



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
2. Aplicação Desenvolvida.....	4
2.1. Estrutura do código	4
2.2. Utilização: ising_main.py.....	4
2.2.1. Escolher nível de detalhe	4
2.2.2. Escolher tipo de teste, e implementação de código	4
2.2.3. Escolher parâmetros do teste.....	5
2.2.4. Correr o programa	5
2.3. Tipos de Métricas e Testes Disponibilizados	6
3. Análise de Código	10
3.1. Implementação de Ising Model – Metrópolis Algorithm.....	10
3.2. Implementações	12
3.3. Implementações Base 2D e 3D, com multi-processing.....	12
3.4. Dynamic Programming.....	16
3.5. Cython	17
3.6. Convolution - Scipy	17
3.7. Numba	20
4. Análise Física.....	22
4.1. Testes e Parâmetros.....	22
4.2. Temperatura Crítica	23
4.3. Ferromagnetismo-Paramagnetismo Histerese.....	25
5. Conclusões	28
6. Bibliografia	29

1. Introdução

Este projeto tem como objetivo o estudo de uma simulação de ferromagnetismo, recorrendo ao modelo de Ising, com vista a analisar a Histerese Ferromagnética-Paramagnética e a Temperatura de Curie. Para tal são calculadas várias métricas e grandezas físicas, nomeadamente, para uma determinada rede sujeita a uma Temperatura e Campo Magnético externos, pretende-se observar a evolução do estado e obter o Momento Magnético, Energia média, Suscetibilidade Magnética e Capacidade Térmica da rede.

O estudo destas métricas será através de uma rede de 3 dimensões, para a qual podemos pensar em cada posição como sendo um átomo, com apenas um eletrão de valência, com um determinado spin. Cada spin de cada átomo / posição da rede, vai interagir com as posições vizinhas mais próximas fazendo evoluir o estado, o qual se pode caracterizar a cada momento com um determinado Momento Magnético e Energia média, tanto por ponto de rede como na rede como um todo.

Dado uma rede inicial sujeita a uma temperatura e campo magnético externo, o sistema vai evoluir naturalmente por forma a convergir para um estado de maior equilíbrio. No entanto, existirão sempre interações entre os spins vizinhos pelo que o estado vai sempre oscilar em torno de um conjunto de estados relativamente estáveis consoante a temperatura e campo magnético externo. Por esta razão, para se calcular a Suscetibilidade Magnética e a Capacidade Térmica da rede, é preciso deixar o sistema evoluir até chegar a estes estados de maior equilíbrio, desprezando-se esta parte da simulação; e depois então deixar a rede evoluir e ir variando entre estes vários estados de maior estabilidade consoante a temperatura e campo magnético externos para se poder calcular a Suscetibilidade Magnética e a Capacidade Térmica da rede. Quantos mais ciclos se deixar o sistema evoluir mais precisas serão estas medições e a simulação como um todo. Importante será encontrar um ponto de equilíbrio entre a precisão dos dados que se pretende, e o tempo e recursos disponíveis em termos da máquina, tanto em termos de computação como de memória, para conseguir executar essa simulação. Por forma a diminuir o tempo de execução implementaram-se várias otimizações ao código para que se consigam obter bons resultados em menor tempo de execução.

Feita a simulação, foram então implementados vários tipos de gráficos por forma a facilitar a análise dos dados e permitir tirar conclusões sobre as várias métricas, para vários pares de Temperatura e Campo Magnético externo, e assim conseguir alcançar os objetivos do projeto, nomeadamente identificar a Temperatura de Curie para uma rede em 3D e comparar com a Temperatura de Curie para uma rede em 2D; bem como observar o fenómeno da Histerese Ferromagnética-Paramagnética, ou seja, a resistência da rede em mudar o seu spin para se orientar com um Campo Magnético externo. Isto quando a temperatura externa é inferior à Temperatura de Curie, uma vez que quando a temperatura é superior, então a energia é tal na rede que faz com que a mesma nunca consiga encontrar um estado estável com os spins alinhados, e ficando assim muito mais suscetível de se orientar na direção geral do Campo Magnético externo existente.

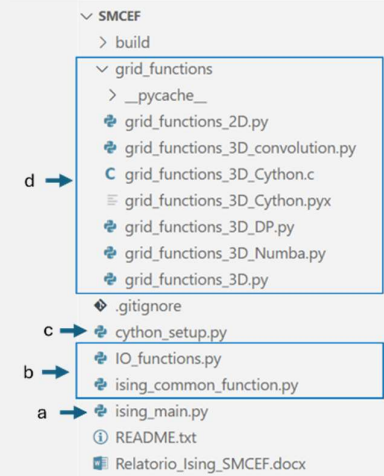
O objetivo inicial seria desenvolver uma implementação que permitisse efetuar a simulação necessária para observar a Temperatura de Curie e Histerese Ferromagnética-Paramagnética, no entanto, para melhor se observarem estes fenómenos e esta relação entre a temperatura e o campo magnético externo, e os efeitos que isso tem no material magnético, foi desenvolvida uma aplicação que disponibiliza vários tipos de testes, podendo mesmo observar de forma iterativa a evolução e comportamento da rede em diferentes situações.

2. Aplicação Desenvolvida

Esta app foi desenvolvida no sentido de permitir de forma simples, a execução de diferentes tipos de testes, com diferentes parâmetros, com diferentes outputs e plots, e com a execução com diferentes tipos de implementações. Esta versatilidade acaba por incrementar ligeiramente o tempo de execução dos testes, embora não de forma significativa, uma vez que cada implementação foi otimizada por forma a obter a melhor performance possível.

