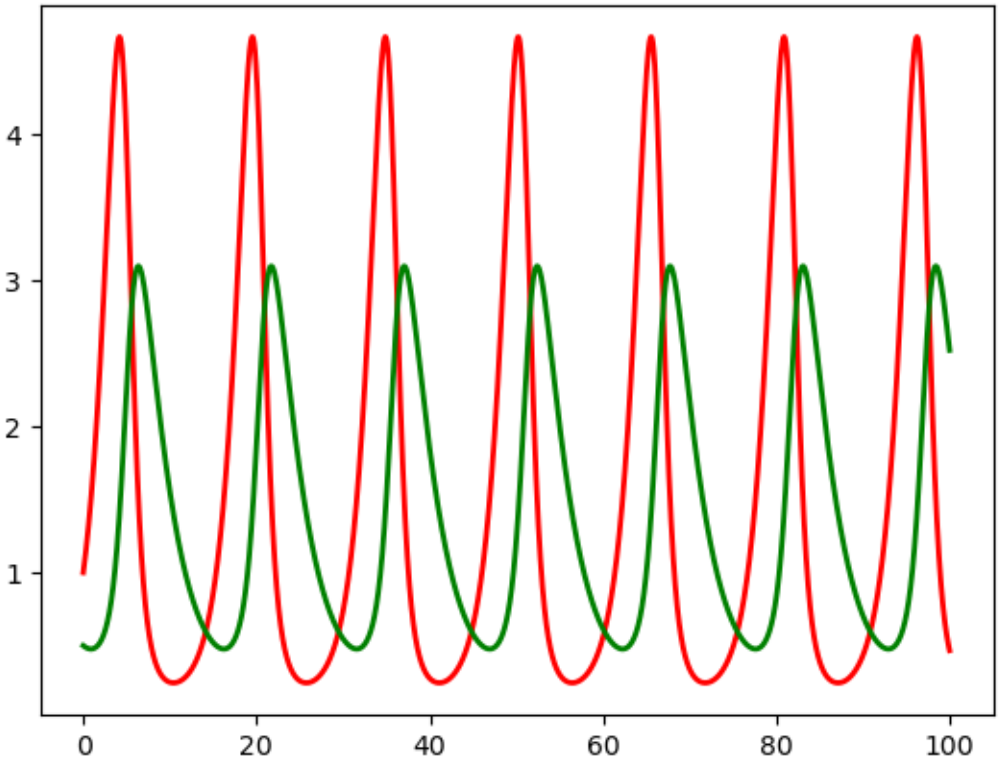
- Nota: Para las gráficas el color rojo pertenece a las presas, y el color verde a los depredadores

[ ]:
```
# Definimos los parámetros del modelo
```

[7]:
```
# model parameters
a = 0.7; b = 0.5; c = 0.3;  e = 0.2
dt = 0.001; max_time = 100

# initial time and populations
t = 0; x = 1.0; y = 0.5
```

[ ]:
```
# Resto del código donde se definen las ecuaciones de Lotka - Volterra y se␣
↪muestra la gráfica
```

[8]:
```
# empty lists in which to store time and populations
t_list = []; x_list = []; y_list = []

# initialize lists
t_list.append(t); x_list.append(x); y_list.append(y)

while t < max_time:
    # calc new values for t, x, y
    t = t + dt
    x = x + (a*x - b*x*y)*dt
    y = y + (-c*y + e*x*y)*dt

    # store new values in lists
    t_list.append(t)
    x_list.append(x)
    y_list.append(y)

# Plot the results
p = plt.plot(t_list, x_list, 'r', t_list, y_list, 'g', linewidth = 2)
```



