Uma imagem com captura de ecrã, padrão, Saturação de cores, Simetria

Descrição gerada automaticamente

**Simulação e Modelação Computacional em Engenharia Física 2023/24 Projeto 1 - Ferromagnetism**

João Garção (Nº 62663), Fábio Silva (Nº 62730), Francisco Silva (Nº 63585)

NOVA School of Science and Technology - Universidade NOVA de Lisboa

**Maio de 2024**

**Índice**

Conteúdo

[1. Introdução 3](#_Toc167062169)

[2. Aplicação Desenvolvida 4](#_Toc167062170)

[2.1. Estrutura do código 4](#_Toc167062171)

[2.2. Utilização: ising\_main.py 4](#_Toc167062172)

[2.2.1. Escolher nível de detalhe 4](#_Toc167062173)

[2.2.2. Escolher tipo de teste, e implementação de código 4](#_Toc167062174)

[2.2.3. Escolher parâmetros do teste 5](#_Toc167062175)

[2.2.4. Correr o programa 5](#_Toc167062176)

[2.3. Tipos de Métricas e Testes Disponibilizados 6](#_Toc167062177)

[3. Análise de Código 10](#_Toc167062186)

[3.1. Implementação de Ising Model – Metrópolis Algorithm 10](#_Toc167062187)

[3.2. Implementações 12](#_Toc167062191)

[3.3. Implementações Base 2D e 3D, com multi-processing 12](#_Toc167062192)

[3.4. Dynamic Programming 16](#_Toc167062198)

[3.5. Cython 17](#_Toc167062200)

[3.6. Convolution - Scipy 17](#_Toc167062202)

[3.7. Numba 20](#_Toc167062206)

[4. Análise Física 22](#_Toc167062208)

[4.1. Testes e Parâmetros 22](#_Toc167062209)

[4.2. Temperatura Crítica 23](#_Toc167062210)

[4.3. Ferromagnetismo-Paramagnetismo Histerese 25](#_Toc167062211)

[5. Conclusões 28](#_Toc167062212)

[6. Bibliografia 29](#_Toc167062213)

# Introdução

.

# Aplicação Desenvolvida

Para maior versatilidade e reaproveitamento de código, e testar diferentes funcionalidades, começou-se por fazer app.

Dp explicito e implícito que importa todas as constantes e funções de app como air density

Para plots fez-se plots e todos importam…

## Correr programas

Para correr cada um dos programas, por exemplo no VS Code, basta abrir o documento respetivo, e.g. reentry\_app.py e clicar no botão “Run”. Para correr num terminal, ir para o diretório que tem o documento reentry\_app.py (“cd <path>”) e correr o comando: python reentry\_app.py.

O mesmo para os restantes documentos: reentry\_implicit.py, reentry\_explicit.py, reentry\_solver.py.

A primeira vez que se corre o programa é normal ocorrerem alguns erros caso o computador não tenha as bibliotecas necessárias instaladas na máquina. Nesse caso é preciso instalar, fazendo:

pip install numpy

pip install scipy

…

Quando o teste apresenta plots, é necessário fechar cada plot para depois ser apresentado o seguinte.

## Reentry\_app.py

Quando executando outros documentos como reentry\_implicit.py, os mesmos poderão ser executados sem qualquer configuração, estando prontos para correr o teste default pretendido.

Já o documento reentry\_app.py permite selecionar várias opções de simulação, com diferentes parâmetros, forças aplicadas, tipo de teste, etc. nomeadamente:

SHOW\_CYCLE\_DETAILS = False

…

Algumas das constantes e funções deste documento serão depois importadas pelos restantes, por forma a evitar repetição de código, nomeadamente funções como get\_air\_density(y).

## Métricas apresentadas

Quando correndo os vários testes é possível observar várias métricas, nomeadamente evolução do deslocamento horizontal vs altitude, bem como evolução da velocidade e da aceleração, ambas em relação à altitude e ao tempo. Exemplo de plots da evolução da posição e da velocidade vs altitude:

Uma imagem com texto, Gráfico, diagrama, file

Descrição gerada automaticamente

Ao observar os plots de velocidade ou aceleração vs altitude, no caso das simulações de reentrada, é importante observar o gráfico da direita para a esquerda, por forma a observar a altitude a diminuir.

Ao observar os gráficos por vezes o valor base aparece referenciado como notação científica, apresentando-se o gráfico apenas com o intervalo de valores mais específico, devendo-se ter então cuidado na análise dos dados. Por exemplo numa simulação em que observamos uma velocidade orbital com angulo 0, a altitude mantem-se a 130.000, com pequenas oscilações sem relevância, mas que no caso de um plot podem ser apresentados e parece muita variância. (ou vertical)

# Análise das Simulações

## Suposições

Como é normal, numa simulação não é possível contabilizar todas as forças existentes que poderão afetar o sistema a qualquer momento, tendo-se feito as simplificações/suposições descritas de seguida.

