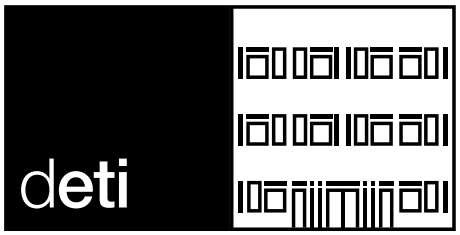
Simulation Mini-Projects

Simulation and Optimization - Project 1



universidade de aveiro
departamento de electrónica,
telecomunicações e informática

Alexandre Pedro Ribeiro
NMec: 108122

Miguel da Silva Pinto
NMec: 107449

May 12, 2025

Contents

1 Introduction

Computational simulation is a fundamental tool for understanding and predicting the behavior of complex systems across scientific and engineering domains. This report presents two distinct simulation mini-projects that showcase different numerical approaches to solving practical problems.

The first project applies discrete-event simulation (DES) [5] to model a bus maintenance facility, demonstrating how this approach can effectively capture queue dynamics, resource utilization, and system capacity in service-oriented processes. DES is particularly valuable for systems where state changes occur at discrete points in time triggered by specific events.

The second project focuses on numerical integration methods for solving ordinary differential equations, specifically applied to projectile motion with air resistance. By implementing both the Forward Euler method [2] and the 4th-order Runge-Kutta method (RK4) [3], we examine the accuracy, stability, and convergence properties of these approaches for nonlinear dynamical systems.

Both projects were implemented in Python, leveraging specialized libraries where appropriate. The bus maintenance simulation utilizes SimPy [4] for discrete-event modeling, while the projectile motion simulation uses NumPy and SciPy for numerical computations and analysis [7],. All source code is available in our GitHub repository [1], with detailed documentation and configuration files to ensure reproducibility.

2 Bus Maintenance Facility Simulation

This section presents a discrete-event simulation [5] of a bus maintenance facility, analyzing queue dynamics, resource utilization, and system stability under various operational conditions. We begin with a description of the problem, followed by our implementation approach and analysis of the results.

2.1 Problem Description

City buses arrive at a maintenance facility at random intervals, averaging one every two hours. Upon arrival, each bus first goes through an inspection process that takes between 15 minutes and just over an hour. About 30% of the buses are found to need repairs, after which they proceed to one of two identical repair stations. Repair times range between just over two hours and four and a half hours. Both the inspection and repair areas operate using FIFO (first-in, first-out) queues.

The operational flow of the bus maintenance facility can be better understood by organizing the relevant details into two main categories: key system parameters and desired performance metrics.

2.1.1 System Parameters

- **Arrival Process:** Buses arrive according to a Poisson process with a mean interarrival time of 2 hours (i.e., arrival rate $\lambda = 0.5$ buses/hour).
- **Inspection Station:**
  - A single inspection station serves all arriving buses.
  - Inspection times follow a uniform distribution between 15 minutes (0.25 hours) and 1.05 hours.

- The inspection queue operates under a First-In-First-Out (FIFO) discipline.

- **Repair Station:**
  - 30% of the inspected buses require repair.
  - The repair system consists of two identical parallel servers.
  - Repair times are uniformly distributed between 2.1 and 4.5 hours.
  - Buses waiting for repair are served from a single FIFO queue.

2.1.2 Performance Metrics

- Average and maximum queue lengths at both the inspection and repair stages.
- Average waiting times (delays) for buses at each station.
- Utilization rates of the inspection and repair servers.

2.2 Implementation Approach

2.2.1 Simulation Model

The simulation model for the bus maintenance facility is based on a **discrete-event simulation (DES)** framework [5]. In this approach, the system state changes only at discrete points in time, specifically when events such as arrivals, repairs, or departures occur. The core logic revolves around processing a sequence of timestamped events stored in an event list, which drives the progression of the simulation clock.

