exercicio-03-giuliavieira

December 13, 2023

UNIVERSIDADE FEDERAL DE MINAS GERAIS INSTUTUTO DE CIÊNCIAS EXATAS GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DISCIPLINA: Introdução a Física Estatística e Computacional

ALUNA: Giulia Monteiro Silva Gomes Vieira MATRICULA: 2016006492

EXERCÍCIO AVALIATIVO 03: Propriedades Ising 2D

[58]:
```
import numpy as np
from numba import jit
import matplotlib.pyplot as plt
from scipy.optimize import fsolve
```

[2]:
```
def initialize_lattice(size, initial_state):
    if initial_state == 'random':
        return 2 * np.random.randint(2, size=(size, size)) - 1
    elif initial_state == 'aligned':
        return np.ones((size, size), dtype=int)
    else:
        raise ValueError("Invalid value for beta. Use 0 or np.inf.")
```

[3]:
```
def calculate_energy(lattice, i, j):
    size = lattice.shape[0]
    spin = lattice[i, j]
    neighbors_sum = lattice[(i + 1) % size, j] + lattice[i, (j + 1) % size] + \
                    lattice[(i - 1) % size, j] + lattice[i, (j - 1) % size]
    return -spin * neighbors_sum
```

[4]:
```
def calculate_magnetization(lattice):
    return np.sum(lattice)
```

[5]:
```
def metropolis(lattice, temperature):
    size = lattice.shape[0]
    i, j = np.random.randint(0, size, size=2)
    current_energy = calculate_energy(lattice, i, j)
    flip_spin = -lattice[i, j]
    new_energy = calculate_energy(lattice, i, j)
```

```
    if new_energy < current_energy or np.random.rand() < np.exp((current_energy␣
 ↪- new_energy) / temperature):
        lattice[i, j] = flip_spin
```

[6]:
```
def find_thermalization_steps(lattice, temperature, threshold=0.01,␣
 ↪max_steps=5000):
    prev_energy = 0.5 * np.sum(-lattice * (np.roll(lattice, 1, axis=0) + np.
 ↪roll(lattice, 1, axis=1)))

    for step in range(max_steps):
        metropolis(lattice, temperature)
        energy = 0.5 * np.sum(-lattice * (np.roll(lattice, 1, axis=0) + np.
 ↪roll(lattice, 1, axis=1)))

        if np.abs(energy - prev_energy) < threshold:
            return step

        prev_energy = energy

    return max_steps
```

[7]:
```
def ising_model_simulation(size, temperature, num_steps, initial_state):
    lattice = initialize_lattice(size, initial_state)
    energies = np.zeros(num_steps)
    magnetizations = np.zeros(num_steps)

    thermalization_steps = find_thermalization_steps(lattice, temperature)
    #print(f'Thermalization steps: {thermalization_steps}')

    for step in range(num_steps):
        metropolis(lattice, temperature)
        energy = 0.5 * np.sum(-lattice * (np.roll(lattice, 1, axis=0) + np.
 ↪roll(lattice, 1, axis=1)))
        magnetization = calculate_magnetization(lattice)

        energies[step] = energy
        magnetizations[step] = magnetization

    return lattice, energies, magnetizations
```

[8]:
```
def specific_heat(energies, temperature, N):
    return (np.average(np.power(energies, 2)) - np.power(np.average(energies),␣
 ↪2)) / (
        N * np.power(temperature, 2))
```

[9]:
```
def susceptibility(magnetizations, temperature, N):
    return (np.average(np.power(magnetizations, 2)) - np.power(np.
 ↪average(magnetizations), 2)) / (N * temperature)
```

[10]:
```
def error(arr):
    return np.sqrt(
        np.sum(np.power(arr - np.average(arr), 2)) / (arr.size * (arr.size - 1))
    )
```

[11]:
```
def metrics(step, safe, mc_steps, temperature, numSpins, energies,␣
 ↪magnetizations,):
    energies_att = np.array([energy for energy in energies[safe:]])

    magnetizations_mod = np.abs(magnetizations)
    magnetizations_att = np.array([magnetization for magnetization in␣
 ↪magnetizations_mod[safe:]])

    batches = int((mc_steps - safe) / step)
    tensor = np.zeros((batches, 4))

    for j in range(0, mc_steps - safe, step):
        cal = specific_heat(energies_att[j : j + step], temperature, numSpins)
        sus = susceptibility(magnetizations_att[j : j + step], temperature,␣
 ↪numSpins)

        ene = np.average(energies_att[j : j + step]) / numSpins
        mag = np.average(magnetizations_att[j : j + step]) / numSpins

        tensor[int(j / step)] = np.array([cal, sus, ene, mag])

    matrix = tensor.T
    erro_cal = error(matrix[0])
    erro_sus = error(matrix[1])
    erro_ene = error(matrix[2])
    erro_mag = error(matrix[3])
    errors = np.array([erro_cal, erro_sus, erro_ene, erro_mag])

    m = [np.average(matrix, axis=0) for matrix in tensor]

    return m, errors
```

01. Escolha dos parâmetros: motivação Tamanho do Sistema ( ): Depende da capacidade computacional e do objetivo da simulação. Tamanhos pequenos facilitam simulações inicias, tamanhos maiores são geralmente mais fidedignos a realidade. Dadas as simulações do exercício anterior eu julgaria que tamanhos maiores também compensam ruídos e têm comportamento mais ordenado.Temperatura de Simulação ( ): Grandes variações poderiam trazer mais informações sobre os comportamentos da energia e magnetismo. Uma forma de abordar esse problema seria começar

próximo a T = 0C e aumentar graduamente.Número de Passos para Termalização (456): Grande o suficiente para garantir que o sistema alcance o equilíbrio térmico.Número de Passos para Médias Termodinâmicas (789): Suficiente para garantir boas estatísticas, o que é complicado de definir, sua escolha é possivelmente mais baseada em tentativa e erro.

02. Comportamento das principais grandezas termodinâmicas Energia por Spin: Deve aumentar à medida que a temperatura aumenta.Magnetização por Spin: Deve diminuir à medida que a temperatura aumenta.Calor Específico: Pode apresentar picos indicando transições de fase.Susceptibilidade Magnética: Pode apresentar picos indicando transições de fase. **02.1 Calor específico e susceptibilidade magnética em função da temperatura** Quando variamos a temperatura em um sistema que se inicia com spins aleatórios vemos que o calor especifico e a susceptibilidade magnética funcionam em picos, ou seja, os valores crescem repentinamente próximos a uma temperatura t, e caem também repentinamente após t. Isso indica mudança de fase.

Quanto a ambos os parametros podemos avaliar que o comportamento é como se houvesse uma "explosão" que ocasiona a mudança de fase, retornando ao estado anterior, não há novo padrão de valor para nova fase.

[14]:
```
size = 32
t_base = 1.0
n_exp = 20
T = np.linspace(-2, 2, n_exp)
L = np.linspace(32, 100, n_exp)
num_steps = 1000
N = 1
PARTITION = 100
SAFETY_NUM = 500
```

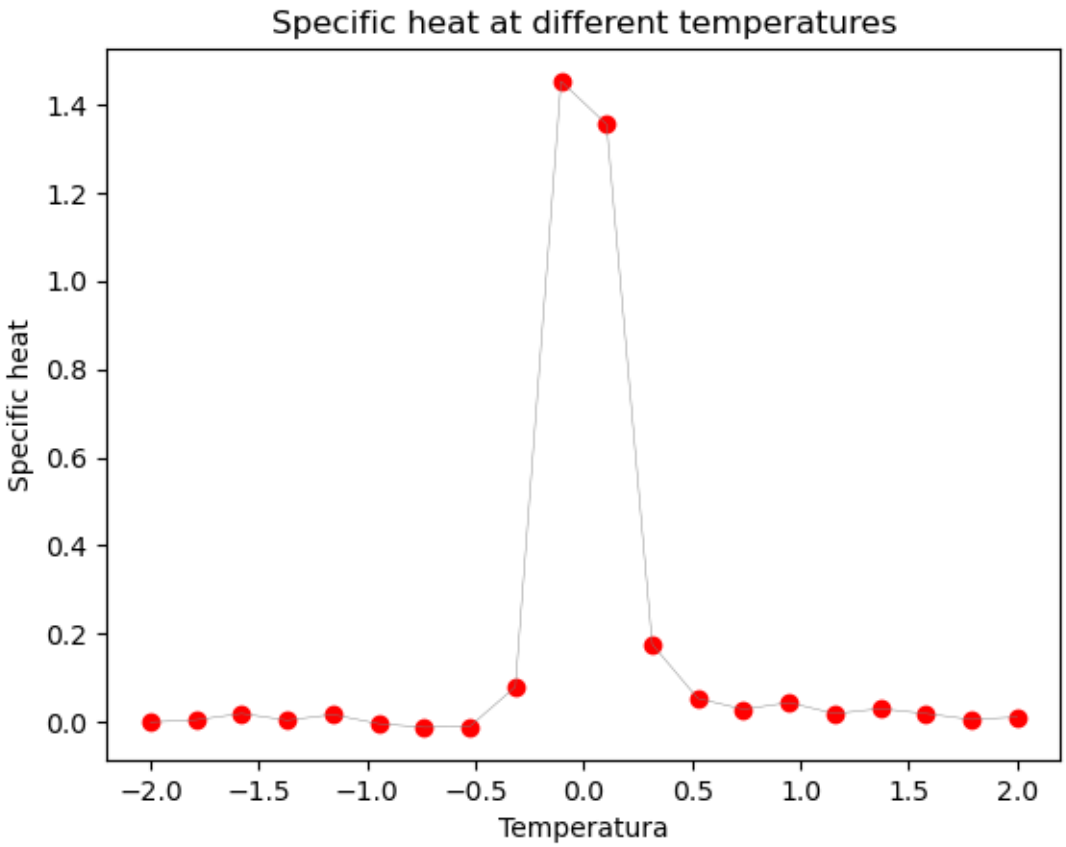
[15]:
```
plt.figure(1)
plt.figure(2);
```

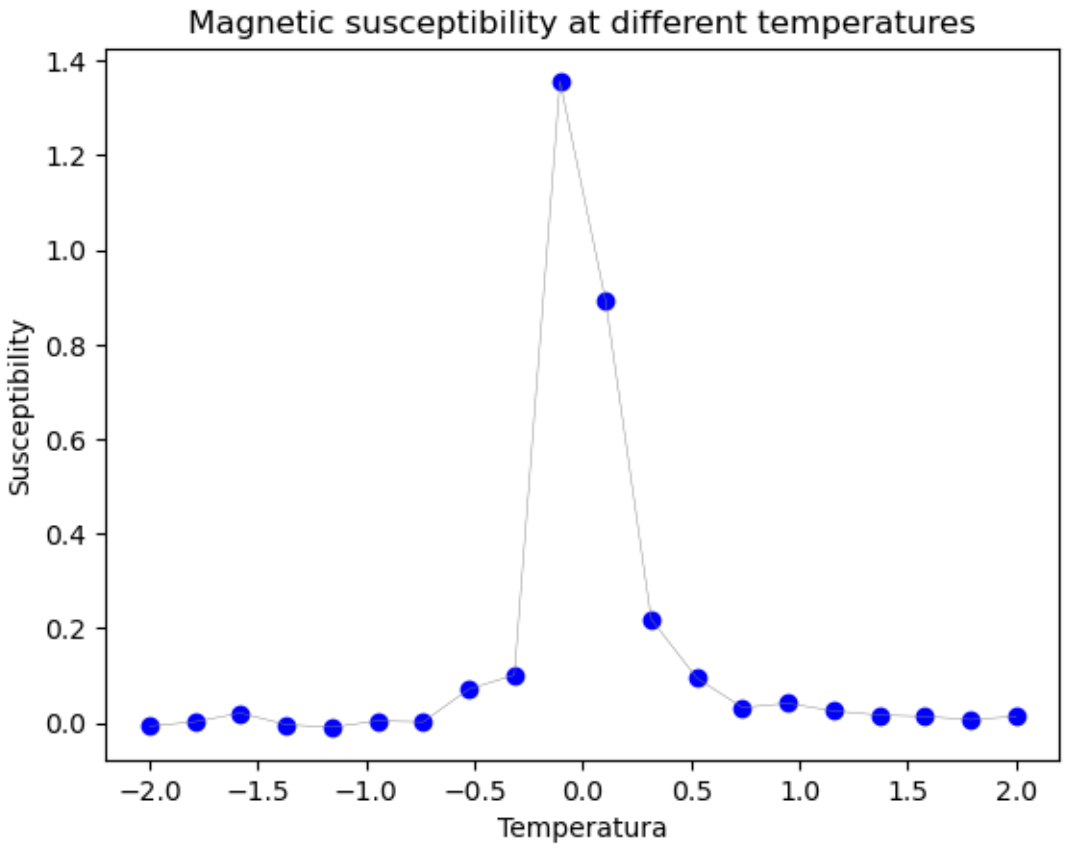
```
<Figure size 640x480 with 0 Axes>
```

```
<Figure size 640x480 with 0 Axes>
```

[17]:
```
m_vec_t = []
e_vec_t = []

for i, t in enumerate(T):
    # Simulation for equally distributed spins initial state
    lattice_random, energies_random, magnetizations_random =␣
 ↪ising_model_simulation(
        size, t, num_steps, initial_state='random')

    num_spins = np.power(size,2)

    m, e = metrics(PARTITION,
                   SAFETY_NUM,
```

```
                    num_steps,
                    t,
                    num_spins,
                    energies_random,
                    magnetizations_random,)

    m_vec_t.append(m)
    e_vec_t.append(e)


    # Plot energy for current temperature
    plt.figure(1)
    plt.scatter(t, m[0], color='Red')
    # Plot magnetization current temperature
    plt.figure(2)
    plt.scatter(t, m[1], color='Blue')


# Set plot labels and title
plt.figure(1)
heat_vec = [row[0] for row in m_vec_t]
plt.plot(T, heat_vec, color='gray', linewidth=0.3)
plt.title(f'Specific heat at different temperatures')
plt.xlabel('Temperatura')
plt.ylabel('Specific heat')

plt.figure(2)
mag_vec = [row[1] for row in m_vec_t]
plt.plot(T, mag_vec, color='gray', linewidth=0.3)
plt.title(f'Magnetic susceptibility at different temperatures')
plt.xlabel('Temperatura')
plt.ylabel('Susceptibility')

plt.show()
```


Specific heat at different temperatures
Specific heat
Temperatura
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0



03. Variação com o Tamanho do Sistema Em sistemas pequenos, o número limitado de partículas pode levar a variações no calor específico devido a efeitos de superfície e confinamento quântico.

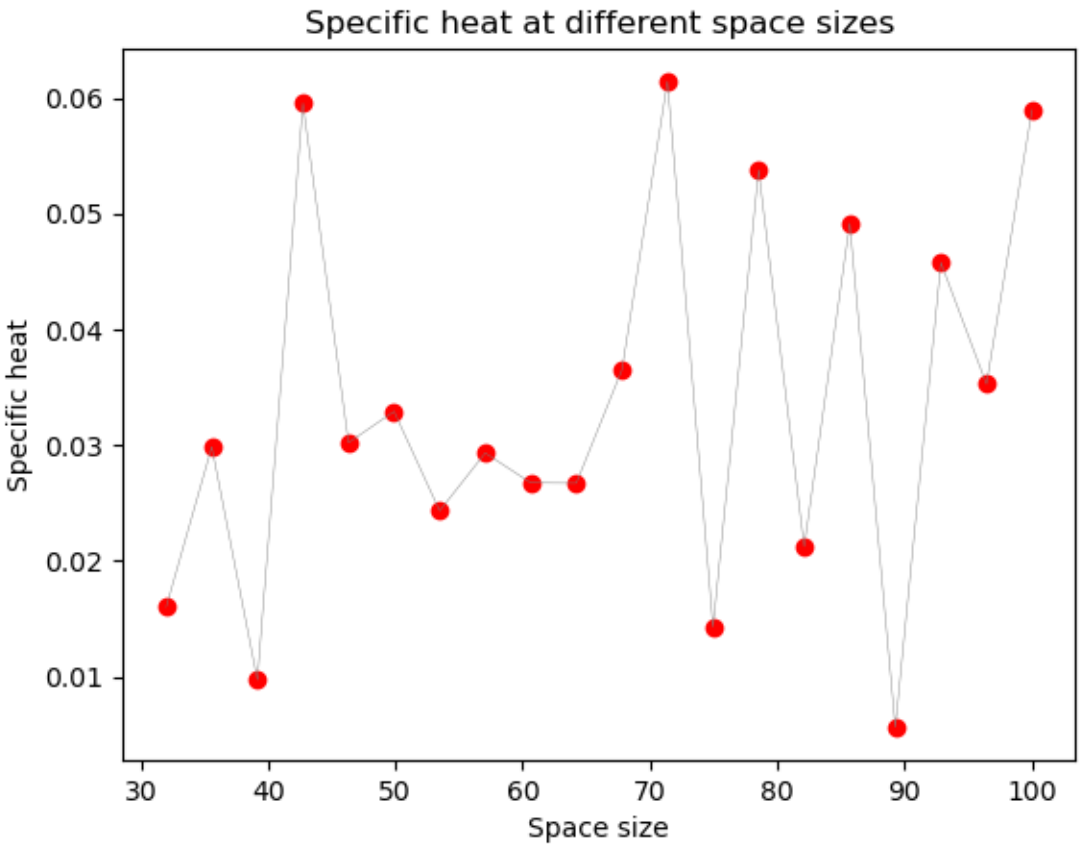
Em amostras pequenas, efeitos de borda e superfície podem desempenhar um papel importante, afetando as propriedades térmicas e magnéticas.

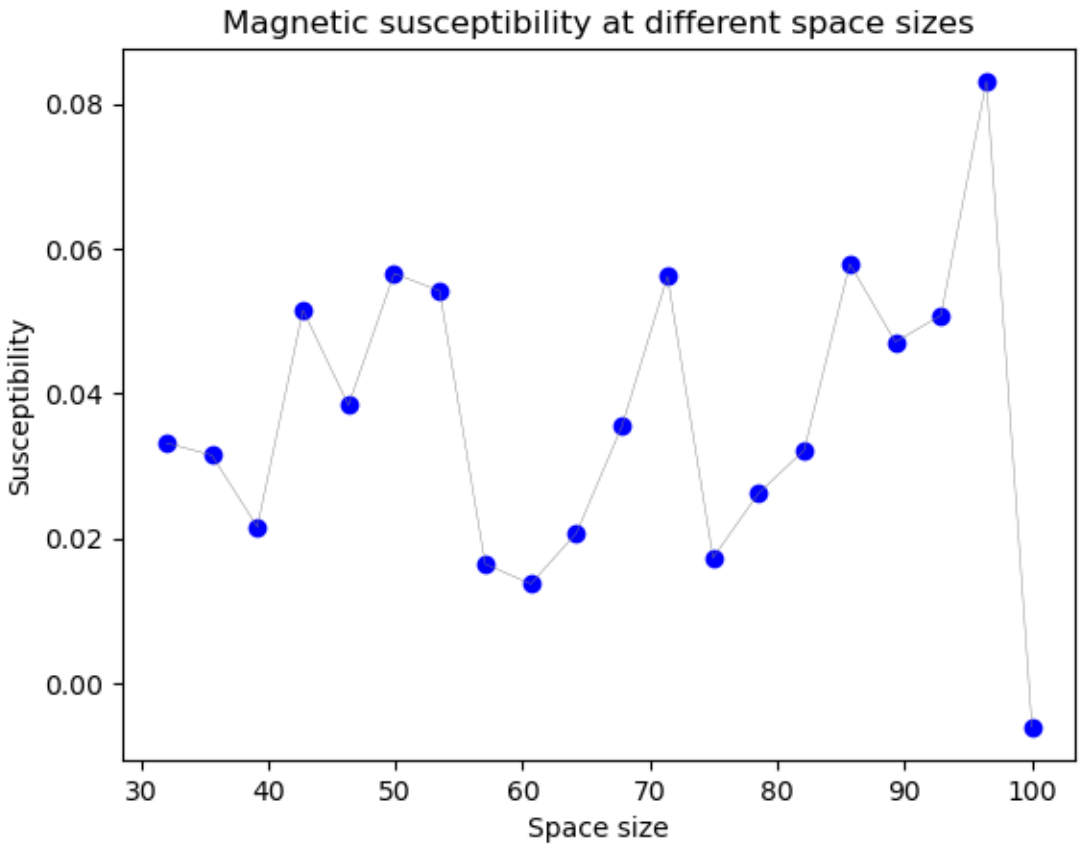
[ ]:
```
plt.figure(1)
plt.figure(2);
```

[18]:
```
t = 1.0
```

[19]:
```
m_vec_l = []
e_vec_l = []

for i, l in enumerate(L):
    # Simulation for equally distributed spins initial state
    lattice_random, energies_random, magnetizations_random =␣
 ↪ising_model_simulation(
        int(l), t, num_steps, initial_state='random')
```

```
    num_spins = np.power(size,2)

    m, e = metrics(PARTITION,
                   SAFETY_NUM,
                   num_steps,
                   t,
                   num_spins,
                   energies_random,
                   magnetizations_random,)

    m_vec_l.append(m)
    e_vec_l.append(e)


    # Plot energy for current temperature
    plt.figure(1)
    plt.scatter(l, m[0], color='Red')
    # Plot magnetization current temperature
    plt.figure(2)
    plt.scatter(l, m[1], color='Blue')


# Set plot labels and title
plt.figure(1)
heat_vec = [row[0] for row in m_vec_l]
plt.plot(L, heat_vec, color='gray', linewidth=0.3)
plt.title(f'Specific heat at different space sizes')
plt.xlabel('Space size')
plt.ylabel('Specific heat')

plt.figure(2)
mag_vec = [row[1] for row in m_vec_l]
plt.plot(L, mag_vec, color='gray', linewidth=0.3)
plt.title(f'Magnetic susceptibility at different space sizes')
plt.xlabel('Space size')
plt.ylabel('Susceptibility')

plt.show()
```


Specific heat at different space sizes
Specific heat
0.06
0.05
0.04
0.03
0.02
0.01
30
40
50
60
70
80
90
100
Space size



04. Comportamento dos erros estatísticos: Dados os erros calculados para as chamadas anteriores:

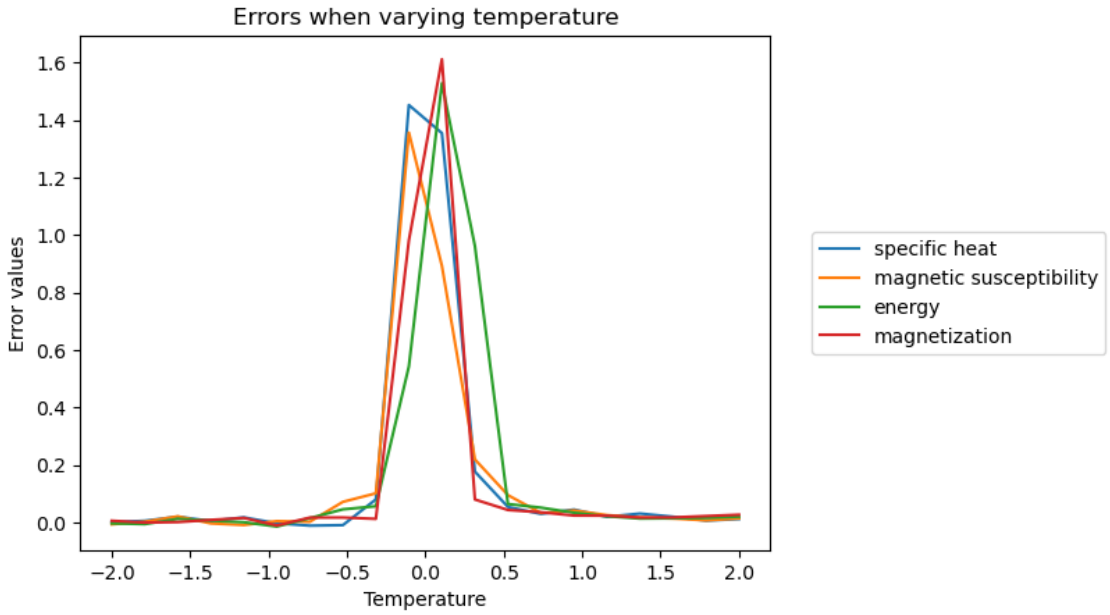
Podemos perceber que quando há variação de temperatura os valores de erro permanecem baixos, a não ser que estejamos próximos a um momento de troca de fase, no caso temperatura igual a zero, onde então há um pico de erros. Isso pode se dar ao fato de que as características do sistema mudam na mudança de fase, o tornando mais imprevisível.

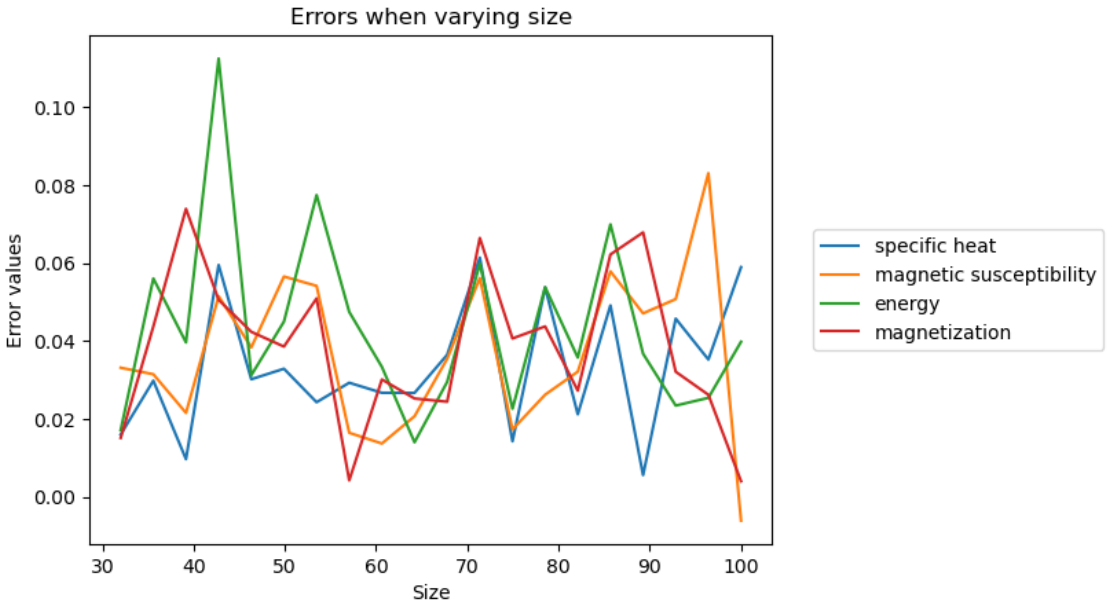
Podemos também observar que quando há variação no espaço do sistema os erros não possuem padrão específico, mas se mantém entre 0 e 0.1, valores baixos. Mudanças no tamanho podem influenciar mudanças de fase, mas normalmente, dadas outras variáveis fixas, o sistema se comporta da mesma maneira independente de seu tamanho. Sendo assim, os valores de erro permanecem numa mesma faixa sem movimento direcionado ao variarmos o tamanho do sistema.

[ ]:
```
plt.figure(1)
plt.figure(2);
```

[54]:
```
plt.figure(1)
plt.plot(T, [row[0] for row in m_vec_t], label='specific heat')
plt.plot(T, [row[1] for row in m_vec_t], label='magnetic susceptibility')
plt.plot(T, [row[2] for row in m_vec_t], label='energy')
```

```
plt.plot(T, [row[3] for row in m_vec_t], label='magnetization')
plt.title(f'Errors when varying temperature')
plt.xlabel('Temperature')
plt.ylabel('Error values')
plt.legend(loc='center right', bbox_to_anchor=(1.5, 0.5))

plt.figure(2)
plt.plot(L, [row[0] for row in m_vec_l], label='specific heat')
plt.plot(L, [row[1] for row in m_vec_l], label='magnetic susceptibility')
plt.plot(L, [row[2] for row in m_vec_l], label='energy')
plt.plot(L, [row[3] for row in m_vec_l], label='magnetization')
plt.title(f'Errors when varying size')
plt.xlabel('Size')
plt.ylabel('Error values')
plt.legend(loc='center right', bbox_to_anchor=(1.5, 0.5))

plt.show()
```





0.5 Fases do sistema e temperatura de transição Fase 01: fase de temperaturas negativas.

Calor específico, susceptibilidade magnética e erros baixos.Transição: temperatura 0

Pico de calor específico e susceptibildiade magnética.Fase 02: temperaturas positivas.

Calor específico, susceptibilidade magnética e erros baixos. **06. Temperatura de transição do sistema no limite termodinâmico** Dadas as análises anteriores, no limite termodinâmico teríamos uma temperatura de transição igual ou próxima a zero.

Com o uso do scipy decidi calcular o valor "real" desta transição, que foi estimado por volta de 1.29 graus. Para um sistema macro, está próximo o suficiente da estimativa

[56]:
```
def mean_field_approximation(temperature, coupling_constant):
    return temperature - 2 * coupling_constant * np.tanh(1 / temperature)
```

[71]:
```
coupling_constant = 1.0  # Interaction strength

# Range of temperatures
temperatures = np.linspace(0.01, 5.0, 100)

# Calculate the mean-field approximation for each temperature
mean_field_results = mean_field_approximation(temperatures, coupling_constant)

# Estimate the transition temperature using scipy's fsolve
transition_temperature_estimate = fsolve(mean_field_approximation, 2.0,␣
 ↪args=(coupling_constant,))

print("Transition temperature estiamte:", transition_temperature_estimate[0])
```

```
Transition temperature estiamte: 1.2958364580592054
```

[70]:
```
# Plot the results
plt.plot(temperatures, mean_field_results, label='Mean-Field Approximation')
plt.axhline(transition_temperature_estimate[0], color='red', linestyle='--',␣
 ↪label='Transition Point')
plt.xlabel('Temperature')
plt.ylabel('Order Parameter')
plt.title('Mean-Field Approximation for Transition Temperature')
plt.legend()
plt.show()
```