deti universidade de aveiro
departamento de electrónica, telecomunicações e informática

Simulation Mini-project

Simulação e Otimização
Assignment 1

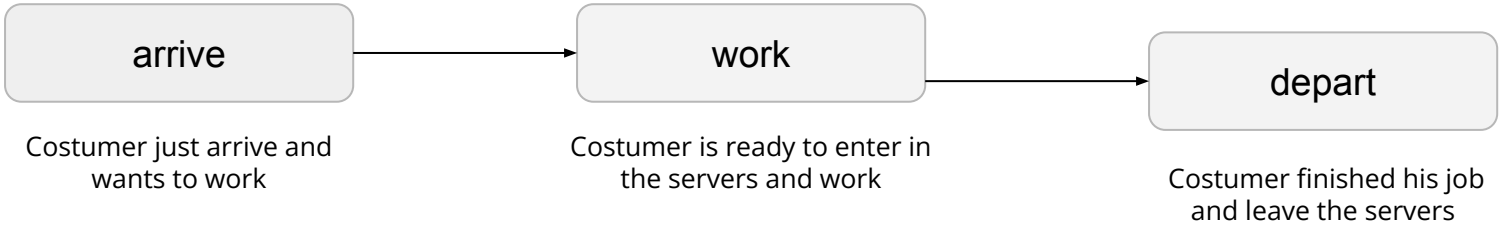
Artur Romão, 98470
João Reis, 98474

PROBLEM 1
service facility

Implemented Strategy

- **Class *Costumer*** →creating several different clients and storing all the information associated with costumers, such as, type, event type, server type, arrival time,...

- A customer has to go through **3 events**.



- The program has 4 relevant functions: ***timing()***, ***arrive()***, ***work()*** and ***depart()***

Implemented Strategy

- **function *timing()*** →Identify which is the next costumer to hold an event.
  - **How?** select, among the active costumers, the one that has the event **closest** to the **current simulation time**.
  - and check if there are any costumer on the waiting queues with the **necessary conditions** to work

Note: It should be noted that we are giving preference to type 2 clients over type 1 clients, when it comes to choosing which of the clients on the waiting queue to choose from.

- **function *arrive()*** →Check if there are servers available for the current costumer to work.
  - **if no,** it goes to the waiting queue.
  - **regardless if yes or no**, the event type of the costumer change to “work”, because the costumer is ready to start working.

Implemented Strategy

- **function *work()*** →Decrement the variables that indicate the availability of the servers and set the server depart time of the current_costumer.
  - For type 1 costumers, preference is given to **type A servers**.

- **function *depart()*** → Increment the variables that control the availability of the servers, since the client has just done its job.

Results obtained for Question 1.1

- using a seed, 98474, to compare results

```
---------------/ statistics \---------------
>> average delay in:
        # queue for type 1 costumer = 0.32
        # queue for type 2 costumer = 0.80

>> average number of costumers in queue:
        # type 1 costumer = 0.06
        # type 2 costumer = 0.05

>> time that servers A spend on:
        # type 1 costumer = 587.20
        # type 2 costumer = 115.94

>> time that server B spends on:
        # type 1 costumer = 62.98
        # type 2 costumer = 115.94
-----------------------------------------------
```

Results obtained for Question 1.2

- using the same seed, 98474

One more server of type A

```
--------------/ statistics \---------------
>> average delay in:
        # queue for type 1 costumer = 0.18
        # queue for type 2 costumer = 0.44

>> average number of costumers in queue:
        # type 1 costumer = 0.01
        # type 2 costumer = 0.03

>> time that servers A spend on:
        # type 1 costumer = 647.34
        # type 2 costumer = 114.15

>> time that server B spends on:
        # type 1 costumer = 12.39
        # type 2 costumer = 114.15
-----------------------------------------
```

One more server of type B

```
--------------/ statistics \---------------
>> average delay in:
        # queue for type 1 costumer = 0.32
        # queue for type 2 costumer = 0.57

>> average number of costumers in queue:
        # type 1 costumer = 0.01
        # type 2 costumer = 0.04

>> time that servers A spend on:
        # type 1 costumer = 576.12
        # type 2 costumer = 115.83

>> time that server B spends on:
        # type 1 costumer = 85.91
        # type 2 costumer = 115.83
-----------------------------------------
```

>> **One more server of type A** is better in reducing the maximum of the **average delay in queue** for both types of costumers.

