Aqui está um exemplo de código em Python que utilizamos para nossas simulações:

```python
import numpy as np
import matplotlib.pyplot as plt
import random
import time
import math
import matplotlib.ticker as ticker
from scipy.special import hankel1
from matplotlib.patches import Circle
from matplotlib.colors import LogNorm
import matplotlib.patches as patches

# Parâmetros:
NAtoms = 1000  # Número de átomos
canal = 0      # Canal: 0 (escalar), 1 (vetorial)
rho = 1        # Densidade da nuvem

R = np.sqrt((NAtoms / rho) / np.pi)  # Calcula o raio da nuvem
print(f"\r Raio: ({R:.2f})", end='')

# Geração da nuvem em coordenadas aleatórias
x = 2 * R * (np.random.rand(NAtoms) - 0.5)
y = 2 * R * (np.random.rand(NAtoms) - 0.5)

# Garantindo que os átomos estão dentro do raio da nuvem
for j in range(NAtoms):
    while np.sqrt(x[j]**2 / R**2 + y[j]**2 / R**2) > 1:
        x[j] = 2 * R * (np.random.rand() - 0.5)
        y[j] = 2 * R * (np.random.rand() - 0.5)

# Visualização dos átomos
plt.scatter(x, y, color='blue')
circle = Circle((0, 0), R, color='black', fill=False, linestyle='--',
↪  linewidth=2)
plt.gca().add_patch(circle)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Átomos dentro da nuvem')
plt.grid(True)
```

```
plt.axis('equal')
plt.show()

# Distâncias entre pares de átomos
xij = np.outer(x, np.ones(NAtoms)) - np.outer(np.ones(NAtoms), x) + np.eye(NAtoms)
yij = np.outer(y, np.ones(NAtoms)) - np.outer(np.ones(NAtoms), y) + np.eye(NAtoms)

# Distância 2D
dij = np.sqrt(xij**2 + yij**2)
phiij = np.arctan2(yij, xij)

# Matriz auxiliar
aux = np.ones((NAtoms, NAtoms)) - np.eye(NAtoms)

# Matriz de kernel escalar
gamma00 = np.eye(NAtoms) + aux * hankel1(0, dij)

# Verifica se o canal é vetorial (1) ou escalar (0)
if canal == 1:
    hankel2 = aux * hankel1(2, dij)
    gammaMP = -np.exp(2j * phiij) * hankel2
    gammaPM = -np.exp(-2j * phiij) * hankel2
    Gamma = np.block([[gamma00, gammaMP], [gammaPM, gamma00]])
else:
    Gamma = gamma00

# Diagonalização
diag_time = time.time()
e_vec_0, v_vec0 = np.linalg.eig(Gamma)
demorou_diag_time = time.time() - diag_time
print(f'Levou {demorou_diag_time:.3f} segundos para diagonalizar a matriz')

# Limpeza de memória
dij, phiij, aux, hankel2, gammaMP, gammaPM = [None] * 6

# Extração de parte real e imaginária dos autovalores
Gamma_real = np.real(e_vec_0)
Gamma_imag = np.imag(e_vec_0)
```

```

# Filtragem por autovalores reais acima de um corte
corte = 1e-2
mask = Gamma_real > corte
Gamma_14_np = Gamma_real[mask]
omega_14_np = Gamma_imag[mask] if len(Gamma_imag) > 0 else np.array([])
autovec_14_np = v_vec0[:, mask] if len(v_vec0) > 0 else np.array([])

# Cálculo do IPR
def ipr(autovetor):
    return sum(np.abs(autovetor)**4) / (sum(np.abs(autovetor)**2)**2)

ipr_n = np.array([ipr(autovec) for autovec in autovec_14_np.T])

# Visualização do espectro
fontsize_value = 40
fontsize_value_solo = fontsize_value - 20

fig, axs = plt.subplots(1, figsize=(13, 10))
scatter = axs.scatter(omega_14_np, Gamma_14_np, c=ipr_n,
↪  cmap='viridis', s=200, norm=LogNorm())
cbar = fig.colorbar(scatter, ax=axs)
cbar.set_label('IPR', fontsize=fontsize_value_solo)
axs.set_xlabel('', fontsize=fontsize_value_solo)
axs.set_ylabel('', fontsize=fontsize_value_solo)
axs.set_yscale('log')
axs.tick_params(axis='both', labelsize=fontsize_value_solo)
cbar.ax.tick_params(labelsize=fontsize_value_solo)
plt.show()

```

Código usado no cluster para gerar apenas uma nuvem, em que aparece o espectro e o modo

```

import numpy as np
import matplotlib.pyplot as plt
import random
import time
import datetime
import math
```

```
import matplotlib.ticker as ticker
import inspect
import os
import tracemalloc

from sklearn.preprocessing import StandardScaler
from sympy import *
from scipy.special import hankel1
from matplotlib.patches import Circle
from matplotlib.colors import LogNorm


def convert_size(size_bytes):
    if size_bytes == 0:
        return "0B"
    size_name = ("B", "KB", "MB", "GB", "TB", "PB", "EB", "ZB", "YB")
    i = int(math.floor(math.log(size_bytes, 1024)))
    p = math.pow(1024, i)
    s = round(size_bytes / p, 2)
    return f"{s} {size_name[i]}"

def get_variable_name(variable):
    callers_local_vars = inspect.currentframe().f_back.f_locals.items()
    for var_name, var_val in callers_local_vars:
        if var_val is variable:
            return var_name
    return None

def print_variable_size(variable):
    var_name = get_variable_name(variable)
    if var_name is None:
        print("Variable not found, size is 0.")
        size_str = "0 B"
    else:
        # Usando __sizeof__() em vez de sys.getsizeof()
        size_bytes = variable.__sizeof__()
        size_str = convert_size(size_bytes)
        print(f"Size of '{var_name}': {size_str}.")

# Defining function to calculate IPR.
def ipr(autovetor):
```

```
    ipr = sum(np.power(abs(autovetor), 4))/
    ↪  (sum(np.power(abs(autovetor), 2))**2)

    return ipr


def
↪  fazendoImagem(NAtoms,rho,vetorial,output_dir,output_filename_0,output_filename_1,xlim,task_id):

    # Inicia o monitoramento
    tracemalloc.start()

    # Cria a pasta se ela não existir
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)
        print(f"Pasta '{output_dir}' criada.")