Each bus passing through the system initiates a sequence of discrete events that model its journey through the maintenance facility:

- **Arrival Event:** A bus arrives at the facility following an exponential interarrival time distribution. Upon arrival, it joins the inspection queue. If the inspection station is idle, the bus proceeds directly to inspection; otherwise, it waits in the FIFO queue.
- **Start Repair Event:** After completing the inspection process, the bus is evaluated for repair. With a probability of 30%, it is deemed to require repair and enters the repair queue. This event is only triggered for buses identified as needing repair. If a repair server is available, the bus begins service immediately; otherwise, it waits in the queue.
- **Departure Event:** A bus departs the system under two conditions: (1) it does not require repairs (which occurs with a 70% probability), or (2) it has completed the repair process. In either case, the departure event marks the end of the bus's lifecycle in the simulation.

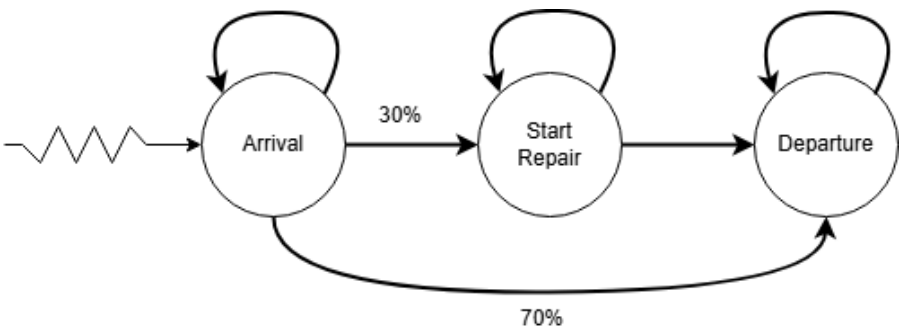


Figure 1: Event Graph - Bus Maintenance Facility

2.2.2 Code

Our codebase for this exercise is located at *src/bus* and consists of five Python files:

- **config.py**: Handles simulation configuration parameters. Provides the 'SimConfig' dataclass to store configuration values and functionality to load configuration from YAML files. The parameters control simulation time, arrival rates, service times, resource capacities, and output directories.
- **simulation.py**: Implements the core simulation model using SimPy [4]. Contains the 'BusMaintenanceSimulation' class that creates the simulation environment, resources, and processes. Handles bus arrivals, inspection, repairs, and collects statistics throughout the simulation run.
- **visualization.py**: Provides functions for visualizing simulation results through various plots and charts. Includes functions to generate time series charts of queue lengths and utilization, histograms of waiting times, and visualizations of experiment results comparing different system parameters.
- **experiments.py**: Contains the experiment framework for determining system stability and capacity limits. Implements functions to vary arrival rates and assess how the system performs under different load conditions to find the maximum stable throughput.
- **main.py**: Serves as the entry point for the simulation. Handles command-line arguments, orchestrates the running of simulations (single, multiple, or experiments), and calls the appropriate visualization functions to display results.

2.3 Results

The base scenario configuration, as described in the project description, is as follows:

Parameter	Value
Simulation Time	160.0
Mean Arrival Time	2.0
Inspection Time (Min)	0.25
Inspection Time (Max)	1.05
Repair Probability	0.3
Repair Time (Min)	2.1
Repair Time (Max)	4.5
Number of Inspectors	1
Number of Repair Stations	2
Random Seed	42
Output Directory	`results/bus`

Table 1: Base Scenario Configuration Parameters

2.3.1 Base Scenario (Single simulation)

Using the base scenario configurations and running a single simulation, we obtained the following results:

- **Buses processed**: 75
- **Average inspection queue length**: 0.09
- **Average repair queue length**: 0.00

- **Average inspection delay**: 0.20 hours
- **Average repair delay**: 0.02 hours
- **Inspection station utilization**: 0.29
- **Repair stations utilization**: 0.22

Figure 2 shows the time-series data for queue lengths and resource utilization from the single simulation run, visually confirming the system's stability under base conditions.