2.1. Estrutura do código

Por forma a permitir uma maior extensibilidade e menor repetição de código, a aplicação divide-se em alguns ficheiros cada um com competências distintas:



- a) main() – documento para se escolher o tipo de teste a correr, os seus parâmetros, o tipo de implementação de código, e para meter o programa a correr
- b) helper functions, comuns a todos os testes:
 - funções de preparação e execução dos ciclos de monte carlo e cálculo das variáveis e métricas necessárias
 - funções de IO para apresentar os resultados obtidos
- c) Documento para desenvolvedor, para gerar implementações em linguagem c, automaticamente, a partir de um documento python
- d) Pasta grid_functions, com diferentes implementações específicas da simulação

2.2. 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:

2.2.1. Escolher nível de detalhe

```
SHOW_CYCLE_DETAILS = False
SHOW_TEST_DETAILS = False
```

2.2.2. 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

2.2.3. 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)]
```

2.2.4. 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
...
```

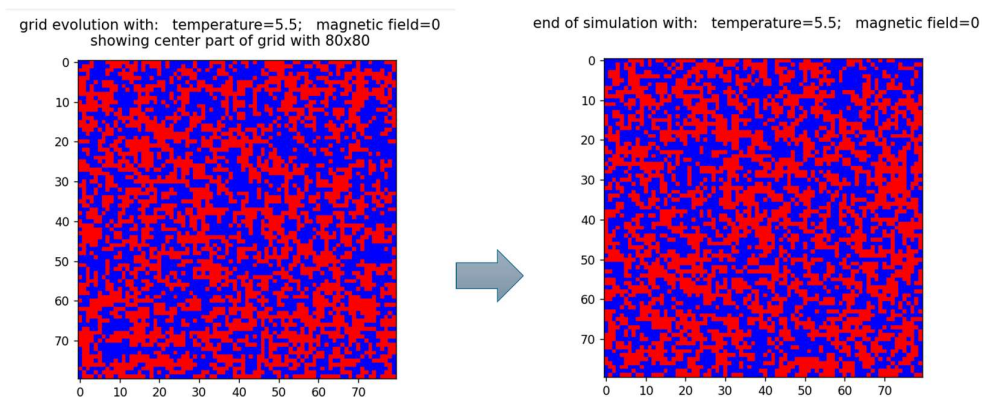
Quando o teste apresenta plots, é necessário fechar cada plot para depois ser apresentado o seguinte.

2.3. Tipos de Métricas e Testes Disponibilizados

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

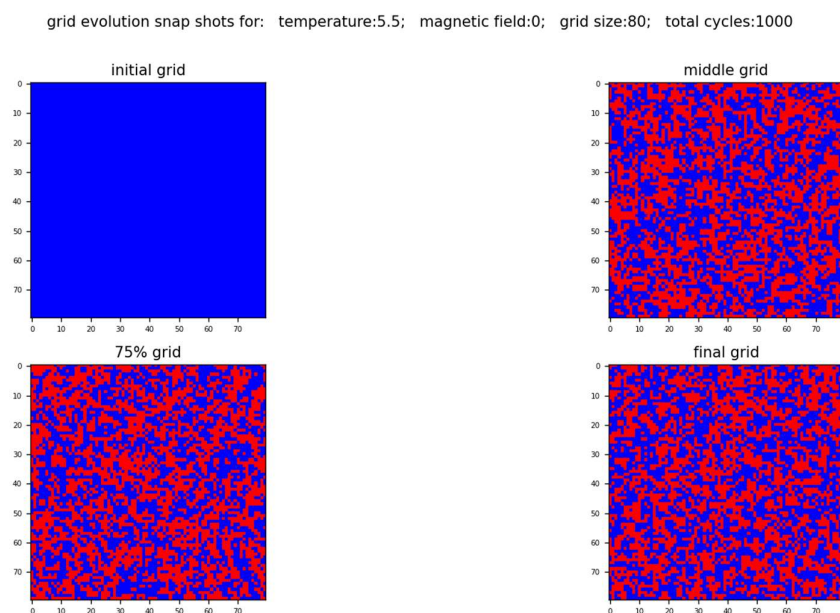
2.3.1. Evolução do estado da rede

Permite ver a evolução do estado da rede, apresentando a azul o $\text{spin} = -1$ (sul) e vermelho o $\text{spin} = 1$ (norte). Para redes de grandes dimensões é apresentada apenas uma amostra. Evitar seleccionar esta opção quando se executa uma simulação com redes de grandes dimensões pois, consoante o computador, a simulação pode ficar muito lenta. No caso de 3D é apresentada uma fatia da rede/cubo.



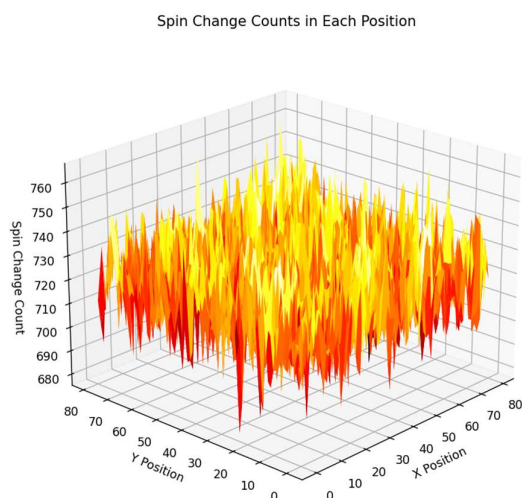
2.3.2. Snapshots da evolução do estado da rede

No fim de correr a simulação, apresenta alguns estados da rede ao longo do tempo. No caso de 3D é apresentada uma fatia da rede/cubo.



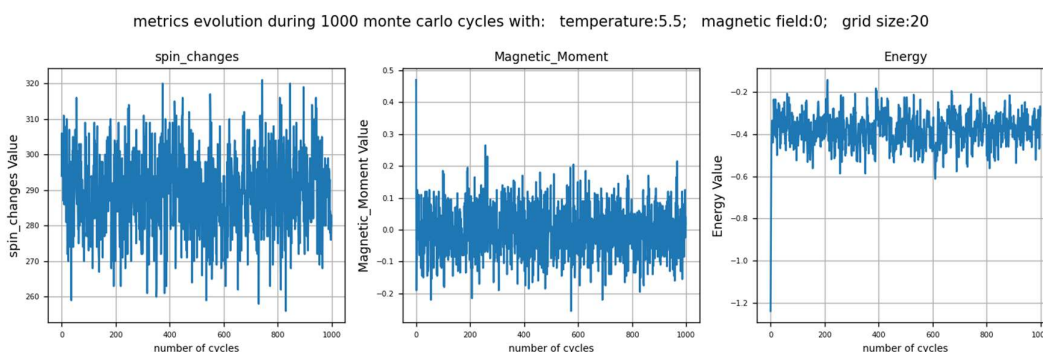
2.3.3. Contagem de Spins em cada posição da rede

Permite ver o total de mudanças de spin que ocorreu em cada posição. No caso de 3D é apresentada uma fatia da rede/cubo.



