Departamento de Eletrónica, Telecomunicações e Informática

Simulação e Otimização

Relatório Final do Mini-Projeto de Simulação

André Oliveira, 107637
Alexandre Cotorobai, 107849

Mestrado em Engenharia Informática

Professor: Prof. Nuno Lau

12 de maio de 2025

Índice

Capítulo 1

Introdução

O presente relatório foi desenvolvido no âmbito da unidade curricular de Simulação e Otimização, integrada no Mestrado em Engenharia Informática da Universidade de Aveiro. Este trabalho visa aplicar técnicas de simulação a problemas com características distintas, explorando a sua modelação computacional e análise de resultados.

Ao longo do projeto, foram utilizados métodos numéricos e estatísticos para representar e estudar o comportamento de sistemas dinâmicos e estocásticos. As abordagens adotadas permitiram não só compreender o impacto de diferentes parâmetros no desempenho dos sistemas simulados, como também avaliar a eficácia dos métodos implementados.

A realização deste trabalho permitiu reforçar competências na construção de simuladores, na análise de dados obtidos e na avaliação de desempenho de sistemas complexos. Os resultados obtidos são discutidos e interpretados de forma a fundamentar as decisões tomadas ao longo do desenvolvimento.

Capítulo 2

Exercício 1

2.1 Introdução

A manutenção eficiente de frotas de transporte urbano é um fator crítico para assegurar a fiabilidade e a sustentabilidade operacional dos serviços públicos. Neste contexto, a simulação de sistemas de manutenção oferece uma abordagem poderosa para analisar e otimizar o desempenho de instalações complexas, onde múltiplos recursos e eventos aleatórios influenciam diretamente na capacidade de resposta e na qualidade do serviço.

Este projeto tem como objetivo a modelação e simulação de uma instalação de manutenção de autocarros urbanos, utilizando conceitos de Simulação de Eventos Discretos (*Discrete Event Simulation – DES*). A instalação em estudo é composta por uma estação de inspeção única e duas estações paralelas de reparação. Os autocarros chegam à instalação com tempos entre chegadas (*interarrival times*) exponencialmente distribuídos com média de 2 horas, sendo todos sujeitos a uma inspeção com duração uniformemente distribuída entre 15 minutos e 1,05 horas. Historicamente, 30% dos autocarros requerem reparações, as quais são realizadas numa das duas estações paralelas, também alimentadas por uma única fila (*FIFO*), com duração uniformemente distribuída entre 2,1 e 4,5 horas.

A simulação foi desenvolvida com recurso à linguagem `Python` e à biblioteca `SimPy`, que oferece uma estrutura orientada a processos para a simulação de eventos discretos. Nesta abordagem, os autocarros são representados como processos individuais que evoluem ao longo do tempo, interagindo com recursos limitados como a estação de inspeção e as estações de reparação. A simulação decorre durante um total de 160 horas simuladas, e recolhe métricas de desempenho do sistema, nomeadamente:

- Atraso médio em cada fila;
- Comprimento médio de cada fila;
- Utilização da estação de inspeção;
- Utilização das estações de reparação.

A estrutura da simulação foi organizada da seguinte forma:

- As chegadas de autocarros são geradas segundo um processo estocástico com distribuição exponencial, utilizando um gerador de chegadas contínuo;
- A estação de inspeção é modelada como um recurso com capacidade unitária;
- As estações de reparação são representadas como um recurso com capacidade dois, partilhado entre todos os autocarros que requerem reparação;

- São utilizados temporizadores internos para modelar os tempos de serviço (inspeção e reparação), com distribuições uniformes conforme os parâmetros fornecidos;
- Estatísticas como tempos de espera e utilização de recursos são calculadas com base no histórico de eventos e acumuladores dedicados.

Além da simulação base, será realizada uma experiência adicional com o objetivo de determinar a taxa de chegada de autocarros máxima (mínimo tempo médio entre chegadas) que o sistema consegue suportar de forma eficiente, sem gerar atrasos excessivos ou sobrecarga nos recursos. Esta análise permitirá estimar a capacidade máxima da instalação e identificar eventuais pontos críticos operacionais.

Esta introdução estabelece assim o enquadramento teórico e técnico para a realização do exercício 1, servindo de base para a descrição da implementação, análise dos resultados e eventuais melhorias ao modelo.

2.2 Metodologia de Implementação

Para a implementação do simulador de manutenção de autocarros foi utilizada a linguagem Python, recorrendo à biblioteca `SimPy`, que permite a modelação orientada a processos em simulações de eventos discretos. Esta abordagem possibilita a definição modular dos componentes do sistema, nomeadamente a estação de inspeção, as estações paralelas de reparação e o mecanismo de monitorização.

2.2.1 Inicialização e Definição de Parâmetros

Para assegurar a reprodutibilidade dos resultados e estabelecer o ambiente simulado, definiram-se diversos parâmetros constantes que determinam os tempos de operação e as probabilidades de transição dos processos. Segue um excerto ilustrativo da sua definição:

```
RANDOM_SEED = 42
SIMULATION_TIME: float = 160
MEAN_INTERARRIVAL: float = 2.0
INSPECTION_CAPACITY: int = 1
INSPECTION_TIME_MIN: float = 0.25
INSPECTION_TIME_MAX: float = 1.05
REPAIR_CAPACITY: int = 2
REPAIR_TIME_MIN: float = 2.1
REPAIR_TIME_MAX: float = 4.5
REPAIR_PROB: float = 0.3
```

Estes valores estabelecem o tempo total da simulação (160 horas), os intervalos de tempo entre chegadas de autocarros (média exponencial de 2 horas), bem como os tempos de serviço para inspeção e reparação (distribuições uniformes). A probabilidade de um autocarro necessitar de reparação foi fixada em 30%. Além disso, a capacidade da estação de inspeção é de 1 autocarro, enquanto a capacidade da estação de reparo é de 2 autocarros simultaneamente.