Figure 2: Bus Single Simulation Results

2.3.2 Base Scenario (Multiple simulations)

Using the base scenario configurations and running 10000 simulations, we obtained the following results:

- **Total buses processed across 10000 simulations**: 811250
- **Average buses processed per simulation**: 81.12 (min: 50, max: 117)
- **Average inspection queue length**: 0.09 (min: 0.01, max: 0.41)
- **Average repair queue length**: 0.02 (min: 0.00, max: 0.26)
- **Average inspection delay**: 0.17 hours (min: 0.03, max: 0.66)
- **Average repair delay**: 0.10 hours (min: 0.00, max: 1.49)

As shown in Figure 3, the distribution of outcomes across multiple simulations provides a more comprehensive view of system behavior, capturing the stochastic variability inherent in the process.

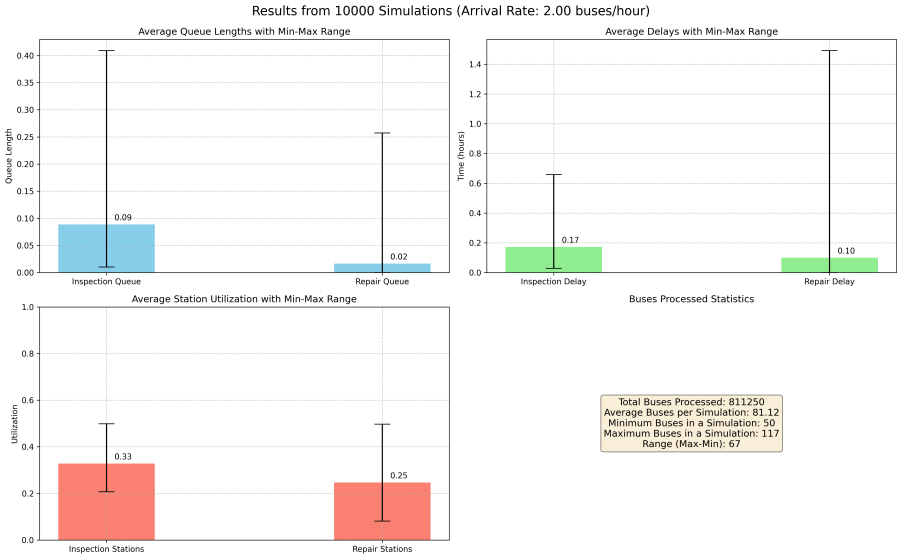


Figure 3: Bus Multiple (10000) Simulations Results

2.3.3 Maximum Arrival Rate Analysis

The arrival rate experiment systematically tests the bus maintenance facility's capacity limits by:

- Testing a range of arrival rates from low to high (buses/hour)
- For each rate, running a full simulation and measuring:
  - Queue lengths (for both inspection and repair)
  - Delays experienced by buses
  - Resource utilization (inspectors and repair stations)
- Determining stability at each rate based on thresholds for:
  - Maximum acceptable queue lengths
  - Maximum acceptable delays
  - Maximum acceptable resource utilization
- Identifying the maximum stable arrival rate - the highest rate at which the system can operate without exceeding any threshold
- Using early stopping when the system clearly becomes unstable to improve computational efficiency

Regarding system stability, we followed the criteria that average queue lengths should remain below 10 buses, average delays for both inspection and repair should stay under 5 hours, and resource utilization at both stages should not exceed 90%.

Using the configurations of the base scenario, the previously mentioned thresholds and running the experiment, we obtained the following results:

- **Maximum stable arrival rate**: 1.374 buses/hour
- **At maximum stable rate** - Inspection utilization: 0.87, Repair utilization: 0.72
- **Equivalent mean interarrival time**: 0.728 hours

Figures 4 and 5 illustrate how key performance metrics change as the arrival rate increases, with the smoothed version highlighting the critical transition point where the system becomes unstable.