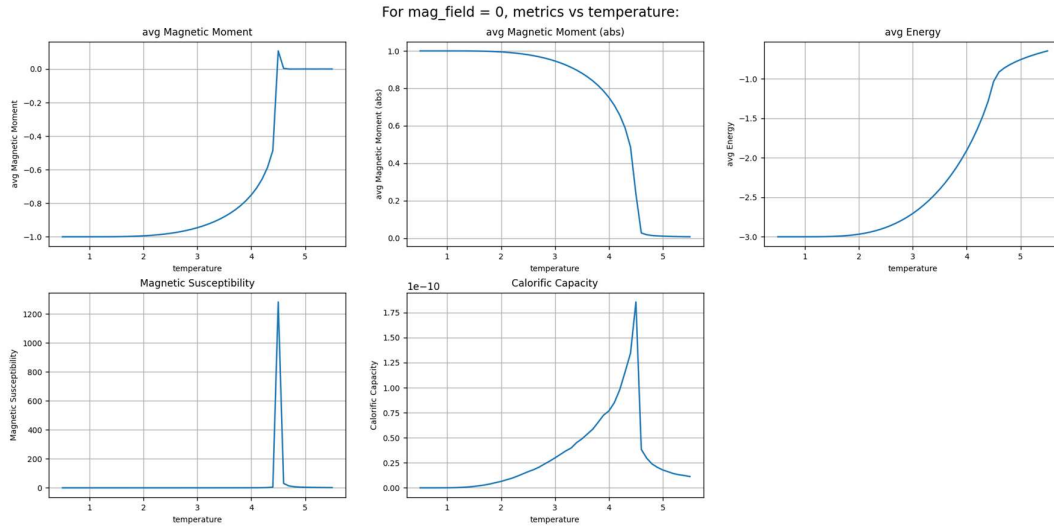
2.3.4. Evolução de Métricas ao Longo de x Ciclos Monte Carlo

São disponibilizadas a variação do número total de spins da rede em cada ciclo de Monte Carlo, bem como o momento magnético médio e a energia média:



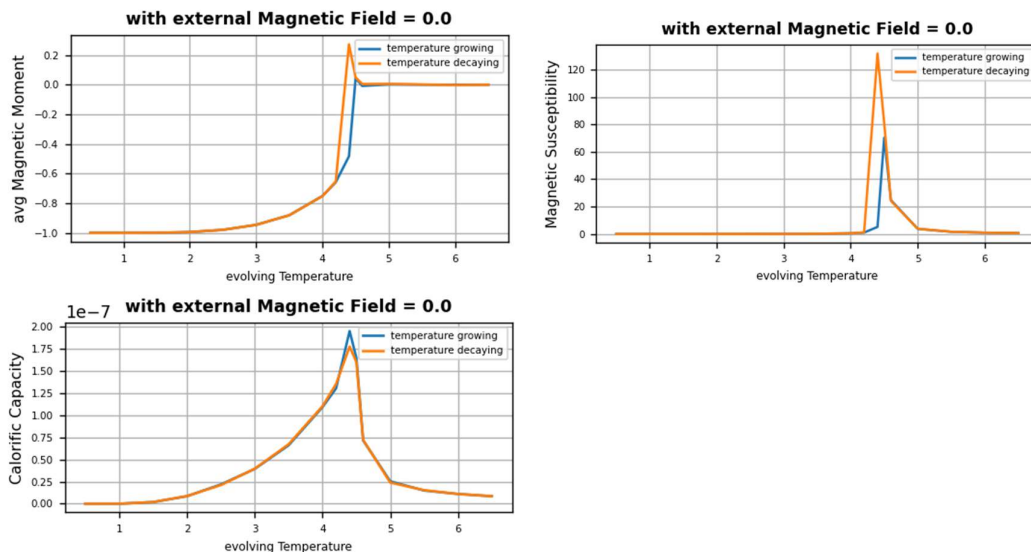
2.3.5. 1 Campo Magnético sujeito a diferentes Temperaturas

Dado um campo magnético externo, podemos ver como cada métrica responde perante diferentes níveis de temperatura. As métricas disponibilizadas são o momento magnético médio e absoluto, a energia média, a suscetibilidade magnética e capacidade calorífica.