    agora = datetime.datetime.now()
    hora_atual_Comeco = agora.strftime("%Y %m %d  %H %M")
    print(f"Task ID: {task_id} - A hora atual é:", hora_atual_Comeco)
    tempo_sim_start = time.time()

    Nr = 1
    # X and Y sizes

    R = np.sqrt((NAtoms / rho) / (np.pi * Nr))                          #
    ↪  Calcula o raio baseado na densidade
    print(f'Task ID: {task_id} - Raio: {R:.2f} e rho:{rho}')

    # Homogeneous distribution
    x = 2 * R * (np.random.rand(NAtoms) - 0.5)                          #
    ↪  Subtrai esse meio para o circulo ficar de -r a r.
    y = 2 * R * (np.random.rand(NAtoms) - 0.5)



    for nj in range(NAtoms):
        while np.sqrt(x[nj]**2 / R**2 + y[nj]**2 / R**2) > 1:
        ↪  # Roda a matriz N atoms, substituindo numeros fora do
        ↪  circulo.
```

```
            x[nj] = 2 * R * (np.random.rand() - 0.5)
            y[nj] = 2 * R * (np.random.rand() - 0.5)


    # Distance between each pair of atoms
    xij = np.outer(x, np.ones(NAtoms)) - np.outer(np.ones(NAtoms), x) +
    ↪  np.eye(NAtoms)
    yij = np.outer(y, np.ones(NAtoms)) - np.outer(np.ones(NAtoms), y) +
    ↪  np.eye(NAtoms)

    # # Limpar x e y
    # x = None
    # y = None


    # 2D Distance modulus
    dij = np.sqrt(xij**2 + yij**2)
    phiij = np.arctan2(yij,xij)


    # # Limpar xij e yij
    xij = None
    yij = None

    # Auxiliar matrix
    aux = np.ones((NAtoms, NAtoms)) - np.eye(NAtoms)

    # Scalar kernel matrix
    gamma00 = np.eye(NAtoms) + aux * hankel1(0, dij)



    if vetorial == 1:
        # Vector kernel matrix

        hankel2 = aux*(hankel1(2,dij))
        # gammaMM = gamma00
        gammaMP = -np.exp(2*1j*phiij)*hankel2

        gammaPM = -np.exp(-2*1j*phiij)*hankel2
        # gammaPP = gamma00
```

```
        Gamma = np.block([[gamma00, gammaMP], [gammaPM, gamma00]])
    else:
        Gamma = gamma00

    print_variable_size(Gamma)

    matrix_memory = Gamma.nbytes / (1024 ** 3)  # Em megabytes (GB)

    ###LIMPANDO###
    gamma00 = None
    dij = None
    phiij = None
    aux = None
    gammaMP = None
    gammaPM = None
    hankel2 = None



    start_time = time.time()
    # Diagonalization
    e_vec_0, v_vec0 = np.linalg.eig(Gamma)
    Gamma_real = np.real(e_vec_0)
    Gamma_imag = np.imag(e_vec_0)
    # Medindo o tempo final
    end_time = time.time()

    ###LIMPANDO###
    Gamma = None
    e_vec_0 = None


    # Calculando o tempo decorrido
    execution_time = (end_time - start_time)/3600
    print(execution_time)

    nome_arquivo = f"dados_da_figura-{output_filename_0}.txt"
    # Caminho completo do arquivo
    caminho_arquivo = os.path.join(output_dir, nome_arquivo)

    # Salvar as variáveis no arquivo .txt
```

```
    with open(caminho_arquivo, "w") as arquivo:
        arquivo.write(f"NAtoms:{NAtoms}\t tempo: {execution_time}\t
        ↪  rho: {rho}  \t canal:{vetorial} \n")

    print(f"Arquivo salvo em: {caminho_arquivo}")



    corte = 1e-14
    # Criar uma máscara booleana para filtrar os índices
    mask = Gamma_real > corte

    # Aplicar a máscara para filtrar os valores
    ind_14_np_1 = np.nonzero(mask)[0]  # Obtém os índices onde
    ↪  `Gamma_real > corte`
    Gamma_14_np = Gamma_real[mask]
    omega_14_np = Gamma_imag[mask] if len(Gamma_imag) > 0 else
    ↪  np.array([])
    autovec_14_np = np.transpose(v_vec0[:, mask] if len(v_vec0) > 0
    ↪  else np.array([]))


    ###LIMPANDO###
    v_vec0 = None

    ipr_n = np.empty(len(autovec_14_np))
    for i in range(len(autovec_14_np)):
        ipr_n[i] = ipr(autovec_14_np[i, :])

    menor_valor = np.nanmin(Gamma_14_np)
    # Encontrando o índice do menor valor
    i_gamma_min = np.where(Gamma_14_np == menor_valor)[0][0]
    i_gamma_max = 1
    fontsize_value = 14

    if vetorial == 1:
        print('Vetorial')
        # Inicialize prob_vetorial com zeros, sem loop explícito
        prob_vetorial = np.zeros((len(Gamma_14_np), len(x)))


```

```
        # Seleciona os elementos de autovec_14_np nos índices pares (2*i) e ímpares (2*i + 1)
        pares = np.abs(autovec_14_np[:, ::2])**2
        impares = np.abs(autovec_14_np[:, 1::2])**2

        # Somando os quadrados dos pares e ímpares para obter o resultado final
        prob_vetorial = pares + impares

        ###LIMPANDO###
        pares = None
        impares = None

        # Calculate IPR for all modes.
        ipr_n = np.empty(len(autovec_14_np))
        for i in range(len(autovec_14_np)):
            ipr_n[i] = ipr(autovec_14_np[i, :])


        # Criando os subplots (1 linha, 2 colunas)
        fig1, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

        # Primeiro scatter plot com barra de cores
        scatter1 = ax1.scatter(omega_14_np, Gamma_14_np, c=(ipr_n), cmap='viridis',norm=LogNorm())
        ax1.set_yscale('log')
        ax1.set_xticks([-10000, 10000])
        fig1.colorbar(scatter1, ax=ax1, label='IPR')

        # Segundo scatter plot com barra de cores
        scatter2 = ax2.scatter(omega_14_np, Gamma_14_np, c=(ipr_n), cmap='viridis',norm=LogNorm())
        ax2.set_yscale('log')
        ax2.set_xlim(-70, 70)
        fig1.colorbar(scatter2, ax=ax2, label='IPR')


        output_path_0 = os.path.join(output_dir, output_filename_0)

        fig1.savefig(output_path_0)  # Salvar a figura como PDF
        print(f'Figura salva em: {output_path_0}')
```

```
238         # Exibindo os gráficos
239         plt.close(fig1)  # Fechar a figura corretamente
240         # Calculate center of mass (x, y, r) for all modes.