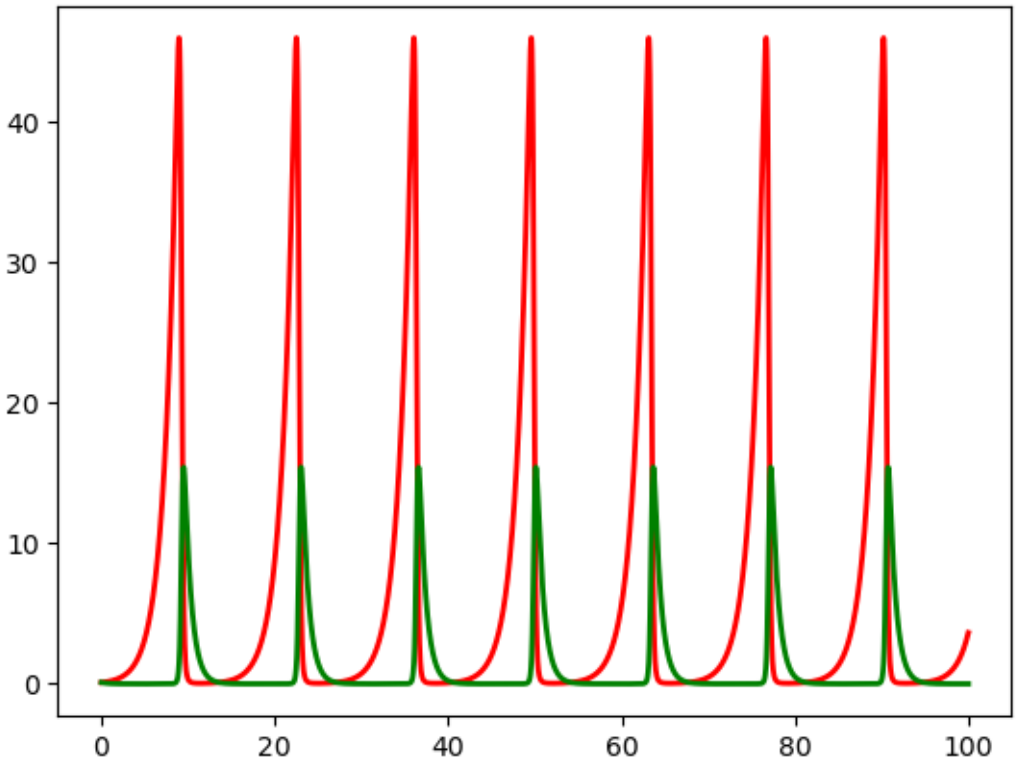
Estado 1 Se toman los parámetros anteriores, pero se aumenta considerablemente el parámetro C a 1.3. Este parámetro indica la muerte del depredador en ausencia de la presa, por lo tanto,

podemos deducir que en la gráfica de salida veremos una drástica disminución de los depredadores, y por ende, un gran aumento de la especie 'presa'.

[ ]:
```
# Definimos los parámetros del modelo
```

[12]:
```
# model parameters
a = 0.7; b = 0.5; c = 1.3;  e = 0.2
dt = 0.001; max_time = 100

# initial time and populations
t = 0; x = 0.1; y = 0.1
```

[13]:
```
# empty lists in which to store time and populations
t_list = []; x_list = []; y_list = []

# initialize lists
t_list.append(t); x_list.append(x); y_list.append(y)

while t < max_time:
    # calc new values for t, x, y
    t = t + dt
    x = x + (a*x - b*x*y)*dt
    y = y + (-c*y + e*x*y)*dt

    # store new values in lists
    t_list.append(t)
    x_list.append(x)
    y_list.append(y)

# Plot the results
p = plt.plot(t_list, x_list, 'r', t_list, y_list, 'g', linewidth = 2)
```