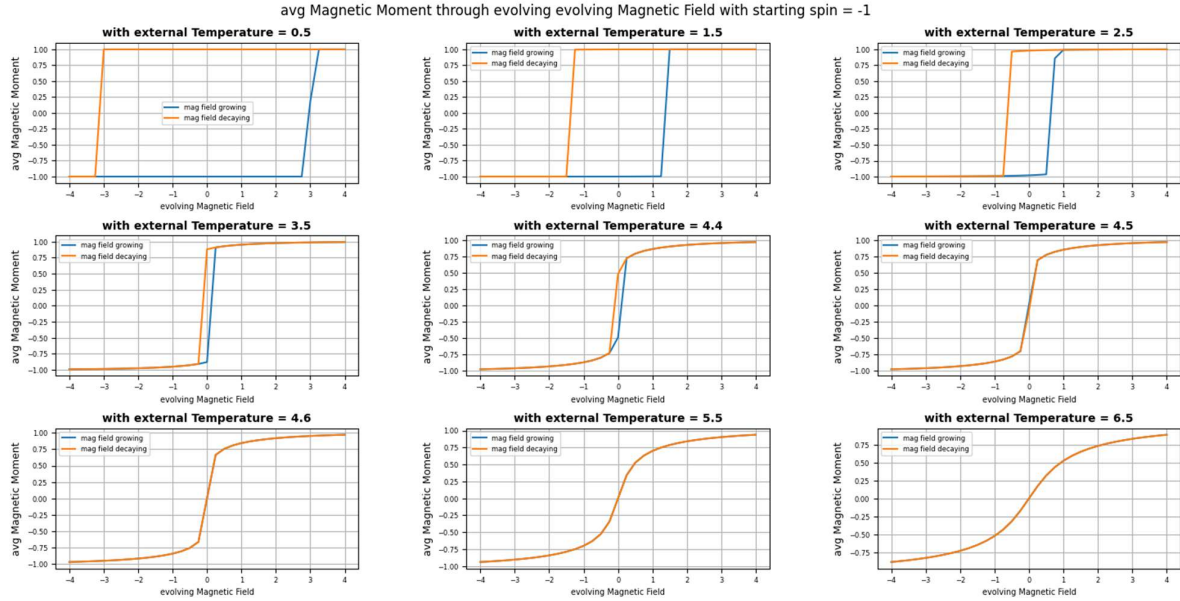
2.3.6. Evolução de Temperatura Externa

Dada uma lista de campos magnéticos, para cada um deles, aplicamos uma evolução da temperatura, por exemplo desde 0.5 até 6, e assim conseguimos perceber o impacto que a temperatura tem em cada métrica ao longo da evolução. Por exemplo, reparamos que temperaturas altas destabilizam o ordenamento de spins do material, tendendo a média de momento magnético para 0 caso o campo magnético externo também seja igual a 0. Com campo magnético externo o momento magnético da rede já vai tender na mesma direção, mas, com temperaturas elevadas, a instabilidade é tal que mesmo com campo magnético externo forte, positivo, ou negativo, a rede nunca vai ficar totalmente alinhada com o mesmo. Já quando a temperatura volta a descer, então o material magnético vai começar a estabilizar com toda a rede com os spins na mesma direção. Se o campo magnético externo for positivo então a rede fica na mesma direção e se for negativo então na direção oposta. Se o campo magnético externo for = 0, então será aleatória qual a direção geral que irá predominar nos spins da rede. Com redes muito grandes poderão mesmo criarem-se domínios de campos magnéticos distintos, nunca convergindo a rede na sua totalidade. Neste teste são apresentadas várias métricas, para vários campos magnéticos externos sobre os quais fazemos evoluir a temperatura, podendo-se verificar situações que serão discutidas mais à frente, nomeadamente sobre quando acontece o pico de suscetibilidade magnética e capacidade calorífica para um campo magnético externo = 0:



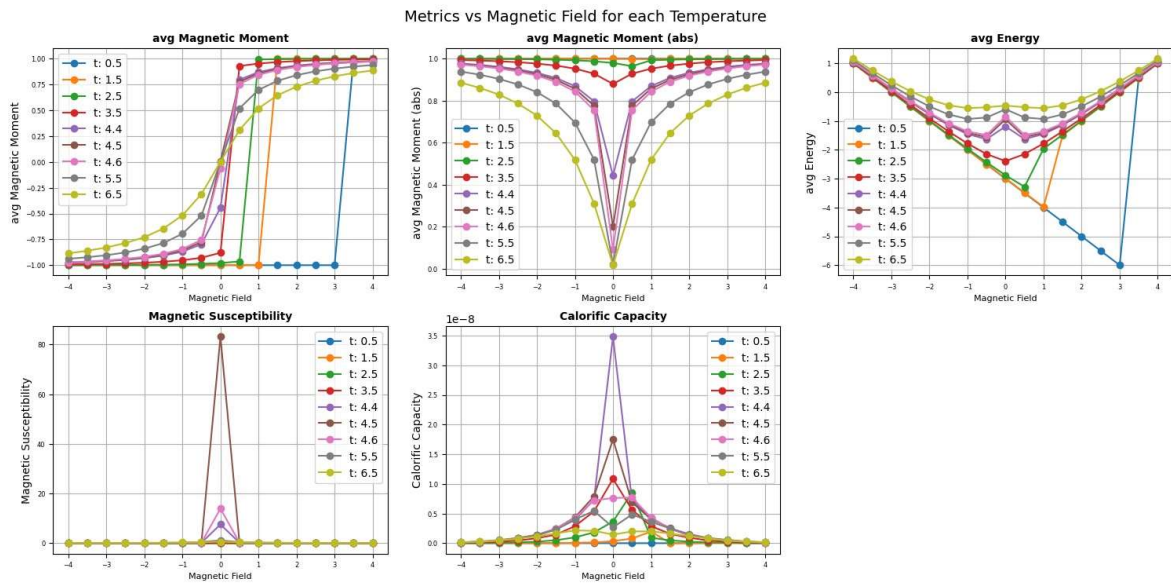
2.3.7. Evolução do Campo Magnético Externo

Dada uma lista de temperaturas, para cada uma delas, aplicamos uma evolução do campo magnético externo, por exemplo desde -4 até 4, e assim conseguimos perceber o impacto que o campo magnético externo tem em cada métrica ao longo da evolução, permitindo assim observar claramente o fenómeno da histerese Ferromagnetismo - Paramagnetismo que será analisado mais à frente no relatório.



2.3.8. Métricas para pares de Temperatura - Campo Magnético Externo

Dada uma lista de temperaturas e campos magnéticos externos, para cada par, calcular as métricas obtidas ao fim dos ciclos de Monte Carlo, útil por exemplo para calcular a temperatura de Curie, como será discutido mais à frente no relatório.



3. Análise de Código

3.1. Implementação de Ising Model – Metrópolis Algorithm

Para implementar o modelo de Heisenberg podem ser tomadas algumas suposições por forma a simplificar o problema, nomeadamente, para o presente projeto, consideraram-se as seguintes características:

- 1 elétron desemparelhado por átomo da rede cristalina. Ou seja, os elétrons do átomo ou ião estão todos emparelhados, exceto 1 que vai então interagir com os átomos vizinhos. Assim cada átomo irá corresponder a uma posição da matriz, 2D ou 3D, e com o spin representado por valores discretos, 1 para spin up, -1 para spin down. Não se consideram por isso outras direções de spin, restringindo-se assim direção do spin a 1 eixo, o que permite simplificar o cálculo da interação entre spins.
- As interações entre elétrons de diferentes nós, ocorrem apenas entre os nós mais próximos, ou seja, na matriz 3D, as posições que estão acima, abaixo, esquerda, direita, a frente, atrás. Não se consideram as posições vizinhas diagonais. E esta interação é determinada apenas pelos valores de spins respetivos de uma posição com os seus vizinhos mais próximos. É por isso um tipo de simulação adequada apenas para alguns materiais, nomeadamente alguns rare earth metals onde o magnetismo surge precisamente desta interação entre elétrons de átomos vizinhos, ao contrário de outros materiais em que o magnetismo surge pela interação de elétrons que se movimentam parcialmente pelo sistema.
- Considera-se uma matriz 2D ou 3D “circular”. Ou seja, e.g. numa matriz 2D, o spin da posição [0,0], vai sofrer influência dos vizinhos mais próximos dentro da matriz [0, 1] e [1,0], e como os outros 2 vizinhos estariam já fora da matriz, então consideram-se os pontos do extremo oposto da matriz, ou seja [último, 0] e [0, último].
- As simulações recorrem ao uso de variáveis reduzidas, nomeadamente para os valores de temperatura e campo magnéticos externos, os quais terão um impacto distinto em diferentes composições de materiais, com diferentes constantes de Boltzmann, etc, e que não é aqui considerado, tendo-se então que:

$$\text{temperatura} = \frac{KT}{J} \quad \text{campo magnetico} = \frac{\mu_B B}{J}$$

