universidade de aveiro

DEPARTAMENTO DE ELECTRÓNICA, TELECOMUNICAÇÕES E INFORMÁTICA

# Simulação e Otimização (2024/2025)

# Simulation Mini-Projects

Daniel Pedrinho Nº107378

Sara Almeida Nº108796

**13th of May 2025**

# Contents

**1 Introduction** **3**

**2 Exercise 1.1** **3**

2.1 Approach . . . 3

2.2 Results . . . 4

**3 Exercise 1.2** **5**

3.1 Approach . . . 5

3.2 Results . . . 6

**4 Exercise 2.1** **7**

4.1 Approach . . . 7

4.2 Results . . . 7

**5 Exercise 2.2** **8**

5.1 Approach . . . 8

5.2 Results . . . 8

**6 Exercise 2.3** **9**

6.1 Approach . . . 9

6.2 Results . . . 10

6.3 Edge-Cases . . . 11

# 1 Introduction

This project was developed for the *Simulação e Otimização* course of the University of Aveiro and the objective of this assignment is to familiarize ourselves with the tools commonly used in Simulation and Optimization environments.

We were challenged to develop solutions for two simulation problems. The first models a bus maintenance facility to analyze queue delays, resource utilization, and system capacity. The second simulates the motion of a projectile under air resistance, using both Euler and Runge-Kutta4 methods.

# 2 Exercise 1.1

## 2.1 Approach

We implemented a discrete-event simulation to solve this exercise. After registering the initial conditions as indicated in the provided PDF, the simulation begins by entering the inspection station. Here, the **wait** time is registered and the inspection is made, also registering how much time the inspection took.

After that, if a bus is deemed to need repair, it is placed in a waiting queue, waiting for a free repair station. If one is available, the time spent in the queue is recorded, and the bus proceeds to complete the repair. Once completed, the repair time is registered and the repair station is freed.

Two auxiliary functions were implemented to facilitate the simulation. **bus_arrival** makes sure that the bus arrival time follows the indicated constraints, while **monitor_queues** is responsible for periodically recording the queue length. The chosen time was **every 30 minutes**. This choice was arbitrarily made.

## 2.2 Results

With the approach provided above, these were the results obtained, as requested in the PDF, with a **random** seed of 42:

| **Metric** | **Measure** |
|---|---|
| Avg Delay Inspection Queue | 08:03 minutes |
| Avg Delay Repair Queue | 05:29 minutes |
| Avg Inspection Queue Length | 0.06 buses |
| Avg Repair Queue Length | 0.01 buses |
| Usage Inspection Station | 29,85% |
| Usage Repair Stations | 23,48% |

Table 1: Results obtained with random.seed(42)

With usage percentage meaning percentage of total simulation time and queue length being amount of buses.

# 3 Exercise 1.2

## 3.1 Approach

In this exercise, we used the previously developed simulator, for the same 160 hours of simulation runtime, to determine the **maximum bus arrival rate (minimum mean interarrival time)** that the maintenance facility could handle without leading to system overload.

To calculate that value, we extended the simulation from *Exercise 1.1* but this time we gradually decrease the mean interarrival time of buses, starting from **2 hours** down to **half an hour** with 0.1 hour steps.
After each simulation run, we save and print important indicators: *average inspection and repair queues delays*, *average inspection and repair queues lengths* and *both stations utilization.*
Then we defined the limit values for some of those indicators that we consider being **critical thresholds** to indicate system overload:

- Inspection utilization exceeds 90% **or**
- Repair utilization exceeds 85% **or**
- Average inspection queue length exceeds 6 buses **or**
- Average inspection queue delay exceeds 1 hour **or**
- A sudden increase in inspection queue delay over the last step (more than 0.5 hours).

Finally, when any of these thresholds is reached, we record the corresponding interarrival time as the minimum sustainable mean interarrival time and that is how we estimate the **maximum arrival rate**.

## 3.2 Results

Using the approach explained above, the simulation determined that the system can handle a minimum mean interarrival time of **47:59 minutes**, which corresponds to a maximum arrival rate of approximately **1.25 buses per hour** (about **30 buses per day**).

The last interarrival time before exceeding one of the critical thresholds was **0.80 hours**, where the system showed clear signs of overload:

| **Metric** | **Measure** |
|---|---|
| Avg Delay Inspection Queue | 01:36:25 (hh:mm:ss) |
| Avg Delay Repair Queue | 01:00:02 (hh:mm:ss) |
| Avg Inspection Queue Length | 2.12 buses |
| Avg Repair Queue Length | 0.39 buses |
| Usage Inspection Station | 80.36% |
| Usage Repair Stations | 59.55% |

Table 2: Critical Results for Edge Interarrival time

This satisfies the overload condition due to the average inspection queue delay exceeding **1.0 hour**. Thus, the corresponding interarrival time (**0.80 hours** or **47:59 minutes**) was selected as the **minimum sustainable mean interarrival time**, leading to a maximum arrival rate of approximately **1.25 buses per hour**.

