Mini-Projeto Simulação

Simulação e Otimização - 2022/2023

Ricardo Rodriguez - 98388

TABLE OF CONTENTS

1st problem

01

PROBLEM

Simulate a service facility with two type A servers and one type B server.

- Type 1 customers choose a type A server if available, while type 2 customers require simultaneous service from both a type A and type B server. Preference is given to type 2 customers upon completion of service.

- The simulation is run for 1000 minutes to estimate average delay, time average number in queue, and server utilization.

- After, analyze the impact of adding an additional type A or type B server is compared in terms of reducing maximum average delay.

SOLUTION

- The problem was approached by defining constants such as the number of servers, arrival rate, customer probabilities, service times, and simulation time.
- Variables were initialized to calculate wait times, average number of customers in each queue, and server allocation.
- The simulation loop ran until the simulation time limit was reached, processing arrival and departure events.
- Customers were added to their respective queues based on their type and served by available servers. If no servers were available, customers were added to their respective queues.
- Finally, metrics were calculated to analyze the simulation.

RESULTS

```
======= expected average delay in queue =======
[TYPE 1] 0.41141819946269426 seconds
[TYPE 2] 0.381659293624804 5 seconds
======= expected time average number in queue for each type of customer =======
[TYPE 1] 0.0178
[TYPE 2] 0.0166
======= expected proportion of time that each server spends on each type of customer =======
[SERVER A1 & TYPE 1] 0.46660143502496737
[SERVER A1 & TYPE 2] 0.5333985649750326
[SERVER A2 & TYPE 1] 0.22955250894598603
[SERVER A2 & TYPE 2] 0.770447491054014
[SERVER B1 & TYPE 1] 0.1797828775344307
[SERVER B1 & TYPE 2] 0.8202171224655693
```

Default results (2 type A servers and 1 type B server)

```
======= expected average delay in queue =======
[TYPE 1] 0.25960616833228817 seconds
[TYPE 2] 0.37211084080620604 seconds
======= expected time average number in queue for each type of customer =======
[TYPE 1] 0.0028
[TYPE 2] 0.0075
```

Results for 3 type A servers and 1 type B server

```
======= expected average delay in queue =======
[TYPE 1] 0.2520375119865399 seconds
[TYPE 2] 0.3274477067095082 seconds
======= expected time average number in queue for each type of customer =======
[TYPE 1] 0.0025
[TYPE 2] 0.0112
```

Results for 2 type A servers and 2 type B servers

* minutes (not seconds)

2nd problem

02

PROBLEM

Simulate the evolution of prey and predator populations using the Lotka-Volterra model mathematical equations, which depend on some previously defined parameters, using the **Euler's** method and the **Runge Kutta**'s method.

- x0 - Initial number of preys
- y0 - Initial number of predators
- Alpha - Maximum prey per capita growth rate
- Beta - Effect of presence of predators on prey growth rate
- Delta - Effect of prey presence on predator growth rate
- Gamma - Per capita mortality rate of the predator
- *time_step* - Time interval between sampling points
- *max_time* - Maximum simulation time

SOLUTION

Both variant's methods start by:

- Retrieving the needed input parameters
- Create a list of temporal values, from zero to *max_time*, with *time_step* intervals
- Update the first elements of x and y arrays with x0 and y0, respectively
- Run the simulation for *n* sampling points
- Estimate the current number of preys and predators
- Plot the evolution graph of the preys' and predators' populations

SOLUTION

```python
# Exercise 2.1 -  trace the evolution of x(t), and y(t), using the Forward Euler method given the method input arguments
def lotka_volterra_forward_euler(x0, y0, alpha, beta, delta, gamma, time_step, max_time):
    times = np.arange(0, max_time + time_step, time_step)   # Fills up a zero-valued array from [0, max_time[ in *time_step* (delta_t) steps
    n = len(times)
    x = np.zeros(n)     # Number of preys
    y = np.zeros(n)     # Number of predators
    x[0] = float(x0)           # Initial number of preys
    y[0] = float(y0)           # Initial number of predators
    # Compute the predator-prey population evolution over n iterations
    for i in range(1, n):
        x[i] = x[i-1] + (alpha * x[i-1] - beta * x[i-1] * y[i-1]) * time_step
        y[i] = y[i-1] + (- gamma * y[i-1] + delta * x[i-1] * y[i-1]) * time_step
    return times, x, y

def lotka_volterra_runge_kutta(x0, y0, alpha, beta, delta, gamma, time_step, max_time):
    times = np.arange(0, max_time + time_step, time_step)
    n = len(times)
    x = np.zeros(n)
    y = np.zeros(n)
    x[0] = x0
    y[0] = y0
    # Compute the predator-prey population evolution over n iterations
    for i in range(1, n):
        k1x = alpha * x[i-1] - beta * x[i-1] * y[i-1]
        k1y = delta * x[i-1] * y[i-1] - gamma * y[i-1]
        k2x = alpha * (x[i-1] + k1x*time_step/2) - beta * (x[i-1] + k1x*time_step/2) * (y[i-1] + k1y*time_step/2)
        k2y = delta * (x[i-1] + k1x*time_step/2) * (y[i-1] + k1y*time_step/2) - gamma * (y[i-1] + k1y*time_step/2)
        k3x = alpha * (x[i-1] + k2x*time_step/2) - beta * (x[i-1] + k2x*time_step/2) * (y[i-1] + k2y*time_step/2)
        k3y = delta * (x[i-1] + k2x*time_step/2) * (y[i-1] + k2y*time_step/2) - gamma * (y[i-1] + k2y*time_step/2)
        k4x = alpha * (x[i-1] + k3x*time_step) - beta * (x[i-1] + k3x*time_step) * (y[i-1] + k3y*time_step)
        k4y = delta * (x[i-1] + k3x*time_step) * (y[i-1] + k3y*time_step) - gamma * (y[i-1] + k3y*time_step)
        x[i] = x[i-1] + time_step/6 * (k1x + 2*k2x + 2*k3x + k4x)
        y[i] = y[i-1] + time_step/6 * (k1y + 2*k2y + 2*k3y + k4y)
    return times, x, y
```