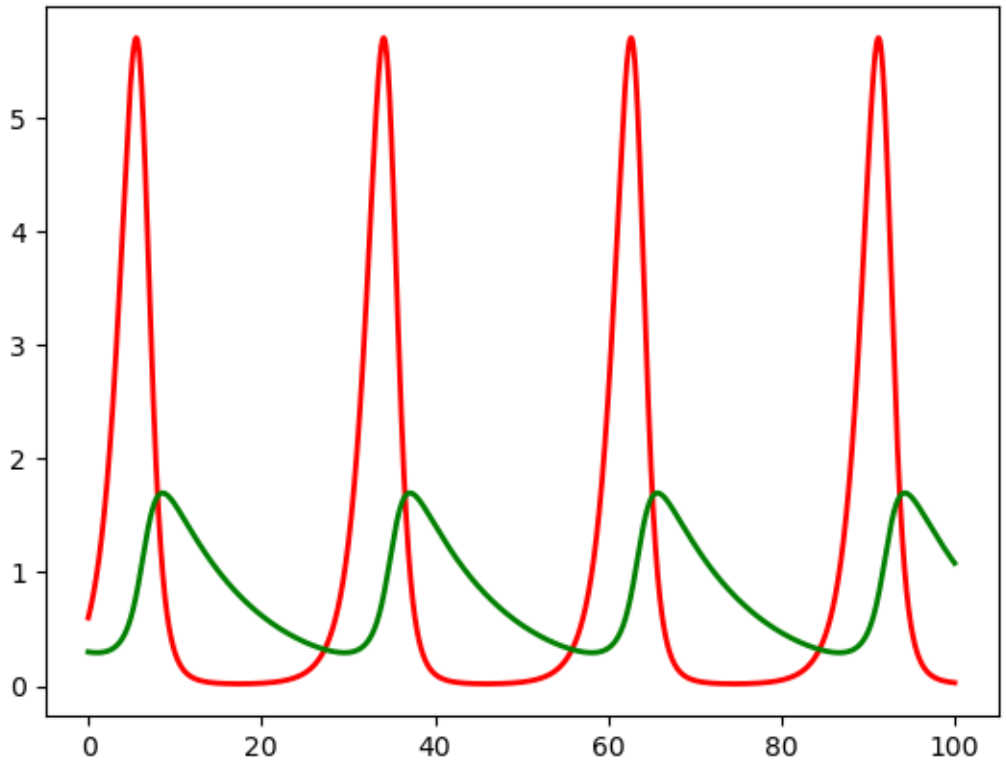


Estado 2 Aquí iniciamos con una número de presas que es el doble que el de depredadores, sin embargo al elevar el factor de destrucción por depredadores (parámetro B) en un 50% al que teníamos anteriormente, y al disminuir el aumento natural de las presas a un 0.4 (parámetro A) podemos deducir que la especie de las presas pasará rápidamente de la sobre población del ecosistema, a un punto cercano de extinción, lo que a su vez provocará la muerte de los depredadores, sin embargo como el factor de muerte en depredadores por ausencia de presa es casi nulo, no veremos un cambio en su población tan drástico, ni tan rápido como en el caso de las presas.

[14]:
```
# model parameters
a = 0.4; b = 1; c = 0.1;  e = 0.3
dt = 0.001; max_time = 100

# initial time and populations
t = 0; x = 0.6; y = 0.3
```

[15]:
```
# empty lists in which to store time and populations
t_list = []; x_list = []; y_list = []

# initialize lists
t_list.append(t); x_list.append(x); y_list.append(y)

while t < max_time:
```