PROBLEM 2
evolution of predator-prey populations

Forward Euler

For the first program, we were asked to use the *Forward Euler* method:

$$\frac{dx(t)}{dt} = \alpha.x(t) - \beta.x(t).y(t)$$
$$\frac{dy(t)}{dt} = \delta.x(t).y(t) - \gamma.y(t)$$

Differential equations that define the *Lotka-Volterra* model

```
def update():
    global x, y, a, b, d, g, dt
    x_new = x + (a * x - b * x * y) * dt
    y_new = y + (d * x * y - g * y) * dt
    x, y  = x_new, y_new
```

Update function for the *Forward Euler* method

Runge-Kutta

For the second program, we were asked to use the *Runge-Kutta* method:

$$y(x+h) = y(x) + \frac{1}{6}(F_1 + 2F_2 + 2F_3 + F_4)$$

where

$$F_1 = hf(x, y)$$

$$F_2 = hf\left(x + \frac{h}{2}, y + \frac{F_1}{2}\right)$$

$$F_3 = hf\left(x + \frac{h}{2}, y + \frac{F_2}{2}\right)$$

$$F_4 = hf(x + h, y + F_3)$$

Explanation of the *Runge-Kutta - RK4* method

```
def f_x(x, y):
    global a, b
    return a * x - b * x * y

def f_y(x, y):
    global d, g
    return d * x * y - g * y
```

Auxiliary methods containing the differential equations to help in the readability of the *RK4* method

```
def update():
    global x, y, a, b, d, g, dt

    k1_x = dt * f_x(x, y)
    k1_y = dt * f_y(x, y)

    k2_x = dt * f_x(x + dt / 2, y + k1_y / 2)
    k2_y = dt * f_y(x + dt / 2, y + k1_y / 2)

    k3_x = dt * f_x(x + dt / 2, y + k2_y / 2)
    k3_y = dt * f_y(x + dt / 2, y + k2_y / 2)

    k4_x = dt * f_x(x + dt, y + k3_y)
    k4_y = dt * f_y(x + dt, y + k3_y)

    x    = x + (k1_x + 2 * k2_x + 2 * k3_x + k4_x) / 6
    y    = y + (k1_y + 2 * k2_y + 2 * k3_y + k4_y) / 6
```

Update function for the *Runge-Kutta* method

Initial values and parameters handling

In order to allow the specification of the initial values and parameters through the *Command Line* and via text file, the *argparse* library was used:

```python
# Initialize the parser and parse the arguments from CLI
parser = argparse.ArgumentParser()

parser.add_argument("--x0", type=int, default=10, help="Initial population of prey")
parser.add_argument("--y0", type=int, default=10, help="Initial population of predators")
parser.add_argument("--a", type=float, default=0.1, help="Parameter alpha")
parser.add_argument("--b", type=float, default=0.02, help="Parameter beta")
parser.add_argument("--d", type=float, default=0.02, help="Parameter delta")
parser.add_argument("--g", type=float, default=0.4, help="Parameter gamma")
parser.add_argument("--dt", type=float, default=0.1, help="Parameter delta_t")
parser.add_argument("--tf", type=int, default=5000, help="Parameter tfinal")
parser.add_argument("--f", type=str, default=None, help="Path to the input file")
```

Definition of the arguments that the program accepts - all of the initial values and parameters and, for last, the text file argument

```
x0 = 10
y0 = 10
a = 0.1
b = 0.02
d = 0.02
g = 0.4
dt = 0.1
tf = 5000
```