3.1.1. Ciclo de Monte Carlo, dada uma rede com spins iniciais:

- Correr todos os pontos da rede
 - Neste caso optou-se por uma iteração linear sequencial por forma a melhorar a eficiência do algoritmo. Outras opção seriam acessos aleatórios a toda a rede, ou então acessos por linhas aleatórias sendo assim um nível estocástico intermédio. Esta opção acabou por ser utilizada no caso da implementação recorrendo à função convolution – scipy, para evitar que toda a matriz fosse analisada ao mesmo tempo.
- Para cada ponto da rede:
 - Calcula-se a variação de energia, caso se inverta o spin
 - Se variação de energia < 0 então é normal o material magnético querer ir para esse estado mais estável, então troca-se sempre o spin.
 - Se a variação de energia > 0, por forma a permitir sair de mínimos locais, nestas situações permite-se trocar de spin com uma dada probabilidade, mesmo que isso acarrete o aumento de energia do sistema:

$$prob = \begin{cases} 1, & \text{se } \Delta \text{energia} < 0 \\ e^{-\frac{2(\delta + spin * mag_{field})}{temperature}}, & \text{se } \Delta \text{energia} > 0 \end{cases}$$

$$\delta = \text{spin} \sum (\text{spin dos vizinhos próximos})$$

3.1.2. Cálculo das métricas no fim de um ciclo de Monte Carlo:

- Momento Magnético médio da rede, é simplesmente somar os spins e dividir pelo tamanho da rede.
- Energia média da rede é a soma do total da energia de cada ponto da rede a dividir pelo tamanho da rede. A energia de cada ponto da rede depende então:
 - da interação com os vizinhos mais próximos $[\text{spin} * (\text{soma dos spins vizinhos})] / 2$ (a dividir por 2 para não contabilizar em duplicado a energia de um ponto e a sua influência para o ponto vizinho)
 - e da interação com o campo magnético externo $-(\text{spin} * \text{mag_field})$

$$\text{point_energy} = \frac{\delta}{2} - \text{spin} * \text{mag_field}$$

3.1.3. Cálculo das métricas no fim de um teste:

Dada uma configuração inicial de uma rede, um campo magnético externo, e uma temperatura externa, isto cria um sistema com excesso de entropia e energia livre que vai tentar procurar o equilíbrio. O Ciclo de Monte Carlo representa essa aproximação a um estado mais equilibrado, com menos energia livre. No entanto nunca vamos conseguir chegar a um estado final terminal em que todos os componentes da rede estão em perfeito equilíbrio com os seus vizinhos e o mundo externo. Vai haver sempre oscilações e pequenas variações, fazendo assim o sistema variar entre um conjunto de estados mais estáveis.

Por isso, para calcular as métricas desse estado final, não devemos olhar apenas para um estado final, mas sim para todo esse conjunto de estados mais estáveis, e então calcular uma média entre eles.

Por isso, para cada teste é bom correr pelo menos, por exemplo 10_000 ciclos de Monte Carlo por forma a conseguirmos encontrar uma boa aproximação a este conjunto de estados mais estáveis. E, como os estados iniciais estão sujeitos a muita entropia, havendo conflito entre a distribuição dos spins da rede e a temperatura e campo magnético externos, não devem por isso ser tidos em conta para o cálculo final das médias do sistema. Por essa razão, não se contam 10% dos ciclos de Monte Carlo efetuados inicialmente.

Quanto às métricas calculadas para cada teste, são as seguintes:

- Momento magnético médio = média (momento magnético médio da rede ao longo dos ciclos de monte carlo)
 - Momento magnético absoluto = média (abs (mom_mag_médio da rede ao longo dos ciclos de Monte Carlo))
- Isto permite ter uma noção mais real de como realmente o momento magnético está a convergir em direção a um valor, permitindo colmatar as pequenas oscilações que sempre ocorrem entre o conjunto de estados mais estáveis, em que por exemplo de um estado quase perfeitamente equilibrado em torno de um momento magnético = 0, mas em que 10 posições trocam de spin, passamos assim para um estado com momento magnético oposto. Mas o que realmente nos interessa estatisticamente é perceber que realmente o momento magnético está a convergir em torno de 0, ou outro valor, consoante a temperatura e campo magnético externo, e a temperatura crítica que vai caracterizar esse sistema, e que em torno da qual vai ocorrer estes saltos de spins mais positivos e mais negativos. E por isso se aplica o absoluto para colmatar estas oscilações, e perceber realmente qual é o valor de convergência do sistema.
- Energia média = média (energia média da rede ao longo dos ciclos de Monte Carlo)
 - Suscetibilidade magnética permite descrever a forma como um determinado material responde perante a aplicação de um campo magnético externo. Alguns materiais são praticamente indiferentes a um campo magnético externo, enquanto outros são altamente responsivos. Esta suscetibilidade magnética é expressa em termos da variância do momento magnético médio e da temperatura:

$$\chi = \frac{\text{var}(\text{momento magnético}) * \text{tamanho da rede}}{\text{temperatura}}$$

- Capacidade calorífica de um sistema permite analisar a quantidade de energia armazenada em flutuações térmicas num sistema magnético, expressa em termos da variância da energia e da temperatura.

$$C = \frac{var(energia)}{temperatura^2}$$

3.2. Implementações

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

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.

<i>Implementação de Código</i>	<i>Tempo do teste</i>
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
```

3.3. 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:

3.3.1. 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()

```

3.3.2. 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)

```

3.3.3. 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:

a) 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
```

b) 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_cycles = 50
for i in range(mc_cycles):
```

10 segundos:	0.023 segundos:
<pre>for i in range(grid_size): for j in range(grid_size): magnetic_moment += grid[i][j]</pre>	<pre>magnetic_moment += np.sum(grid[1:-1, 1:-1])</pre>

3.3.4. 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
```

3.3.5. 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:

```
All tests finished in 26.97 seconds
```

```
18909933 function calls (18821628 primitive calls) in 36.291 seconds
```

```
Ordered by: internal time
```

ncalls	totttime	percall	cumtime	percall	filename:lineno(function)
8000000	11.863	0.000	11.863	0.000	grid_functions_2D.py:34(get_point_energy)
595	4.814	0.008	4.814	0.008	{method 'read' of '_io.BufferedReader' objects}
4000000	4.757	0.000	11.843	0.000	grid_functions_2D.py:44(check_spin)
3	4.357	1.452	4.955	1.652	{built-in method exec}
9	2.469	0.274	2.469	0.274	{built-in method _winapi.WaitForSingleObject}
10000	1.363	0.000	20.334	0.002	grid_functions_2D.py:79(monte_carlo_cycle)