241         xcm = np.dot(x,
            ↪  np.transpose(prob_vetorial))/np.sum(abs(prob_vetorial),
            ↪  axis=1)
242         ycm = np.dot(y,
            ↪  np.transpose(prob_vetorial))/np.sum(abs(prob_vetorial),
            ↪  axis=1)
243
244
245         tamanho = len(autovec_14_np[0])
246         # Calculate |r - rcm|, vector distance between the center of
            ↪  mass and the atoms, for all.
247         rd = []
248         for i in range(tamanho):
249             rd.append(np.sqrt((x-xcm[i])**2 + (y-ycm[i])**2 ))
250
251
252         def ordenando_vect(prob_vetorial,i_g):
253             # Certificando-se de que prob_vetorial[estadooo] seja um
                ↪  array numpy
254             prob_vetorial_estado = np.array(prob_vetorial[i_g])
255             # Reordenando as variáveis 'x', 'y', e 'prob_vetorial'
256             x_ordenado = x[np.argsort(prob_vetorial_estado)]
257             y_ordenado = y[np.argsort(prob_vetorial_estado)]
258             prob_vetorial_ordenado =
                ↪  prob_vetorial_estado[np.argsort(prob_vetorial_estado)]
259             rd_s = rd[i_g]  # Presumo que 'rd' seja uma lista ou array
                ↪  de dados
260             prob_s = np.log10(prob_vetorial[i_g])  # Presumo que
                ↪  'prob_vetorial' seja uma lista ou array de
                ↪  probabilidades
261
262             return
                ↪  prob_vetorial_ordenado,x_ordenado,y_ordenado,prob_s,rd_s
263
264
265         prob_vetorial_ordenado1, x_ordenado1, y_ordenado1,prob_s1,rd_s1
            ↪  = ordenando_vect(prob_vetorial,i_gamma_max)
266
```

```
        prob_vetorial_ordenado2, x_ordenado2, y_ordenado2,prob_s2,rd_s2 = ordenando_vect(prob_vetorial,i_gamma_min)

        # Criando uma figura e subplots
        fig2, axs = plt.subplots(2, 2, figsize=(12, 10))

        # Plotando os gráficos
        axs[0, 0].scatter(rd_s1, prob_s1, c='black')
        axs[1, 0].scatter(rd_s2, prob_s2, c='black')

        # Usando LogNorm para colorbar em escala logarítmica
        scatter = axs[0, 1].scatter(x_ordenado1, y_ordenado1, c=prob_vetorial_ordenado1, cmap='jet', norm=LogNorm())
        axs[0, 1].add_patch(Circle((0, 0), R, color='black', fill=False))  # Adicionando o círculo
        axs[0, 1].set_aspect('equal')  # Definindo a escala igual para os eixos
        scatter2 = axs[1, 1].scatter(x_ordenado2, y_ordenado2, c=prob_vetorial_ordenado2, cmap='jet', norm=LogNorm())
        axs[1, 1].add_patch(Circle((0, 0), R, color='black', fill=False))  # Adicionando o círculo
        axs[1, 1].set_aspect('equal')  # Definindo a escala igual para os eixos
        # Adicionando o colorbar aos gráficos
        cbar1 = fig2.colorbar(scatter, ax=axs[1, 1])
        cbar2 = fig2.colorbar(scatter2, ax=axs[0, 1])
        cbar1.set_label(r'$|\Psi^{(1)}_{j}|^2$', fontsize=fontsize_value)
        cbar2.set_label(r'$|\Psi^{(0)}_{j}|^2$', fontsize=fontsize_value)

        # Ajustando o layout
        plt.tight_layout()

        # Salvar a figura ou mostrar
        output_path_1 = os.path.join(output_dir, output_filename_1)

        fig2.savefig(output_path_1)  # Salvar a figura como PDF
        print(f'Figura salva em: {output_path_1}')
        # Exibindo os gráficos
        plt.close(fig2)  # Fechar a figura corretamente
```

```
299
300         xlim_escalar = 0
301     elif vetorial == 0:
302
303         # Calculate Psi**2 for all modes.
304         prob = (abs(autovec_14_np))**2
305
306         # Calculate center of mass (x, y, r) for all modes.
307         print(f"x shape: {x.shape}")
308         print(f"prob shape: {prob.shape}")
309         # print(f"autovec_14_np shape: {xcm.shape}")
310         xcm = np.dot(x, np.transpose(prob))/np.sum((prob), axis=1)
311         ycm = np.dot(y, np.transpose(prob))/np.sum((prob), axis=1)
312
313         # Calculate |r - rcm|, vector distance between the center of
            ↪  mass and the atoms, for all.
314         rd = []
315         for i in range(len(autovec_14_np)):
316             rd.append(np.sqrt((x-xcm[i])**2 + (y-ycm[i])**2 ))
317
318         ipr_n = np.empty(len(autovec_14_np))
319         for i in range(len(autovec_14_np)):
320             ipr_n[i] = ipr(autovec_14_np[i, :])
321
322         # Criando o gráfico
323         fig3, axs = plt.subplots(1, figsize=(12, 10))
324         scatter = axs.scatter(omega_14_np, Gamma_14_np, c=ipr_n,
            ↪  cmap='viridis', norm=LogNorm())
325         # Definindo os rótulos dos eixos
326         cbar = fig3.colorbar(scatter, ax=axs)
327         cbar.set_label('IPR')
328         axs.set_xlabel('')  # Eixo x
329         axs.set_ylabel('')  # Eixo y
330
331         # Aplicando escala logarítmica ao eixo y
332         axs.set_yscale('log')
333
334         output_path_0 = os.path.join(output_dir, output_filename_0)
335
336         fig3.savefig(output_path_0)  # Salvar a figura como PDF
337         print(f'Figura salva em: {output_path_0}')
```

```
338
339         # Exibindo os gráficos
340         plt.close(fig3)  # Fechar a figura corretamente
341
342         # Calculate Psi**2 for all modes.
