Projeto 2 - Reentrada

Unidade Curricular: Simulação e Modelação Computacional em Engenharia Física

62568 - Alexandre Tashchuk
62637 - José Costa
68839 - Carolina Saraiva

Licenciatura em Engenharia Informática
Licenciatura em Engenharia Física

Índice

1. Introdução

Neste segundo projeto, iremos recorrer ao modelo de um módulo espacial, reentrando na Terra, de forma a conseguirmos obter os parâmetros de reentrada, capazes de garantir uma aterragem segura, dentro de algumas restrições. Os parâmetros, aqui a analisar, serão, naturalmente, a velocidade e o ângulo descendente com o horizonte, na interface de reentrada. Notemos que, se quisermos saber, aproximadamente, o local de aterragem do módulo espacial, teríamos de conhecer a posição deste, acima da Terra, bem como a direção do voo. No entanto, não vamos considerar esta abordagem.

O processo de reentrada constitui uma das etapas mais críticas a completar, em missões espaciais. Nesta fase, a remoção de energia cinética do módulo produz enormes quantidades de energia, que o podem danificar, ou mesmo destruir, como, infelizmente, já aconteceu.

No nosso modelo, para garantirmos o sucesso da trajetória de reentrada, teremos de considerar três aspetos: o nível máximo de múltiplos da aceleração gravítica, suportado pelo módulo espacial e pelos astronautas, no seu interior; a velocidade de aterragem e, ainda, a distância da interface de reentrada à posição de aterragem, projetada, na superfície da Terra.

A resistência do ar leva a uma diminuição da velocidade do módulo e a presença da força de sustentação permite manter a trajetória, com maior facilidade. Desta forma, o módulo espacial encontra-se, durante mais tempo, numa trajetória descendente, o que garante uma maior redução de energia cinética.

O modelo de reentrada a considerar envolve, somente, a força de resistência do ar e algumas Leis de Newton, nomeadamente, a Lei da Gravitação Universal.
A expressão seguinte representa a Lei Empírica da Resistência do Ar:

$$F_d = -\frac{1}{2}\rho A C_d v^2$$

Sendo ρ a resistência do ar; A a superfície considerada, para o coeficiente de arrasto; C_d o coeficiente de arrasto, dependente da geometria do corpo considerado, e v a velocidade.

Nas nossas simulações, iremos analisar as duas etapas do processo de reentrada: antes e depois da abertura do paraquedas. Na primeira fase, o módulo espacial está sujeito à força de arrasto, F_d, que depende da sua velocidade total, estando, sempre, orientada, na mesma direção, e com sentido contrário ao vetor velocidade. A força de sustentação, F_l, está, igualmente, presente. Esta tem o sentido positivo do eixo Oy, sendo obtida, a partir da Lei Empírica da Resistência do Ar, já apresentada, com um coeficiente de sustentação, C_l. Com a abertura do paraquedas, a resistência de arrasto do módulo mantém-se. Contudo, deixamos de considerar a força de sustentação, F_l. Agora, teremos de assumir a resistência de arrasto do paraquedas, com diferentes parâmetros A e C_d, alinhada com o vetor velocidade e sentido oposto a este.

Neste estudo, não poderemos considerar a aceleração gravítica, g, constante. Esta terá de ser calculada, para cada posição vertical da trajetória.

2. Estrutura e Funcionamento do Código

O código inclui duas funções principais de simulação: *forward_simulation(vx, vy, x, y)* e *backward_simulation(vx, vy, x, y)*. A função *forward_simulation* simula a reentrada da cápsula, usando o método *forward*, onde a posição e velocidade são atualizadas, diretamente, a cada instante de tempo, considerando forças de arrasto e sustentação, e considerando as propriedades do paraquedas, quando este é aberto. A função *backward_simulation* simula a reentrada da cápsula de maneira semelhante, mas atualiza a posição e velocidade, iterativamente, dentro de cada passo de tempo principal.

O código contém várias funções auxiliares. A função *calc_v0_components(v0, alpha)* calcula as componentes horizontal e vertical da velocidade inicial, a partir do módulo da velocidade *v0* e do ângulo *alpha*. A função *exponential_model(x, A, B)* define um modelo exponencial para o ajuste de dados e *get_exponential_fit(altitude, density)* ajusta um modelo exponencial aos dados da densidade do ar, para diferentes altitudes. A função *air_density(altitude)* interpola a densidade do ar, numa determinada altitude, usando o ajuste exponencial. As funções *drag_force(v, rho, Cd, A)* e *lift_force(v, rho, Cl, A)* permitem determinar as forças de arrasto e sustentação presentes, respetivamente. A função *calculate_accelaration_due_to_gravity(y)* calcula a aceleração, devido à gravidade, em função da altitude. A função *calculate_horizontal_distance(x, y)* calcula a distância horizontal percorrida, na trajetória de reentrada. A função *calculate_g_value(velocities, time)* determina a aceleração total e o valor de *g*, experimentado, durante a reentrada. Por fim, a função *calculate_final_velocity(vx, vy)* calcula a velocidade final da cápsula.