# 4 Exercise 2.1

## 4.1 Approach

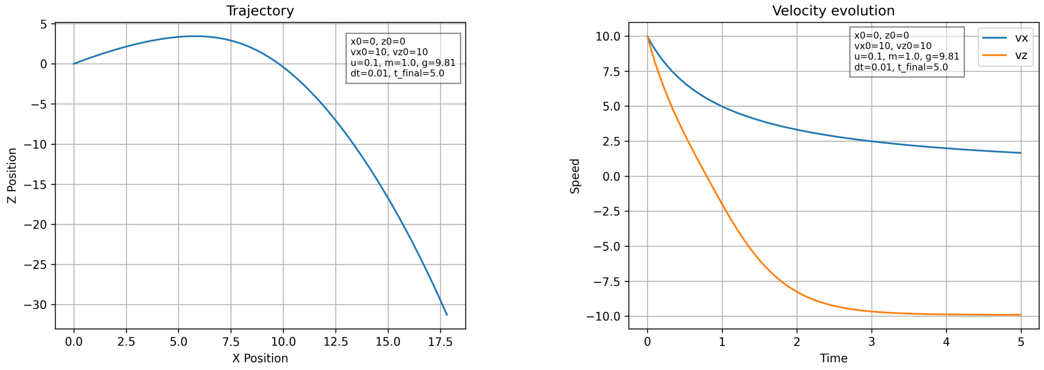
For this exercise, we implemented a numerical simulation to model the trajectory. The initial conditions, as stated in the original PDF, are read from the command line as parameters or from a **.json** file.

The simulation begins by initializing all of the state variables and then progresses in fixed time intervals, updating both the projectile's velocity and position, using Newton's Second Law of Motion.

At each step, horizontal and vertical accelerations are calculated taking into account both gravity and air resistance. These accelerations are then used to update the velocities and position for the next step in time.

## 4.2 Results

With the values present in the **README.md**, under **Usage**, the following results were obtained:



(a) Trajectory

(b) Velocity

Figure 1: Trajectory and velocity side by side - Euler method

# 5 Exercise 2.2

## 5.1 Approach

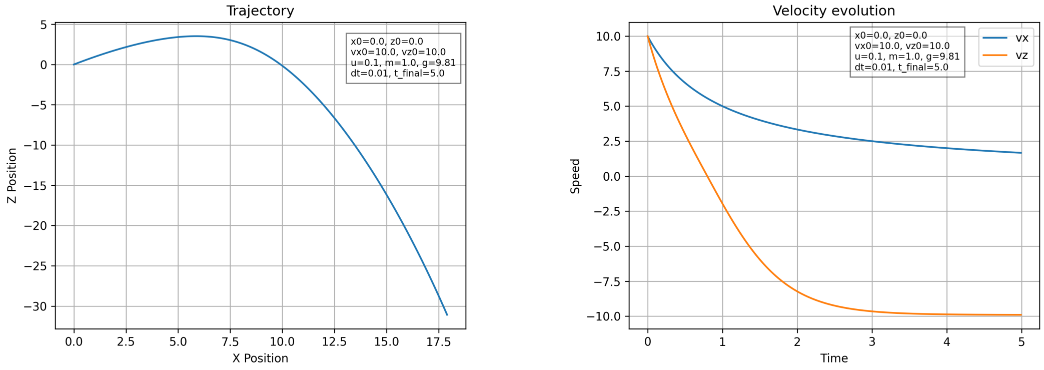
For this exercise, we implemented the Runge-Kutta method of 4th order (RK4), explained in the class's slides, to simulate the trajectory of a projectile under the influence of gravity and air resistance. Again, the initial conditions are read from the command line as parameters or from a **.json** file.

More complex than the last one, the RK4 method uses a weighted average of four increment estimates (k1 through k4) to compute the next value of the state variables *x, z, vx, vz.* At each time step, accelerations due to gravity and drag are recalculated with intermediate velocities.

To help with these incremental steps, we create the function *axay* to calculate the accelerations given the current velocity values. This function encapsulates the drag and gravity forces applied to the projectile and is reused at each intermediate RK4 step.

## 5.2 Results

With the values present in the **README.md**, under **Usage**, the following results were obtained:



(a) Trajectory

(b) Velocity

Figure 2: Trajectory and velocity side by side - Runge-Kutta 4th order

# 6 Exercise 2.3

## 6.1 Approach

Finally, in this exercise, the goal was to compare the precision of both previous methods. For this we compared our solutions applied to the same projectile moving problem, which means they both simulate the trajectory influenced by gravity and air resistance fo the same input parameters. Once again, the initial conditions are read from the command line as parameters or from a **.json** file.

To compare the results, we printed and compared key metrics:

- Final horizontal and vertical positions
- Final horizontal and vertical velocities
- Maximum height reached

Additionally, the trajectory and velocity profiles of both methods were plotted together for a more visual comparison.