343
344
345         # Criando uma figura e subplots
346         fig4, axs = plt.subplots(2, 2, figsize=(12, 10))
347
348         # Plotando os gráficos
349         axs[0, 1].scatter(rd[i_gamma_min], prob[i_gamma_max],
            ↪  color='blue')
350         axs[0, 1].set_yscale('log')
351         #axs[0, 1].set_ylim(bottom=10**-55)  # Definindo o limite
            ↪  inferior do eixo y
352         #axs[0, 1].set_xlim(0,100)
353         axs[0, 1].set_xlabel(r'$k|r_{j} - r_{cm}|$',
            ↪  fontsize=fontsize_value)
354         axs[0, 1].set_ylabel(r'$|\Psi^{(0)}_{j}|^2$',
            ↪  fontsize=fontsize_value)
355         axs[0, 1].xaxis.set_major_locator(ticker.MaxNLocator(nbins=6,
            ↪  integer=True))
356         axs[0, 1].yaxis.set_major_locator(ticker.LogLocator(base=10.0,
            ↪  numticks=5))
357
358
359         axs[1, 1].scatter(rd[i_gamma_min], prob[i_gamma_min],
            ↪  color='blue')
360         axs[1, 1].set_yscale('log')
361         axs[1, 1].set_xlabel(r'$k|r_{j} - r_{cm}|$',
            ↪  fontsize=fontsize_value)
362         axs[1, 1].set_ylabel(r'$|\Psi^{(1)}_{j}|^2$',
            ↪  fontsize=fontsize_value)
363         axs[1, 1].xaxis.set_major_locator(ticker.MaxNLocator(nbins=6,
            ↪  integer=True))
364         axs[1, 1].yaxis.set_major_locator(ticker.LogLocator(base=10.0,
            ↪  numticks=5))
365
366         # Usando LogNorm para colorbar em escala logarítmica
367         scatter = axs[1, 0].scatter(x, y, c=prob[i_gamma_min],
            ↪  cmap='jet', norm=LogNorm())
```

```
        axs[1, 0].add_patch(Circle((0, 0), R, color='black', fill=False))  # Adicionando o círculo
        axs[1, 0].axis('equal')
        axs[1, 0].set_xlabel(r'$x_{j}$', fontsize=fontsize_value)
        axs[1, 0].set_ylabel(r'$y_{j}$', fontsize=fontsize_value)
        axs[1, 0].xaxis.set_major_locator(ticker.MaxNLocator(nbins=5, integer=True))
        axs[1, 0].yaxis.set_major_locator(ticker.MaxNLocator(nbins=5, integer=True))

        scatter2 = axs[0, 0].scatter(x, y, c=prob[i_gamma_max], cmap='jet', norm=LogNorm())
        axs[0, 0].add_patch(Circle((0, 0), R, color='black', fill=False))  # Adicionando o círculo
        axs[0, 0].axis('equal')
        axs[0, 0].set_xlabel(r'$x_{j}$', fontsize=fontsize_value)
        axs[0, 0].set_ylabel(r'$y_{j}$', fontsize=fontsize_value)
        axs[0, 0].xaxis.set_major_locator(ticker.MaxNLocator(nbins=5, integer=True))
        axs[0, 0].yaxis.set_major_locator(ticker.MaxNLocator(nbins=5, integer=True))

        # Adicionando o colorbar aos gráficos
        cbar1 = fig4.colorbar(scatter, ax=axs[1, 0])
        cbar2 = fig4.colorbar(scatter2, ax=axs[0, 0])
        cbar1.set_label(r'$|\Psi^{(1)}_{j}|^2$', fontsize=fontsize_value)
        cbar2.set_label(r'$|\Psi^{(0)}_{j}|^2$', fontsize=fontsize_value)


        # Ajustando o layout
        plt.tight_layout()

        # Salvar a figura como PDF
        output_path_1 = os.path.join(output_dir, output_filename_1)
        fig4.savefig(output_path_1)  # Salvar a figura como PDF
        print(f'Figura salva em: {output_path_1}')
        # Exibindo os gráficos
        plt.close(fig4)  # Fechar a figura corretamente
    else:
```

```
        print('Erro')

    diferencas = -1*np.diff(omega_14_np[np.argsort(-omega_14_np)])

    g = (1/np.mean(1/Gamma_14_np))/np.mean(diferencas)

    ###LIMPANDO###
    prob = None
    rd = None
    prob_vetorial = None

    print('Teste de memoria')
    ##Tentar limpar um pouco da memoria
    # import sys

    # # Listar variáveis e seus tamanhos
    # variables = {name: sys.getsizeof(value) for name, value in
    ↪  locals().items()}
    # sorted_variables = sorted(variables.items(), key=lambda x: x[1],
    ↪  reverse=True)

    # # Exibir apenas as 5 variáveis maiores
    # print("As 5 variáveis que ocupam mais memória:")
    # for var_name, size in sorted_variables[:3]:  # Limitar às 5
    ↪  primeiras
    #     print(f"{var_name}: {size / (1024**3):.2f} GB")


    ########################


    tempo_sim = (time.time() - tempo_sim_start)/3600

    print(f"Task ID: {task_id} - Simulação concluída em
    ↪  {tempo_sim:.3f} horas.")
    print(f"Task ID: {task_id} - Diagonalização concluída em
    ↪  {execution_time:.3f} horas.")
    print(f"Task ID: {task_id} - A matriz ocupa aproximadamente
    ↪  {matrix_memory:.2f} GB de memória.")
    # Obtém estatísticas de pico
    current, peak = tracemalloc.get_traced_memory()
```

```
435     print(f"Task ID: {task_id} - Uso atual: {(current /(1024 **
        ↪  3)):.2f} GB")
436     print(f"Task ID: {task_id} - Pico de uso: {(peak / (1024 **
        ↪  3)):.2f} GB")
437
438     # Finaliza o monitoramento
439     tracemalloc.stop()
440
441
442     return execution_time
443
444
445
446 if __name__ == '__main__':
447
448     # Dicionário de parâmetros para cada gráfico
449     parametros = {
450         1: {"rho": 0.1, "vetorial": 0, "caso": "esclar"},
451         2: {"rho": 0.1, "vetorial": 1, "caso": "vetorial"},
452     }
453
454     # Obtém o ID do job array (passado pelo SLURM)
455     job_id = int(os.getenv('SLURM_ARRAY_TASK_ID', 1))
456
457     # Seleciona os parâmetros com base no ID
458     param = parametros[job_id]
459
460     NAtoms = 10000                      # Number of atoms
461     rho = param["rho"]                          # Homogeneous density
        ↪  of scatterers
462     vetorial = param["vetorial"]
463     caso = param["caso"]
464
465     print(f"RAM: 1*7")
466     print(f"rho: {rho}")
467     print(f"vetorial: {vetorial}")
468     print(f"caso: {caso}")
469
470     output_filename_0 =
        ↪  f'Imagem_espectro_{caso}_de_densidade_{rho}_N_{NAtoms}_210125.pdf'
471     output_filename_1 =
        ↪  f'Imagem_modo_{caso}_de_densidade_{rho}_N_{NAtoms}_210125.pdf'
```

```
    output_dir = 'meta20k'
    print('Entrou na simulação')

    a =
    ↪  fazendoImagem(NAtoms,rho,vetorial,output_dir,output_filename_0,output_filename_1,(-30,30),job
    print(f'Tempo:{a}')
    print('Saiu na simulação')
```

Codigo do **job_sing**:

```
#!/usr/bin/bash

#SBATCH -J N_10k2r                          # Nome do job
#SBATCH -o outputs/%j.out                   # Salvar saída padrão em
↪  "outputs/%j.out"
#SBATCH -e outputs/%j.err                   # Salvar saída de erro em
↪  "outputs/%j.err"
#SBATCH -t 11:59:00                         # Tempo limite de execução
↪  (11 horas e 59 minutos)
#SBATCH --cpus-per-task=1                   # Número de CPUs por tarefa
#SBATCH --ntasks-per-node=1                 # Tarefas por nó (ajustado
↪  para 1 por tarefa)
#SBATCH --partition=fast                    # Partição do cluster
↪  (ajustar conforme necessidade)
#SBATCH --mem-per-cpu=20G                   # Memória alocada por CPU
↪  (20 GB)
#SBATCH --mail-user=fuzitaalexandre@estudante.ufscar.br # E-mail para
↪  notificações
#SBATCH --mail-type=ALL                     # Notificações de todas as
↪  atualizações
#SBATCH --array=1-2                         # Array de jobs (de 1 a 2)

# Caminho para o interpretador Python no ambiente do cluster
PYTHON_EXEC=/usr/local/bin/python3          # Verifique se este é o
↪  caminho correto para o Python

# Início da simulação
echo "---- Simulação de diagonalização iniciada com Task
↪  ID=£{SLURM_ARRAY_TASK_ID}, N_10k_RAM20 ----"

# Executa o Python dentro do container Singularity
```

```
srun singularity exec Singularity.simg $PYTHON_EXEC test.py
↪ £{SLURM_ARRAY_TASK_ID}

# Fim da simulação
echo "---- Simulação de diagonalização FINALIZADA para Task
↪ ID=£{SLURM_ARRAY_TASK_ID}! ----"
```

Codigo de calcular a condutancia

```
#!/usr/bin/bash

#SBATCH -J N_10k2r                          # Nome do job
#SBATCH -o outputs/%j.out                   # Salvar saída padrão em
↪ "outputs/%j.out"
#SBATCH -e outputs/%j.err                   # Salvar saída de erro em
↪ "outputs/%j.err"
#SBATCH -t 11:59:00                         # Tempo limite de execução
↪ (11 horas e 59 minutos)
#SBATCH --cpus-per-task=1                   # Número de CPUs por tarefa
#SBATCH --ntasks-per-node=1                 # Tarefas por nó (ajustado
↪ para 1 por tarefa)
#SBATCH --partition=fast                    # Partição do cluster
↪ (ajustar conforme necessidade)
#SBATCH --mem-per-cpu=20G                   # Memória alocada por CPU
↪ (20 GB)
#SBATCH --mail-user=fuzitaalexandre@estudante.ufscar.br # E-mail para
↪ notificações
#SBATCH --mail-type=ALL                     # Notificações de todas as
↪ atualizações
#SBATCH --array=1-2                         # Array de jobs (de 1 a 2)

# Caminho para o interpretador Python no ambiente do cluster
PYTHON_EXEC=/usr/local/bin/python3          # Verifique se este é o
↪ caminho correto para o Python

# Início da simulação
echo "---- Simulação de diagonalização iniciada com Task
↪ ID=£{SLURM_ARRAY_TASK_ID}, N_10k_RAM20 ----"

# Executa o Python dentro do container Singularity
srun singularity exec Singularity.simg $PYTHON_EXEC test.py
↪ £{SLURM_ARRAY_TASK_ID}
```

```

# Fim da simulação
echo "---- Simulação de diagonalização FINALIZADA para Task ID=${SLURM_ARRAY_TASK_ID}! ----"
```

Usado para fazer o conteiner:

```
Bootstrap: docker
From: python

%help
# Informações sobre o container
# CLUSTER AJF
# Este container configura um ambiente Python com suporte a MPI e bibliotecas científicas.

%files
# Adicionar arquivos necessários ao container
test.py ./  # Copiar o arquivo test.py para o container

%environment
# Variáveis de ambiente para o container
export PYTHONPATH=/usr/local/bin/python3
export PATH="/usr/lib/openmpi/bin:$PATH"  # Adicionar o caminho do MPI ao PATH

%post
# Etapas de configuração do container
apt-get update
apt-get install -y apt-utils
apt-get install -y software-properties-common
apt-get install -y build-essential
apt-get install -y python3
apt-get install -y python3-pip
apt-get install -y openmpi-bin libopenmpi-dev  # Instalar OpenMPI

# Atualizar o pip e instalar pacotes necessários
pip install --upgrade pip
pip install numpy
pip install matplotlib
pip install scikit-learn
```

```
pip install sympy
pip install scipy
pip install mpi4py  # Instalar mpi4py para suporte a MPI no Python

%runscript
# Script executado quando o container é chamado
echo "---- Testando ambiente configurado ----"

# Testar as versões dos pacotes instalados
python -c "import numpy; print('NumPy versão:', numpy.__version__)"
python -c "import scipy; print('SciPy versão:', scipy.__version__)"
python -c "import matplotlib; print('Matplotlib versão:',
↪  matplotlib.__version__)"
python -c "import sklearn; print('Scikit-learn versão:',
↪  sklearn.__version__)"
python -c "import sympy; print('SymPy versão:', sympy.__version__)"

# Executar o script Python
echo "Executando test.py..."
python test.py
```

Comandos utilizados no Cluster Ufscar.

Comandos para Uso do Cluster at UFSCar e Terminal (usuário: u123456)

- **Acesso ao cluster:** `ssh u123456@openhpc.ufscar.br`
- **Enviar arquivo:** `scp <arquivo> u123456@openhpc.ufscar.br:`
- **Copiar arquivo:** `scp u123456@openhpc.ufscar.br:<arquivo> <caminho>`
- **Checar nós:** `scontrol show nodes`
- **Submeter trabalho:** `sbatch job_sing.sh`
- **Ver fila:** `squeue`
- **Cancelar trabalho:** `scancel <número>`
- **Listar arquivos:** `ls`
- **Caminho atual:** `pwd`
- **Visualizar arquivo:** `cat <arquivo>`

- **Monitorar arquivo:** `tail -f <arquivo>`
- **Auto-completar:** Pressione `Tab` (duas vezes)
- **Histórico de comandos:** Use as setas ↑ e ↓

Codigo que le os dados e faz um FIT:

```
import matplotlib.pyplot as plt
import numpy as np
from collections import Counter
from matplotlib.lines import Line2D
import datetime
import os
import numpy as np
import matplotlib.pyplot as plt
import random
import time
import math
import matplotlib.ticker as ticker
import inspect
import os
from matplotlib.lines import Line2D
from sklearn.preprocessing import StandardScaler
from sympy import *
from scipy.special import hankel1
from matplotlib.patches import Circle
from matplotlib.colors import LogNorm
from scipy.stats import linregress

import matplotlib.patches as patches
import pandas as pd

# Obtém a data e hora atuais
agora = datetime.datetime.now()
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
# Formata a hora atual como string
hora_atual = agora.strftime("%H:%M:%S")

print("A hora atual é:", hora_atual)

'''Funções Secundarias:'''
```

```python


'''Responsaveis por ler arquivo .txt'''
def lendoAqruivo(file_path,tamanhodcabeca):

    # Lendo o arquivo
    with open(file_path, "r") as file:
        # Lendo o cabeçalho
        header = [file.readline().strip() for _ in range(tamanhodcabeca)]  # Lê as 11 linhas do cabeçalho

        # Lendo os dados
        dataleitura = []
        for line in file:
            values = line.strip().split(", ")
            dataleitura.append([float(value) for value in values])

    # Armazenando uma linha específica do cabeçalho em uma variável
    print("Cabeçalho:")
    for line in header:
        print(line)
    return dataleitura,header

lista_cores = ['red', 'blue', 'green', 'yellow', 'purple', 'orange', 'pink', 'brown', 'gray', 'cyan']


def funcaoCor(array, lista_cores):
    # Contando a frequência de cada número
    contador = Counter(array)

    # Dicionário para armazenar a associação número-cor
    numero_para_cor = {}

    # Índice para a próxima cor na lista
    indice_cor = 0

    # Associando as cores aos números no array
    for numero in array:
        if numero not in numero_para_cor:
            numero_para_cor[numero] = lista_cores[indice_cor]
```

```
            indice_cor += 1
            if indice_cor >= len(lista_cores):
                break  # Certificando-se de não exceder o número de cores disponíveis

    # Criando uma lista de cores correspondente ao array original
    cores_array = [numero_para_cor[num] for num in array]

    # Armazenando os números e suas cores correspondentes em uma lista de tuplas
    numeros_cores = [(numero, cor) for numero, cor in numero_para_cor.items()]

    return cores_array, numero_para_cor, numeros_cores

def filtro(dataleitura,coluna_filtro,coluna_resposta,valores_especificos):
    # Usar np.isin para criar uma máscara booleana
    filtro = np.isin(dataleitura[:, coluna_filtro], valores_especificos)

    # Selecionar as colunas 0 e 1 onde o filtro é verdadeiro
    resultado = dataleitura[filtro, :][:, coluna_resposta]
    return resultado

fontsize_value = 40
# Aumentar ainda mais os tamanhos das fontes
plt.rc('font', size=24)  # Tamanho padrão das fontes
plt.rc('axes', titlesize=35)  # Tamanho do título dos eixos
plt.rc('axes', labelsize=24)  # Tamanho dos rótulos dos eixos
plt.rc('xtick', labelsize=35)  # Tamanho das labels do eixo x
plt.rc('ytick', labelsize=35)  # Tamanho das labels do eixo y
plt.rc('legend', fontsize=35)  # Tamanho da legenda
plt.rc('figure', titlesize=28)  # Tamanho do título da figura
labels = ['a)', 'b)','c)', 'd)', 'e)']

 ##################   ORDENADOR DE VETOR      ###################
def indices_vetor_ordenado(vetor):
    # Retorna os índices do vetor ordenado sem alterar a ordem original
    indices_ordenados = sorted(range(len(vetor)), key=lambda k: vetor[k])
```

```
110     return indices_ordenados
111 
112  ##################   FAZ O FIT   ###################
113 
114 def ajuste_linregress(x, log_dados):
115     slope, intercept, r_value, _, _ = linregress(x, log_dados)
116     return slope, intercept, r_value**2
117 
118 def le_fazFit(filename,corte_fit,com_legenda):
119     def fit(dataleitura,rho,canal,corte_fit):
120 
121         # Lista de cores que desejamos usar
122 
123         # Lista de cores que desejamos usar
124         lista_cores = ['red', 'blue', 'green', 'yellow', 'purple',
        ↪  'orange', 'pink', 'brown', 'gray', 'cyan']
125         labels = ['a)', 'b)','c)', 'd)', 'e)']
126 
127         coluna_filtro = 3 #Qual coluna está os dados do canal? 3
128         coluna_resposta = [0, 1, 2] #g,R,rho
129 
130         matriza =
        ↪  filtro(dataleitura,coluna_filtro,coluna_resposta,canal)
131 
132         coluna_filtro = 2
133         coluna_resposta = [0, 1]
134 
135         resultado = filtro(matriza,coluna_filtro,coluna_resposta,rho)
136 
137         # Salva variaveis do modo
138         ##Cortando g menores que 10**-13
139         if canal[0]==1.0:
140             mascara2 = resultado[:,1] <= matriz[0][-1]  # Seleciona a
            ↪  coluna 1 e cria a máscara
141             resultado = resultado[mascara2]
142         ##Cortando g menores que 10**-13 PARA O GRAFICO
143         mascara = resultado[:,0] >= (10**-13)  # Seleciona a coluna 1 e
        ↪  cria a máscara
144 
145         resultado_com_mascara = resultado[mascara]
146 
```

```
147     #         if resultado_com_mascara == 0:
148     #             print('Todos os dados são menores que 10**-13')
149
150         y_cond_g = resultado_com_mascara[:,0]
151         x_tamanhoR = resultado_com_mascara[:,1]
152
153         # Pega os indices da menor posição para a maior
154         indices_ordenado = indices_vetor_ordenado(x_tamanhoR)
155         # Ordena modo pela menor possição para a maior
156         y_cond_g_ordenado = y_cond_g[indices_ordenado]
157         x_tamanhoR_ordenado  = x_tamanhoR[indices_ordenado]
158
159
160         ##Cortando g menores que 10**-13 PARA O FIT
161         if np.any(y_cond_g_ordenado < (10**-corte_fit)):
162             indice = np.argmax(y_cond_g_ordenado < (10**-corte_fit))
163             print('ENTROU')
164         else:
165             indice = len(y_cond_g_ordenado)
166         if indice == 0:
167             print('Todos os dados são menores que 10**-1')
168             indice = len(y_cond_g_ordenado)
169
170         # Filtrando as linhas com base na máscara
171         y_cond_g_ordenado_filtrada = y_cond_g_ordenado[:indice]
172         x_tamanhoR_ordenado_filtrada = x_tamanhoR_ordenado[:indice]
173
174
175         log_dados = np.log10(y_cond_g_ordenado_filtrada)
176
177         slope, intercept, r2 =
            ↪  ajuste_linregress(x_tamanhoR_ordenado_filtrada, log_dados)
178
179         print(f'slope do {rho}: {slope}')
180         print(f'intercept do {rho}: {intercept}')
181
182         return x_tamanhoR_ordenado,
            ↪  y_cond_g_ordenado,x_tamanhoR_ordenado_filtrada,
            ↪  y_cond_g_ordenado_filtrada,
            ↪  slope,intercept,x_tamanhoR_ordenado[-1]
183         # Obtém a data e hora atuais
```

```
184
185     ##############################################
186     def graficando(matrizteste, canais, rhos, com_legenda,output_path):
187         from matplotlib.lines import Line2D
188
189         # Criação da figura e dos eixos
190         fig, ax = plt.subplots(figsize=(13, 10))
191         mapeamento_cores = {
192             0.01: 'yellow',
193             0.02: 'brown',
194             0.03: 'purple',
195             0.05: 'pink',
196             0.1: 'tab:red',
197             1.0: 'tab:green',
198             0.3: 'tab:blue',
199             1.1: 'tab:cyan',
200             2.0: 'tab:purple',
201             1.3: 'tab:orange',
202             1.4: 'magenta',
203             1.5: 'gold',
204             1.6: 'silver',
205             1.7: 'beige',
206             1.8: 'turquoise',
207             1.9: 'lilac',
208             1.2: 'salmon'
209         }
210         legenda1 = r'vector $\rho/ k^2$'
211         legenda2 = r'scalar $\rho / k^2$'
212
213         # Lista para capturar handles automáticos
214         line_handles = []
215
216         for rho in rhos:
217             print(f'rho: {rho}')
218             somadosrho = 0
219             if rho == 0.3:
220                 somadosrho = 2
221             for canal in canais:
222                 canal = int(canal)
223                 modos_canais = canal + somadosrho
224
```

```
                # Gráfico de dispersão
                ax.scatter(
                    matrizteste[modos_canais][2],
                    matrizteste[modos_canais][3],
                    marker=mapeamento_simbolos[canal],
                    c='black',
                    s=200
                )
                ax.scatter(
                    matrizteste[modos_canais][2],
                    matrizteste[modos_canais][3],
                    c=mapeamento_cores[rho],
                    marker=mapeamento_simbolos[canal],
                    s=100
                )

                corlinha = float(1 + rho)

                # Linha ajustada
                line, = ax.plot(
                    matrizteste[modos_canais][2],
                    10 ** (matrizteste[modos_canais][5] +
                    ↪  matrizteste[modos_canais][4] *
                    ↪  matrizteste[modos_canais][2]),
                    color=mapeamento_cores[corlinha],
                    linestyle=mapeamento_tracejado[canal],
                    linewidth=4
                )
                # Adicionando handles automáticos
                line_handles.append(line)

        # Configurando o eixo y para escala logarítmica
        ax.set_yscale('log')

        # Configuração de rótulos
        ax.set_xlabel(r'$kR$', fontsize=fontsize_value)
        ax.set_ylabel(r'g', fontsize=fontsize_value)

        # Configuração dos ticks do eixo x e y
        ax.xaxis.set_major_locator(ticker.MaxNLocator(nbins=6,
        ↪  integer=True))
```

```
263         ax.yaxis.set_major_locator(ticker.LogLocator(base=10.0,
            ↪  numticks=6))
264
265         # Grid no gráfico
266         ax.grid(True)
267
268         if com_legenda:
269             if len(rhos)==2:
270             # Handles personalizados
271                 custom_handles = [
272                     Line2D([0], [0], marker='o', color='w',
                        ↪  markerfacecolor=mapeamento_cores[rhos[0]],
                        ↪  markersize=20, label=f'{legenda1}: 0.10'),
273                     Line2D([0], [0], marker='v', color='w',
                        ↪  markerfacecolor=mapeamento_cores[rhos[0]],
                        ↪  markersize=20, label=f'{legenda2}: 0.10'),
274                     Line2D([0], [0], marker='v', color='w',
                        ↪  markerfacecolor=mapeamento_cores[rhos[1]],
                        ↪  markersize=20, label=f'{legenda2}: 0.30'),
275                     Line2D([0], [0], marker='o', color='w',
                        ↪  markerfacecolor=mapeamento_cores[rhos[1]],
                        ↪  markersize=20, label=f'{legenda1}: 0.30'),
276                 ]
277             if len(rhos)==1:
278                 custom_handles = [
279                     Line2D([0], [0], marker='o', color='w',
                        ↪  markerfacecolor=mapeamento_cores[rhos[0]],
                        ↪  markersize=20, label=f'{legenda1}:
                        ↪  {rhos[0]}'),
280                     Line2D([0], [0], marker='v', color='w',