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.

```
18 All tests finished in 611.71 seconds
```

```
19
```

```
20 308600652 function calls (308517363 primitive calls) in 617.554 seconds
```

```
21
```

```
22 Ordered by: internal time
```

```
23
```

ncalls	totttime	percall	cumtime	percall	filename:lineno(function)
16000000	415.147	0.000	415.147	0.000	grid_functions_3D.py:34(get_point_energy)
8000000	114.431	0.000	342.597	0.000	grid_functions_3D.py:44(check_spin)
10000	28.378	0.003	604.466	0.060	grid_functions_3D.py:83(monte_carlo_cycle)
10000	25.012	0.003	233.397	0.023	grid_functions_3D.py:66(calc_cycle_metrics)
64515669	20.883	0.000	20.883	0.000	{method 'random' of 'numpy.random.mtrand.RandomState' objects}
3	4.114	1.371	5.103	1.701	{built-in method exec}
593	2.471	0.004	2.471	0.004	{method 'read' of '_io.BufferedReader' objects}

```
31
```

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.

3.4. 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.

3.4.1. Resultados

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

```
All tests finished in 674.76 seconds
```

```
330882340 function calls (330798028 primitive calls) in 680.747 seconds
```

```
Ordered by: internal time
```

ncalls	totttime	percall	cumtime	percall	filename:lineno(function)
168000000	431.747	0.000	431.747	0.000	grid_functions_3D.py:34(get_point_energy)
88000000	115.696	0.000	363.914	0.000	grid_functions_3D_DP.py:16(check_spin)
10000	86.990	0.009	665.353	0.067	grid_functions_3D_DP.py:67(monte_carlo_cycle)
70785948	23.208	0.000	23.208	0.000	{method 'random' of 'numpy.random.mtrand.RandomState' objects}
3	6.460	2.153	7.492	2.497	{built-in method exec}

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.

3.5. 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:

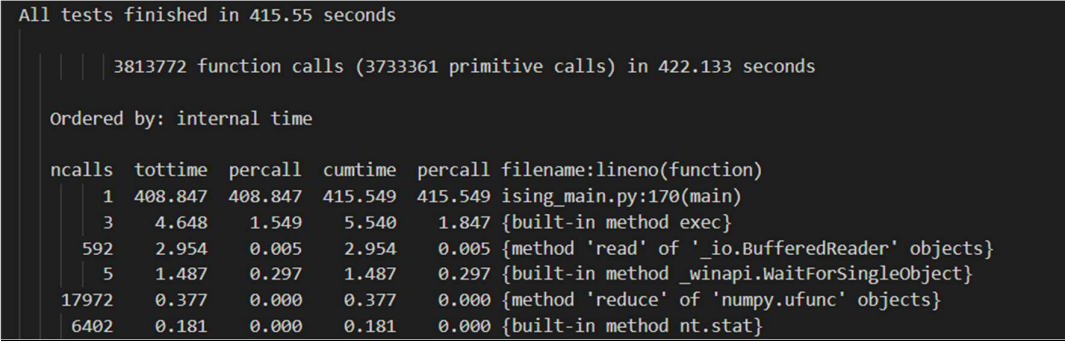
```
import pyximport
pyximport.install()
```

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")
)
```

3.5.1. Resultados

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



```
All tests finished in 415.55 seconds
| | | 3813772 function calls (3733361 primitive calls) in 422.133 seconds

Ordered by: internal time
```

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
1	408.847	408.847	415.549	415.549	ising_main.py:170(main)
3	4.648	1.549	5.540	1.847	{built-in method exec}
592	2.954	0.005	2.954	0.005	{method 'read' of '_io.BufferedReader' objects}
5	1.487	0.297	1.487	0.297	{built-in method _winapi.WaitForSingleObject}
17972	0.377	0.000	0.377	0.000	{method 'reduce' of 'numpy.ufunc' objects}
6402	0.181	0.000	0.181	0.000	{built-in method nt.stat}

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 correremos 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.

3.6. 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.

3.6.1. 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:

a) 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])
```

b) 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.

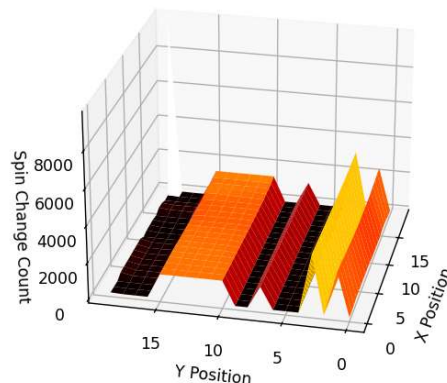
```
for i in range(10_000_000):
    spin = random.choice([UP, DOWN])
    delta = random.choice(DELTA_OPTIONS)
    w = transition_values_w[spin][delta]
    # no caso de matriz numpy é igual, apenas o acesso tem syntax diferente:
    # w = transition_values_w[spin][delta]
```

3.6.2. 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:

Spin Change Counts in Each Position

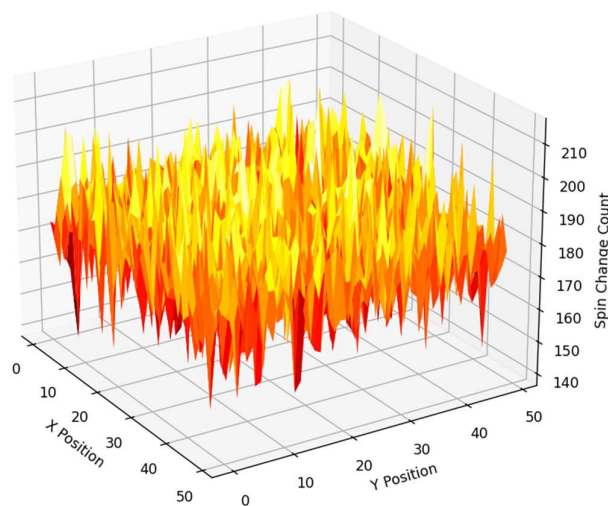


Como podemos ver, existe uma certa aleatoriedade, mas apenas entre linhas da matriz. Para os pontos da mesma linha aplicam-se as fórmulas ao mesmo tempo e assim os resultados são sempre iguais.