Theoretically, it is expected that the Euler method is simpler and computationally faster but the RK4 is much more precise and suitable for application where accuracy is crucial. This last method improves accuracy by sampling the derivative at multiple points within each time interval and computing a weighted average. It has a total accumulated error of order $\mathcal{O}(h^4)$, making it significantly more reliable for physical simulations with forces that vary rapidly, like air resistance applicable in this exercise case.

## 6.2 Results

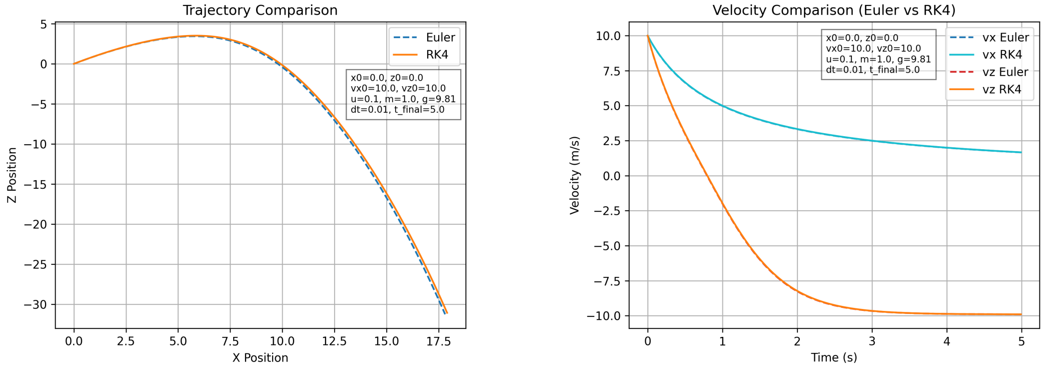
In general, the results show that the Runge-Kutta method provides a more stable and precise trajectory than the Euler method. In particular, the final positions and velocities calculated using RK4 are smoother and more physically consistent, especially over longer time intervals or when air resistance significantly affects the motion. Numerically, the final values obtained were:

| Metric | Euler | RK4 |
|---|---|---|
| **Final x position** | 17.81 m | 17.90 m |
| **Final z position** | -31.27 m | -31.08 m |
| **Final vx** | 1.66 m/s | 1.67 m/s |
| **Final vz** | -9.90 m/s | -9.90 m/s |
| **Max height** | 3.44 m | 3.51 m |

Table 3: Comparison of final values obtained

While both methods follow similar trends, RK4 consistently remains closer to the expected path and exhibits less numerical dissipation, particularly near the trajectory apex and final descent. Velocity profiles also confirm that RK4 better preserves the expected decay pattern under drag.

Below are presented the comparative results, with the values present in the **README.md**, under **Usage**:



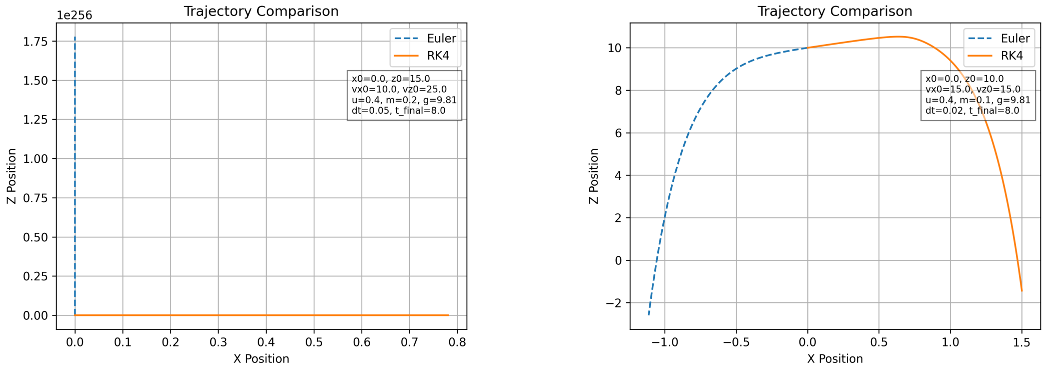
(a) Trajectory

(b) Velocity

Figure 3: Trajectory and velocity side by side - Euler VS Runge-Kutta 4th order

### 6.3 Edge-Cases

It is important to note that the Euler method shows instability when applied to certain edge-case scenarios, as illustrated in the graphics below. Under these conditions, the method produces diverging or physically implausible results.



(a) Trajectory

(b) Velocity

Figure 4: Instability in Euler vs RK4 trajectory analysis

In contrast, the Runge-Kutta 4th Order (RK4) method demonstrates greater stability and accuracy across the same cases. These observations support the conclusion that RK4 is a more robust and dependable algorithm overall, particularly when dealing with stiff or sensitive systems where precision is critical.

# References

[1] A. M. Law, *Simulation Modeling & Analysis*, 5th. McGraw-Hill, 2014, ISBN: 9780073401324.

[2] GeeksforGeeks. “Runge-kutta 4th order method to solve differential equation”. Accessed: 2025-05-10. (2020), [Online]. Available: `https://www.geeksforgeeks.org/runge-kutta-4th-order-method-solve-differential-equation/`.