Para a visualização dos resultados, recorremos a duas funções de plotagem: *plot_trajectory* e *plot_velocities*. A função *plot_trajectory* representa a trajetória da cápsula, para ambos os métodos de simulação. Por sua vez, a função *plot_velocities* plota as componentes da velocidade, ao longo do tempo, para os dois métodos de simulação.

A função *simulation_handler(v0, alpha*) coordena a simulação, executando ambas as simulações *forward* e *backward*. Permite fazer uma avaliação dos resultados obtidos, exibindo os gráficos e mensagens de aceitação/rejeição, com base em critérios específicos. Esta função começa por calcular as componentes da velocidade inicial através da função *calc_v0_components(v0, alpha)*. Em seguida, executa a simulação *forward* e a simulação "backward", avaliando os resultados de ambas, em termos de distância horizontal percorrida, velocidade final, valor de *g* e outros parâmetros. A análise destes critérios permite decidir se a simulação é aceite ou rejeitada. Por fim, são chamadas as funções de plotagem, para ser possível visualizar, graficamente, a trajetória e as velocidades da cápsula, ao longo do tempo, para ambos os métodos de simulação.

Melhorias do Código da Versão v0 para a Versão v1:

Performance:

- *Multiprocessing*: permite que várias simulações sejam executadas, ao mesmo tempo, utilizando múltiplos processos. Assim, é possível encontrar os valores de v_0 e α, mais rapidamente, do que executando uma única simulação, de cada vez.
- Otimização: o *get_exponential_fit*, em vez de ser feito em cada cálculo de uma densidade a determinada altura, é calculado no início e os valores vão ser guardados como variáveis globais. Desta forma, apenas é necessário fazer a interpolação, quando precisamos de calcular uma densidade.

Legibilidade:

- *Comentários*: junto de cada método, temos uma breve descrição do que este faz, bem como o tipo dos parâmetros que recebe e o que retorna.

Extras:

- *Possível escolha entre 3 modos de execução*: o código, agora, permite ao utilizador selecionar, entre três modos diferentes de simulação: automático, manual e rápido. No modo automático, o utilizador pode definir espaçamentos entre os valores de v_0 e o número de processos a serem usados. No modo manual, o utilizador pode especificar, diretamente, os valores de v_0 e α. No modo rápido, são usados valores predefinidos para v_0 e α, com um menor tempo total de execução. Nos modos manual e rápido, são apresentados os plots e os valores, na consola. No modo automático, os valores são guardados em ficheiros *.tsv*. Ao correr este modo, caso existam, os ficheiros *.tsv* da simulação anterior serão apagados.

$$\text{total\_combinations} = \left(\left\lfloor \frac{15000}{n} \right\rfloor + 1 \right) \times 16$$

Esta equação dá o número total de simulações, que vão ser executadas com espaçamento (n).

- *Aspeto da consola*: para melhorar a legibilidade e a experiência do utilizador, o código utiliza cores diferentes, nas mensagens da consola, facilitando a distinção entre diferentes tipos de informação.
- *Mais prints a indicar diferentes ações, ao longo da simulação*: o código, agora, inclui mais mensagens, durante a execução, para manter o utilizador informado sobre o progresso da simulação.

3. Complexidade Computacional

- **Complexidade Temporal**

Funções com complexidade temporal constante ($O(1)$):

- *calc_v0_components*, *exponential_model*, *drag_force*, *lift_force*, *calculate_accelaration_due_to_gravity*, *calculate_final_velocity, air_density, residual, jacobian, get_exponential_fit*

Funções com complexidade temporal linear ($O(n)$):

- *calculate_horizontal_distance*, *calculate_g_value*

Funções com complexidade temporal linear, multiplicado por fator constante ($O(k \times n)$):

- *forward_simulation*, *backward_simulation*

A análise de complexidade revela que as funções mais críticas, em termos de tempo e espaço, são as duas simulações (*forward_simulation* e *backward_simulation*), que têm complexidade linear, com relação ao número total de intervalos de tempo. A maioria das outras funções tem complexidade constante ou linear, o que é adequado para o tipo de problema a ser resolvido. A escolha cuidadosa do número de intervalos de tempo (dt) e do intervalo de simulação é essencial, para garantir um equilíbrio entre a precisão dos resultados obtidos e a eficiência computacional.

- **Complexidade Espacial**

- $O(1)$, para variáveis individuais e constantes.
- $O(n)$, para listas de posições, velocidades e acelerações e para dados de plotagem.

A complexidade espacial do programa é dominada pelas listas, usadas para armazenar as posições, velocidades e acelerações, durante as simulações *forward_simulation* e *backward_simulation*. Assim sendo, a complexidade espacial total do programa é $O(n)$, onde n é o número de intervalos de tempo, na simulação.

4. Métodos Utilizados e Simulações

Para obtermos uma aproximação dos valores da densidade do ar, recorremos a uma função exponencial, dada por:

$$f(y) = A \times e^{B \cdot y}$$

Com este método, conseguimos uma boa aproximação para a generalidade dos valores, com pequenas diferenças registadas, na parte menos linear. O facto de ser uma função, com derivada definida, em todos os pontos, constitui uma clara vantagem. Quando a altitude tende para infinito, a densidade do ar aproxima-se de zero.

A simulação *forward* é um método simples e explícito, sendo, por isso, fácil de implementar e, computacionalmente, pouco dispendioso, por etapa. Contudo, apresenta como desvantagens a menor estabilidade e precisão, o que pode comprometer os resultados obtidos, quando são considerados intervalos de tempo muito significativos.

A simulação *Euler backward method* consiste num método implícito e, consequentemente, mais estável e preciso, com a necessidade de ser resolvido um sistema de equações, em cada intervalo de tempo (mais dispendioso, computacionalmente).

Façamos, agora, uma comparação, mais detalhada, entre os dois métodos.

- Tempo de execução:
  - A simulação *forward* permite concluir, mais rapidamente, cada etapa, uma vez que requer menos cálculos. Para problemas menos rigorosos, ou quando são utilizados intervalos de tempo reduzidos, o tempo total de execução é, geralmente, reduzido.
  - O método *backward* leva mais tempo a executar cada etapa, dado que recorre a um processo de solução iterativo. Um número de iterações elevado aumenta, significativamente, o tempo total de execução.
- Previsibilidade:
  - O método *forward* é menos previsível, em termos de estabilidade. Assim sendo, é maior a probabilidade de encontrar algumas instabilidades numéricas.
  - A simulação *backward* é mais estável e previsível, fornecendo resultados mais fiáveis, para problemas complexos.
- Complexidade:
  - A simulação *forward* tem uma complexidade $O(1)$, por etapa, já que são feitos, somente, cálculos diretos.
  - O método *backward* apresenta uma complexidade $O(N)$, por etapa, sendo N o número de iterações necessárias, para resolver a equação implícita.

Concluída a avaliação dos dois métodos, apresentamos, em seguida, os resultados esperados, para a reentrada do módulo.

- Trajetória de Reentrada:
  - A trajetória resultante do método *backward* deverá ser mais suave e estável, quando comparada com a da simulação *forward*, que poderá apresentar mais oscilações e instabilidade numérica, sobretudo, para velocidades iniciais elevadas.
- Tempo de Execução:
  - Espera-se, pelo exposto acima, que a simulação *forward* seja executada, mais rapidamente, dado que tem uma menor complexidade, por etapa. No entanto, para o problema em questão, com intervalos de tempo significativos,

poderemos obter um menor tempo total de execução, para o método *forward*, graças à sua maior estabilidade, nestas situações.

- Estabilidade:
  - As diferenças mais notórias entre as duas simulações estarão presentes, quando a reentrada não é bem sucedida, com as condições iniciais a originarem instabilidade numérica, para o método *forward*.

5. Análise dos Resultados Obtidos

- **Trajetória de Reentrada**

Comecemos por analisar a trajetória de reentrada do módulo espacial obtida, com recurso aos dois métodos.

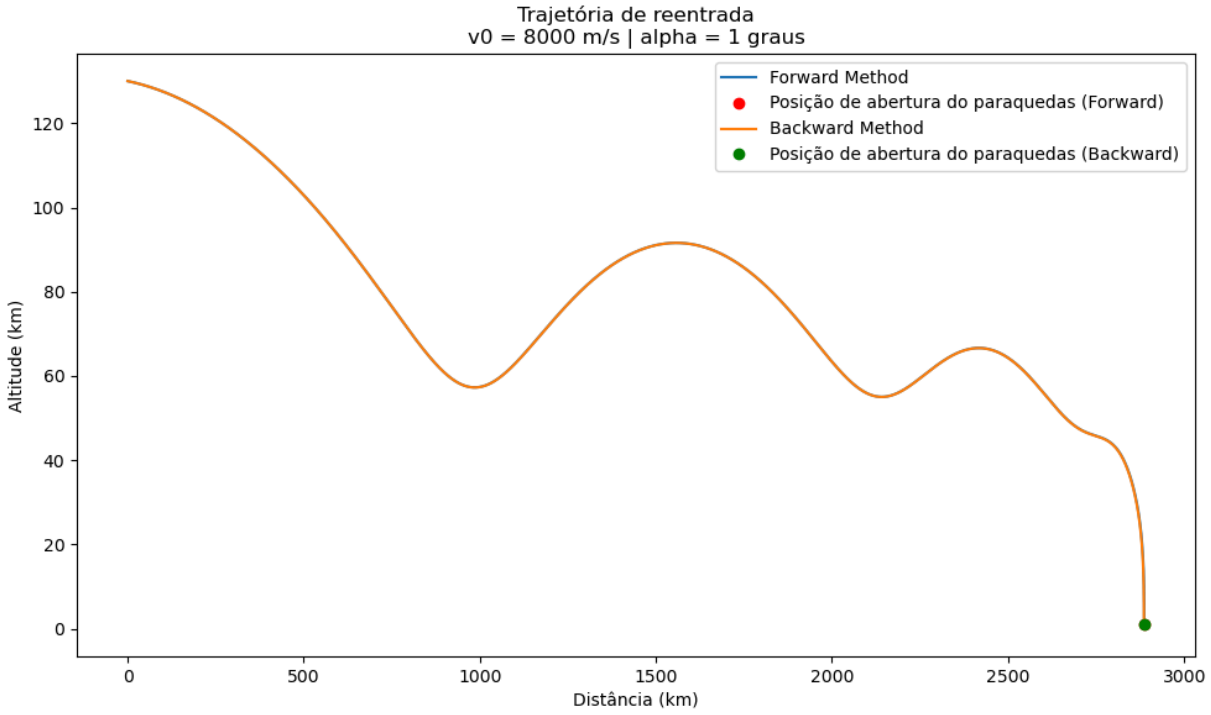

Figura 1: Gráfico da Trajetória de Reentrada, para os Dois Métodos

Deparamo-nos com uma aparente sobreposição das trajetórias. Contudo, ampliando o gráfico acima, na zona vertical, obtemos o seguinte.

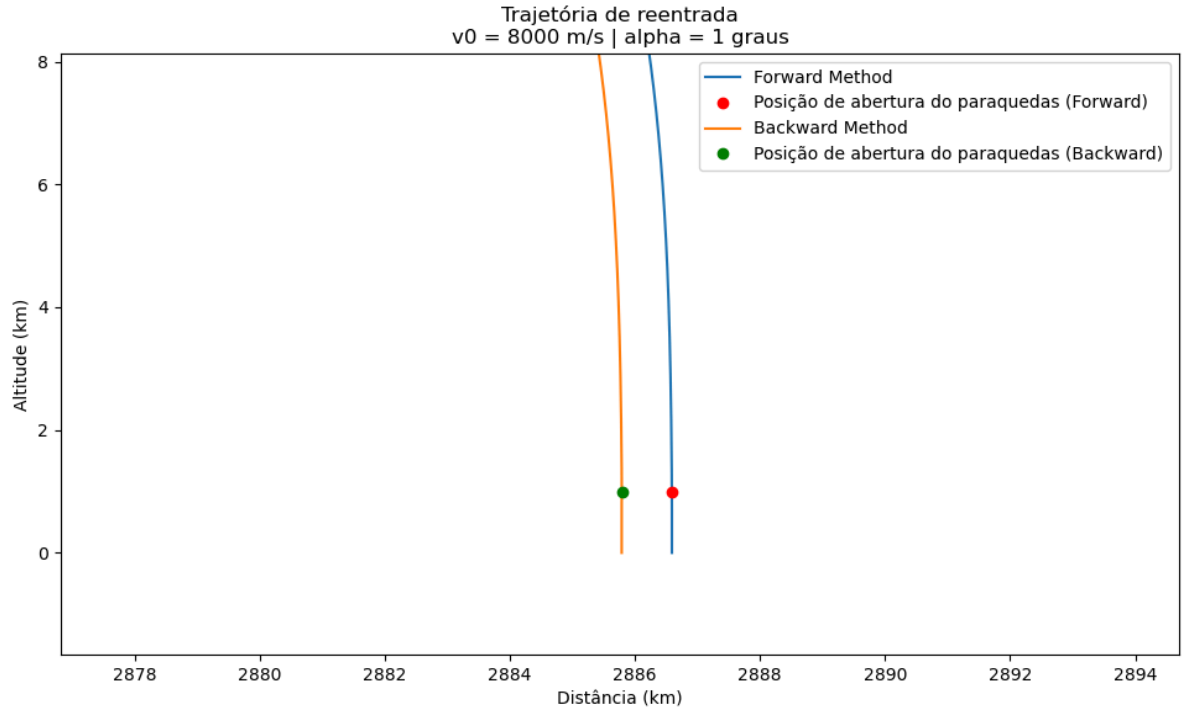

Figura 2: Ampliação do Gráfico da Trajetória de Reentrada, na Zona Vertical

Verificamos, deste modo, que não existe sobreposição entre as trajetórias, com o paraquedas a ser aberto, para uma distância percorrida inferior, com o método *backward*.

- **Velocidades**

Traçamos, igualmente, o gráfico das velocidades conseguido, com as duas simulações.

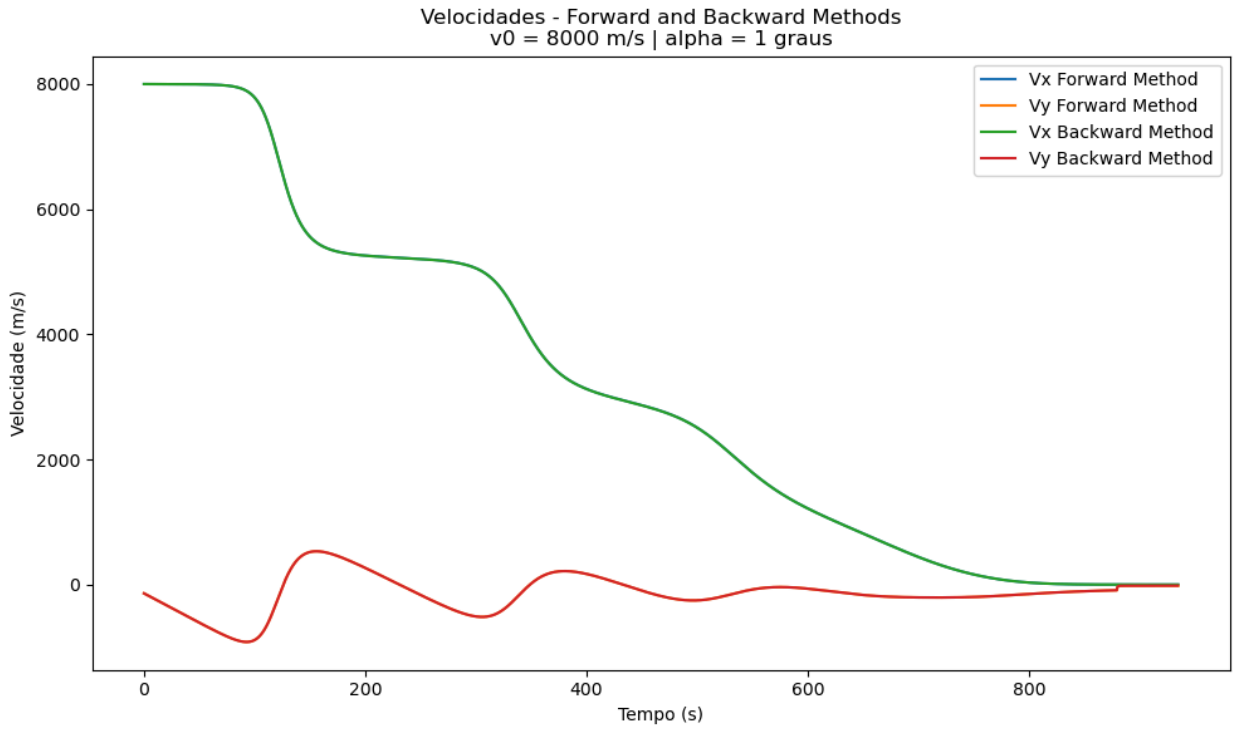

Figura 3: Gráfico das Componentes da Velocidade, obtidas pelos Dois Métodos

Também, aqui, parece haver uma total sobreposição entre os valores conseguidos. Fazemos, por isso, uma nova ampliação do gráfico.

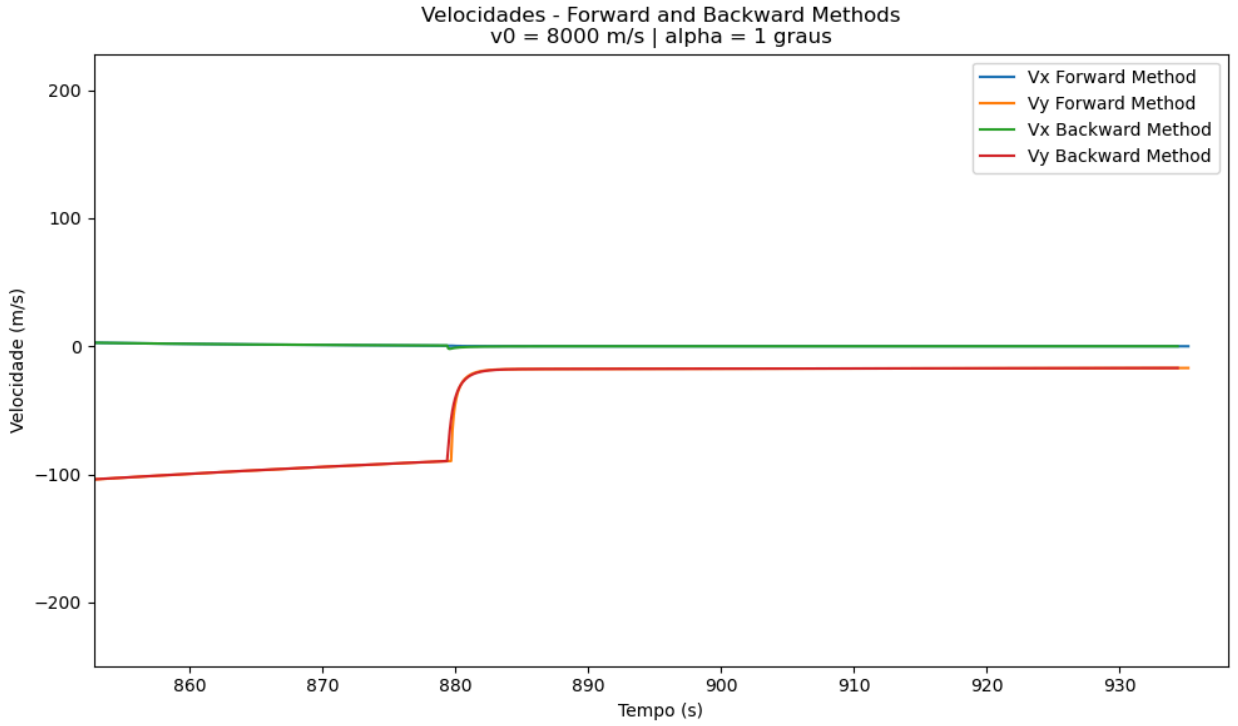

Figura 4: Ampliação do Gráfico das Componentes da Velocidade, na Fase Final da Trajetória

- **Acelerações**

Estudamos, agora, a influência do método usado, nos valores obtidos para as acelerações. O momento da abertura do paraquedas é claramente visível. O pico de aceleração do *Ay*, no método *forward*, é muito superior ao da simulação *backward*.

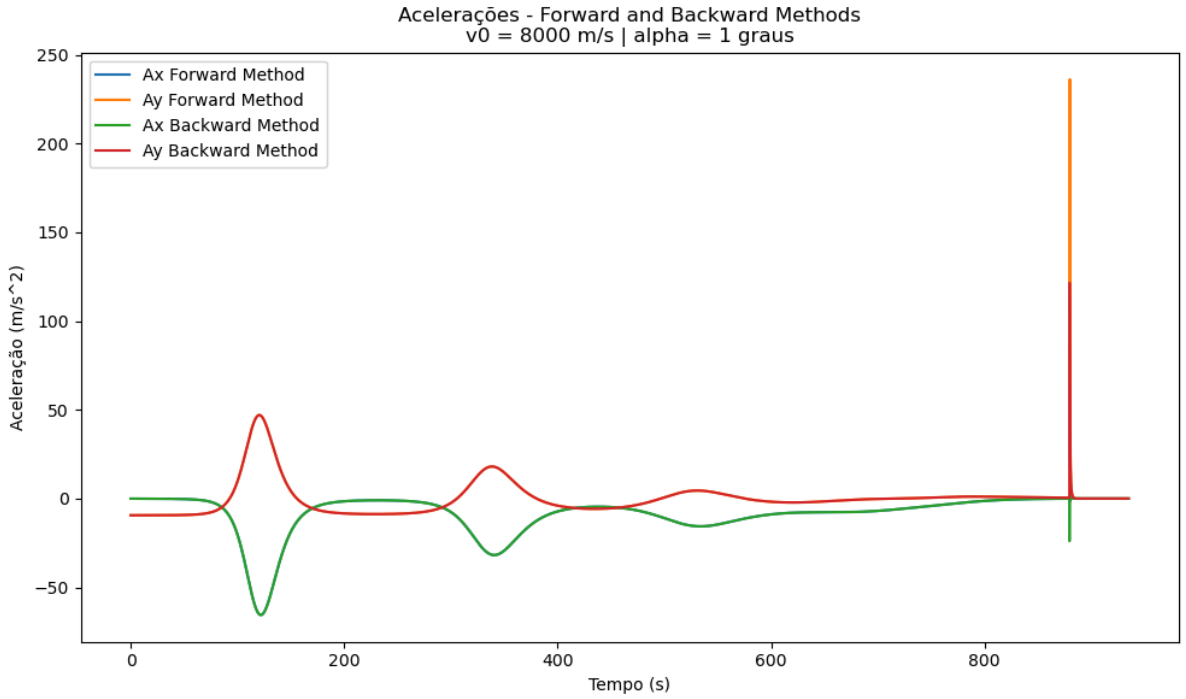

Figura 5: Gráfico das Componentes da Aceleração, obtidas pelos Dois Métodos

Ampliando o gráfico acima, na reta final da trajetória, obtemos o seguinte:

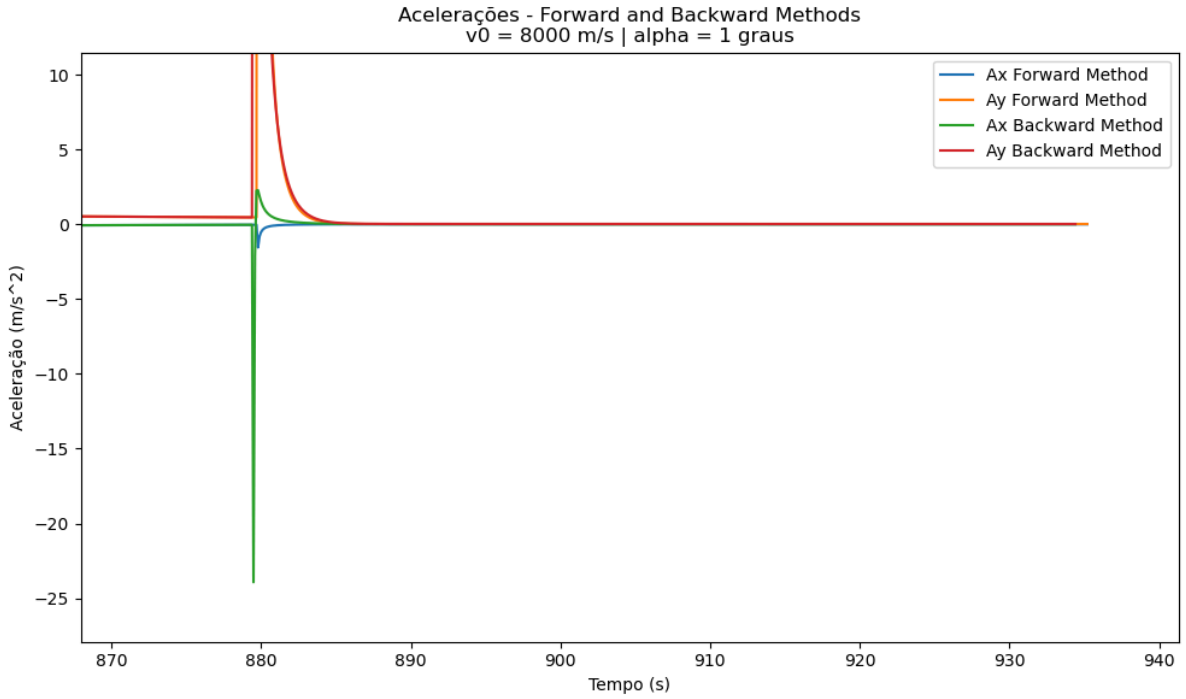

Figura 4: Ampliação do Gráfico das Componentes da Aceleração, na Fase Final da Trajetória

Se compararmos os valores numéricos obtidos, para a distância horizontal, velocidade final e aceleração da gravidade, com recurso aos dois métodos, deparamo-nos com diferenças mais significativas, quando a velocidade inicial assume um valor mais elevado. No entanto, fica clara a concordância entre os resultados conseguidos, pelas duas simulações.

- **Influência do Espaçamento nas Simulações**

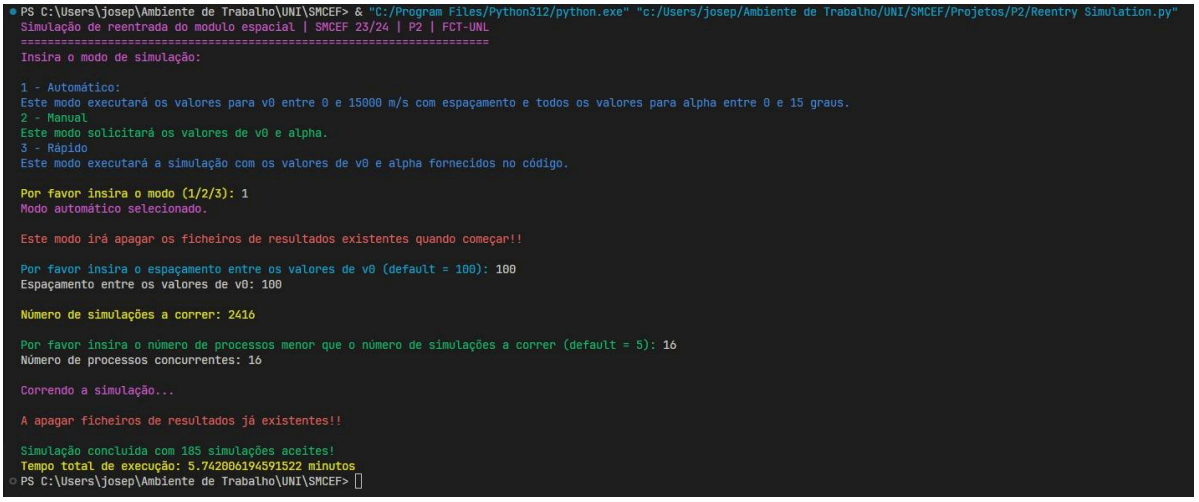

Figura 5: Resultado da Consola, para Espaçamento = 100

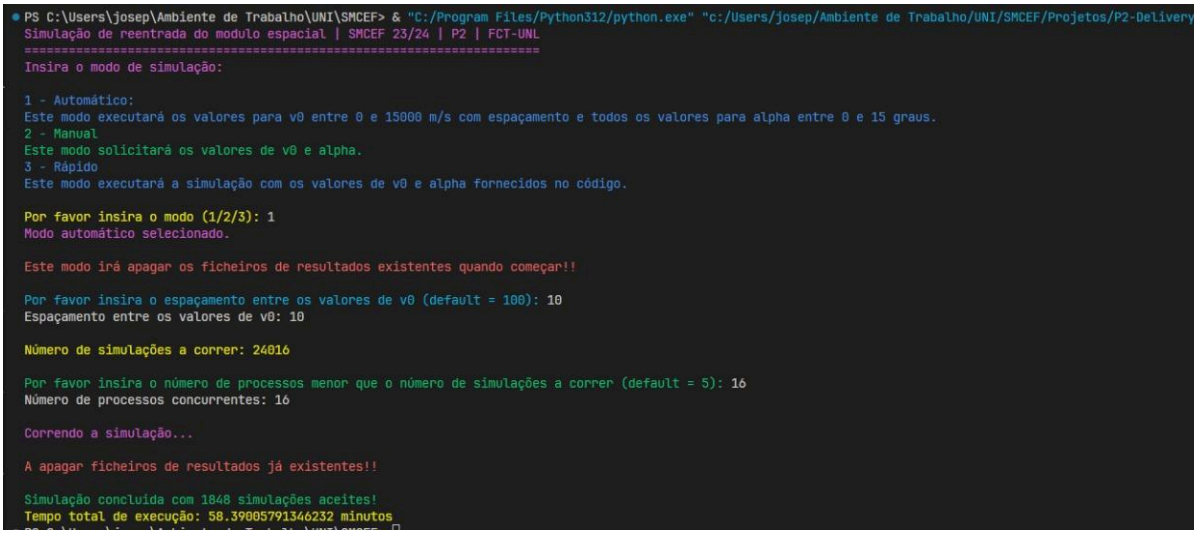

Figura 6: Resultado da Consola, para Espaçamento = 10

Como é possível observar nas figuras 5 e 6, o valor do espaçamento influencia diretamente a quantidade de sucessos obtidos, pois testa mais possibilidades. Como consequência, o tempo de execução vai ser maior.

- Tempo de execução para espaçamento **100** e pool com **16** processos: 5 minutos e 44 segundos

- Tempo de execução para espaçamento **10** e pool com **16** processos: 58 minutos e 23 segundos

Nota: Estes valores foram obtidos, usando um Intel Core i9-11900H (Núcleos físicos - 8, Threads - 16, Velocidade base - 2.5Ghz), com 32 Gb de RAM.

6. Conclusões

Neste projeto, desenvolvemos um modelo simplificado, para simular a reentrada de um módulo espacial, na atmosfera terrestre, tendo em consideração os parâmetros que garantem uma trajetória segura e dentro de certas restrições pré-definidas. Utilizámos métodos numéricos de integração, tanto explícitos quanto implícitos, para realizar as simulações. Aqui estão as principais conclusões:

- **Parâmetros de Reentrada**:

Identificámos os parâmetros de velocidade inicial v_0 e ângulo de descida α, que resultam numa reentrada segura. Estes parâmetros foram ajustados, de forma a garantir que a desaceleração máxima não excedesse $150\ m/s^2$ e que a velocidade de impacto, no oceano, fosse inferior a $25\ m/s$.

A distância horizontal projetada na superfície da Terra, variou entre $2500\ km$ e $4500\ km$, conforme estipulado, nas restrições do projeto.

- **Métodos Numéricos**:

- O método de Euler *forward* mostrou-se eficaz, graças à sua simplicidade e rapidez, na obtenção de soluções iniciais. Contudo, apresenta limitações de precisão, para intervalos maiores.

- O método de Euler *backward* e o Método dos Trapézios, ambos métodos implícitos, foram implementados, na tentativa de melhorar a precisão das soluções. Estes métodos revelaram-se mais robustos, especialmente, em simulações com intervalos de tempo maiores.

- **Principais Dificuldades Encontradas**

- Precisão na Densidade do Ar: a obtenção de uma função precisa, para a densidade do ar, em diferentes altitudes foi um desafio, dado que não existe uma equação simples e universal. Optámos por realizar um ajuste exponencial, para obtermos uma interpolação dos dados disponíveis.

- Integração Numérica: a escolha do tamanho do intervalo de tempo dt também levantou algumas questões. Intervalos de tempo maiores permitiam a execução mais rápida das simulações, comprometendo a precisão dos resultados. Por sua vez, o recurso a intervalos de tempo menores aumenta, significativamente, a carga computacional.

- Parâmetros de Modelo: a modelação das forças de arrasto e de sustentação, sobretudo, após a abertura do paraquedas, exigiu uma consideração cuidadosa dos coeficientes aerodinâmicos e das áreas de superfície, a considerar.

- Optámos por remover o quadrado da velocidade, no cálculo da força de resistência do ar, para evitar uma possível divisão por 0, no cálculo das acelerações.