Simulação e Otimização

Mini-Project 1

92952: Rui Fernandes
98430 Paulo Pereira

Topics

Exercise 1

Objective

Simulate a facility with two types of servers and two types of customers.

Customers that can't be served must be added to different FIFO queues (according to type) and type 2 customers must be prioritized over type 1.

Customers have different serving times and probability of arriving.

Must answer the question, which is better for the system, one more type A server or one more type B server?

Solution

We decided to use the simpy library over the method we used in class.

Developed 4 functions:

- Customer arrival
- Queue Manager
- Type 1 and Type 2 life cycle

Usage of simpy

Simpy is able to simulate and manage the servers and queues. Using its own classes like **Resource** and **Store.**

The way this library works is simply by starting an environment and initializing the needed classes.

It provides methods like **request** that request the server if possible. We could've used this feature to completely manage the servers but we decided it was more in line with the assignment to still do some of the management manually.

Developed functions

A deeper look into each function

Customer arrival

```
def customer_arrival(env, servers_type_A, servers_type_B):
    global NUM_TYPE1_CUSTOMERS, NUM_TYPE2_CUSTOMERS
    global count
    while True:

        # Determine the type of customer
        if random.uniform(0, 1) < 0.8:
            customer_type = 1
            NUM_TYPE1_CUSTOMERS += 1
        else:
            customer_type = 2
            NUM_TYPE2_CUSTOMERS += 1
        count+=1

        # Add the customer to the appropriate queue
        if customer_type == 1:
            arrival_time = env.now
            print("[A] type1 customer arrived at time %f" % arrival_time)
            env.process(type1_customer(queue_type1, env, servers_type_A, servers_type_B,server_usage,waiting_times, arrival_time,waiting_times_type1,None))
            yield env.timeout(random.expovariate(ARRIVAL_RATE))
        else:
            arrival_time = env.now
            print("[A] type2 customer arrived at time %f" % arrival_time)

            env.process(type2_customer(queue_type2, env, servers_type_A, servers_type_B,server_usage,waiting_times, arrival_time,waiting_times_type2,None))
            yield env.timeout(random.expovariate(ARRIVAL_RATE))
```

Developed functions

A deeper look into each function

Queue Manager

```python
def queue_manager(env,queue_type1, queue_type2):
    server=None
    server_A=None
    server_B=None

    while True:

        # Check if there are customers in the queue 2 because it's the priority queue
        if len(queue_type2.items) > 0:

            # Check if there are available servers type A
            for i in range(0, NUM_TYPEA_SERVERS):
                if not servers_type_A.requested[i]:
                    server_A = i
                    break

            # Check if there are available servers type B
            for i in range(0, NUM_TYPEB_SERVERS):
                if not servers_type_B.requested[i]:
                    server_B = NUM_TYPEA_SERVERS + i
                    break

            # If there are available servers type A and B
            if server_A is not None and server_B is not None:

                # Get the customer from the queue
                customer = queue_type2.get()

                # Request a server to serve the customer
                env.process(type2_customer(queue_type2, env, servers_type_A, servers_type_B,server_usage,waiting_times, env.now,waiting_times_type2,customer.value))
                print("[Q] customer %s was released from queue 2" % customer.value)
```

```python
# Check if there are customers in the queue 1
if len(queue_type1.items) > 0:

    # Check if there are available servers
    for i in range(0, NUM_TYPEA_SERVERS):
        if not servers_type_A.requested[i]:
            server = i
            break

    # If no Type A server is available, check for available Type B servers
    if server is None:
        for i in range(0, NUM_TYPEB_SERVERS):
            if not servers_type_B.requested[i]:
                server = NUM_TYPEA_SERVERS + i
                break

    # If there are available servers
    if server is not None:
        # Get the customer from the queue
        customer = queue_type1.get()

        # Request a server to serve the customer
        env.process(type1_customer(queue_type1, env, servers_type_A, servers_type_B,server_usage,waiting_times,
         env.now,waiting_times_type1,customer.value))
        print("[Q] Customer %s was released from queue 1" % customer.value)

yield env.timeout(0.1)
```