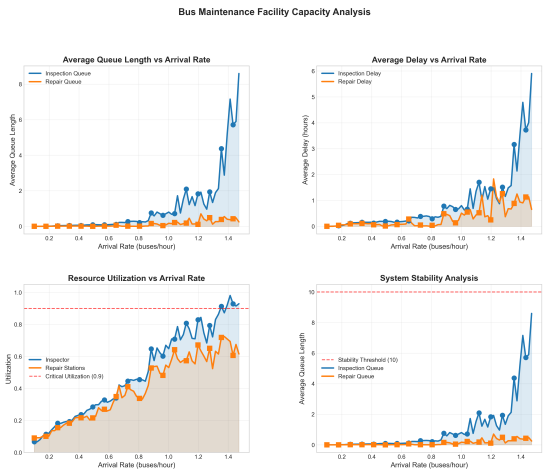


Figure 4: Arrival Rate Results

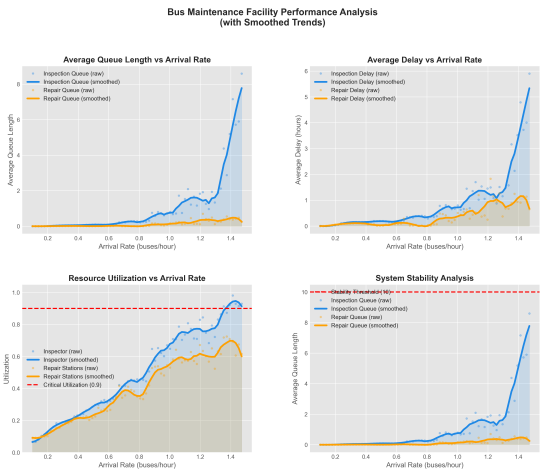


Figure 5: Arrival Rate Results (Smoothed)

2.3.4 Analysis

The simulation results indicate that the system behaves as expected under the base scenario and under varied arrival rates. The low average queue lengths and delays in both the single and multiple simulation cases confirm that the base arrival rate (0.5 buses/hour) does not overload the system. The utilization levels of the inspection and repair resources further support this conclusion, remaining well below saturation.

In the multiple simulation runs, the variability observed across the minimum and maximum values (e.g., inspection delays ranging from 0.03 to 0.66 hours) demonstrates the importance of stochastic effects and justifies the need for repeated runs to derive statistically reliable metrics.

The arrival rate experiment shows that the system can sustain up to 1.374 buses per hour without entering an unstable regime. Beyond this threshold, queues grow uncontrollably, and resource utilization nears full capacity, indicating bottlenecks, especially at the inspection station. This insight helps identify potential scaling needs or optimization opportunities.

2.4 Verification and Validation

To ensure correctness, the following measures were taken:

- **Code Verification:** The simulation logic was traced using logging and visual inspection to confirm that events occurred in the intended sequence (e.g., no bus skips inspection).
- **Distribution Checks:** Empirical distributions of interarrival, inspection, and repair times were plotted and compared against their theoretical counterparts (exponential and uniform).
- **Extreme Case Testing:** Scenarios with 0% repair probability and with very frequent arrivals were tested to observe edge behaviors (e.g., bypassing repair stations or inducing overload).
- **Repeatability:** A fixed random seed was used to enable deterministic runs for debugging and result comparison.

3 Projectile Motion Simulation

In this section, we develop a numerical simulation of projectile motion under the influence of gravity and air resistance. We implement and compare two integration methods—Forward Euler [2] and 4th-order Runge-Kutta [3] to evaluate their relative accuracy and computational efficiency for this physical system.

3.1 Problem Description

The projectile motion simulation models the flight of a projectile under the influence of gravity and air resistance. Unlike simpler models that only consider gravitational force, this simulation incorporates quadratic air resistance [6], making it more realistic but requiring numerical methods to solve.

3.1.1 Physical Model and Differential Equations

The motion of a projectile in a two-dimensional plane (x-z, where z represents height) is governed by the following system of differential equations:

$$\frac{dx}{dt} = v_x \quad \frac{dz}{dt} = v_z \quad \frac{dv_x}{dt} = -\frac{u}{m} v_x |v_x| \quad \frac{dv_z}{dt} = -g - \frac{u}{m} v_z |v_z| \tag{1}$$

Where:

- x, z represent the position coordinates
- v_x, v_z represent velocity components
- m is the mass of the projectile
- u is the air resistance coefficient
- g is the gravitational acceleration (9.81 $\mathrm{m/s^2}$)
- The terms $-\frac{u}{m} v_x |v_x|$ and $-\frac{u}{m} v_z |v_z|$ model quadratic air resistance [6]