2.2.2 Modelação dos Recursos e Entidades

No modelo, a estação de inspeção é implementada como um recurso com capacidade unitária, enquanto as estações de reparação são simuladas como um recurso com duas unidades, partilhadas entre os autocarros que requerem reparação. A criação destes recursos é efetuada conforme o excerto abaixo:

```
inspection_station: InspectionStation = InspectionStation(env) # Capacidade 1
repair_station: RepairStation = RepairStation(env) # Capacidade 2
```

Cada recurso possui um acumulador que regista o tempo em que se encontra ocupado, permitindo posteriormente o cálculo da sua utilização.

2.2.3 Implementação do Processo bus

Cada autocarro é modelado como um processo que passa por duas fases principais: inspeção e, se necessário, reparação. O processo implementa os seguintes passos:

Fase de Inspeção O autocarro solicita acesso à estação de inspeção, sendo registado o tempo de espera até ao início do serviço. Após a obtenção do recurso, procede-se à inspeção:

```
arrival_time: float = env.now

with inspection_station.resource.request() as req:
    yield req
    wait_time = env.now - arrival_time
    inspection_wait_times.append(wait_time)
    yield env.process(inspection_station.inspect(bus_id))
```

Fase de Reparação Com probabilidade definida em 30%, o autocarro é encaminhado para a estação de reparação. O tempo de espera para este serviço é igualmente registado:

```
if random.random() < REPAIR_PROB:
    with repair_station.resource.request() as req:
        yield req
        repair_wait = env.now - repair_arrival_time
        repair_wait_times.append(repair_wait)
        yield env.process(repair_station.repair(bus_id))
```

2.2.4 Geração dos Processos e Monitorização

A chegada de autocarros é simulada através de um processo estocástico, que gera intervalos entre chegadas segundo uma distribuição exponencial:

```
bus_count: int = 0
while True:
    bus_count += 1
    env.process(bus(env, f"Bus {bus_count}", inspection_station, repair_station))
    interarrival_time: float = random.expovariate(1.0 / MEAN_INTERARRIVAL)
    yield env.timeout(interarrival_time)
```

Simultaneamente, o sistema regista, a intervalos regulares, o comprimento das filas de espera para cada recurso, permitindo a monitorização dinâmica do estado do sistema:

```
inspection_queue_lengths.append(len(inspection_station.resource.queue))
repair_queue_lengths.append(len(repair_station.resource.queue))
yield env.timeout(sample_interval)
```

2.2.5 Cálculo das Estatísticas e Execução da Simulação

Ao final da execução, são calculadas diversas métricas de desempenho, que permitem avaliar a eficácia do sistema simulado. Entre as estatísticas computadas destacam-se os tempos médios de espera nas filas, os comprimentos médios das mesmas e a utilização dos recursos:

```
avg_inspection_wait: float = (
    statistics.mean(inspection_wait_times) if inspection_wait_times else 0.0
)
avg_repair_wait: float = (
    statistics.mean(repair_wait_times) if repair_wait_times else 0.0
)
avg_inspection_queue: float = (
    statistics.mean(inspection_queue_lengths) if inspection_queue_lengths else 0.0
)
avg_repair_queue: float = (
    statistics.mean(repair_queue_lengths) if repair_queue_lengths else 0.0
)
utilization_inspection: float = (
    inspection_station.busy_time / (INSPECTION_CAPACITY * SIMULATION_TIME) * 100
)
utilization_repair: float = (
    repair_station.busy_time / (REPAIR_CAPACITY * SIMULATION_TIME) * 100
)
```

Finalmente, a função principal orquestra o fluxo global da simulação, iniciando os processos de geração e monitorização dos autocarros:

```
env.process(bus_generator(env, inspection_station, repair_station))
env.process(monitor_queues(env, inspection_station, repair_station))
env.run(until=SIMULATION_TIME)
```

Todos os tempos são posteriormente convertidos para um formato compreensível (horas, minutos e segundos) através da função `convert_hours_to_hms()`.

Esta arquitetura modular e claramente delineada permite não só a análise detalhada dos indicadores de desempenho do sistema (tais como atrasos e utilização dos recursos) como também a realização de experiências adicionais. Por exemplo, foi possível avaliar a capacidade máxima da instalação, determinando o menor intervalo médio entre chegadas de autocarros que o sistema consegue suportar sem que sejam gerados atrasos excessivos.

2.2.6 Fluxogramas e Considerações sobre a Implementação

A Figura 2.1 apresenta os fluxogramas que representam os principais eventos da simulação: chegada dos autocarros, fim da inspeção e fim da reparação. Estes diagramas serviram como ferramenta essencial para a conceção inicial da lógica do sistema, descrevendo o fluxo sequencial e as condições decisivas que regem o comportamento dos autocarros na instalação.

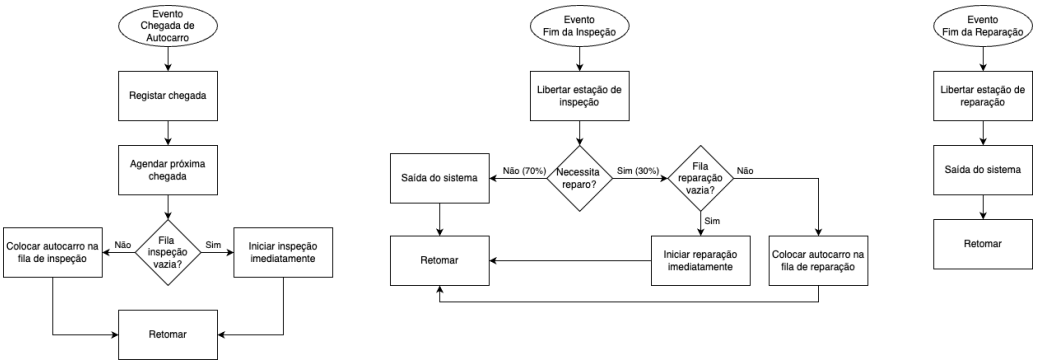

Figura 2.1: Fluxogramas dos eventos principais da simulação.