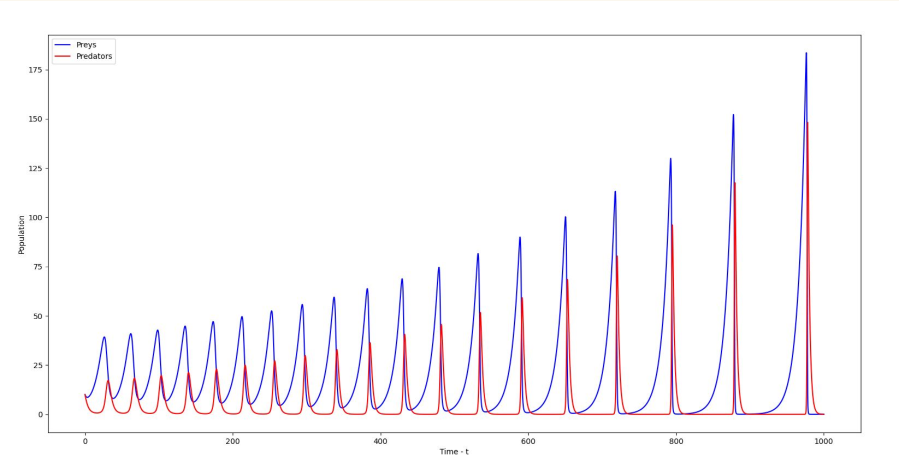
Implementation

SOLUTION

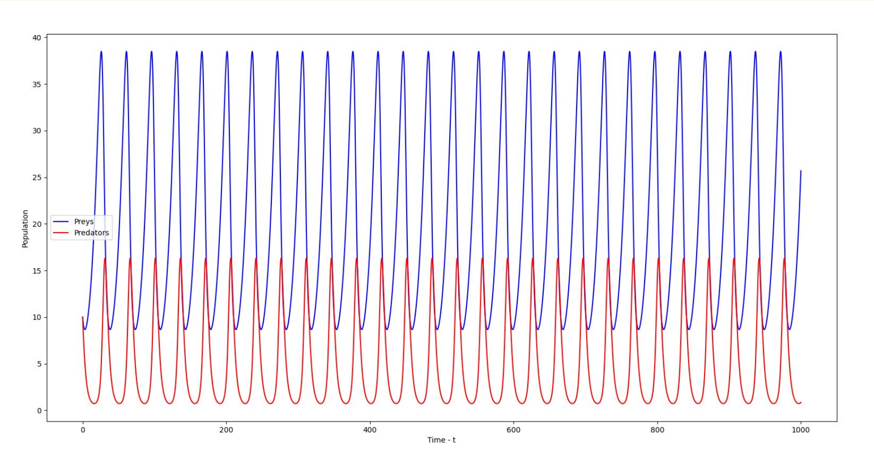
```
parser = argparse.ArgumentParser(description='CLI Interface to the predator-prey populations simulation program using the Lotka-Volterra model')
parser.add_argument('x0', type=float, help='Initial number of preys (population density)')
parser.add_argument('y0', type=float, help='Initial number of predators (population density)')
parser.add_argument('alpha', type=float, help='Maximum prey per capita growth rate')
parser.add_argument('beta', type=float, help='Effect of the presence of predators on the prey growth rate')
parser.add_argument('delta', type=float, help="Effect of the presence of prey on the predator's growth rate")
parser.add_argument('gamma', type=float, help="Predator's per capita death rate")
parser.add_argument('time_step', type=float, help='Time step interval')
parser.add_argument('max_time', type=float, help='Time of the simulaiton')
parser.add_argument('--method', type=str, default='euler', choices=['euler', 'runge_kutta'], help='Lotka Volterra variation method')
args = parser.parse_args()
```

Command line arguments parsing

RESULTS



Results for **Euler method**
python3 ex2.py 10 10 0.1 0.02 0.02 0.4 0.1 1000 --method euler



Results for **Runge Kutta method**
python3 ex2.py 10 10 0.1 0.02 0.02 0.4 0.1 1000 --method runge_kutta

* minutes (not seconds)

THANKS