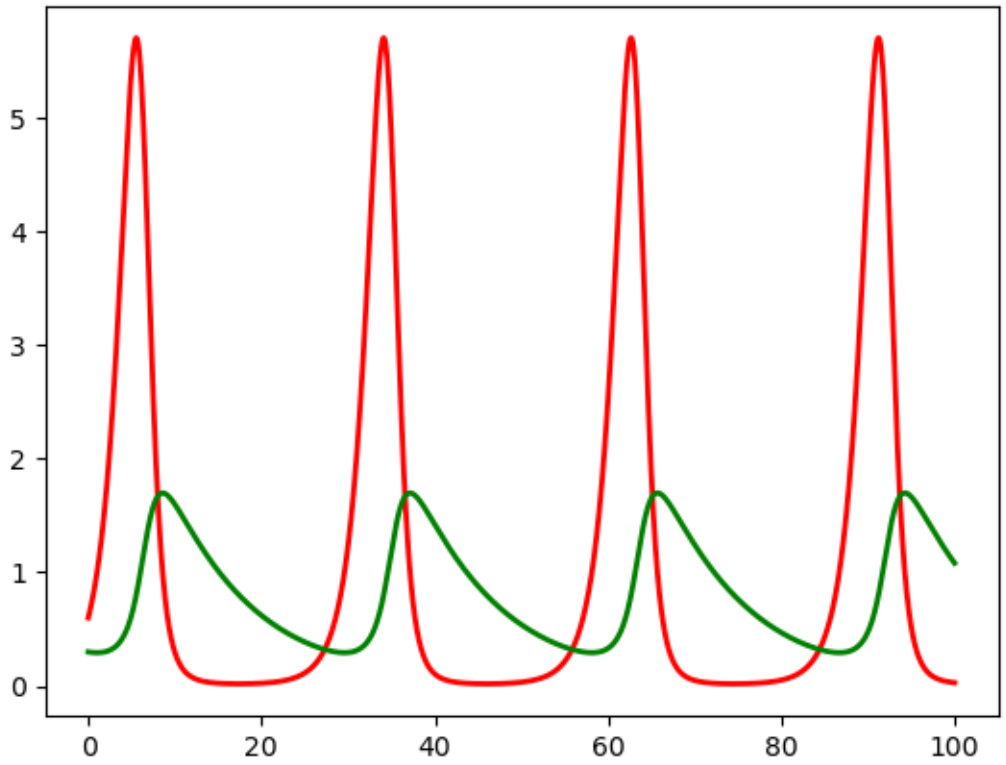
```
    # calc new values for t, x, y
    t = t + dt
    x = x + (a*x - b*x*y)*dt
    y = y + (-c*y + e*x*y)*dt

    # store new values in lists
    t_list.append(t)
    x_list.append(x)
    y_list.append(y)

# Plot the results
p = plt.plot(t_list, x_list, 'r', t_list, y_list, 'g', linewidth = 2)
```



Estado 3 Retomamos el estado anterior, pero esta vez aumentamos el parámetro A con un valor de 0.8 y disminuimos el parámetro E a 0.1 Esto significa que las presas tendrán un mayor índice de crecimiento, mientras que los depredadores tendrán un crecimiento de población mucho más lento. Tomando esto en cuenta, junto con los valores de población inicial, podemos asegurar que en primer momento habrá una sobre población de presas, lo que a su vez propiciará un aumento en los depredadores, sin embargo, el punto máximo de los depredadores estará muy por debajo del que pueden lograr las presas, esto es por el factor de crecimiento, así como la población inicial.

[16]:
```
# model parameters
a = 0.8; b = 1; c = 0.1;  e = 0.1
dt = 0.001; max_time = 100

# initial time and populations
t = 0; x = 0.6; y = 0.3
```

[17]:
```
# empty lists in which to store time and populations
t_list = []; x_list = []; y_list = []

# initialize lists
t_list.append(t); x_list.append(x); y_list.append(y)

while t < max_time:
    # calc new values for t, x, y
    t = t + dt
    x = x + (a*x - b*x*y)*dt
    y = y + (-c*y + e*x*y)*dt

    # store new values in lists
    t_list.append(t)
    x_list.append(x)
    y_list.append(y)

# Plot the results
p = plt.plot(t_list, x_list, 'r', t_list, y_list, 'g', linewidth = 2)
```