Developed functions

A deeper look into each function
Type 1 customer life-cycle

1. Check for type A server
2. Check for type B server (if no A server)
3. If no servers -> put into queue
4. If a server is available serve
5. Release server

```python
# Define the service process for type 1 customers
def type1_customer(queue, env, servers_type_A, servers_type_B, server_usage,waiting_times, arrival_time,waiting_times_type1,customer):

    #Count the number of customers in queue 1 at any given time
    global count_type1_queue
    count_type1_queue+=len(queue.items)

    server = None

    #just to make sure we request the right server type this flag will be 0 if we request type A server and 1 if we request type B server
    flag=0

   #create customer
    if customer == None:
        customer = {'type': 'Type 1', 'arrival_time': arrival_time}

    #System status
    print("[S] Current systems status: \n \t Servers Type A -> %s \n \t Servers Type B -> " % servers_type_A.requested, servers_type_B.requested)
    print(f"[S] Number of customers in queue 1 : {len(queue.items)}")

    # First, check for available Type A servers
    for i in range(0, NUM_TYPEA_SERVERS):
        if not servers_type_A.requested[i]:
            server = i
            break

    # If no Type A server is available, check for available Type B servers
    if server is None:
        for i in range(0, NUM_TYPEB_SERVERS):
            if not servers_type_B.requested[i]:
                server = NUM_TYPEA_SERVERS + i
                flag=1
                break

    # If no server is available, server remains None and the customer joins the queue
    if server is None:
        queue.put(customer)
        print("[Q] type1 customer joined the queue at time %f" % arrival_time)
        yield env.timeout(0)
```

Developed functions

A deeper look into each function

Type 1 customer life-cycle

```python
# Wait for service
with servers[flag].request() as req:
    yield req

    # Request the server
    if server <= NUM_TYPEA_SERVERS-1:
        servers_type_A.requested[i] = True
        print("[W] Customer of type 1 that arrived at %f is being served by server A%d at time %f" % (customer['arrival_time'],server+1, env.now))

    else:
        servers_type_B.requested[i] = True
        print("[W] Customer of type 1 that arrived at %f is being served by server B%d at time %f" % (customer['arrival_time'],server - NUM_TYPEA_SERVERS+1, env.now))

    # Record server usage and waiting time
    server_name = 'A' + str(server+1) if server <= NUM_TYPEA_SERVERS-1 else 'B' + str(server - NUM_TYPEA_SERVERS+1)
    waiting_times.append(env.now - customer['arrival_time'])
    waiting_times_type1.append(env.now - customer['arrival_time'])
    serving_time=random.expovariate(SERVICE_RATE_TYPE1)
    server_usage[server_name][0] += serving_time

    yield env.timeout(serving_time)

    # Release the server
    if server <= NUM_TYPEA_SERVERS-1:
        print("[R] Released server A%d at time %f" % (server+1, env.now))
        servers_type_A.requested[server] = False
    else:
        print("[R] Released server B%d at time %f" % (server - NUM_TYPEA_SERVERS+1, env.now))
        servers_type_B.requested[server - NUM_TYPEA_SERVERS] = False
```

Developed functions

A deeper look into each function

Type 2 customer life-cycle

1. Check for type A and B servers
2. If missing one or more servers
   - put into queue
3. If both servers are available serve
4. Release servers

```python
# Define the service process for type 2 customers
def type2_customer(queue, env, servers_type_A, servers_type_B, server_usage,waiting_times, arrival_time,waiting_times_type2,customer):

    #Count the number of customers in queue 2 at any given time
    global count_type2_queue
    count_type2_queue+=len(queue.items)

    server_A = None
    server_B = None

    #create customer
    if customer == None:
        customer = {'type': 'Type 2', 'arrival_time': arrival_time}

    #System status
    print("[S] Current systems status: \n \t Servers Type A -> %s \n \t Servers Type B -> " % servers_type_A.requested, servers_type_B.requested)
    print(f"[S] Number of customers in queue 2: {len(queue.items)}")

    # Request available Type A server
    for i in range(0, NUM_TYPEA_SERVERS):
        if not servers_type_A.requested[i]:
            server_A = i
            break

    # Request available Type B server
    for i in range(0, NUM_TYPEB_SERVERS):
        if not servers_type_B.requested[i]:
            server_B = NUM_TYPEA_SERVERS + i
            break

    # If either Type A or Type B server is not available, join queue
    if server_A is None or server_B is None:
        queue.put(customer)
        print("[Q] type2 customer joined the queue at time %f" % arrival_time)
        yield env.timeout(0)
```