3.1.2 Initial Conditions and Parameters

The default simulation parameters are:

- Initial position: $(x_0, z_0) = (0, 0)$ m
- Initial velocity: $(v_{x0}, v_{z0}) = (50, 50)$ m/s
- Mass: 1.0 kg
- Air resistance coefficient: 0.01
- Gravity: 9.81 $\mathrm{m/s^2}$
- Time step (dt): 0.01 s
- Simulation time: 10.0 s

3.2 Implementation Approach

The simulation implements two numerical methods to solve the system of differential equations: the Forward Euler method and the 4th-order Runge-Kutta method (RK4). Both methods are implemented and compared to evaluate their accuracy and performance.

3.2.1 Forward Euler Method

The Forward Euler method is a first-order numerical approach that approximates the solution using the following update rule [2]:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}(\mathbf{y}_n, t_n) \tag{2}$$

Where $\mathbf{y}$ represents our state vector $[x, z, v_x, v_z]$, h is the time step, and $\mathbf{f}$ is the right-hand side of our differential equations.

The Euler method is simple to implement but has limited accuracy for nonlinear systems, especially with larger time steps.

3.2.2 Runge-Kutta Method

The 4th-order Runge-Kutta method (RK4) provides a higher-order approximation by [3]:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{h}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4)$$

Where:

$$\mathbf{k}_1 = \mathbf{f}(\mathbf{y}_n, t_n) \quad \mathbf{k}_2 = \mathbf{f}\left(\mathbf{y}_n + \frac{h}{2}\mathbf{k}_1, t_n + \frac{h}{2}\right) \quad \mathbf{k}_3 = \mathbf{f}\left(\mathbf{y}_n + \frac{h}{2}\mathbf{k}_2, t_n + \frac{h}{2}\right) \quad \mathbf{k}_4 = \mathbf{f}\left(\mathbf{y}_n + h\mathbf{k}_3, t_n + h\right)$$

RK4 generally offers much better accuracy than Euler, especially for nonlinear systems, but requires more computational effort per step.

3.3 Results and Analysis

3.3.1 Trajectory Comparison

The projectile simulation generates trajectory plots comparing the Euler and Runge-Kutta (RK4) methods. Figure 6 shows the results of both numerical integration methods for a projectile with the default configuration mentioned in the problem description.

Method	Time (s)	X (m)	Z (m)	VX (m/s)	VZ (m/s)
Euler	10.00	179.12	-80.93	8.32	-30.44
RK4	10.00	179.18	-80.88	8.33	-30.43

Table 2: Final state comparison between Euler and RK4 methods at t=10s

Method	Time (s)	Distance (m)	Found
Euler	7.21	152.68	Yes
RK4	7.22	152.82	Yes

Table 3: Landing position comparison between Euler and RK4 methods

Method	X Error (m)	Z Error (m)	Total Error (m)
Euler	0.036414	0.051208	0.087622
RK4	0.000000	0.000000	0.000000

Table 4: Precision comparison against a high-precision reference solution

As visible in the trajectory plot (top left), both methods produce similar parabolic paths, with the projectile reaching a maximum height of approximately 51 meters before returning to