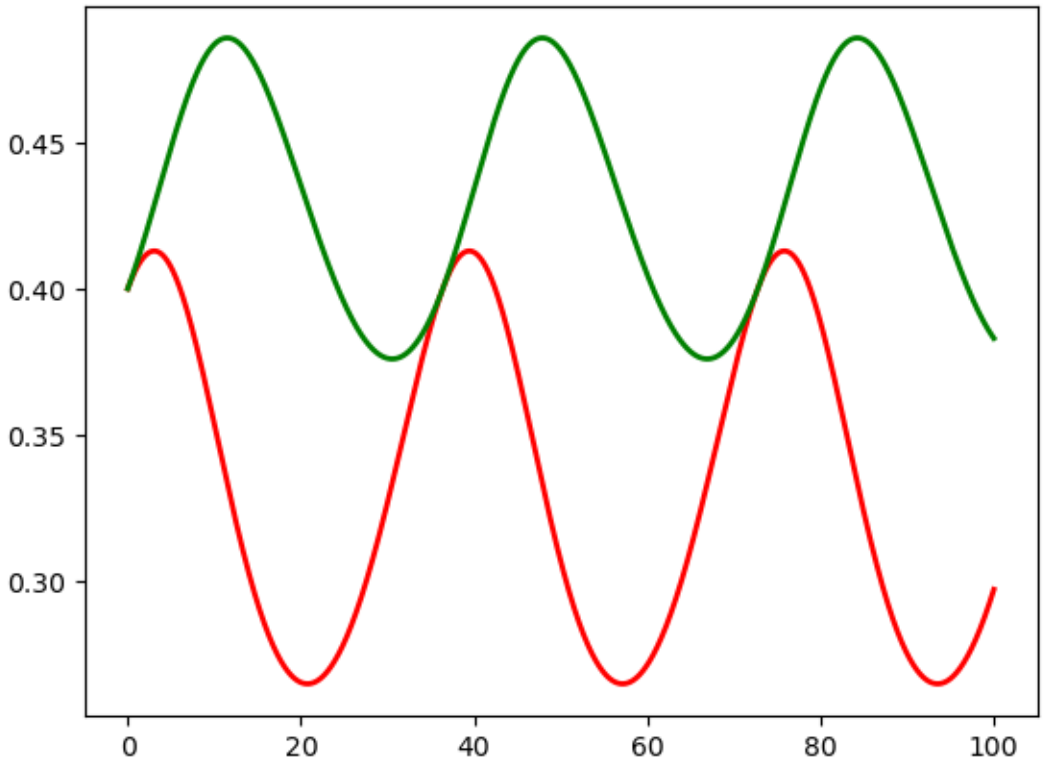


Estado 4 Ahora bien, si queremos que sean los depredadores quienes estén en el pico máximo de población, podemos hacer lo siguiente. Comenzamos con una población inicial igual para ambas especies, y definimos su ritmo de crecimiento con el mismo índice, para mantener a las especies en niveles similares, sin embargo, para lograr que los depredadores sean la especie dominante debemos aumentar el parámetro de muerte de las presas por depredadores, en este caso en 0.7. Esto causará el rápido deceso de las presas, lo que posteriormente causaría la muerte de los depredadores, para amortiguar esta situación, establecemos el índice de muerte por ausencia de presa en un nivel bajo como 0.1.

[18]:
```
# model parameters
a = 0.3; b = 0.7; c = 0.1;  e = 0.3
dt = 0.001; max_time = 100

# initial time and populations
t = 0; x = 0.4; y = 0.4
```

[19]:
```
# empty lists in which to store time and populations
t_list = []; x_list = []; y_list = []

# initialize lists
t_list.append(t); x_list.append(x); y_list.append(y)
```

```
while t < max_time:
    # calc new values for t, x, y
    t = t + dt
    x = x + (a*x - b*x*y)*dt
    y = y + (-c*y + e*x*y)*dt

    # store new values in lists
    t_list.append(t)
    x_list.append(x)
    y_list.append(y)

# Plot the results
p = plt.plot(t_list, x_list, 'r', t_list, y_list, 'g', linewidth = 2)
```