### Gravidade variável consoante a altitude

Pegando na segunda equação de Newton, temos que , sendo “m” a massa em kg do objeto e “a” a aceleração em m/s2.

No documento reentry\_app podemos selecionar a opção para utilizar a aceleração da gravida constante, ou seja …

No entanto a opção por defeito da gravidade ocorre recorrendo à Lei da Gravitação Universal de Newton, ou seja, variando consoante a distância entre os 2 objetos, neste caso, a distância entre o centro da terra e o objeto, sendo então a força definida pela fórmula seguinte, com “M” e “m” a massa da terra e do objeto em kg, “r” a distância entre o centro da terra e o objeto em m, e G, a constante gravitacional, que quantifica a intensidade da força gravitacional, com o valor de 6.67430×10−11 N⋅(m2/kg2):

No caso das simulações pretendemos avaliar a força gravitacional por forma a determinar a aceleração do objeto a cada momento, daí que, como sabemos que , então temos que a aceleração devido à força da gravidade “g” seja:

Esta força da gravidade é aplicada apenas sobre a componente y, fazendo com que o y seja reduzido, ou seja levando à redução da altitude, daí que se subtraia a aceleração da gravidade à restante aceleração do objeto, na sua componente y:

ay - = g

é normal acelerações terminais e velocidades serem todas iguais, porque estabilizamos ao fim de uns segundos. P.ex. se metermos a altitude inicial 1\_000 e sem paraquedas já dá acelerações terminais bem diferentes porque não tem tempo de estabilizar. O mesmo acontece se iniciarmos a 130\_000 e abrirmos o paraquedas muito perto do chão, ou seja não estabiliza a tempo…

**Densidade do Ar**

**Lift**

Lift tem sempre influencia em y a 100%.

O que resulta num lift superior ao real… como podemos ver numa simulação de projetil com lift real é igual a não ter drag… e com lift a somar sempre a 100% a y vai sempre a subir… é como se tivesse motor…

O programa corre por default com lift a 100% pro y… e caso se pretenda testar com lift mais real é escolher opção.

De qualquer forma, este lift real é também ele sempre positivo. Assume-se que a capsula mantém a sua posição, não rodando (banking…??) nem fazendo loops no ar quando velocidade é muito grande…

O que faz sim é nose up e down e assim o lift vai ter maior impacto na componente x ou y do referencial (não da capsula pois aí é sempre para y) mas nunca controlado por nós. O que é diferente por exemplo do space shuttle em que pode ajustar nose up ou down consoante velocidade, aceleração, temperatura, etc… e aqui vamos ao sabor das acelerações que a capsula sofre ao longo do tempo

**Uma imagem com file, texto, diagrama, design

Descrição gerada automaticamente**

Figure 2.2: Representation of an Aircraft’s Pitch Angle, adapted from [25].

[1]

A. Miguel and P. Moreira, “Optimal and Robust Control of Atmospheric Reentry Trajectories (Versão corrigida após defesa).” Accessed: Jul. 02, 2024. [Online]. Available: https://ubibliorum.ubi.pt/bitstream/10400.6/13015/1/8802\_19157.pdf

‌

‌

In Sections 4.1.7.1 through 4.1.7.3, we assumed the force of lift on our reentering vehicle was zero, so we could more simply investigate the tradeoffs between re-entry characteristics. Adding lift to the problem takes it beyond the scope of our simple model but gives us more flexibility. For example, we can use the lifting force to “stretch” the size of the corridor and allow a greater margin of error in re-entry velocity or angle. Controlling lift also improves accuracy over a strictly ballistic re-entry. We can change the vehicle’s angle of attack (angle between the vehicle’s nose and its velocity vector) to improve lift, making the vehicle fly more like an airplane than a rock. This allows the pilot or onboard computer to guide the vehicle directly to the desired landing area, as shown in Figure 4.1.7-23. The Space Shuttle is a great example of a lifting-re-entry vehicle. About one hour before landing, re-entry planners send the Shuttle crew the necessary information to do a deorbit burn. This burn changes the Shuttle’s trajectory to re-enter the atmosphere by establishing a –1° to – 2° re-entry flight-path angle. After this maneuver, the Shuttle is on “final approach.” Because it has no engines to provide thrust in the atmosphere, it gets only one chance to make a landing! Preparing to hit the atmosphere (just like a skipping stone), the Shuttle rotates its nose to a 40° angle of attack, that means the nose is pitched up 40° with respect to the velocity vector. This high angle of attack exposes it’s wide, flat bottom to the atmosphere. At an altitude of about 122,000 m (400,000 ft.), the re-entry interface takes place. Here the atmosphere begins to be dense enough for the re-entry phase to begin. From this point, more than 6400 km (4000 mi.) from the runway, the Shuttle will land in about 45 minutes! Figure 4.1.7-24 shows a graph of the Shuttle’s re-entry profile. Throughout re-entry, the Shuttle rolls to change lift direction in a prescribed way, keeping maximum deceleration well below 2 g’s. These roll maneuvers allow the Shuttle to use its lift to steer toward the runway. In contrast, Apollo and Gemini capsules had minimal lifting ability, so they re-entered much more steeply and didn’t roll much, so they endured up to 12 g’s. Figure 4.1.7-25 compares these re-entry profiles. Another exciting application of lifting re-entry is aerobraking, which uses aerodynamic forces (drag and lift) to change a vehicle’s velocity and, therefore, its trajectory. In Section 4.1.6 we explored the problem of interplanetary transfer, and we saw that to get from Earth orbit to another planet required us to use the spacecraft’s rockets twice: one ∆V to start the transfer at Earth and a second ∆V to capture it into orbit around the target planet. But if the target planet has an atmosphere, there’s another option. Instead of using engines to slow the spacecraft enough to enter a parking orbit, we can plan the hyperbolic approach trajectory to take it right into the atmosphere and then use drag to do the equivalent of the second ∆V burn. We then use its lift to pull it back out of the atmosphere before it crashes into the planet! By getting this “free” ∆V, we can save a huge amount of fuel.

Calculations show that using aerobraking, instead of conventional rocket engines, is almost ten times more efficient. This efficiency could mean a tremendous savings in the amount of material that must be put into Earth orbit to mount a mission to Mars. Figure 4.1.7-26 shows an artist’s conception of an aerobraking vehicle. In his novel 2010: Odyssey Two, Arthur C. Clarke uses aerobraking to capture a spaceship into orbit around Jupiter. The movie made from this novel dramatically depicts the aerobraking maneuver. Figure 4.1.7-27 shows an aerobraking scenario. On an interplanetary transfer, the spacecraft approaches the planet on a hyperbolic trajectory (positive specific mechanical energy with respect to the planet). During aerobraking, it enters the atmosphere at a shallow angle to keep maximum deceleration and heating rate within limits. Drag then reduces its speed enough to capture it into an orbit (now it has negative specific mechanical energy with respect to the planet). To “pull out” of the atmosphere, it changes its angle of attack, lift. Basically, the vehicle dives into the atmosphere, and then “bounces” out. In the process it loses so much energy that it is captured into orbit. This atmospheric encounter now leaves the vehicle on an elliptical orbit around the planet. Because periapsis is within the atmosphere, the vehicle would re-enter if it took no other actions. Finally, it completes a single burn, much smaller than the ∆V needed without the aerobraking to put the vehicle into a circular parking orbit well above the atmosphere.

Uma imagem com texto, captura de ecrã, file, diagrama

Descrição gerada automaticamente

Figure 4.1.7-25. Re-entry Profiles for the Shuttle Versus Gemini and Apollo.This graph shows the difference between re-entry profiles for Apollo, Gemini, and the Space Shuttle. Notice Gemini and Apollo re-entered much more steeply than the Space Shuttle. The Shuttle’s re-entry profile must stay within a tight corridor between equilibrium glide, which ensures it will slow enough to avoid skipping out and not over shoot the runway, and surface temperature/load factor requirements, which determine maximum heating and deceleration.

[1]

“Returning from Space: Re-entry.” Available: <https://16streets.com/39-B/PDF%20files/Re-entry_returning_from_space.pdf>

Forças com lift perpendicular

Force diagram

Uma imagem com file, diagrama

Descrição gerada automaticamente

[1]

A. Mazyku and H. Blunton, “OPTIMUM EARTH RE-ENTRY CORRIDORS.” Available: https://ntrs.nasa.gov/api/citations/19660008888/downloads/19660008888.pdf

‌Uma imagem com diagrama, file, Gráfico

Descrição gerada automaticamente

Lift and drag coeficients vs angle of attack

**Drag**

**Não há outras forças**

Não há vento - ar está parado, nã há força do vento( o que sobretudo após abertura do paraquedas tem sempre influencia...); então não é considerado essa influencia nem no drag nem no lift.

**Magnus force**

**Earth**

É esférica e força g é constante ao longo da superfície e não um geoide como de facto é.

Não se movimenta

Não roda, ex não há coriolis …

Se round earth atualizamos vetores de velocidade para apontarem na direção certa, e efetuar movimento na direção certa. Usando o make\_round\_earth com setp e sem / cos…

Mas posições mantemos na flat earth pra continuarmos a fazer as contas no step seguinte. O que guardamos sim é outra copia com a posição na round earth. E por isso, em vez de se guardar uma copia e fazer-se as contas ex earthangle\*RADIUS\_EARTH em cada step, com numpy é melhor vectorizar, e guardar so x,y da flat earth e converter no fim.

**Referencial de Coordenadas**

Y é a origem, podemos pensar que é o zenit que passa no polo norte desde o centro da terra; x é o zenit que passa no equador. Mas na pratica como consideramos a terra redonda estes eixos podem passar em qualquer ponto do globo.

Além disso na app podemos escolher um x diferente de 0.

Flat earth vs round earth

Podemos escolher que referencial…

Temos simulações sem força, ou seja só com vel horizontal assim vemos se o movimento na round earth continua a ser horizontal

E vertical, iniciando x fora do eixo y, ex 1000 metros ao lado e projetil a subir e descer, ou capsula a descer sem velocidade inicial, e assim tem de cair na vertical, e realmente confirma-se que cai… com margem de erro de… ??

**Paraquedas**

**Podemos ver que acelerações estabilizam em queda livre entre 130\_000 e (no caso de velocidade inicial 0, ou seja caindo na vertical apenas sob a força da gravidade), a partir do 40\_000 começa a ganhar mais aceleração com o drag e lift, e depois estabiliza novamente desta quando há equilíbrio entre drag e lift vs gravidade.**

**E quando abre paraquedas há grande aumento de aceleração na abertura, e depois estabiliza novamente (e rapidamente), mantendo uma velocidade praticamente constante até ao solo.**

**» pq aceleração dp de abrir pq não fica a 0, estabilizando??**

**Uma imagem com texto, diagrama, file, Gráfico

Descrição gerada automaticamente**

## Implementações

**App**

Pra testar código e diferentes resultados das várias opções de forças, ângulos, velocidades, etc.

**Projetil**

Para confirmer formulas aplicadas, e por forma a mais facilmente encontrar outras simulações disponíveis online, começou-se por implementar uma simulação de projétil, primeiro apenas com gravidade, depois com drag e finalmente com lift

**Round Earth vs Flat Earth**

O que pretendemos não é realmente obter as coordenadas de x e y na superfície da terra, mas sim obter a distancia horizontal em x e vertical em y.

Ex com opção round earth podemos confirmar que realmente com a velocidade orbital a altitude mantém-se constante (exceto uma pequena margem de erro);

Com velocidade de escape a altitude aumenta.

Round earth também permite ver diferença entre realmente aplicar modelo round earth vs apenas ir contando a componente x como round earth. No segundo caso, imaginando por exemplo que estamos na posição (R + 10\_000, R+10\_000), ou seja na diagonal, a 45º da origem, se aplicarmos um novo vetor de velocidade, por exemplo um vetor de velocidade horizontal, sem componente y; na flat earth, esse vetor realmetne é horizontal, mas na round earth, ser horizontal nessa posição na verdade significa estar a apontar 45º para baixo. Como ilustra de forma simplificada a figura seguinte:

Uma imagem com texto, file, diagrama, Gráfico

Descrição gerada automaticamente

Analisando as tarefas para converter flat earth em round earth:

1. Aplicamos conversão dos vetores de velocidade para depois serem utilizados para calcular a aceleração no passo seguinte:

vx\_step = ax \* dt

vy\_step = ay \* dt

vx += y\_step \* np.sin(earth\_angle) + x\_step \* np.cos(earth\_angle)

vy += y\_step \* np.cos(earth\_angle) - x\_step \* np.sin(earth\_angle)

# y\_flat is decreasing because we are going down

1. Calculamos a posição x e y normalmente (mantemos x e y na flat earth, mas tendo esta opção obtida através de movimentos da round earth):

x += vx \* dt

y += vy \* dt

1. Atualizamos o earth\_angle com base na nova posição. Este é o angulo formado pelo eixo y e a direção da linha que passa pela origem e pelo ponto. Podíamos considerar o eixo x como sendo o eixo de referencia, mas para melhor pensar o problema físico de reentrada, é bom ter o eixo y como a referência:

earth\_angle = np.radians(90) - np.arctan2(y, x)  # angle in origin from y axis to current position

1. Calculamos x e y na round earth (este valor é utilizado para observação do problema, mas não é utilizado para as iterações seguintes, utilizando-se antes x e y da flat earth do passo anterior).

Para calcular a distância x, utilizamos a formula trigonométrica de calculo do comprimento de um arco, ou seja raio \* angulo:

x\_round = RADIUS\_EARTH \* earth\_angle

Para calcular a altitude y, é no fundo a distancia desde a origem até ao ponto, logo podemos utilizar o teorema de Pitágoras, e subtrair o raio da terra para ficarmos com a altirude acima da superfície:

y\_round = (np.sqrt(x\*\*2 + y\*\*2) - RADIUS\_EARTH)

**Vários testes foram executados por forma a validar a implementação, nomeadamente:**

(meter uns plots)

Horizontal

Vertical

Vel orbital

G = constante gravitacional

M = massa da terra

R = Raio da Terra + Altura da orbita sobre a Superfície

Vel escape

O cálculo das posições finais ou ao longo de uma trajetória não é possível fazer através da resolução de uma equação de cinemática por um método analítico direto, uma vez que as acelerações não são constantes.

**Lift**

Com lift real em vez de ser todo totalmente somado a ay nota-se que a trajetória é mais suave, uma vez que por exemplo ao cair na vertical, fruto da gravidade, na pratica vai ter uma força lateral do lift evitando que caia mesmo na vertical, ou seja a trajetoria fica mais suave…

perpendicular ao movimento.

ex avião, lift puxa pra cima (só componente y)

ex se sky diver sai de um helicoptero sem qualquer velocidade inicial, e se mantiver o seu corpo com uma configuração simétrica tipo pena de badminton... (imagem)... vai cair na vertical (ex quando sky diver treina num tunel de vento e se mantem no centro do tunel de vento...). mas se mudar posição vai conseguir andar para os lados ou frente/tras devido ao lift que cria. e então saindo do heli não vai cair na vertical mas vai por exemplo andar para a frente, caso estique o corpo todo; e se usar wing suit isso vai criar ainda mais lift, parecendo uma pequena asa de avião e vai andar ainda mais na horizontal... e se abrir o paraquedas, então cria ainda mais lift e consegue andar ainda mais na horizontal, considerando que não há vento...

testar lift na simulação vertical pra ver se vai pró lado

Então implementou-se integração para a frente (Forward Euler) bem como Backward Euler, como descrito de seguida…

E o solver\_ivp com o kutpa… ODE45 ???

Sistema de ODEs (ordinary differential equation), e resolução com solver, neste caso usando scipy.integrate.solve\_ivp . Para o efeito definimos as funções que constituem o sitema de ODEs, garantindo que apenas utilizamos derivadas de 1º grau. Definimos condições iniciais, especificamos o time span, chamamos o solver para encontrar as soluções, filtramos resultados como pretendido e fazemos plot de resultados.

Fez-se 4 implementações:

» tipo app pra ser fácil perceber e testar diferentes opções

» foward (com arrays e multi processing e numba)

» dp meter num vetor e criar logo arrays de tamanho grande e se for preciso crescer, então duplica (ou meter limite igual ao max time da simulação \* dt… e ver se é mais rápido

» backward (com arrays e multi processing e numba)

Solver\_ivp pra comparar resultados

Tempos

|  |  |
| --- | --- |
| **Implementação de Código** | **Tempo do teste** |
| **2D\_MP** | 26.97 segundos (neste caso a rede é apenas uma matriz de 20\*20) |
| **3D\_MP** | 611 segundos |
| **3D\_MP\_DynamicProg** | 674 segundos |
| **3D\_MP\_Cython** | 415 segundos |
| **3D\_MP\_Convolution** | 34.6 segundos |
| **3D\_MP\_Numba** | 16.8 segundos |

Para se analisar os pontos críticos do código, correram-se os testes recorrendo à ferramenta cProfile que permite recolher informações sobre o tempo consumido por cada função ao longo da execução. Para correr o programa com esta ferramenta basta correr no terminal:

python -m cProfile -s tottime ising\_main.py > out.txt

Numba é um compilador JIT (Just in Time) para python que usa LLVM (Low-Level Virtual Machine) para compilar funções python para um código máquina altamente otimizado. Muito útil para cálculos numéricos e loops.

Para usar esta ferramenta basta instalar a biblioteca, fazer o import, e decorar as funções que queremos com @numba.jit

Para forçar a compilação e desativar o modo de interpretação do python (que interpreta e compila o código apenas à medida que vai precisando), podemos utilizar a anotação @numba.njit (no python JIT), permitindo um desempenho ainda maior.

Algumas funções python e numpy não são suportadas pelo numba, sendo preciso testar bem o código para confirmar que não usamos funções não suportadas, como e.g.:

np.random.choice ([options], (shape), p=[probabilities])

**Resultados para o teste de grid = 20x20x20; ciclos MC = 10.000, com a versão 3D Numba.**

Uma imagem com texto, captura de ecrã, software, Tipo de letra

Descrição gerada automaticamente

Tal como cython, como numba compila o código python em código máquina, a ferramenta cProfiler não permite identificar pontos de pior desempenho. No entanto, com a pequena tarefa de fazer import de numba e colocar as anotações nas funções, conseguimos executar o teste que de outras formas demorou cerca de 10 minutos, em apenas cerca de 17 segundos.

Precisão:

### Forward Euler (explicit)

Integração para a frente, em que, em cada ponto, analisamos o campo vetorial nessa posição, e tendo esse vetor extrapolamos que essa é a derivada correta e executamos o movimento para a próxima posição. Ou seja para calcular o próximo ponto, usamos a informação vectorial do ponto presente, na forma geral:

\* S é o estado, o qual pode ser representado por um conjunto de várias ODEs, exemplo a posição, a velocidade, a aceleração, etc.

Isto pode criar erros de “overshoot” face à solução exata, apresentando normalmente uma ondulação das curvas mais expressiva que a solução exata.

Caso o step usado (dt) seja demasiado grande podemos entrar em estados instáveis em que as soluções podem por exemplo oscilar a tender para o infinito, afastando-se assim da realidade. Por isso dt deve ser o menor possível, adequando-se à capacidade computacional que temos para resolve ro problema.

### Backward Euler (implicit)

Ao contrário do método explícito, em que calculamos o estado n+1 usando a informação do estado n, aqui vamos usar a própria informação do estado n+1. Ou seja vamos ter estas variáveis n+1 em ambos os lados das equações e por isso se conhece como método implícito.

The Euler implicit method, also known as the backward Euler method, is a numerical technique used to solve ordinary differential equations (ODEs). Unlike the explicit Euler method, which calculates the state of a system at the next time step using the current state, the implicit method involves solving an equation that depends on the unknown future state. This generally makes it more stable, especially for stiff equations.

**Podemos resolver este esquema de equações implícitas usando fzero ou fsolve em matlab;**

**Ou então newton… achar raízes…**

Isto pode criar erros de “undershoot” face à solução exata. Tal como no metodo

É geralmente mais estável que a integração para a frente, uma vez que tende a manter oscilações abaixo da solução exata, e então não tendem a afastarem-se para o infinito como a integração para a frente, nos casos em que usamos um step muito grande e/ou temos picos??

1. Stability:

o Explicit Method: Can be unstable for stiff equations or large time steps.

o Implicit Method: Generally more stable, especially for stiff equations, allowing larger time steps.

2. Computational Effort:

o Explicit Method: Easier to implement, directly calculates the next step.

o Implicit Method: Requires solving an implicit equation at each step, which can be computationally intensive.

### Solver ivp

Uma forma de conseguir melhores resultados é combinado os 2 métodos anteriores. Por exemplo fazendo uma média entre o estado de n com n+1:

Tendo integração para a frente:

Fazemos média com método implícito:

Com

Isto é conhecido como o método de Euler modificado, e é a primeira aproximação para métodos mais precisos como o método Runge-Kutta ('RK23', 'RK45', 'DOP853'), ou 'Radau' ou 'BDF' para problemas com picos.

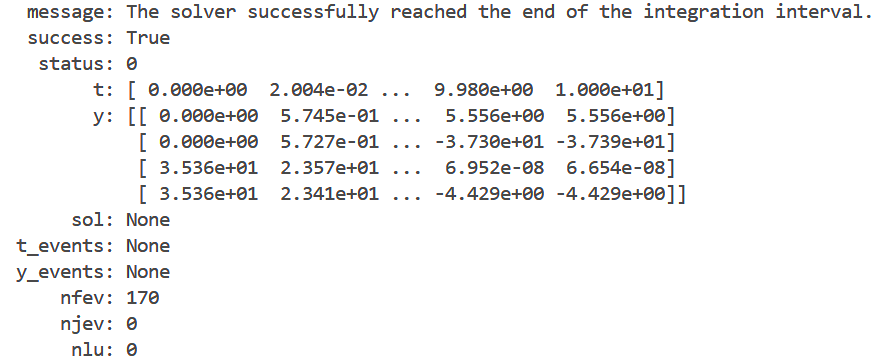
Em python temos estes métodos disponíveis na biblioteca: scipy.integrate.solve\_ivp, o que torna a implementação deste tipo de sistemas mais fácil, e precisa. Basta então, definir as funções ODE’s (todas de 1º nível), em relação a uma variável, exemplo tempo ou outra qualquer. Escolher condições iniciais para as variáveis dependentes. Especificar o tempo durante o qual queremos simular e a precisão, ex:

t\_eval = np.linspace(0, 5, 100)

Chamar o solver para resolver o ODE e obtermos os resultados, ex:

solution = solve\_ivp(ode\_functions, t\_span, y0, t\_eval=t\_eval)

O método por defeito do solver é o method='RK45', mas podemos passar outros métodos apresentados acima. O solver retorna um dicionário com vários campos, incluindo os tempos para os quais foram calculada a simulação, e a simulação que pretendemos. O nome que dão ao estado ou seja ao sistema de funções derivadas é y, então é nesse campo y que acedemos a cada função derivada que definimos (s.y[0] - resultados para a 1ª função; s.y[1] - resultados par a 2ª função, etc). Ex de solution returnada:



...

# Análise Física

## Testes e Parâmetros

Para estudo dos fenómenos seguintes recorreu-se a 2 testes disponíveis na aplicação, os quais foram executados com os seguintes parâmetros:

**4 - EVOLVING\_MAG\_FIELD\_TEST**

ão bem global da resposta do sistema perante diferentes situações.

Para a rede 2D realizaram-se testes com valores semelhantes, aumentando-se o tamanho da rede e ajustando-se os valores próximos da temperatura crítica.

## Distância Horizontal

Podiamos ir aplicando logo o angulo para calcular dist horizontal iterativamente, mas x não faz diferença nas contas das acelerações então podemos fazer só no fim assim fica operação vectorizada mais eficiente antes de se fazer o plot.

Uma imagem com texto, diagrama, file, captura de ecrã

Descrição gerada automaticamente

Bounds on entry flight-path: angle vs entry speed

Uma imagem com texto, esboço, diagrama, desenho

Descrição gerada automaticamente

Entry arcs for re-entry speed 7833 m/s (for manned

Uma imagem com diagrama, file, Gráfico, texto

Descrição gerada automaticamente

A typical re-entry altitude vs time history

[1]

A. Mazyku and H. Blunton, “OPTIMUM EARTH RE-ENTRY CORRIDORS.” Available: https://ntrs.nasa.gov/api/citations/19660008888/downloads/19660008888.pdf

‌

# Conclusões

Neste projeto foi implementada uma simulação de ferromagnetismo utilizando o modelo de Ising, por forma a observar 2 objetivos principais, nomeadamente a Temperatura de Curie e a Histerese ferromagnética-paramagnética.

O objetivo inicial passava por desenvolver 2 implementações para um teste específico, tendo-se optado no entanto por desenvolver uma aplicação que permita observar como um todo o fenómeno do ferromagnetismo, através de vários tipos de testes. Estes testes são importantes para perceber como realmente uma rede evolui perante diferentes temperaturas e campos magnéticos externos, servindo assim de base para o estudo dos 2 objetivos principais do projeto.

Em termos de implementações, acabou-se por testar diferentes aproximações e explorar várias opções disponíveis atualmente de ferramentas de otimização, que permitem de uma forma mais ou menos simples melhorar em muito a performance do código desenvolvido, nomeadamente através de compiladores para linguagem c (Cython) ou mesmo máquina (numba); ou bibliotecas que disponibilizam funções vetorizadas, nomeadamente Scipy e numpy, o que aumenta muito a performance ao conseguirmos deste modo rentabilizar as VPUs ou SIMD dos CPUs e efetuar operações sobre todo um vetor, em vez de realizar essas operações valor a valor. Além destas, uma otimização sempre importante a considerar é o multiprocessamento, para permitir a rentabilização dos vários cores da máquina que possuímos. A estrutura global do código permite boa extensibilidade para futuras funcionalidades que se queiram implementar.

Quanto às várias simulações implementadas, as mesmas permitiram observar e confirmar os fenómenos da temperatura de curie, a obtenção dos seus valores específicos, e a diferença que se verifica entre redes 2D e 3D, nomeadamente com a temperatura de curie de 26.97 para redes 2D versus 4.51 para redes 3D. Quanto à histerese ferromagnética-paramagnética ficaram igualmente claros os efeitos deste fenómeno, visíveis pelas curvas de evolução do momento magnético perante diferentes campos magnéticos externos, observáveis, por exemplo no teste 4 (EVOLVING\_MAG\_FIELD\_TEST) e 5 (ALL\_TEMP\_AND\_MAG\_FIELD\_TEST), tendo-se assim confirmado os valores teóricos esperados.

O desenvolvimento deste projeto foi desafiante, tanto em termos do domínio da física e ferromagnetismo, como em termos computacionais. Em termos do ferromagnetismo, existe muita literatura sobre o assunto, sendo no entanto difícil por vezes encontrar uma explicação do fenómeno com fórmulas matemáticas simples e otimizadas, que facilitem assim uma implementação mais eficiente. Além disso, nem sempre se encontram modelos e gráficos que demonstram o efeito esperado para diferentes valores de input e parâmetros, o que torna a avaliação do código gerado mais difícil. Uma possível forma de melhor apoiar os grupos que desenvolvem projetos deste tipo, será por exemplo fornecer uma pequena lista com alguns valores de input e parâmetros e quais os valores esperados para cada uma das métricas, possivelmente com uma margem de erro associada. Isto tornaria mais fácil confirmar o código à medida que se vai desenvolvendo.

Em termos do domínio da computação, foi exigente explorar todas as ferramentas de otimização, não sendo óbvio por vezes quais os pormenores que temos de ter em atenção para que a simulação continue a ser correta. Por exemplo o aspeto de ser mais difícil executar operações sobre pontos aleatórios da rede usando as funções da biblioteca scipy como convolution. Um desenvolvimento futuro interessante de executar será ainda implementar a simulação recorrendo ao GPU, nomeadamente através de bibliotecas disponíveis atualmente como Cuda, para placas gráficas Nvidia, etc.

De uma forma geral, o projeto apresentou um nível de desafio considerável, sendo gratificante observar que bons resultados foram alcançados.

# Bibliografia

[1]

“Returning from Space: Re-entry.” Accessed: Jul. 02, 2024. [Online]. Available: https://16streets.com/39-B/PDF%20files/Re-entry\_returning\_from\_space.pdf

‌

[1]

J. Orlando *et al.*, “TRAJECTORY AND ATTITUDE MODELING AND PROPAGATION FOR REENTRY DEBRIS WITH FRAGMENTATION IMPLEMENTING VOXELS MESHS.” Available: http://mtc-m21c.sid.inpe.br/col/sid.inpe.br/mtc-m21c/2018/05.02.13.07/doc/publicacao.pdf

‌

[1]

“youtube\_channel/Python Metaphysics Series/vid22.ipynb at main · lukepolson/youtube\_channel,” *GitHub*, 2024. https://github.com/lukepolson/youtube\_channel/blob/main/Python%20Metaphysics%20Series/vid22.ipynb (accessed Jul. 02, 2024).

‌

[1]

“lukepolson/youtube\_channel: Notebooks for the python tutorials of my youtube channel. See specific youtube video for link to specifc notebook.,” *GitHub*, 2024. https://github.com/lukepolson/youtube\_channel (accessed Jul. 02, 2024).

‌ [1]

“youtube\_channel/Python Metaphysics Series/vid22.ipynb at main · lukepolson/youtube\_channel,” *GitHub*, 2024. https://github.com/lukepolson/youtube\_channel/blob/main/Python%20Metaphysics%20Series/vid22.ipynb (accessed Jul. 02, 2024).

‌

[1]

Miscellaneous Bits, “Simulating Rocket Trajectories with Python,” *YouTube*. Apr. 21, 2023. Accessed: Jul. 02, 2024. [YouTube Video]. Available: https://www.youtube.com/watch?v=22OCPbfY5SE

‌

[1]

A. Miguel and P. Moreira, “Optimal and Robust Control of Atmospheric Reentry Trajectories (Versão corrigida após defesa).” Available: https://ubibliorum.ubi.pt/bitstream/10400.6/13015/1/8802\_19157.pdf

‌ [1]

“ferrolho/space-shuttle-reentry-trajectory: Space Shuttle Reentry Trajectory,” *GitHub*, 2024. https://github.com/ferrolho/space-shuttle-reentry-trajectory (accessed Jul. 02, 2024).

‌

[1] P. Gallais, *Atmospheric Re-Entry Vehicle Mechanics*. Berlin: Springer, 2007.

[1]

D. Chapman, “AN ANALYSIS OF THE CORRIDOR AND GUIDANCE REQUIREMENTS FOR SUPERCIRCULAR ENTRY INTO PLANETARY ATMOSPHERES.” Accessed: Jul. 02, 2024. [Online]. Available: https://ntrs.nasa.gov/api/citations/20040030504/downloads/20040030504.pdf

‌

[1]

J. Orlando *et al.*, “TRAJECTORY AND ATTITUDE MODELING AND PROPAGATION FOR REENTRY DEBRIS WITH FRAGMENTATION IMPLEMENTING VOXELS MESHS.” Accessed: Jul. 02, 2024. [Online]. Available: http://mtc-m21c.sid.inpe.br/col/sid.inpe.br/mtc-m21c/2018/05.02.13.07/doc/publicacao.pdf

‌

**Ising model:**

H. Gould, J. Tobochnik, and D. E. Harrison, “An Introduction to Computer Simulation Methods: Applications to Physical Systems, Part 1 and Part 2,” *Computers in Physics*, vol. 2, no. 1, pp. 90–91, Jan. 1988, doi: https://doi.org/10.1063/1.4822668.

‌

“Parallelization and implementation of multi-spin Monte Carlo simulation of 2D square Ising model using MPI and C++,” *ar5iv*, 2024. https://ar5iv.labs.arxiv.org/html/1811.04384 (accessed May 20, 2024).

‌“IsingModel2D\_MonteCarlo/Ising2D.ipynb at master · lorenzomancini1/IsingModel2D\_MonteCarlo,” *GitHub*, 2024. https://github.com/lorenzomancini1/IsingModel2D\_MonteCarlo/blob/master/Ising2D.ipynb (accessed May 18, 2024).

‌“IsingModel,” *Github.io*, 2024. https://rajeshrinet.github.io/blog/2014/ising-model/ (accessed May 18, 2024).

‌J. Vanderplas, “Optimization with Cython: Ising Models (Part 1),” *YouTube*. Dec. 11, 2017. Accessed: May 18, 2024. [YouTube Video]. Available: <https://www.youtube.com/watch?v=rN7g4gzO2sk>

“youtube\_channel/Python Metaphysics Series/vid14.ipynb at main · lukepolson/youtube\_channel,” *GitHub*, 2024. https://github.com/lukepolson/youtube\_channel/blob/main/Python%20Metaphysics%20Series/vid14.ipynb (accessed May 18, 2024).

‌“GPU-based single-cluster algorithm for the simulation of the Ising model : Yukihiro Komura : Free Download, Borrow, and Streaming : Internet Archive,” *Internet Archive*, Jan. 09, 2012. https://archive.org/details/arxiv-1110.0899/page/n13/mode/2up (accessed May 18, 2024).

[1]

A. Mazyku and H. Blunton, “OPTIMUM EARTH RE-ENTRY CORRIDORS.” Available: https://ntrs.nasa.gov/api/citations/19660008888/downloads/19660008888.pdf

‌

‌