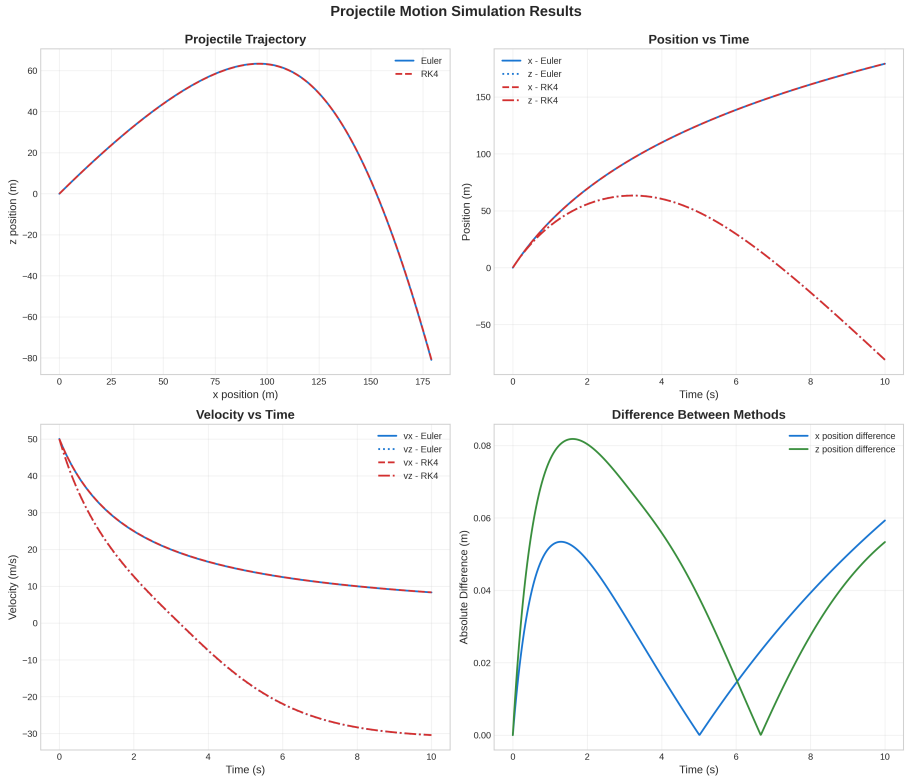


Figure 6: Comparison of projectile motion simulation results using Euler and RK4 methods.

ground level. The landing positions differ slightly, with Euler predicting a landing at 152.68 meters at 7.21 seconds, while RK4 predicts 152.82 meters at 7.22 seconds—a difference of only 14 centimeters (Table 3).

The position and velocity plots (top right and bottom left) also demonstrate high visual agreement between the methods, indicating that even the simpler Euler method provides reasonable approximations for this problem. However, the difference plot (bottom right) reveals subtle but important discrepancies, with maximum position differences of about 8 centimeters occurring around mid-flight.

These differences, while small in this case, illuminate an important property of numerical methods: the errors accumulate over time, becoming more pronounced in longer simulations or those with higher air resistance. For the default parameters, these errors remain relatively contained, but they highlight why higher-order methods like RK4 are preferred for applications requiring precision.

The simulation report quantifies the accuracy difference (Table 4), showing that RK4 is substantially more accurate when compared to a high-precision reference solution. This becomes particularly important for scenarios involving longer flight times or greater physical forces. The analysis indicates that RK4 is approximately 8.7×10^6 times more accurate than Euler for this particular simulation configuration, which is an abnormally high value and very likely linked to the fact that the reference solution is also RK4 with a lower time step.

3.3.2 Convergence Analysis

A central part of our analysis was a convergence study, where we systematically varied the time step size to observe how the error changes. This analysis helps determine the empirical order of convergence for each method, while providing insights into their practical performance.

Test Case	Initial Velocity (m/s)	Air Resistance
No air resistance	$(50.0, 50.0)$	0.00
Default	$(50.0, 50.0)$	0.01
High velocity	$(100.0, 80.0)$	0.01
High air resistance	$(50.0, 50.0)$	0.05

Table 5: Test scenario configurations for convergence analysis

We conducted the convergence study using five different time steps ($\Delta t = 0.1, 0.05, 0.01, 0.005, 0.001$ s) across four distinct test scenarios described in Table 5. For each simulation, we measured the landing position error against a high-precision reference solution computed with an extremely small time step ($\Delta t = 0.0001$ s).