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:

Spin Change Counts in Each Position



3.6.3. Resultados

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

```
All tests finished in 34.59 seconds

| 6867499 function calls (6801493 primitive calls) in 35.387 seconds

Ordered by: internal time

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
60000   8.567    0.000   15.451    0.000 grid_functions_3d_convolution.py:45(check_spin)
3        5.755    1.918    7.021    2.340 {built-in method exec}
120000  5.739    0.000    5.739    0.000 {method 'random' of 'numpy.random.mtrand.RandomState' objects}
70000   5.081    0.000    5.081    0.000 {built-in method scipy.ndimage.nd_image.correlate}
70000   2.269    0.000    9.207    0.000 grid_functions_3d_convolution.py:35(get_energy_grids)
1        1.733    1.733    3.267    3.267 IO_functions.py:89(plot_total_spin_changes_of_each_point)
```

Usando 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.

3.7. 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])
```

3.7.1. Resultados

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

```
All tests finished in 16.87 seconds
| | | 9090320 function calls (8675590 primitive calls) in 23.588 seconds

Ordered by: internal time

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
   8953    6.203    0.001    6.362    0.001 ffi.py:190(__call__)
    990    2.733    0.003    2.733    0.003 {method 'read' of '_io.BufferedReader' objects}
      3    2.686    0.895    3.367    1.122 {built-in method exec}
   10000    2.263    0.000    2.269    0.000 grid_functions_3D_Numba.py:88(monte_carlo_cycle)
      1    1.303    1.303   16.873   16.873 ising_main.py:170(main)
```

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.

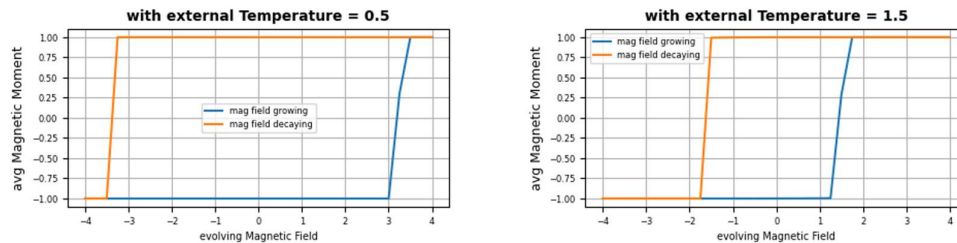
4. Análise Física

4.1. 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:



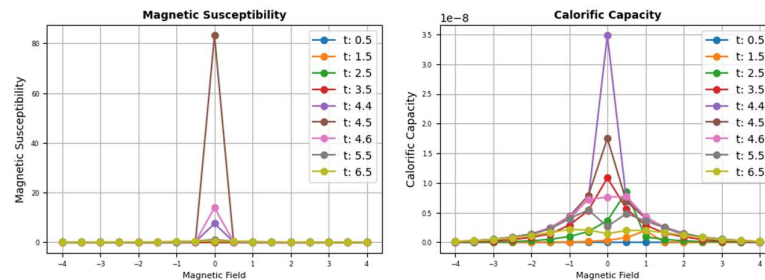
Os parâmetros utilizados foram os seguintes:

```
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.



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.

4.2. 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.

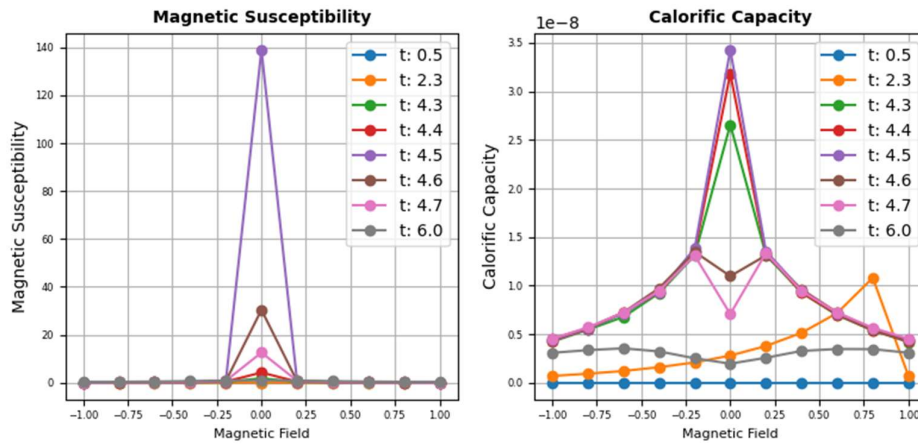
Capacidade Calorífica

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:

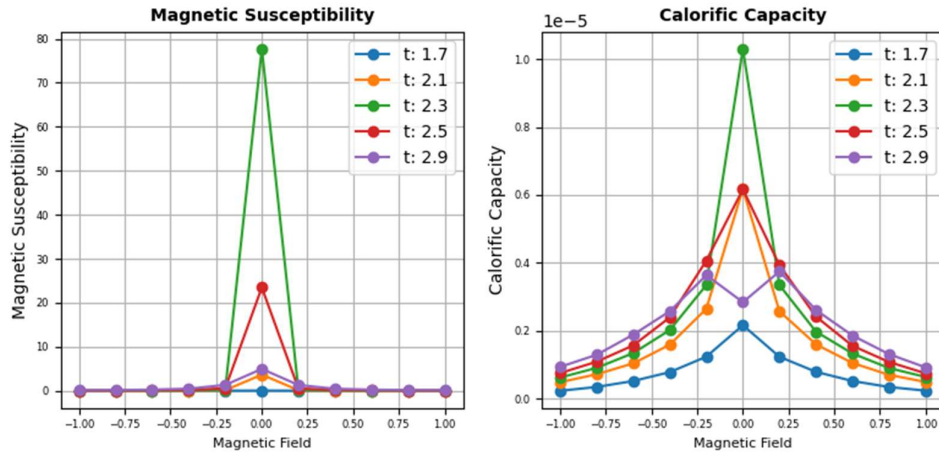
[ALL_TEMP_AND_MAG_FIELD_TEST](#), e analisando a suscetibilidade magnética e a capacidade calorífica, com um campo magnético externo = 0 (ou seja, a linha vertical central do gráfico), vemos que tanto a suscetibilidade magnética como a capacidade calorífica atingem o pico com uma temperatura externa de cerca de 4.5, para uma rede 3D, próximo do valor teórico conhecido de 4.51:



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:

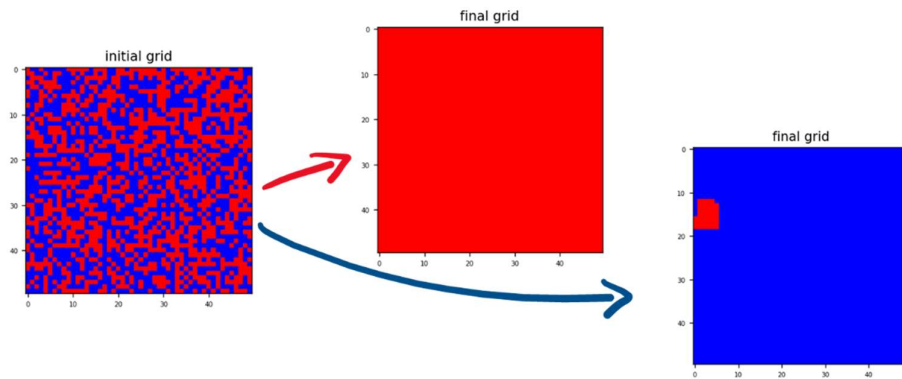
$$\text{temperatura} = \frac{KT}{J} \quad \text{campo magnético externo} = \frac{\mu_B B}{J}$$

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.



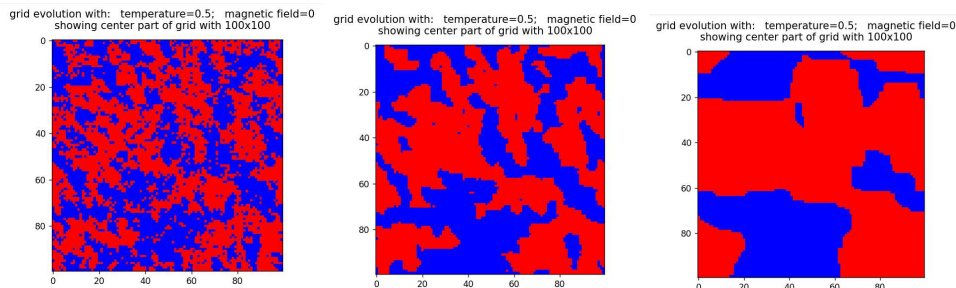
4.3. Ferromagnetismo-Paramagnetismo Histerese

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. Uma vez os vermelhos ganham, outras vezes os azuis:



Nota complementar:

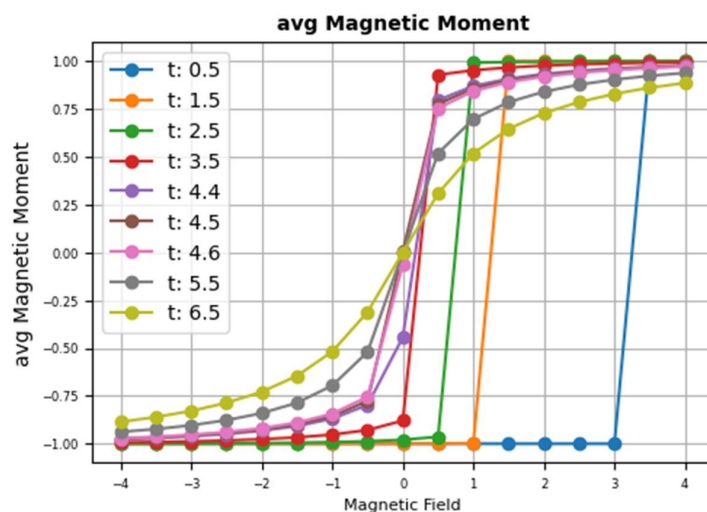
Como vemos, a rede a azul ficou com uma “ilha” de spins vermelhos. Isto acontece em redes muito grandes, em que de facto os spins vão se todos orientar no mesmo sentido, mas como a rede é grande de mais, vão-se criar domínios de spin igual, dentro dos quais o material fica estável e não há troca de spin, 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 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.

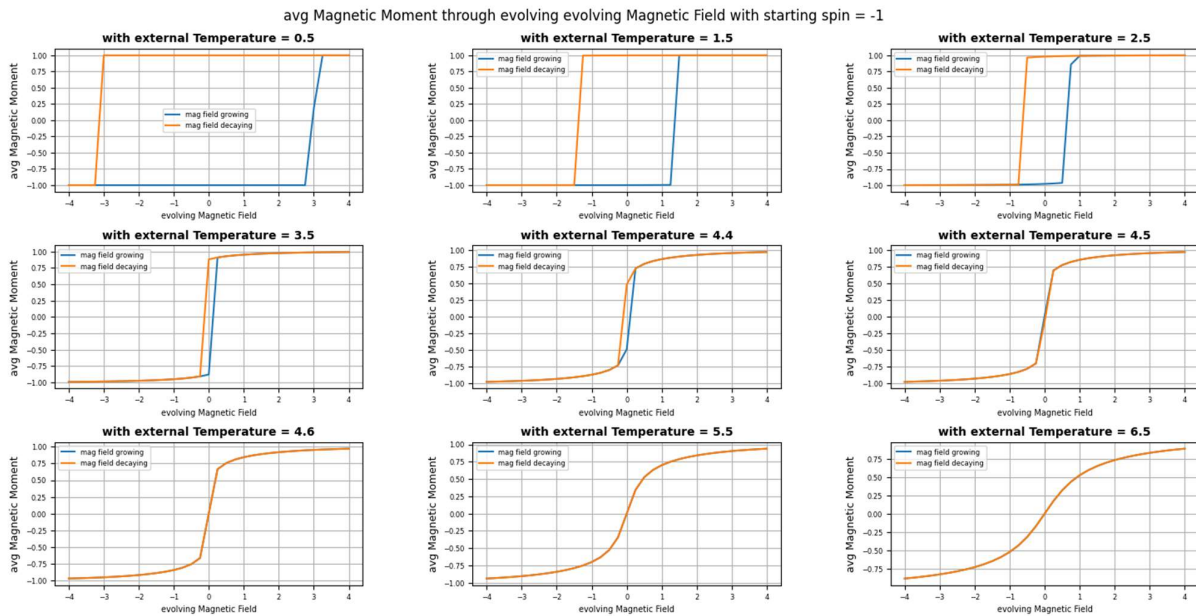
Olhando para o centro do gráfico, percebemos que com temperaturas mais elevadas, o momento magnético da rede se vai aproximando de 0. Mas com temperaturas baixas, o spin praticamente não se altera e o momento magnético continua próximo de -1.

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.



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 correremos 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.



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.

5. 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 VPU's ou SIMD dos CPU's 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 demonstrem 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.

6. 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).