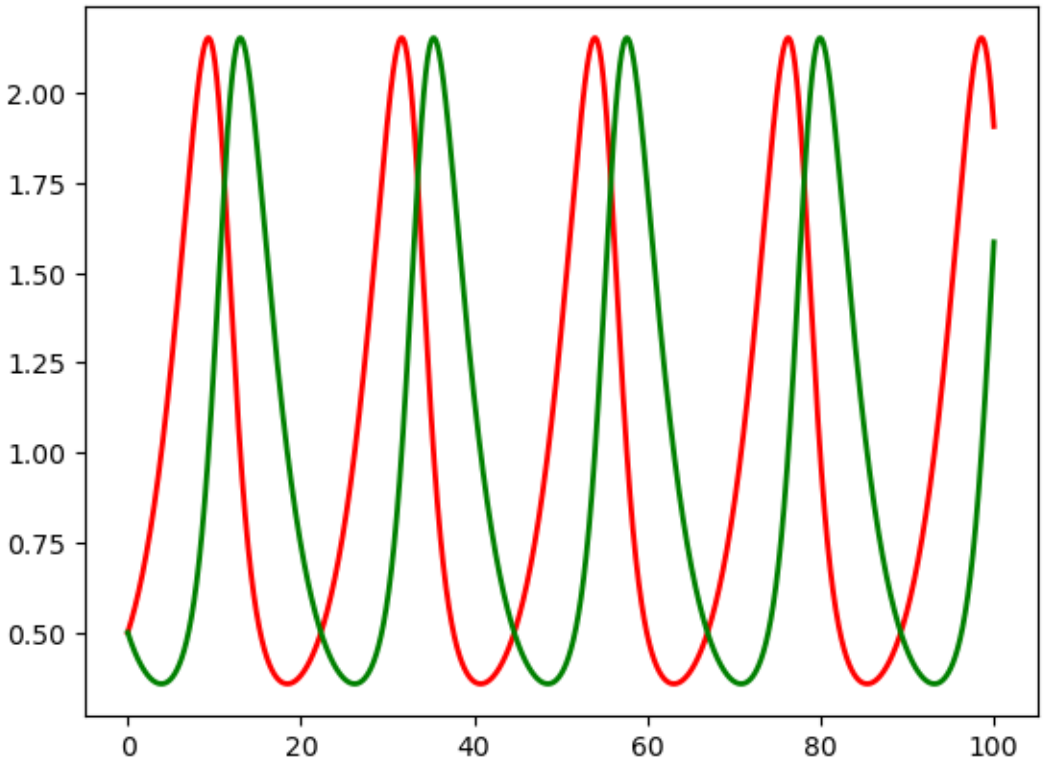


Estado 5 En este ejemplo, se inicializaron ambas poblaciones (presas y depredadores) en 0.5 y se ajustaron todos los parámetros para que tuvieran el mismo valor. Esto asegura que tanto las presas como los depredadores se reproducen al mismo ritmo. Por lo que si solo consideráramos estas condiciones el crecimiento de ambas poblaciones sería idéntico. Sin embargo, al tomar en cuenta las interacciones entre las dos especies, esta igualdad se altera. Para mantener en equilibrio las condiciones, se ha puesto el mismo valor (0.3) para los factores de muerte para ambas especies. Esto implica que los valores de aparición y desaparición de presas y depredadores son iguales, pero no simultáneo. La ausencia de presas provoca la disminución de los depredadores, mientras que

la falta de depredadores permite que las proliferación de las presas, lo que a su vez, el lleva a un incremento de depredadores. Entonces es de suponer que la gráfica que se obtiene muestra patrones similares de aumento y disminución para presas y depredadores pero de forma desfasada.

[20]:
```
# model parameters
a = 0.3; b = 0.3; c = 0.3;  e = 0.3
dt = 0.001; max_time = 100

# initial time and populations
t = 0; x = 0.5; y = 0.5
```

[21]:
```
# empty lists in which to store time and populations
t_list = []; x_list = []; y_list = []

# initialize lists
t_list.append(t); x_list.append(x); y_list.append(y)

while t < max_time:
    # calc new values for t, x, y
    t = t + dt
    x = x + (a*x - b*x*y)*dt
    y = y + (-c*y + e*x*y)*dt

    # store new values in lists
    t_list.append(t)
    x_list.append(x)
    y_list.append(y)

# Plot the results
p = plt.plot(t_list, x_list, 'r', t_list, y_list, 'g', linewidth = 2)
```


2.00
1.75
1.50
1.25
1.00
0.75
0.50
0
20
40
60
80
100

[ ]: