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

Esta

## Estrutura do código

Por

## Utilização: ising\_main.py

O ficheiro ising\_main.py é o documento de ligação com o utilizador. Aqui ele escolhe os parâmetros e testes que pretende correr, e executa o método main(), iniciando então o programa.

Para tal deve seguir os 4 passos seguintes:

### Escolher nível de detalhe

SHOW\_CYCLE\_DETAILS = False

SHOW\_TEST\_DETAILS = False

### Escolher tipo de teste, e implementação de código

TEST\_TO\_RUN = 1

Escolher entre 1 a 5, consoante o tipo de teste que pretende:

SINGLE\_TEST = 1

SINGLE\_MAG\_FIELD\_TEST = 2

EVOLVING\_TEMP\_TEST = 3

EVOLVING\_MAG\_FIELD\_TEST = 4

ALL\_TEMP\_AND\_MAG\_FIELD\_TEST = 5

CODE\_VERSION = 4

Escolher entre 1 a 6 consoante o tipo de teste que pretende. A implementação 1 é para 2D e as restantes para 3D. De entre estas, as implementações mais rápidas são a 5 e 6:

MULTI\_PROCESSING\_2D = 1

MULTI\_PROCESSING\_3D = 2

DYNAMIC\_PROGRAMMING\_3D = 3

CYTHON\_3D = 4

CONVOLUTION\_3D = 5

NUMBA\_NJIT\_3D = 6

### Escolher parâmetros do teste

Exemplo para SINGLE\_TEST, pode escolher entre diversas opções, nomeadamente:

    GRID\_SIDE = 100

    MC\_CYCLES = 10\_000

    INITIAL\_GRID\_SPIN = -1    # -1=down, 1=up, 0=random, +.decimal, -.decimal

    SINGLE\_TEST\_TEMPERATURE = 5.5

    SINGLE\_TEST\_MAGNET\_FIELD = 0

SHOW\_GRID\_EVOLUTION = False     # don't choose this for big grids, it will slow the simulation

    SNAP\_SHOTS\_TO\_PLOT = [("initial grid", 0), … , ("final grid", MC\_CYCLES)]

### Correr o programa

Para correr o programa, por exemplo no VS Code, basta abrir este documento ising\_main.py e clicar no botão “Run”. Para correr num terminal, ir para o diretório que tem o documento ising\_main.py (“cd <path>”) e correr o comando: python ising\_main.py

A primeira vez que corre o programa é normal ocorrerem alguns erros caso o computador não tenha as bibliotecas necessárias instaladas na máquina. E.g. correndo a implementação com numba, se não tivermos essa biblioteca instalada, vai aparecer o erro:

File "c:\...\grid\_functions\grid\_functions\_3D\_Numba.py", line 4, in <module>

import numba

ModuleNotFoundError: No module named 'numba'

Nesse caso é preciso instalar, fazendo:

pip install numba

pip install numpy

pip install cython

pip install scipy

…

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

## Tipos de Métricas e Testes Disponibilizados

Consoante o tipo de teste há diferentes métricas disponibilizadas que podemos observar, nomeadamente:

# Análise de Código

## Implementação de Reentry

Várias considerações:

- terra redonda

- aceleração da gravidade consoante altura

## Implementações

De seguida serão descritas as diferentes implementações do algoritmo presentes na aplicação.

Com fórmula analítica

Implicito

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.

(dp meter numba pra ver se fica mais rápido)

Para se comparar o desempenho de cada implementação, executou-se o SINGLE\_TEST, efetuando-se então para um par de temperatura e campo magnético, 10\_000 ciclos de Monte Carlo, sobre uma rede/cubo de 20\*20\*20.

|  |  |
| --- | --- |
| **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

## Implementações Base 2D e 3D, com multi-processing

Desde logo nas implementações base, utilizam-se um conjunto de otimizações mínimas para conferir um bom nível de performance do código, nomeadamente:

### Multi-processing

Todo o procedimento descrito na secção anterior é repetido para cada teste, que, de uma forma não otimizada demoraria bastante tempo a executar. A fim de diminuir esse tempo de execução, optámos por implementar multiprocessamento da seguinte forma: inicialmente calcula-se o número de tarefas a realizar, que irá depender do número de Temperaturas reduzidas e do número de campos Magnéticos externos do teste. De seguida, usufruindo da biblioteca multiprocessing, calcula-se um número máximo entre 1 e o número total de CPUs da máquina – 2, e divide-se o número de tarefas igualmente pelo número de CPUs.

Por forma a diminuir a memória necessária para comunicação entre processos, nomeadamente através de uma queue onde cada processo coloca os resultados dos seus testes, foi importante implementar um mecanismo para ir retirando os dados da queue por forma a manter o seu tamanho reduzido, atribuindo-se esta tarefa ao processo principal:

…

        for process in processes:

            process.start()

        def process\_queue():

            while not tasks\_queue.empty():

                try:

                    task\_value, metrics = tasks\_queue.get()

                    idx\_task\_value = np.where(tasks\_values == task\_value)[0]

                    for metric\_idx, (metric\_name, metric\_matrix) in enumerate(results\_per\_metric.items()):

                        metric\_matrix[:, idx\_task\_value] = metrics[:, metric\_idx][:, np.newaxis]

                except mp.queues.Empty:

                    break

        while any(process.is\_alive() for process in processes):

            process\_queue()

            time.sleep(0.1)

        process\_queue()

        for process in processes:

            process.join()

### Tabulação

Analisando o código verifica-se que repetidamente é preciso fazer cálculos com as mesmas constantes. Então uma forma de otimizar é efetuar os cálculos uma vez no início do programa, e assim ao longo dos ciclos, em vez de se fazerem essas contas novamente, apenas se consultam os valores guardados inicialmente. Isto acontece para o cálculo da variação de energia, para confirmar se se muda o spin do ponto ou não, para cada posição da matriz, ao longo de todos os ciclos de Monte Carlo:

def check\_spin(grid, i, j, k, transition\_values\_w, mag\_field):

    curr\_spin = grid[i][j][k]

    delta , \_ , energy\_diff = get\_point\_energy(grid, i, j, k, mag\_field)

    if( energy\_diff < 0 or np.random.random() < transition\_values\_w[curr\_spin][delta]):

        grid[i][j][k] \*=  -1

Como os valores de probabilidade de transição são sempre iguais para cada par de temperatura e campo magnético, variando apenas consoante os spins do ponto em questão e dos vizinhos, e como esses spins são valores discretos {-1, 1}, então estamos perante um conjunto limitado de opções, que podemos então calcular logo no início do programa:

DELTA\_OPTIONS = [-6, -4, -2, 0, 2, 4, 6]

transition\_values\_w = {UP:{}, DOWN:{}}

for spin in transition\_values\_w.keys():

    for idx, delta in enumerate(DELTA\_OPTIONS):

        d\_sm = delta + (spin \* mag\_field)

        if (d\_sm <= 0):

            transition\_values\_w[spin][delta] = 1

        else:

            transition\_values\_w[spin][delta] = math.exp(-(2\*d\_sm) / temp)

### Funções de Alto Nível

Normalmente é preferível usar funções de alto nível da linguagem do que tentarmos nós programar tudo ao baixo nível, nomeadamente, a título de exemplo, apresentam-se os dois casos seguintes:

#### Exemplo simples:

Um simples exemplo pode ser percorrer um array e ir acedendo e/ou alterando os seus campos, por exemplo tendo um array com 100 chars:

arr\_of\_chars = [chr(i) for i in range(97, 97+100)]

Para percorrer o array, para aceder/alterar o valor de cada posição, podemos usar funções mais baixo nível, demorando, para 1.000.000 de ciclos, em média, 9.4 segundos:

for test in range(1\_000\_000):

    for i in range (len(arr\_of\_chars)):

        arr\_of\_chars[i] = chr(97+i)

Ou usar função built in de mais alto nível, demorando neste caso, em média, 7.4 segundos:

for test in range(1\_000\_000):

    for i, ch in enumerate(arr\_of\_chars):

        arr\_of\_chars[i] = ch

#### numpy vs python

Para operações numéricas sobre arrays e matrizes é mais eficiente usar biblioteca numpy, uma vez que esta biblioteca disponibiliza várias funções vetorizadas, que aproveitam as novas componentes dos CPU’s atuais que permitem executar operações sobre vetores, em vez de executarem apenas célula a célula.

**Exemplo 1:** criando um array, atualizando todos os seus valores, e somando tudo no fim.

python = 2 segundos

py\_matrix = [[0] \* 1\_000 for \_ in range(1\_000)]

for i in range(1\_000):

    for j in range(1\_000):

        py\_matrix[i][j] = py\_matrix[i-1][j-1] - py\_matrix[i-1][j] + py\_matrix[i][j-1]

sum = sum(sum(py\_matrix, []))

numpy = 0.6 segundos

np\_matrix = np.arange(1\_000\_000).reshape(1\_000, 1\_000)

for i in range(1\_000):

    for j in range(1\_000):

        np\_matrix[i,j] = np\_matrix[i-1,j-1] - np\_matrix[i-1,j] + np\_matrix[i,j-1]

sum = np\_matrix.sum()

**Exemplo 2:** ir somando valores de magnetic moment em cada ciclo ou somar tudo no fim com uma função vectorized np.sum():

grid\_size = 1\_000

mc\_cicles = 50

for i in range(mc\_cicles):

|  |  |
| --- | --- |
| **10 segundos:** | **0.023 segundos:** |
| for i in range(grid\_size):          for j in range(grid\_size):              magnetic\_moment += grid[i][j] | magnetic\_moment += np.sum(grid[1:-1, 1:-1]) |

### Padding

Para evitar a execução de vários if’s ao percorrer cada posição da rede para ver se estamos nas fronteiras, então cria-se logo uma rede com um padding. No entanto, como ao percorrer uma linha, para calcular o spin da última posição, já vamos precisar de ter o padding atualizado com o spin obtido na 1ª posição, então temos sempre que fazer essa atualização na hora:

def check\_spin():

…

if( energy\_diff < 0 or np.random.random() < transition\_values\_w[curr\_spin][delta]):

        grid[i][j] \*=  -1

        if(i == 1):

            grid[-1][j] \*=  -1

        if(j == 1):

            grid[i][-1] \*=  -1

### Resultados

Através da implementação tendo em conta os aspetos descritos acima, entre outros, obtém-se os seguintes resultados:

**Resultados para o teste de grid = 20x20x20; ciclos MC = 10.000, com a versão 2D:**

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

Descrição gerada automaticamente

Analisando os resultados podemos concluir que o tempo necessário para percorrer os ciclos de Monte Carlo foi 26.97 segundos, sendo o total do teste de 36 segundos, incluindo-se aqui o tempo necessário de IO para apresentar os gráficos sobre as várias métricas calculadas para o teste.

Nas colunas tottime e ncalls podemos observar 11.8 segundos serviram para executar 80 000 000 de vezes a função get\_point\_energy. Apesar do tempo de execução desta função ser negligível, o número de vezes que a função é chamada faz com que se perca aproximadamente 12 segundos, o que representa que praticamente metade do tempo dos ciclos de Monte Carlo é dedicado a calcular a energia de cada ponto, primeiro para confirmar se se troca de spin ou não, e depois para contabilizar a energia do sistema.

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

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

Descrição gerada automaticamente

A diferença mais notória neste teste, devido à adição de mais uma dimensão à rede, é certamente o aumento significativo no tempo de execução, tanto no tempo dos ciclos de Monte Carlo, passando para 611 segundos (10 minutos e 11 segundos), como no tempo total, incluindo operações de IO para apresentação de resultados, num total de 617 segundos (10 minutos e 17 segundos)

De igual forma, grande parte do tempo é despendido no cálculo da energia de cada, nas funções get\_point\_energy e check\_spin, que ocupam um tempo somado de 529 segundos (8 minutos e 49 segundos), ou seja cerca de 86% do tempo de execução dos ciclos de Monte Carlo.

## Dynamic Programming

Para evitar a execução dos if’s dentro do método check\_spin(), como descrito acima, pode-se pensar uma solução baseada na lógica de Dynamic Programming. Nomeadamente, quando estamos a percorrer a matriz para verificar a alteração de spin, então calculamos logo no momento o momento magnético dessa posição [i,j], por exemplo. Para calculo da energia, como ao alterarmos o spin em [i,j], a última posição que não vai sofrer mais influência é a posição [i-1, j-1], então aproveitamos o percorrer a rede para fazer este somatório de energia, e no fim temos de calcular a energia dos últimos planos.

Esta metodologia é ainda benéfica para grandes estruturas de dados, nomeadamente quando o computador não tem capacidade de ter toda a matriz em memória RAM para acesso rápido. Então uma forma de otimizar é programar de forma a conseguir uma boa aproximação espacial e temporal. Ou seja, quando acedemos a uma linha de uma matriz, então fazemos tudo nesse momento.

### Resultados

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

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

Descrição gerada automaticamente

Numa tentativa de otimizar a versão anterior onde voltamos a percorrer a rede sempre que queremos calcular as métricas relativas a cada ciclo, acabou por não resultar numa melhoria do desempenho, chegando até a piorá-lo.

Nesta versão calculámos as métricas dentro do ciclo de Monte Carlo o que provoca um aumento no tempo por chamada de função 3 vezes superior quando comparado ao anterior, o que se reflete no tempo cumulativo e nos resultados obtidos.

2 justificações deste aumento do tempo em vez de melhorarmos eficiência poderá advir das estruturas de dados que utilizamos para as simulações não ser exageradamente grande, então o computador consegue ter a matriz toda em RAM e possivelmente em cache próxima ao CPU, daí que o acesso à matriz se consiga fazer rápido de qualquer forma. Além disso, ao percorrermos a matriz uma posição de cada vez nesta programação dinâmica e por exemplo ir somando o momento magnético, na prática cada soma do spin de cada posição é um pequeno conjunto de operações que tem de ser feito de forma sequencial ou com reduzido paralelismo no CPU, usando-se ALUs e registos simples. Ao passo que ao usar-se as funções vetorizadas do numpy, conseguimos aproveitar as valências dos CPUs modernos que tem hardware próprio para fazer operações sobre vetores, ou seja em vez de cada operação tratar de uma posição da matriz, com função vetorizada pode tratar de várias posições ao mesmo tempo.

## Cython

Cython é uma biblioteca que permite converter código python em código c de forma automática. Por exemplo tendo criado uma implementação qualquer num documento doc.py, podemos em seguida mudar o nome do documento para doc.pyx, correr no terminal, no diretório onde temos o documento cython\_setup.py, o comando:

python cython\_setup.py build\_ext --build-lib build

Isto vai gerar de forma automática um ficheiro doc.c o qual depois podemos importar diretamente para o nosso código python restante. Para compilação e import em tempo de execução, sem termos de efetuar configurações de “build” do projeto, é bom usar a biblioteca:

import pyximport

pyximport.install()

No documento de setup identificamos então os documentos para os quais pretendemos gerar código c:

from setuptools import setup

from Cython.Build import cythonize

setup(

    ext\_modules = cythonize("sub\_folder/…/doc.pyx")

)

### Resultados

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

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

Descrição gerada automaticamente

Como cython compila o código python em c, a ferramenta cProfiler não permite observar em que pontos exatos o código apresenta pior desempenho, mas podemos observar que com cython poupamos cerca de 3 minutos neste teste. Ou seja uma melhoria de 30%, apenas por pegarmos num ficheiro normal python, renomear como \*.pyx, e corrermos o cython setup para converter para linguagem c.

Melhorias mais significativas poderiam ser feitas ao atribuir o tipo de dados a cada variável, evitando assim o trabalho de determinação do tipo de dados que python faz de forma dinâmica, sempre que vai aceder a uma variável.

## Convolution - Scipy

Convolution é uma função da biblioteca scipy que permite fazer operações em toda a matriz ao mesmo tempo. Para programarmos para estas funções temos que deixar de pensar em loops que percorrem a matriz e pensar que quando escrevemos uma operação qualquer, isso está a ser aplicado a toda a matriz ao mesmo tempo, o que traz um grande aumento de performance.

### Teste de Opções Locais

Quando estamos a analisar localmente se algumas das opções realmente trazem maior performance, os resultados podem enganar, e realmente a performance é melhor se programarmos em mais baixo nível.

e.g. vendo os 2 casos seguintes:

#### Acesso direto à matriz - vs - kernel e sub\_matrix

e.g. para calcular (delta = somar os vizinhos \* centro), uma solução que à primeira vista parece elegante, será usando um kernel para encontrar os vizinhos e fazer slice da matriz. No entanto assim de forma isolada acaba por não ser tão eficiente. Fazendo o teste para um cubo de 100\*100\*100 e executando 10 ciclos Monte Carlo, temos:

kernel + sub\_matrix = 160 segundos:

kernel = np.array([[[0, 1, 0], [1, 0, 1], [0, 1, 0]],

                    [[1, 0, 1], [0, 0, 0], [1, 0, 1]],

                    [[0, 1, 0], [1, 0, 1], [0, 1, 0]]])

delta = spin \* np.sum(sub\_matrix \* kernel)

Acesso direto à rede = 85 segundos:

delta = grid[i][j][k] \* (grid[i-1][j][k] + grid[i+1][j][k] + grid[i][j-1][k] + grid[i][j+1][k] + grid[i][j][k-1] + grid[i][j][k+1])

#### Dicionário - vs - numpy com índices negativos

Como descrito acima, os cálculos dos valores de probabilidades de transição de spins são calculados no início do programa para evitar repetição dos mesmos. Para guardar os valores podemos usar um dicionário, de fácil entendimento para o ser humano quando for ler o código (e.g. um dicionário com chaves = spins, e para cada spin um sub\_dicionário com chaves = delta):

transition\_values\_w = {UP:{}, DOWN:{}}

for spin in transition\_values\_w.keys():

    for idx, delta in enumerate(DELTA\_OPTIONS):

        d\_sm = delta + (spin \* mag\_field)

        if (d\_sm <= 0):

            transition\_values\_w[spin][delta] = 1

        else:

            transition\_values\_w[spin][delta] = math.exp(-(2\*d\_sm) / temp)

Ou, como python permite aceder a arrays com índices negativos, acedendo ao array a ontar do fim, podemos guardar os valores da probabilidade de transição para cada spin e delta numa matriz, ficando os valores de delta positivo no início do array, e os valores de delta negativo no fim do array. Exemplo:

arr = [0,1,2,3,4,5,6,-6,-5,-4,-3,-2,-1]

print(arr[0]) »» 0

print(arr[6]) »» 6

print(arr[-6]) »» -6

print(arr[-2]) »» -2

O tamanho do array neste caso vai depender não do número de valores len(DELTA\_OPTIONS), mas sim dos valores em si, o que pode acarretar um maior uso de memória:

delta\_length = max(DELTA\_OPTIONS) + abs( min(DELTA\_OPTIONS) ) + 1

    spin\_length = 3

    transition\_values\_spin = np.zeros((spin\_length, delta\_length))

    for spin in [UP, DOWN]:

        for idx, delta in enumerate(DELTA\_OPTIONS):

            d\_sm = delta + (spin \* mag\_field)

            if d\_sm <= 0:

                transition\_values\_spin[spin, delta] = 1

            else:

                transition\_values\_spin[spin, delta] = math.exp(-(2\*d\_sm) / temp)

No entanto, mesmo usando arrays numpy em vez de arrays python, esta solução, usada assim de forma isolada, acaba por ser mais lenta do que usar um dicionário. Por exemplo fazendo 10 milhões de acessos:

dicionário = 120 segundos

numpy array com índices negativos = 140 segundos.

for i in range (10\_000\_000):

    spin = random.choice([UP, DOWN])

    delta = random.choice(DELTA\_OPTIONS)

    w = transition\_values\_w[spin][delta]

# no caso de matriz numpy é igual, apenas o acesso tem syntax diferente:

# w = transition\_values\_w[spin][delta]

### Teste de Opções Combinadas com Convolution

As opções descritas anteriormente, usadas de forma isolada, não trazem melhoria de performance, aliás, pelo contrário são prejudiciais. Mas quando combinando estas opções com a função convolution – scipy, isto traz melhorias enormes na performance. E então vale a pena usar as opções anteriores para assim conseguirmos utilizar a função convolution, a qual não funciona com dicionários python por exemplo.

No entanto a utilização destas funções que se aplicam ao mesmo tempo a toda a matriz, representam uma dificuldade quando queremos executar ações sobre a matriz de uma forma aleatória. Por exemplo, ao fazer-se 10\_000 ciclos de Monte Carlo, com uma implementação normal do código scipy, com as mesmas fórmulas que se usam numa implementação normal para cada ponto da matriz, o total de trocas de spins para cada ponto da matriz será algo parecido com o gráfico seguinte:

Uma imagem com texto, captura de ecrã, Retângulo, design

Descrição gerada automaticamente

Como podemos ver, existe uma certa aleatoriedade, mas apenas entre linhas da matriz. Para os pontos da mesma linha aplicam-se as fórmulas ao mesmo tempo e assim os resultados são sempre iguais.

Então temos de introduzir alguma independência e espontaneidade entre as trocas de spin de cada posição, por exemplo analisando apenas um número aleatório de pontos da matriz. No caso específico do problema com a rede 3D, como cada ponto é influenciado por 6 vizinhos mais próximos, uma probabilidade razoável de escolha de pontos da matriz para analisar a troca de spin pode ser precisamente 1/6. E para garantir que no ciclo de Monte Carlo percorremos a rede toda, mesmo que de forma aleatória, então teremos de repetir o processo 6 vezes.

Desta forma conseguimos obter os resultados esperados, por exemplo com uma temperatura de 5.5 e um campo magnético externo = 0, ou seja uma temperatura muito elevada em que não vai haver qualquer orientação dominante na rede, e todos os pontos vão estar constantemente a trocar de spin, teremos então uma amostra com um aleatório bastante mais uniforme:

Uma imagem com captura de ecrã, design

Descrição gerada automaticamente

### Resultados

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

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

Descrição gerada automaticamenteUsando esta biblioteca scipy, nomeadamente função convolution, conseguimos assim ter um aumento significativo no desempenho, passando de 10 minutos para o teste executado com a implementação base, para apenas 34.6 segundos.

## Numba

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

**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.

# 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**

Para efetuar a simulação para uma rede sujeita a uma temperatura externa (simulados para vários casos), e sendo depois aplicado um campo magnético externo crescente de -4 a 4 e depois reduzindo de volta a -4, permitindo obter gráfico semelhante ao seguinte:

Uma imagem com texto, file, Gráfico, recibo

Descrição gerada automaticamente

Os parâmetros utilizados foram os seguintes:

    GRID\_SIDE = 30

    MC\_CYCLES = 20\_000

    INITIAL\_GRID\_SPIN = -1

    TEMPERATURE\_TEST\_VALUES = np.array([0.5, 1.5, 2.5, 3.5, 4.4, 4.5, 4.6, 5.5, 6.5])

    MAGN\_FIELD\_TEST\_VALUES = np.arange(start=-4, stop=4.1, step=0.2)

Este teste demorou 2 horas e 53 minutos,

num processador 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 1.69 GHz, com 16 GB RAM.

**5 - ALL\_TEMP\_AND\_MAG\_FIELD\_TEST:**

Para efetuar a simulação para vários pares de valores de temperatura e campo magnético externos, e obtenção dos gráficos seguintes, entre vários outros com outras métricas observadas.

Uma imagem com texto, Gráfico, file, diagrama

Descrição gerada automaticamente

Os parâmetros utilizados foram os seguintes:

    GRID\_SIDE = 50

    MC\_CYCLES = 20\_000

    INITIAL\_GRID\_SPIN = -1

    TEMPERATURE\_TEST\_VALUES = np.array([0.5, 1.5, 2.5, 3.5, 4.4, 4.5,4.6, 5.5, 6.5])

    MAGN\_FIELD\_TEST\_VALUES = np.arange(start=-4, stop=4.1, step=0.5)

Este teste demorou 2 horas e 7 minutos,

num processador 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 1.69 GHz, com 16 GB RAM.

Este teste também foi feito com valores de campo magnético próximos de 0 para se ver melhor o comportamento das métricas: MAGN\_FIELD\_TEST\_VALUES = np.arange(start=-1, stop=1.1, step=0.1)

Os parâmetros utilizados em ambas as simulações, procuraram encontrar o equilíbrio entre o tempo computacional, o que impede números demasiado elevados, nomeadamente quando se dispõe de um computador pessoal com características limitadas; e os valores necessários para permitir capturar os fenómenos físicos de forma o mais correta possível estatisticamente.

Assim, um cubo de 50x50x50, constitui uma rede de 125.000 posições, o que constitui uma escolha razoável para capturar o comportamento do Ising Model mas mantendo a carga computacional suportável. Para o teste 4, como fazemos evoluir o campo magnético desde -4 a 4 com step de 0.2 e depois voltar a descer para -4, por forma a reduzir a carga computacional, efetuou-se o teste com um cubo de 30x30x30, ou seja 27.000 posições. Uma rede mais pequena poderia deixar passar despercebidos alguns fenómenos, como por exemplo os domínios de campo magnético oposto que se criam numa rede de grandes dimensões perante campo magnético externo = 0 e baixa temperatura, em que o material ferromagnético vai procurar estabilizar com um único spin na rede toda, mas por vezes surgem subdomínios na rede com um spin diferente.

20.000 ciclos de Monte Carlo de forma semelhante, permite o sistema encontrar um estado de equilíbrio (ou conjunto de estados de equilíbrio), e assim obter informação estatística com margem de erro suficientemente reduzida.

Spin inicial de -1, permite ter melhor noção da forma como o sistema evolve. Um spin completamente aleatório na rede por exemplo perante um campo magnético externo = 0 e temperatura externa = 0.5, pode levar a uma rede toda orientada para cima ou toda orientada para baixo, o que pode tornar confuso o estudo dos fenómenos.

Os valores de temperatura focaram-se em permitir uma boa distribuição, entre 0.5 e 6.5, focando-se no entanto também a atenção mais próximo da temperatura crítica, daí se utilizar então os valores de 4.4, 4.5 e 4.6.

Os valores do campo magnético externo variam entre -4 e 4, com step de 0.1 no primeiro caso e 0.5 no segundo caso, permitindo assim ter uma visão 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.

## Temperatura Crítica

A temperatura de Curie é a temperatura acima da qual um material ferromagnético se torna paramagnético. Num sistema magnético podemos identificar esta temperatura pelas mudanças significativas na suscetibilidade magnética e capacidade calorífica do sistema.

**Suscetibilidade magnética**

Como descrito anteriormente, a suscetibilidade de um material magnético é a sua capacidade de ser magnetizado por um campo magnético externo. E é inversamente proporcional à temperatura.

A suscetibilidade magnética atinge o pico próximo da temperatura de curie. Quer dizer que é perante esta temperatura externa que um material fica mais responsivo a qualquer campo magnético externo que lhe seja imposto.

Se a temperatura for inferior à temperatura de curie, o material magnético vai querer resistir à mudança e manter o seu estado interno, e se a temperatura for superior à temperatura de curie o sistema vai-se tornar paramagnético, não havendo qualquer orientação geral dos spins, havendo sim muita instabilidade com mudanças constantes de spin em toda a rede, não permitindo assim que a rede se magnetize.

Ou seja, tanto um material ferromagnético, como paramagnético vão resistir à orientação global dos seus spins internos de acordo com um campo magnético externo. O momento em que o material está sim mais suscetível de ser influenciado pelo campo magnético externo é precisamente com a temperatura externa correspondente à temperatura de curie.

**Capacidade Calorífica**

A capacidade calorífica também atinge um pico próximo da temperatura de curie, indicando grandes flutuações de energia com absorção e libertação significativa de calor devido à reorganização constante dos momentos magnéticos.

**Temperatura de Curie**

Olhando então para o gráfico seguinte, obtido através da realização do teste 5: ALL\_TEMP\_AND\_MAG\_FIELD\_TEST, e analisando a suscetibilidade magnética e a capacidade calorífica, com um campo magnético externo = 0 (ou seja, a linha vertical central do gráfico), vemos que tanto a suscetibilidade magnética como a capacidade calorifica atingem o pico com uma temperatura externa de cerca de 4.5, para uma rede 3D, próximo do valor teórico conhecido de 4.51:

Uma imagem com texto, diagrama, file, Gráfico

Descrição gerada automaticamente

Estes valores são para o material magnético genérico que utilizamos nas simulação, onde apenas utilizamos a temperatura e campo magnético reduzido, sem ter em conta outras características do material:

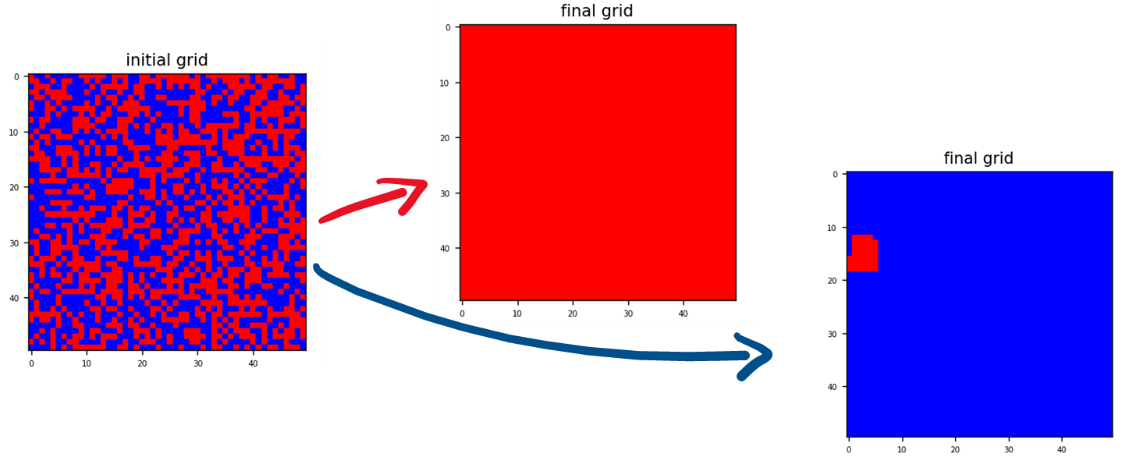
Já olhando para uma rede 2D verificamos que o pico ocorre por volta da temperatura 2.3, bem próximo do valor teórico conhecido de 2.269, derivado por Lars Onsager em 1944, para um campo magnético externo = 0.

Uma imagem com texto, diagrama, file, Gráfico

Descrição gerada automaticamente

## Ferromagnetismo-Paramagnetismo Histerese

Um material ferromagnético vai procurar sempre manter o seu estado interno estável, com todos os spins orientados na mesma direção, independente de orientados para cima ou baixo. Por exemplo iniciando uma rede com spins aleatórios (50% para cima e 50% para baixo), a uma temperatura baixa, por exemplo 0.5, a rede vai convergir para um estado em todos os spins ficam orientados na mesma direção. Umas vezes os vermelhos ganham, outras vezes os azuis:



*Nota complementar:*

*Como vemos, a rede a azul ficou com uma “ilha” de spins vermelhos. Isto acontece em redes muito grandes, em que de facto os spins vão se todos orientar no mesmo sentido, mas como a rede é grande de mais, vão-se criar domínios de spin igual, dentro dos quais o material fica estável e não há troca de spin, havendo sim nas fronteiras entre domínios. Exemplo de evolução de uma rede de 100x100x100, onde se mostra uma fatia central em 2D em diferentes momentos da evolução:*

Uma imagem com texto, captura de ecrã, Saturação de cores

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, mapa, Saturação de cores

Descrição gerada automaticamenteUma imagem com texto, mapa, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Uma vez a rede com os spins todos orientados, o material ferromagnético vai resistir à mudança mesmo quando lhe é aplicado um campo magnético externo. Por exemplo, tendo uma rede com spin -1. Se lhe for aplicado um campo magnético externo negativo, obviamente o spin do material não vai mudar e vai continuar próximo de -1, mesmo temperatura externa mais elevada, embora nestes casos comece a haver maior instabilidade e trocas de spin e por isso o momento magnético pode deixar de ser tão “puro”, igual a -1. Isto pode ver-se no gráfico seguinte, onde, para uma rede com spin inicial de menos 1 se simulam os seus estados finais perante diferentes condições externas de temperatura e campo magnético, e onde se pode ver no lado esquerdo do gráfico, que, quando o campo magnético externo é negativo, então a rede vai manter também a sua orientação geral para -1.

Olhando para o centro do gráfico, percebemos que com temperaturas mais elevadas, o momento magnético da rede se vai aproximando de 0. Mas com temperaturas baixas, o spin praticamente não se altera e o momento magnético continua próximo de -1.

E mesmo, quando avançamos para o lado direito do gráfico, ou seja, quando o material fica sujeito a um campo magnético externo positivo, era esperável que os spins do material também se orientassem para 1, mas tal não acontece com temperaturas mais baixas. Nestes casos o material vai resistir à mudança, e vai preferir manter o seu estado interno. Com temperatura de 0.5 por exemplo, vemos que só quando o campo magnético externo fica próximo de 3 é que realmente a rede inverte o spin e fica orientada para 1. A este atraso na resposta do material perante um campo magnético externo chama-se histerese, e é um fenómeno que ocorre tanto no sentido crescente como no sentido contrário.

Uma imagem com texto, diagrama, file, Esquema

Descrição gerada automaticamente

E esta passagem de uma rede com spins -1 sujeita a um campo magnético externo positivo forte e então o spin da rede passa para 1, acontece de forma repentina. Já com temperaturas mais elevadas, que causam maior instabilidade na rede é que a passagem é mais suave, pois de facto a rede fica com um estado interno menos coeso e assim mais suscetível à mudança perante um campo magnético externo. Com temperaturas muito elevadas, acima da temperatura de curie, a resposta do material ao campo magnético externo é imediata, resultando numa curva mais suave. Perante esta temperatura o material fica paramagnético, e a instabilidade da rede é tanta que os spins nunca se orientam completamente, ficando toda a rede a -1 ou a 1, mas em termos de resposta a um campo magnético externo é praticamente imediata.

Se corrermos o teste 4 do programa (EVOLVING\_MAG\_FIELD\_TEST), podemos ver a resposta do material, exposto a uma determinada temperatura externa, e depois fazendo variar o campo magnético externo, por exemplo de -4 a 4 com step de 0.2. Isto permite observar o que já descrevemos atrás, de forma ainda mais visível. Por exemplo no gráfico da temperatura = 0.5 vemos todo o atraso que o material aguenta até realmente não aguentar mais a pressão do campo magnético externo bastante forte, e então aí sim mudar repentinamente de spin. Já com temperatura de 6.5 a curva de histerese é muito mais suave, ajustando-se de imediato ao campo magnético externo, sem nunca se obter no entanto um spin uniforme da rede para temperaturas muito elevadas.

Uma imagem com texto, diagrama, file, Paralelo

Descrição gerada automaticamente

Outro aspeto que podemos reparar além da curva de histerese é o próprio loop formado pela curva do aumento do campo magnético externo e pela curva da diminuição do campo magnético. Como Podemos ver para temperaturas baixas a área é muito superior. Para temperaturas acima da temperatura de curie a área é praticamente inexistente.

Esta área representa a energia dissipada na forma de calor durante a transição de cada ciclo, e permite também distinguir os materiais ferromagnéticos dos materiais paramagnéticos, os quais sofrem magnetização perante campos magnéticos externos, mas não são capazes de manter essa magnetização quando se retira o campo magnético externo.

# 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

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).

‌