Developed functions

A deeper look into each function

Type 2 customer life-cycle

```
else:

    # Wait for service
    with servers[flag].request() as req:
        yield req

        # Request the server
        if server <= NUM_TYPEA_SERVERS-1:
            servers_type_A.requested[i] = True
            print("[W] Customer of type 1 that arrived at %f is being served by server A%d at time %f" % (customer['arrival_time'],server+1, env.now))

        else:
            servers_type_B.requested[i] = True
            print("[W] Customer of type 1 that arrived at %f is being served by server B%d at time %f" % (customer['arrival_time'],server - NUM_TYPEA_SERVERS+1, env.now))

        # Record server usage and waiting time
        server_name = 'A' + str(server+1) if server <= NUM_TYPEA_SERVERS-1 else 'B' + str(server - NUM_TYPEA_SERVERS+1)
        waiting_times.append(env.now - customer['arrival_time'])
        waiting_times_type1.append(env.now - customer['arrival_time'])
        serving_time=random.expovariate(SERVICE_RATE_TYPE1)
        server_usage[server_name][0] += serving_time

        yield env.timeout(serving_time)

        # Release the server
        if server <= NUM_TYPEA_SERVERS-1:
            print("[R] Released server A%d at time %f" % (server+1, env.now))
            servers_type_A.requested[server] = False
        else:
            print("[R] Released server B%d at time %f" % (server - NUM_TYPEA_SERVERS+1, env.now))
            servers_type_B.requested[server - NUM_TYPEA_SERVERS] = False
```

Conclusion

Comparing all 3 scenarios

Adding one Type A is better

```
----- Results -----

Simulation with 3 servers and 1019 customers
Served 800 Type 1 customers
Served 219 Type 2 customers
Queue Type 1 size at end of simulation: 0
Queue Type 2 size at end of simulation: 0
Average time spent in the system per Type 1 customer: 0.829791296569312
Average time spent in the system per Type 2 customer: 0.5966216418530347
Average delay in queue for any customer: 0.08413702198106095
Average delay in queue for Type 1 customers: 0.036027312176790824
Average delay in queue for Type 2 customers: 0.25988025414277827
Expected time average number in queue for type 1 customers: 0.064
Expected time average number in queue for type 2 customers: 0.185
Proportion of time server A1 was in use by type 1 customers: 37.02
Proportion of time server A1 was in use by type 2 customers: 9.90
Proportion of time server A2 was in use by type 1 customers: 23.08
Proportion of time server A2 was in use by type 2 customers: 3.17
Proportion of time server B1 was in use by type 1 customers: 6.28
Proportion of time server B1 was in use by type 2 customers: 13.07
Maximum of the average delay in queue for both types of customers: 0.25988025414277827
```

Normal case
2 type A
1 type B

```
----- Results -----

Simulation with 4 servers and 1014 customers
Served 792 Type 1 customers
Served 222 Type 2 customers
Queue Type 1 size at end of simulation: 0
Queue Type 2 size at end of simulation: 0
Average time spent in the system per Type 1 customer: 0.8248258222718009
Average time spent in the system per Type 2 customer: 0.5948846840074187
Average delay in queue for any customer: 0.019068323806531562
Average delay in queue for Type 1 customers: 0.0057718888333388235
Average delay in queue for Type 2 customers: 0.06650425398116512
Expected time average number in queue for type 1 customers: 0.003
Expected time average number in queue for type 2 customers: 0.018
Proportion of time server A1 was in use by type 1 customers: 35.63
Proportion of time server A1 was in use by type 2 customers: 9.36
Proportion of time server A2 was in use by type 1 customers: 20.59
Proportion of time server A2 was in use by type 2 customers: 2.84
Proportion of time server B1 was in use by type 1 customers: 1.09
Proportion of time server B1 was in use by type 2 customers: 13.21
Proportion of time server A3 was in use by type 1 customers: 8.02
Proportion of time server A3 was in use by type 2 customers: 1.01
Maximum of the average delay in queue for both types of customers: 0.06650425398116512
```

Using 3 servers of the type A and one of type B

```
----- Results -----

Simulation with 4 servers and 1012 customers
Served 795 Type 1 customers
Served 217 Type 2 customers
Queue Type 1 size at end of simulation: 0
Queue Type 2 size at end of simulation: 0
Average time spent in the system per Type 1 customer: 0.8261351815726923
Average time spent in the system per Type 2 customer: 0.5971135509856436
Average delay in queue for any customer: 0.07856483252339119
Average delay in queue for Type 1 customers: 0.03389862070455592
Average delay in queue for Type 2 customers: 0.2422037191407831
Expected time average number in queue for type 1 customers: 0.059
Expected time average number in queue for type 2 customers: 0.131
Proportion of time server A1 was in use by type 1 customers: 37.00
Proportion of time server A1 was in use by type 2 customers: 9.79
Proportion of time server A2 was in use by type 1 customers: 22.33
Proportion of time server A2 was in use by type 2 customers: 3.17
Proportion of time server B1 was in use by type 1 customers: 3.35
Proportion of time server B1 was in use by type 2 customers: 6.56
Proportion of time server B2 was in use by type 1 customers: 2.99
Proportion of time server B2 was in use by type 2 customers: 6.40
Maximum of the average delay in queue for both types of customers: 0.2422037191407831
```

Using 2 servers of the type A and 2 of type B

Exercise 2

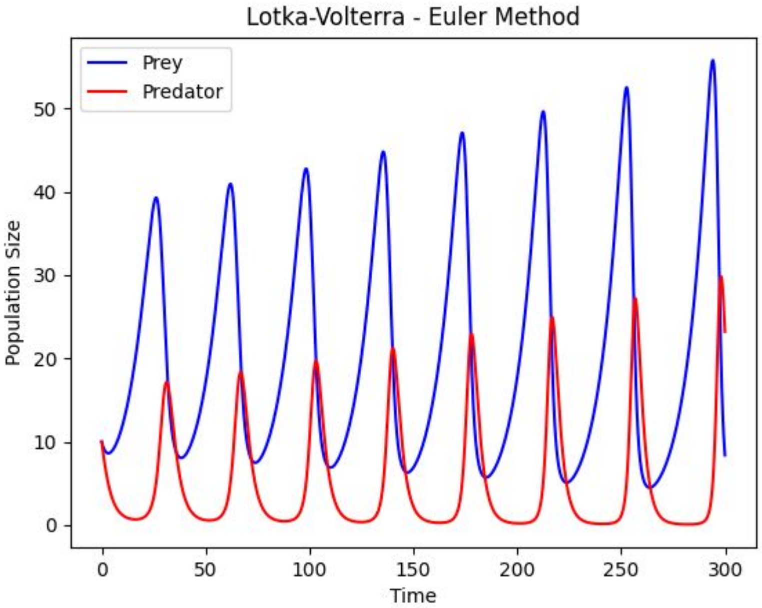
Objective

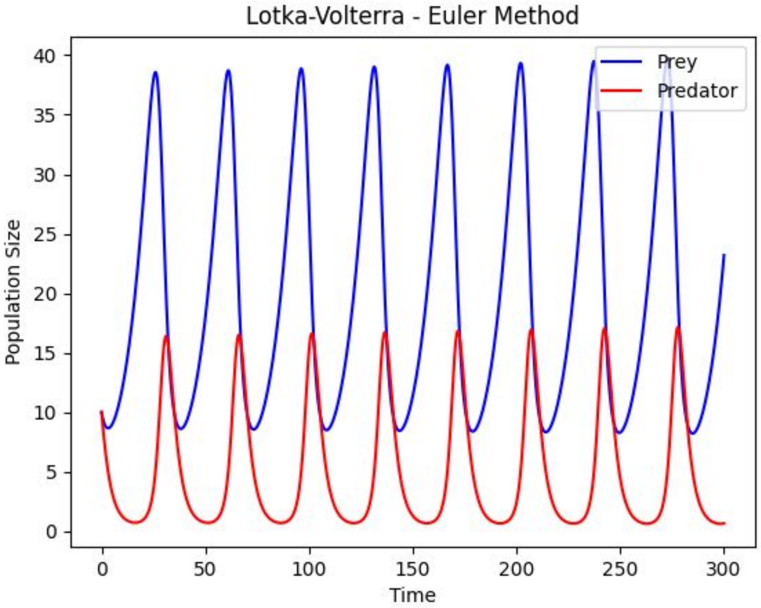
$$\frac{dx(t)}{dt} = \alpha.x(t) - \beta.x(t).y(t)$$
$$\frac{dy(t)}{dt} = \delta.x(t).y(t) - \gamma.y(t)$$

Lotka-Volterra model: a pair of first-order differential equations, used to describe the evolution of the population of two species, one a predator and one a prey in a biological system in which only they interact.

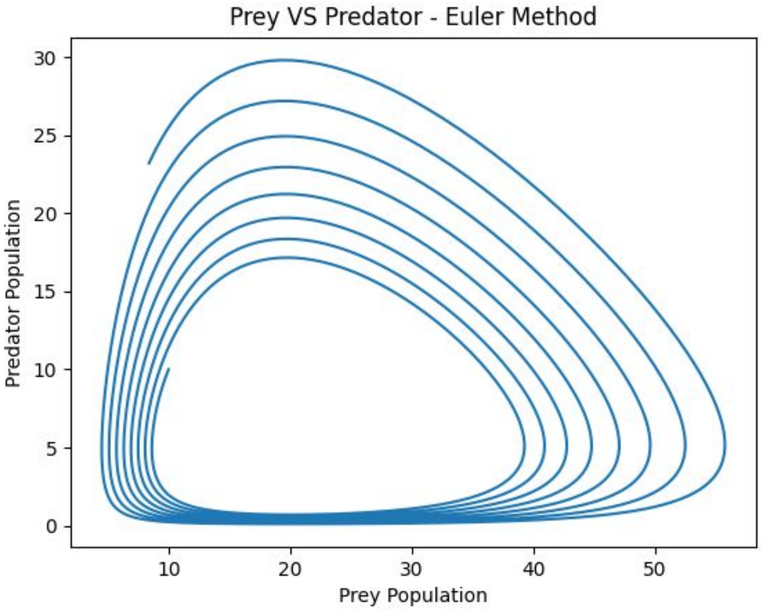
To write programs that simulate the evolution of this model, getting approximate results with the Euler method, and the Runge-Kutta method