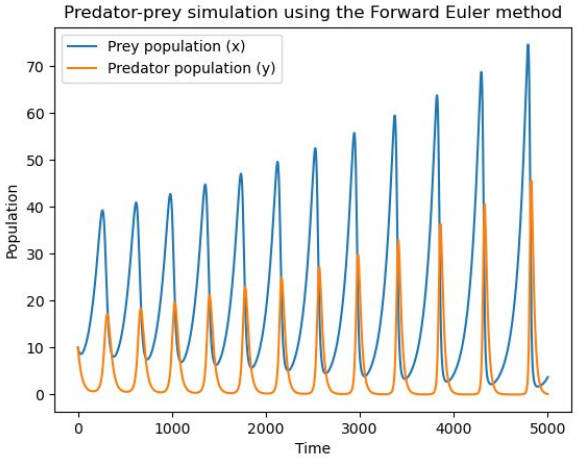
Structure of the text file, which is mandatory - every parameter defined, one parameter per line and in this specific order

```python
args = parser.parse_args()

params = vars(args) # Turn the parsed arguments into a dictionary

if params["f"]: # If the user specified a file, read the parameters from the file
    try:
        with open(params["f"], "r") as f:
            params["x0"] = int(f.readline().split("=")[1])
            params["y0"] = int(f.readline().split("=")[1])
            params["a"]  = float(f.readline().split("=")[1])
            params["b"]  = float(f.readline().split("=")[1])
            params["d"]  = float(f.readline().split("=")[1])
            params["g"]  = float(f.readline().split("=")[1])
            params["dt"] = float(f.readline().split("=")[1])
            params["tf"] = int(f.readline().split("=")[1])
    except FileNotFoundError:
        print("The file you specified does not exist.")
        exit(1)
    f.close()
```

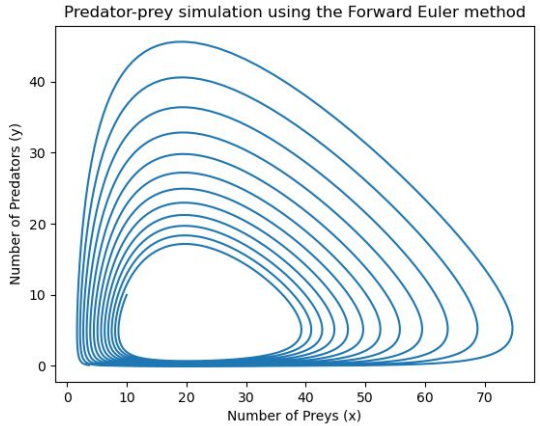
Logic behind the parameter specification - if the user passes the argument "--f" with a valid path to file, the parameters of the text file prevail and the *CLI* ones are ignored

Results obtained for *Forward Euler*

These were the results obtained for the *Forward Euler* method:



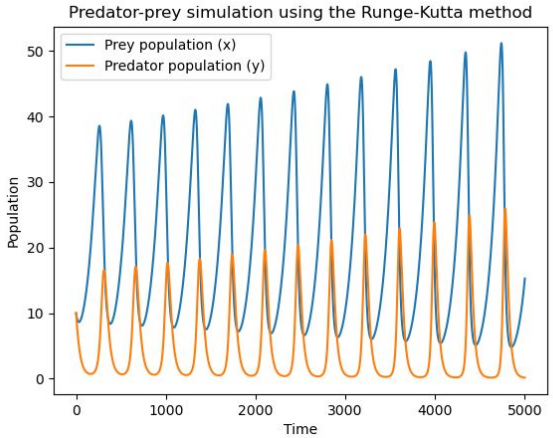
Population evolution over time



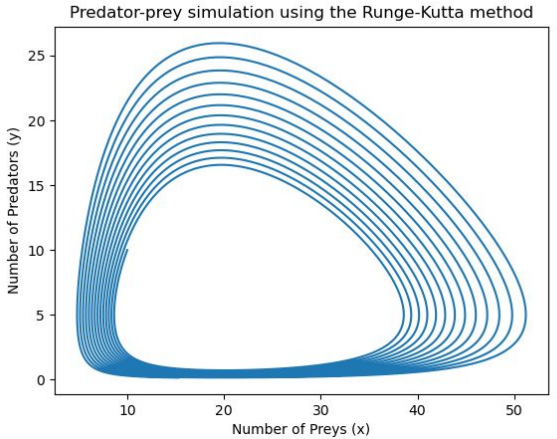
Number of predators in order of the number of preys

Results obtained for *Runge-Kutta*

These were the results obtained for the *Runge-Kutta* method:



Population evolution over time



Number of predators in order of the number of preys