No entanto, é importante salientar que, apesar da utilidade dos fluxogramas para visualizar o processo, a implementação prática diferiu significativamente da abordagem clássica de Simulação de Eventos Discretos. Isto deve-se ao facto de termos utilizado a biblioteca `SimPy`, que segue um paradigma orientado a processos. Assim, não foi necessário implementar manualmente um grafo de eventos nem criar eventos futuros explicitamente.

Com `SimPy`, cada autocarro é encapsulado num processo autónomo, onde a lógica sequencial (por exemplo, inspeção seguida ou não de reparação) é definida diretamente no código através de instruções `yield`. Isso significa que operações como "um evento disparar outro evento futuro" (exemplificado nos fluxogramas quando o fim da inspeção leva ao início da reparação) são geridas de forma automática pela *framework*. A coordenação entre processos e recursos (como filas e estações) é igualmente assegurada internamente, eliminando a necessidade de uma lista global de eventos ou de mecanismos explícitos para agendar eventos futuros.

Assim, enquanto os fluxogramas ilustram claramente o fluxo concetual, a implementação aproveitou a abstração do `SimPy` para simplificar substancialmente o desenvolvimento, focando-se apenas na definição do comportamento sequencial dos processos, deixando a complexidade da gestão de eventos a cargo da *framework*.

Adicionalmente, após a conclusão desta implementação orientada a processos com `SimPy`, foi desenvolvida uma segunda implementação alternativa, desta vez não orientada a processos. Nesta nova versão, aplicou-se a abordagem clássica de Simulação de Eventos Discretos, com a construção explícita de um grafo de eventos, como mostrado na Figura 2.2, e o controlo manual da lista de eventos futuros. Esta implementação permitiu uma comparação direta entre os dois paradigmas e reforçou a compreensão sobre as diferenças práticas na gestão e coordenação dos eventos da simulação.

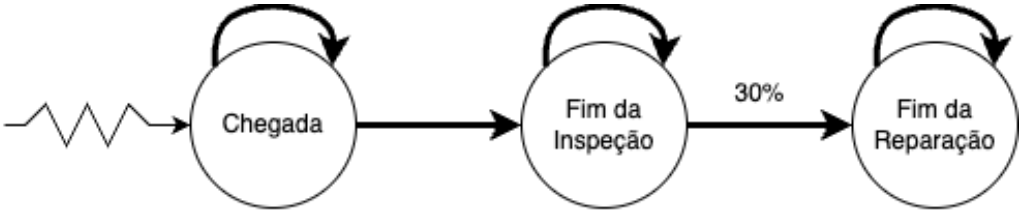

Figura 2.2: Grafo de Eventos.

O grafo de eventos é composto pelos seguintes **nós (eventos)**:

- `Chegada`: Representa a chegada de um autocarro à instalação;
- `Fim da Inspeção`: Representa o término da inspeção de um autocarro;
- `Fim da Reparação`: Representa o término do reparo de um autocarro.

Para clarificar a lógica do agendamento de eventos, o grafo utiliza diferentes tipos de setas, cada uma com um significado específico:

- **Seta Grossa (Heavy Arrow)**

  Representa eventos que são agendados para ocorrer **após um tempo não-nulo** (ou seja, com um atraso programado). Exemplos diretamente observados no código incluem:

  - Agendamento da **próxima chegada** de autocarro;
  - Agendamento do **fim da inspeção** (entre 0,25 e 1,05 horas);
  - Agendamento do **fim do reparo** (entre 2,1 e 4,5 horas);
  - Agendamento da **próxima inspeção** da fila (quando há autocarros a aguardar e a estação fica livre);
  - Agendamento do **próximo reparo** da fila (quando há autocarros a aguardar e a estação fica livre).

- **Seta Fina (Thin Arrow)**

  Representa eventos agendados para ocorrer **imediatamente**, ou seja, com atraso nulo, embora no código atual não existam exemplos concretos deste tipo de evento.

- **Seta Serrilhada (Jagged Arrow)**

  Representa eventos **agendados no início da simulação**. No código, este tipo de seta corresponde ao agendamento inicial do evento `Chegada`, que inicia todo o fluxo da simulação.

O comportamento dinâmico da simulação segue esta sequência:

1. **Início da simulação**:
   - É agendada a primeira `Chegada` de um autocarro.
2. **Quando um autocarro chega (`Chegada`)**:
   - Agenda-se outro evento `Chegada` para manter o fluxo contínuo de chegadas.
   - O autocarro inicia a inspeção **imediatamente** se a estação estiver livre ou entra na fila caso contrário.
   - Se a inspeção começar, é agendado o evento `Fim da Inspeção`.
3. **Quando a inspeção termina (`Fim da Inspeção`)**:
   - Se houver fila de inspeção, agenda-se o próximo evento `Fim da Inspeção` para o autocarro da fila.
   - Com **30% de probabilidade**, o autocarro segue para reparo:
     - Se a estação de reparo estiver livre, inicia-se o reparo e agenda-se o evento `Fim de Reparação`.
     - Caso contrário, o autocarro entra na fila de reparo.
4. **Quando o reparo termina (`Fim de Reparação`)**:
   - Se houver fila de reparo, agenda-se o próximo evento `Fim da Reparação` para o autocarro da fila.

Este esquema de setas e eventos permite compreender não apenas a lógica sequencial da simulação, mas também os diferentes tempos de espera associados a cada transição, reforçando a clareza do funcionamento interno do modelo.

2.3 Resultados

Após executar a simulação durante 160 horas, recolheram-se várias métricas que permitem avaliar o desempenho deste sistema. Os dados foram extraídos a partir dos acumuladores estatísticos referidos na Secção 2.2 e analisados com recurso ao *Jupyter Notebook*, permitindo a geração de gráficos que exploram o comportamento do sistema face a diferentes taxas de chegada.

2.3.1 Métricas Obtidas com a Configuração Base (média com 1000 *seeds* diferentes)

- **Tempo médio de espera na inspeção:** ≈ 10 minutos e 39 segundos;
- **Tempo médio de espera na reparação:** ≈ 5 minutos e 54 segundos;
- **Comprimento médio da fila de inspeção:** ≈ 0.089 autocarros;
- **Comprimento médio da fila de reparação:** ≈ 0.015 autocarros;
- **Utilização da estação de inspeção:** ≈ 32.566%;
- **Utilização das estações de reparação:** ≈ 24.338%.

Estes resultados indicam que o sistema opera de forma eficiente na configuração base, apresentando tempos de espera reduzidos e uma utilização moderada dos recursos.

2.3.2 Análise Gráfica

Para avaliar o comportamento do sistema sob diferentes cargas, realizou-se uma experiência paramétrica variando o tempo médio entre chegadas (alterando assim a taxa de chegada de autocarros). Foram obtidos os seguintes gráficos:

1. **Tempos médios de espera (Figura 2.3):** Demonstra que, para taxas superiores a 1.5 autocarros/hora, os tempos de espera começam a aumentar significativamente na fila de inspeção, atingindo valores superiores a 4 horas nos casos extremos, enquanto a fila de reparação mantém-se relativamente estável, indicando que o ponto crítico situa-se na fase de inspeção.
2. **Comprimentos médios das filas (Figura 2.4):** Verifica-se que a fila de inspeção cresce de forma exponencial com a taxa de chegada, enquanto a fila de reparação permanece quase constante, mais uma vez, indicando que o ponto crítico situa-se na fase de inspeção.
3. **Utilização dos recursos (Figura 2.5):** Observa-se que a estação de inspeção atinge a utilização de 100% a partir de 2.5 autocarros/hora, enquanto as estações de reparação estabilizam entre 70% e 80%.

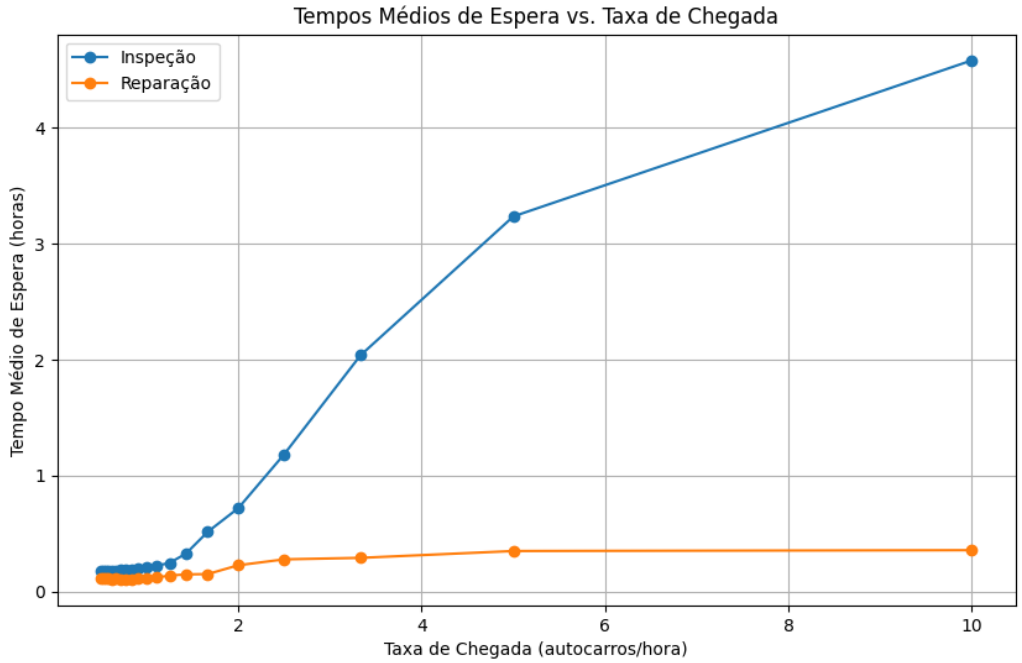

Figura 2.3: Tempos médios de espera em função da taxa de chegada de autocarros.

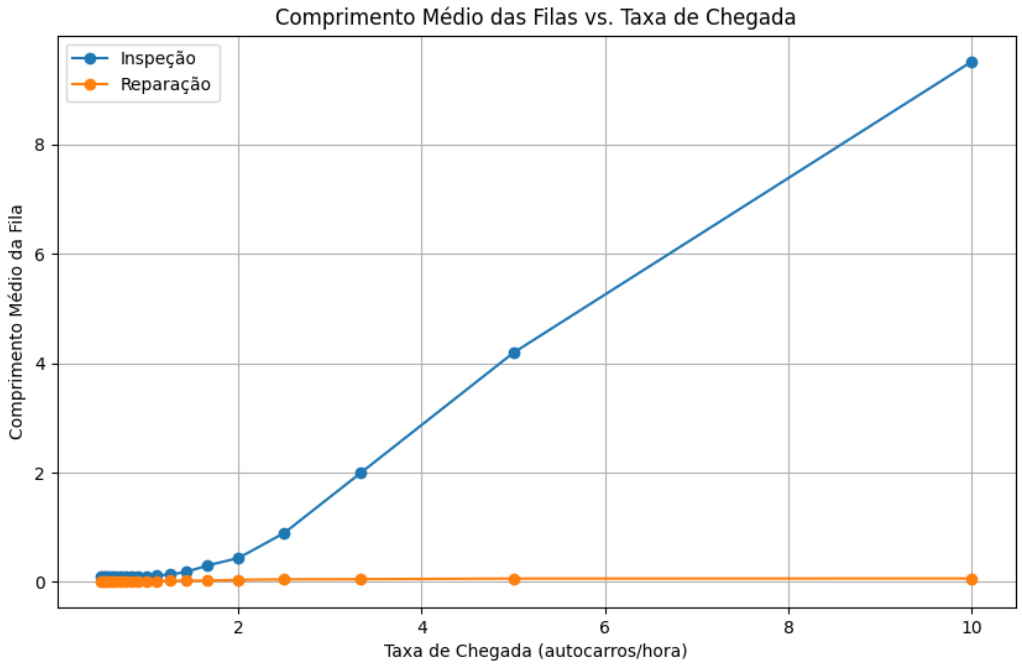

Figura 2.4: Comprimentos médios das filas em função da taxa de chegada de autocarros.

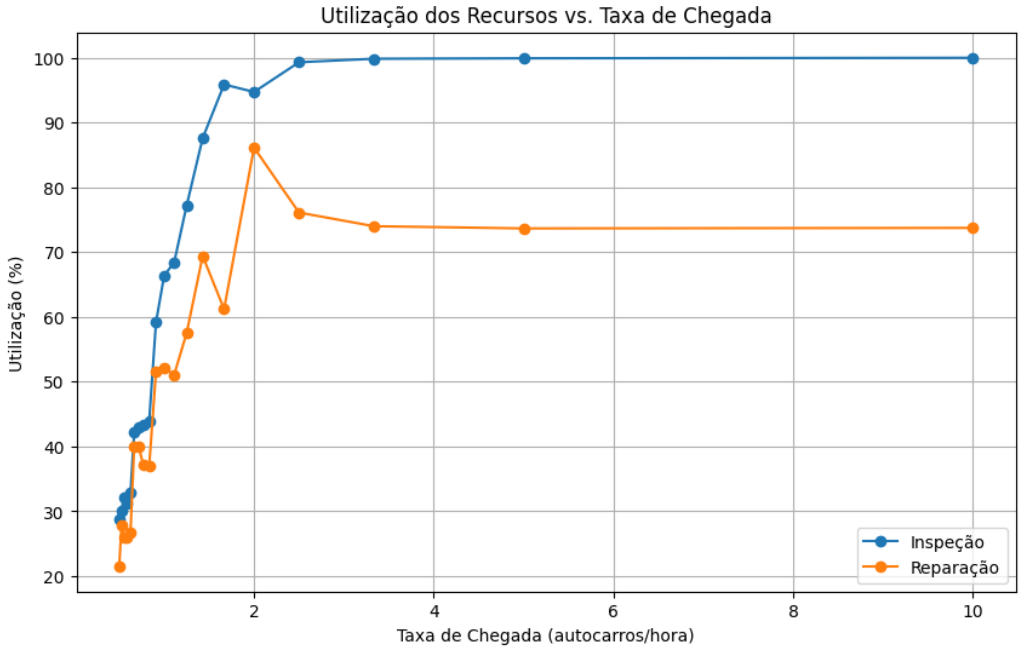

Figura 2.5: Utilização dos recursos em função da taxa de chegada de autocarros.

2.3.3 Determinação da Capacidade Máxima do Sistema

A análise paramétrica permitiu identificar os seguintes pontos críticos:

- A **estação de inspeção atinge 100% de utilização** para uma taxa de chegada próxima dos 2.5 autocarros/hora;
- O **tempo médio de espera na inspeção ultrapassa 2 horas** para taxas acima deste valor, comprometendo a eficiência operacional;
- A fase de reparação mantém uma utilização inferior a 80%, sem apresentar problemas de congestionamento.

Conclui-se, portanto, que a capacidade máxima sustentável do sistema situa-se entre **2.0 e 2.5 autocarros por hora**. Acima deste intervalo, o sistema entra em regime de sobrecarga, com aumento significativo dos tempos de espera e do comprimento das filas, especialmente na inspeção.

2.3.4 Síntese

A análise dos resultados, suportada tanto por dados estatísticos como por representações gráficas, evidencia que o sistema de manutenção de autocarros apresenta um desempenho robusto na configuração base. Contudo, a investigação paramétrica revela um ponto crítico na estação de inspeção, o qual poderá orientar futuras melhorias na infraestrutura, nomeadamente mediante a duplicação da estação de inspeção ou a implementação de estratégias de balanceamento de carga.

Capítulo 3

Exercício 2

3.1 Introdução

O estudo do movimento de projéteis sujeitos à resistência do ar é uma aplicação clássica da modelação e simulação de sistemas dinâmicos contínuos. Este tipo de simulação é fundamental em diversas áreas da engenharia e da física, onde a evolução do sistema depende de variáveis que mudam continuamente no tempo segundo leis expressas por equações diferenciais. Quando essas equações não podem ser resolvidas analiticamente, devido à complexidade ou à não linearidade dos termos, recorre-se a métodos numéricos de integração como alternativa viável e eficaz.

No contexto deste projeto, pretende-se simular a trajetória bidimensional de um projétil lançado no ar, considerando os efeitos da gravidade e da resistência do ar. A evolução do sistema é descrita por duas equações diferenciais de segunda ordem que governam a posição do projétil nos eixos $x(t)$ e $z(t)$ (horizontal e vertical, respetivamente), tendo em conta a aceleração devido à força de arrasto (resistência do ar), que é proporcional ao quadrado da velocidade e atua na direção oposta ao movimento.

As equações diferenciais que regem o sistema são:

$$m \cdot \frac{d^2x(t)}{dt^2} = -u \cdot \left(\frac{dx(t)}{dt}\right)^2 \cdot \text{sign}\left(\frac{dx(t)}{dt}\right)$$

$$m \cdot \frac{d^2z(t)}{dt^2} = -m \cdot g - u \cdot \left(\frac{dz(t)}{dt}\right)^2 \cdot \text{sign}\left(\frac{dz(t)}{dt}\right)$$

Onde:

- m é a massa do projétil;
- g é a aceleração da gravidade;
- u é o coeficiente de resistência do ar;
- $\text{sign}(v)$ representa o sinal da velocidade (positivo ou negativo).

A abordagem adotada consiste na resolução numérica destas equações utilizando dois métodos distintos:

1. **Método de Euler (Forward Euler)**, um método explícito simples e intuitivo, que permite obter aproximações sucessivas das variáveis de estado em passos discretos de tempo Δt;
2. **Método de Runge-Kutta de quarta ordem (RK4)**, que proporciona uma maior precisão na aproximação das soluções, com um custo computacional ligeiramente superior, mas com vantagens significativas em termos de estabilidade e erro numérico.

Foi desenvolvido um único programa de simulação que implementa ambos os métodos de integração numérica, Euler e Runge-Kutta de quarta ordem (RK4), permitindo traçar a trajetória do projétil ao longo do tempo, bem como a evolução das suas velocidades horizontais e verticais. O utilizador pode selecionar qual dos métodos pretende utilizar ou optar por executar ambos, de forma a permitir uma comparação direta dos resultados obtidos. Todos os parâmetros iniciais, incluindo posição inicial, velocidades iniciais, massa, resistência do ar, tempo final e passo temporal, são configuráveis, podendo ser definidos através da linha de comandos.

Por fim, será realizada uma análise comparativa entre os dois métodos de integração, com o intuito de avaliar a **precisão numérica** de cada abordagem, identificar diferenças nos resultados obtidos e discutir as vantagens e limitações de cada técnica de simulação no contexto do problema proposto.

3.2 Fundamentos Teóricos da Aproximação Numérica

Na simulação da trajetória do projétil, o objetivo é resolver equações diferenciais que descrevem a evolução contínua do sistema ao longo do tempo. Na maioria dos casos, não existe uma solução analítica exata para essas equações, especialmente quando o sistema envolve componentes não lineares (como a resistência do ar proporcional ao quadrado da velocidade). Assim, recorremos a **métodos numéricos** para obter aproximações sucessivas do estado do sistema.

3.2.1 Método de Euler

O método de Euler é uma técnica simples para aproximar soluções de equações diferenciais. Baseia-se na expansão em série de Taylor, considerando apenas o primeiro termo:

$$x(t + \Delta t) \approx x(t) + \Delta t \cdot f(x(t), t)$$

Este método aproxima a derivada da função pelo seu valor atual e atualiza o estado com um pequeno incremento temporal Δt. Contudo, esta simplicidade faz com que o método seja **condicionalmente estável** e apresente **erros de truncamento significativos** quando Δt não é suficientemente pequeno. O erro global deste método é da ordem de $O(\Delta t)$.

3.2.2 Método de Runge-Kutta de Quarta Ordem (RK4)

O método RK4 é, também, uma técnica para aproximar soluções de equações diferenciais, que melhora a estimativa ao considerar várias avaliações intermédias da função derivada. O RK4 calcula a inclinação em quatro pontos distintos dentro do intervalo Δt:

$$\begin{aligned}
K_1 &= \Delta t \cdot f(x(t), t) \\
K_2 &= \Delta t \cdot f\left(x(t) + \frac{K_1}{2}, t + \frac{\Delta t}{2}\right) \\
K_3 &= \Delta t \cdot f\left(x(t) + \frac{K_2}{2}, t + \frac{\Delta t}{2}\right) \\
K_4 &= \Delta t \cdot f\left(x(t) + K_3, t + \Delta t\right)
\end{aligned}$$

A atualização final é dada por:

$$x(t + \Delta t) = x(t) + \frac{K_1 + 2K_2 + 2K_3 + K_4}{6}$$

Este método apresenta **ordem de precisão quatro**, ou seja, o erro local por passo é da ordem de $O(\Delta t^5)$ e o erro global é $O(\Delta t^4)$, tornando-o substancialmente mais robusto e preciso do que o método de Euler.

3.2.3 Como as Aproximações se Aproximam da Solução Exata

Ambos os métodos transformam o problema contínuo de integração numa sequência de atualizações discretas. Teoricamente, a solução exata seria:

$$x(t) = x(0) + \int_0^t f(x(s), s)\, ds$$

Os métodos numéricos substituem esta integral contínua por somas finitas, aproximando a área sob a curva derivada por retângulos (Euler) ou por aproximações mais sofisticadas (RK4).

À medida que $\Delta t \to 0$, estas aproximações convergem para a solução exata. Contudo, há sempre um compromisso entre **precisão** e **custo computacional**, onde passos mais pequenos reduzem o erro, mas aumentam o tempo de execução..

3.3 Metodologia de Implementação

A implementação da simulação da trajetória de um projétil foi realizada em *Python*, recorrendo a métodos de integração numérica para resolver as equações diferenciais não lineares que descrevem o sistema. Tanto o método de Euler como o método de Runge-Kutta de quarta ordem (RK4) foram implementados no mesmo programa, permitindo a sua comparação num mesmo ambiente de execução.

Toda a arquitetura do programa segue o ciclo clássico de simulação de modelos contínuos abordado na unidade curricular: **initialize()**, **update()** e **observe()**.

3.3.1 `initialize()`

O programa permite a definição dos parâmetros iniciais da simulação através de argumentos na linha de comandos. Estes incluem a posição inicial (x_0, z_0), as velocidades iniciais (v_{x0}, v_{z0}), a massa m, o coeficiente de resistência do ar u, a aceleração da gravidade g, o passo temporal Δt e o tempo final t_{final}. Um excerto do código que ilustra esta funcionalidade é o seguinte:

```
parser = argparse.ArgumentParser()
parser.add_argument('--x0', type=float, default=0.0)
parser.add_argument('--z0', type=float, default=0.0)
parser.add_argument('--vx0', type=float, default=10.0)
parser.add_argument('--vz0', type=float, default=10.0)
parser.add_argument('--m', type=float, default=1.0)
parser.add_argument('--u', type=float, default=0.1)
parser.add_argument('--g', type=float, default=9.8)
parser.add_argument('--dt', type=float, default=0.01)
parser.add_argument('--tfinal', type=float, default=2.0)
```

Durante esta fase, são também inicializadas todas as variáveis do sistema e estruturas de armazenamento para recolha de dados, tal como as listas que guardam posições e velocidades ao longo do tempo.

3.3.2 `update()`

Método de Euler

O método de Euler foi implementado de forma iterativa, calculando a nova posição e velocidade a cada passo Δt, conforme se segue:

```
def update_euler(self) -> None:
    ax: float = -(self.drag / self.mass) * self.vx * abs(self.vx)
    az: float = -self.gravity - (self.drag / self.mass) * self.vz * abs(self.vz)
    self.x += self.dt * self.vx
    self.vx += self.dt * ax
    self.z += self.dt * self.vz
    self.vz += self.dt * az
```

Esta rotina corresponde à fase de `update()` do ciclo de simulação, onde as variáveis de estado são atualizadas com base nas equações diferenciais do sistema.

Método de Runge-Kutta de Quarta Ordem

Para maior precisão, foi também implementado o método RK4, utilizando a fórmula clássica com os coeficientes K_1 a K_4. A evolução da velocidade e posição é calculada com base em médias ponderadas das derivadas em pontos intermédios:

```
def update_rk4(self) -> None:
    state: np.ndarray = np.array([self.x, self.z, self.vx, self.vz])

    def f(state: np.ndarray) -> np.ndarray:
        x, z, vx, vz = state
        ax: float = -(self.drag / self.mass) * vx * abs(vx)
        az: float = -self.gravity - (self.drag / self.mass) * vz * abs(vz)
        return np.array([vx, vz, ax, az])

    dt: float = self.dt
    k1: np.ndarray = f(state)
    k2: np.ndarray = f(state + 0.5 * dt * k1)
    k3: np.ndarray = f(state + 0.5 * dt * k2)
    k4: np.ndarray = f(state + dt * k3)
    state_next: np.ndarray = state + (dt / 6.0) * (k1 + 2 * k2 + 2 * k3 + k4)
    self.x, self.z, self.vx, self.vz = state_next
```

Tal como no método de Euler, esta fase também representa a componente `update()` da simulação, com a diferença de incorporar várias estimativas intermédias que melhoram a precisão da atualização.

3.3.3 `observe()`

Os resultados de cada simulação, posições e velocidades ao longo do tempo, foram armazenados em estruturas de dados (listas) que representam a fase de `observe()`. Estes dados foram posteriormente exportados para análise em *Jupyter Notebook*, permitindo gerar gráficos comparativos entre os métodos.

Esta separação de responsabilidades, `initialize()`, `update()` e `observe()`, permite não só uma estruturação clara do código como também facilita a sua reutilização, extensão e validação.

3.4 Resultados

Após a implementação dos dois métodos de integração numérica, foram realizados testes com diferentes passos temporais Δt, com o objetivo de comparar a precisão e estabilidade de cada abordagem. Os dados recolhidos permitiram traçar a evolução da posição e da velocidade do projétil ao longo do tempo, analisando a divergência dos resultados entre os métodos conforme a granularidade temporal.

3.4.1 Comparação com $\Delta t = 0.1$

A Figura 3.1 apresenta a trajetória do projétil obtida pelos dois métodos com um passo temporal de $\Delta t = 0.1$. Observa-se uma discrepância significativa entre os resultados de Euler e RK4, sendo evidente que o método de Euler apresenta erros acumulados consideráveis, especialmente na componente vertical da trajetória. Estes erros resultam numa estimativa prematura do impacto no solo, comprometendo a precisão da simulação.

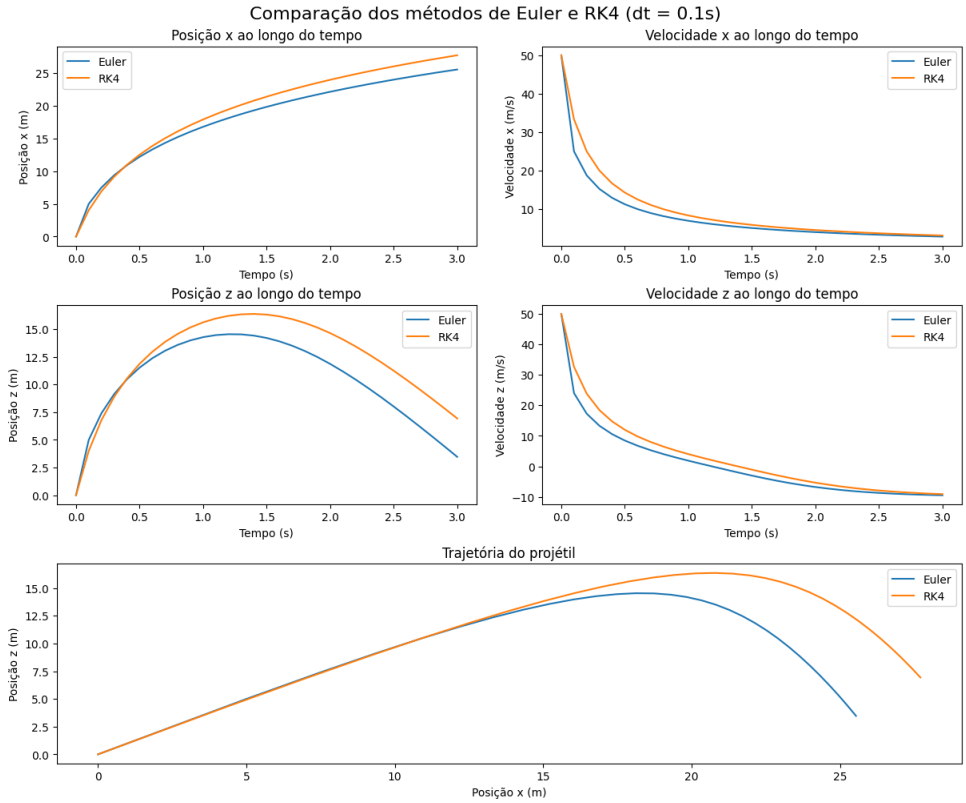

Figura 3.1: Comparação das trajetórias para $\Delta t = 0.1$ entre os métodos de Euler e RK4.

3.4.2 Comparação com $\Delta t = 0.01$

Com um passo temporal mais reduzido, $\Delta t = 0.01$, ambos os métodos convergem para trajetórias muito semelhantes, como ilustrado na Figura 3.2. O método de Euler torna-se mais estável e a sua aproximação melhora consideravelmente, embora ainda se verifiquem pequenas diferenças na posição final do projétil. O método de RK4, por outro lado, mantém a sua elevada precisão mesmo com passos maiores, sendo visivelmente mais robusto a variações no passo temporal.

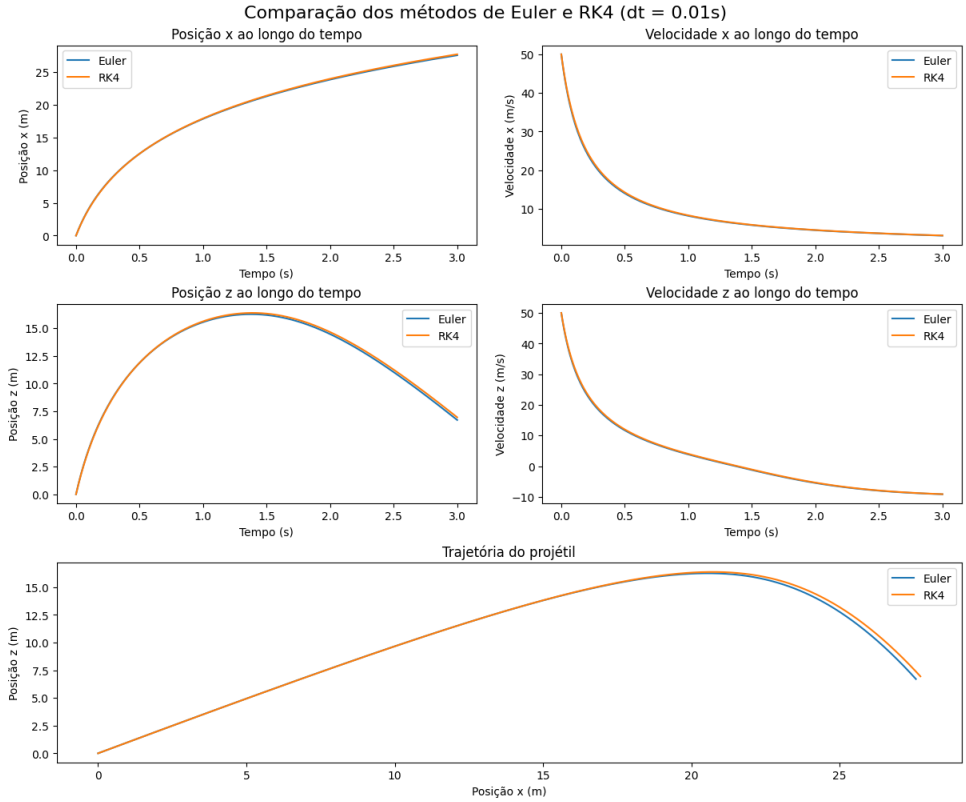

Figura 3.2: Comparação das trajetórias para $\Delta t = 0.01$ entre os métodos de Euler e RK4.

3.4.3 Síntese Comparativa

A análise dos gráficos permite concluir que:

- O método de Euler apresenta erros acumulados mais visíveis com passos de tempo maiores, podendo conduzir a trajetórias irrealistas;
- O método de RK4 é substancialmente mais preciso e estável, mesmo com passos maiores, sendo preferível quando a precisão é prioritária;
- Para passos pequenos ($\Delta t \leq 0.01$), ambos os métodos produzem resultados coerentes, embora RK4 continue a demonstrar superioridade na fidelidade dos valores finais.

Estes resultados demonstram claramente a vantagem do método de Runge-Kutta de quarta ordem na simulação de sistemas contínuos, sobretudo quando o custo computacional adicional é aceitável em troca de maior precisão.

Capítulo 4

Conclusão

A realização deste mini-projeto permitiu consolidar conhecimentos fundamentais no domínio da Simulação e Otimização, através da aplicação prática de duas abordagens distintas: a simulação de eventos discretos e a simulação de sistemas dinâmicos contínuos.

No **Exercício 1**, foi modelada uma instalação de manutenção de autocarros utilizando a biblioteca `SimPy`. Esta abordagem orientada a processos revelou-se eficaz para representar o comportamento de um sistema estocástico com múltiplos recursos e filas. A simulação permitiu obter métricas relevantes como os tempos médios de espera, a utilização dos recursos e o comprimento médio das filas. A análise paramétrica demonstrou que o ponto crítico do sistema reside na estação de inspeção, a qual atinge rapidamente os 100% de utilização quando a taxa de chegada de autocarros se aproxima dos 2.5 autocarros/hora. Esta limitação poderá ser mitigada com a duplicação da estação ou a introdução de estratégias de balanceamento de carga.

No **Exercício 2**, foi desenvolvido um simulador da trajetória de um projétil sujeito à resistência do ar, com a implementação dos métodos numéricos de Euler e Runge-Kutta de quarta ordem. A estrutura modular adotada, baseada no ciclo `initialize()` – `update()` – `observe()`, facilitou a experimentação e comparação entre métodos. Os resultados evidenciaram que o método de Euler apresenta erros significativos com passos de tempo maiores, enquanto o método RK4 mostrou-se consistentemente mais preciso e robusto, mesmo com passos maiores.

Ambos os exercícios demonstraram a importância da simulação como ferramenta de análise e apoio à decisão em sistemas complexos, quer no contexto de eventos discretos com múltiplas entidades e recursos, quer na modelação de sistemas físicos contínuos regidos por equações diferenciais.

Por fim, destaca-se que foi também elaborado um ficheiro `README.md`, o qual contém instruções detalhadas sobre a execução dos programas desenvolvidos, os requisitos necessários, bem como exemplos de utilização e explicações sobre a estrutura do código. Este ficheiro serve de guia útil para qualquer utilizador que pretenda replicar os resultados ou adaptar os simuladores a novos cenários.