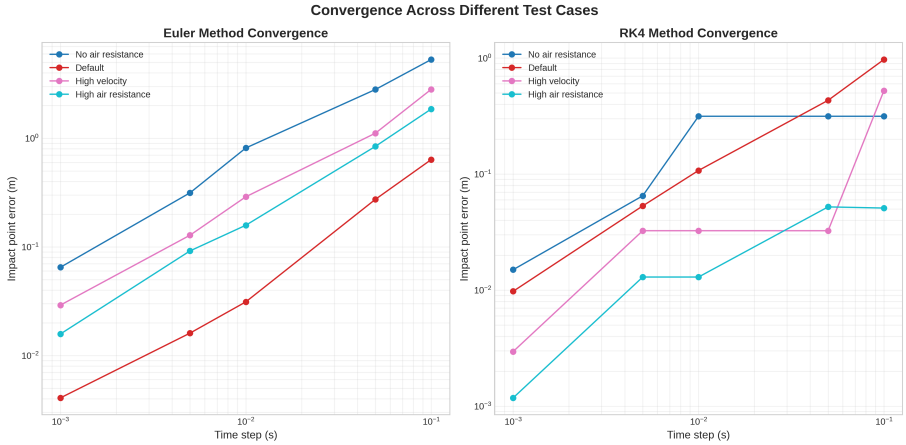


Figure 7: Convergence behavior of Euler and RK4 methods across different test scenarios

Figure 7 presents the convergence behavior across all test cases. An unexpected observation is that the RK4 method exhibits non-smooth convergence patterns, particularly evident in the "No air resistance" scenario where the error plateaus for certain time step ranges.

Test Case	Euler Order	RK4 Order
No air resistance	$O(\Delta t^{0.95})$	$O(\Delta t^{0.66})$
Default	$O(\Delta t^{1.12})$	$O(\Delta t^{0.98})$
High velocity	$O(\Delta t^{0.98})$	$O(\Delta t^{0.88})$
High air resistance	$O(\Delta t^{1.02})$	$O(\Delta t^{0.80})$
Average	$O(\Delta t^{1.02})$	$O(\Delta t^{0.83})$

Table 6: Empirical convergence orders for different test scenarios

Table 6 summarizes the empirical convergence orders calculated through linear regression in log-log space. These results reveal several interesting insights:

1. The Euler method consistently shows approximately first-order convergence ($O(\Delta t^1)$) across all test cases, which aligns with theoretical expectations.
2. The RK4 method exhibits lower than expected convergence rates, averaging $O(\Delta t^{0.83})$ instead of the theoretical fourth-order convergence ($O(\Delta t^4)$).
3. In the "No air resistance" case (Figure 8), RK4 demonstrates step-like convergence behavior, where the error remains nearly constant across certain ranges of time steps.

The detailed results for the default case further illustrate these patterns:

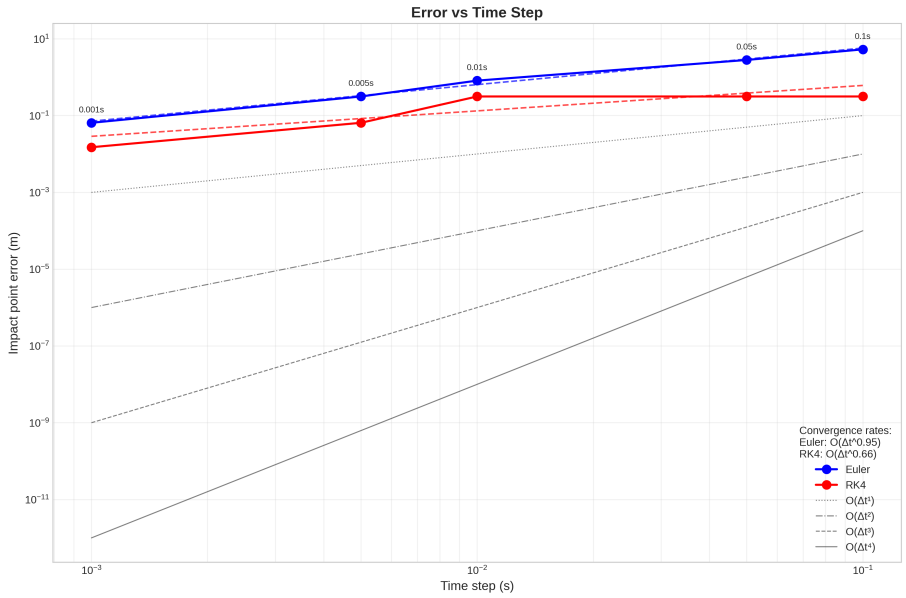


Figure 8: Convergence study for the "No air resistance" case, showing unusual step-like behavior in the RK4 method