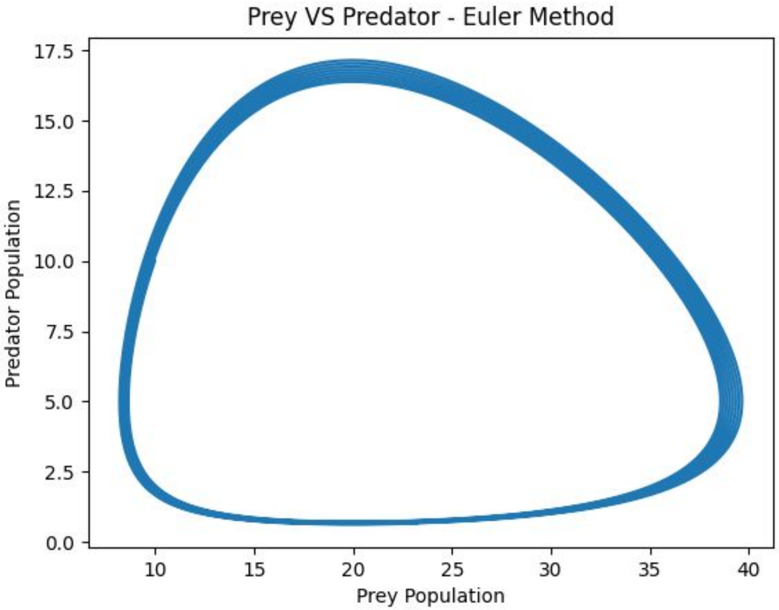
Euler Method



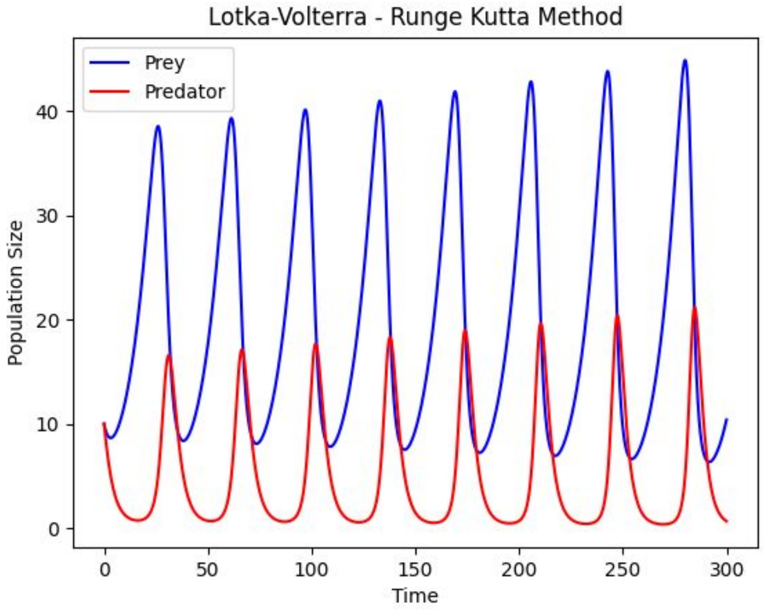


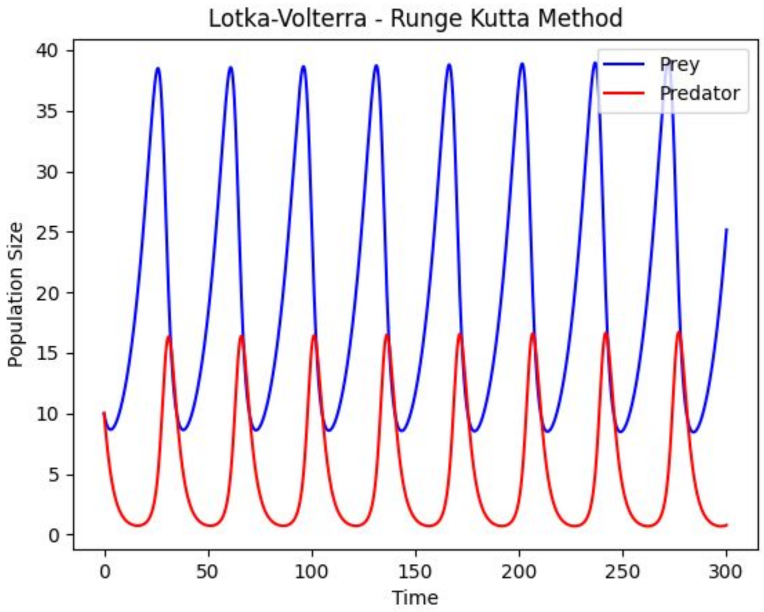
Euler Method





Runge-Kutta Method





Runge-Kutta Method