Time Step (s)	Euler Error (m)	RK4 Error (m)
0.10000	0.636961	0.971361
0.05000	0.274414	0.432284
0.01000	0.031217	0.107433
0.00500	0.016093	0.053188
0.00100	0.004078	0.009771

Table 7: Detailed error measurements for the default test case

This unexpected convergence behavior can be attributed to several factors:

- **Non-smooth dynamics:** The presence of air resistance creates nonlinear dynamics that affect the convergence properties.
- **Landing point sensitivity:** Our error metric (landing position) is particularly sensitive to small variations in the trajectory.
- **Interpolation effects:** The calculation of the exact landing point involves interpolation between discrete time steps, which introduces additional error.

Despite lower-than-expected convergence orders, RK4 still outperforms the Euler method in absolute error across most scenarios, particularly at smaller time steps. This underscores the importance of considering both the convergence rate and absolute error when selecting a numerical method for practical applications.

4 Conclusion

This report has presented two computational simulation projects that demonstrate different approaches to modeling and solving diverse problems in science and engineering.

For the bus maintenance facility simulation, we have shown that discrete-event simulation provides an effective framework for analyzing service systems with queues and limited resources. The base configuration, with an arrival rate of 0.5 buses/hour, operates well within capacity, with low queue lengths and moderate resource utilization. Through systematic experimentation, we determined that the system can sustainably handle up to 1.374 buses per hour before becoming unstable, with the inspection station emerging as the primary bottleneck. This analysis demonstrates how simulation can inform capacity planning and resource allocation decisions in maintenance operations.

The projectile motion simulation highlighted the comparative advantages and limitations of different numerical integration methods. While the Forward Euler method demonstrated expected first-order convergence ($O(\Delta t)$), the 4th-order Runge-Kutta method showed surprisingly lower convergence rates than theoretically predicted, averaging $O(\Delta t^{0.83})$. This unexpected behavior reveals how the nature of the physical system—particularly the nonlinearity introduced by air resistance and the sensitivity of landing position calculations—can affect numerical performance. Despite this, RK4 consistently provided more accurate results than Euler for equivalent time steps, confirming its value for precision-critical applications.

Several limitations and opportunities for future work exist for both simulations. The bus maintenance model could be extended to consider variable staffing levels, scheduled maintenance, or priority queueing for urgent repairs. The projectile simulation could incorporate additional physical effects like wind or variable air density, or implement adaptive time-stepping to optimize computational resources. Additionally, exploring alternative error metrics beyond landing position might provide more insight into the convergence properties of the numerical methods.

Overall, these projects demonstrate the power of computational simulation as a tool for understanding complex systems, whether discrete or continuous in nature. The complementary approaches (discrete-event simulation [5] for service systems and numerical integration for dynamical systems) showcase the extension of techniques available to modern computational scientists and engineers.

5 References

[1] Ribeiro, A., & Pinto, M. (2025). SO-Simulation-MP: Simulation Mini-Projects for Simulation and Optimization Course. [Online]. Available: `https://github.com/Sytuz/SO-Simulation-MP`

[2] Butcher, J. C. (2016). Numerical Methods for Ordinary Differential Equations. John Wiley & Sons. pp. 45-52.

[3] Hairer, E., Nørsett, S. P., & Wanner, G. (1993). Solving Ordinary Differential Equations I: Nonstiff Problems. Springer-Verlag.

[4] Team SimPy (2022). SimPy: Discrete-Event Simulation for Python. [Online]. Available: `https://simpy.readthedocs.io/`

[5] Law, A. M., & Kelton, W. D. (2000). Simulation Modeling and Analysis. McGraw-Hill Higher Education.

[6] Parker, G. W. (1977). Projectile motion with air resistance quadratic in the speed. American Journal of Physics, 45(7), 606-610.

[7] Süli, E., & Mayers, D. (2003). An Introduction to Numerical Analysis. Cambridge University Press.