                        ↪  markerfacecolor=mapeamento_cores[rhos[0]],
                        ↪  markersize=20, label=f'{legenda2}:
                        ↪  {rhos[0]}'),
281                 ]
282             # Combinar handles automáticos e personalizados
283             #plt.legend(handles=custom_handles +line_handles,
                ↪  loc='best', fontsize=25)
284             plt.legend(handles=custom_handles, loc='best', fontsize=25)
285
286         plt.savefig(output_path, bbox_inches='tight')
287         print(output_path)
```

```
        plt.show()
        # Salvar a figura como PDF

    ####################
    agora = datetime.datetime.now()

    # Formata a hora atual como string
    hora_atual = agora.strftime("%H:%M:%S")

    print("A hora atual é:", hora_atual)
    #---LEITURA DOS DADOS--#


    # Escolha o nome do arquivo que será lida
    directory = r'C:\Users\xandy\OneDrive\Área de Trabalho\Dados'

    tamanhodcabeca = 19

    #---Fazendo a Figura--#

    texto_especifico = 'Figura'
    output_dir = r'C:\Users\xandy\OneDrive\Área de Trabalho\Imagens
    ↪  dissertação\g'

    #---Faz sozinho--#
    #---Lendo--#
    file_path = os.path.join(directory, filename)
    dataleitura,header = lendoAqruivo(file_path,tamanhodcabeca)
    dataleitura =  np.array(dataleitura)

    #---Salvar Figura--#


    hora_atual1 = header[3]
    Titulodafig = header[14]
    output_filename =
    ↪  f'{texto_especifico}_{hora_atual1}_{Titulodafig}.pdf'
    output_path = os.path.join(output_dir, output_filename)

    #Extraindo colunas para o gráfico
    gm = dataleitura[:, 0]   # gm
```

```
    kr = dataleitura[:, 1]   # R
    rho  = dataleitura[:, 2]   # rho
    vec = dataleitura[:, 3]  # vetorial ou não
    #time=dataleitura[:, 4]  # tempo de simulação por ponto
    #tm1=dataleitura[:, 5]   # tempo de simulação por região
    #tm2=dataleitura[:, 6]   # tempo de simulação por região :tempo de diagnonalizar
    #tm3=dataleitura[:, 7]   # tempo de simulação por região
    na = dataleitura[:, 8]   # Num Atomos
    # corte=dataleitura[:, 9]  # Corte
    # xi = dataleitura[:, 10]   # Comprimento
    # r2 = dataleitura[:, 10]   # R2 do Fit

    # Plotar
    eixo_x = kr
    eixo_y = gm
    eixo_z = rho
    eixo_t = vec

    valores_unicos = np.round(rho, 2) ################## NEW >>>>>>>>>>>>>>>
    valores_especificos  = np.unique(valores_unicos)################## NEW >>>>>>>>>>>>>>>
    valores_vec = np.unique(vec)


    mapeamento_simbolos = {
        1: 'v',  # Triângulo
        0: 'o'   # Bolinha
    }
    mapeamento_tracejado = {
        1: '--',  # Triângulo
        0: ':'   # Bolinha
    }
    # Mapeamento de cores para valores de z
    cores_array, mapeamento_cores, numeros_cores = funcaoCor(valores_especificos, lista_cores)


    # Criando uma lista para armazenar os resultados
    resultados = []
```

```
    matrizt = []
    densidaddes = []
    a=0
    plt.figure(figsize=(10, 6))
    for i in valores_especificos:
        for vecs in valores_vec:
            print(f'{i} e {vecs}')
            matrizt.append(fit(dataleitura, [i], [vecs],corte_fit))


    graficando(matrizt,valores_vec,valores_especificos,com_legenda,output_path)
    print(type(matrizt))


    # Pegando apenas a segunda matriz de cada linha
    Slopes = [linha[4] for linha in matrizt]
    intercepts = [linha[5] for linha in matrizt]
    print(f'Slope:{Slopes}')
    print(f'intercept :{intercepts}')
    return Slopes,intercepts


le_fazFit("resultados_1601_img_2k_Nr_10_rho_1_mean.txt",11.5,True)

```

Fez o ultimo fit:

```
import matplotlib.pyplot as plt
import numpy as np

cores = {
    0: 'tab:red',
    5: 'tab:green',
    1: 'tab:blue',
    3: 'tab:cyan',
    4: 'tab:purple',
    6: 'tab:orange',
}
mapeamento_simbolos = {
0: 'o',   # Bolinha
1: 'v',  # Triângulo
```

```
2: 'o', # Triângulo
3: 'v',   # Bolinha
}

mapeamento_legenda = {
0: r'Fit scalar $\rho / k^2$ = 0.1 ',   # Bolinha
1: r'Fit vector $\rho / k^2$ = 0.1 ',  # Triângulo
2: r'Fit scalar $\rho / k^2$ = 0.3 ', # Triângulo
3: r'Fit vector $\rho / k^2$ = 0.3 ',   # Bolinha
}
# Criando os dados para o gráfico
x = np.linspace(5, 1000, 50)  # Valores de x
x1 = np.linspace(5, 300, 50)  # Valores de x
plt.figure(figsize=(13, 10))
for i in range(4):
    # Criando o gráfico
    print(i)
    y = a[i] * x + b[i]  # Cálculo de y = ax + b
    if i!=2:
        plt.plot(
            x, y, linestyle='--',marker=mapeamento_simbolos[i],
            markersize=20,c=cores[int(i/2)],markevery=[10, 20, 30,
            40],
            linewidth=4, markerfacecolor=cores[int(i/2)],
            markeredgecolor='black',label=mapeamento_legenda[i]
        )
    if i==2:
        plt.plot(
            x, y, linestyle='--',marker=mapeamento_simbolos[i],
            markersize=20,c=cores[int(i/2)],markevery=[1, 2, 3, 4],
            linewidth=4, markerfacecolor=cores[int(i/2)],
            markeredgecolor='black',label=mapeamento_legenda[i]
        )

# Adicionando título e rótulos

# Customizando os ticks do eixo y em formato LaTeX
yticks = np.arange(-10, 11, 5)  # Valores no eixo y de -10 a 10 com
espaçamento de 5
ytick_labels = [f"$10^{{{tick}}}$" if tick != 0 else "$10^0$" for tick
in yticks]  # LaTeX para 10^n
```

```
49
50 plt.yticks(yticks, ytick_labels)  # Aplica os novos labels no eixo y
51 plt.xlabel("kR", fontsize=fontsize_value)
52 plt.ylabel(r"g$_{im}$", fontsize=fontsize_value)
53 plt.ylim(-12,1)  # Definindo o intervalo do eixo y (10^-10 a 10^-1)
54 plt.axhline(0, color='black', linewidth=0.8, linestyle="--")  # Linha
   ↪  no eixo x
55 plt.axvline(0, color='black', linewidth=0.8, linestyle="--")  # Linha
   ↪  no eixo y
56 plt.legend()
57 plt.grid(True)
58
59 output_filename = f'Fit dos G 17012025.pdf'
60 output_dir = r'C:\Users\xandy\OneDrive\Área de Trabalho\Imagens
   ↪  dissertação\g'
61 output_path = os.path.join(output_dir, output_filename)
62 plt.savefig(output_path, bbox_inches='tight')
63 print(output_path)
64
65 # Mostrando o gráfico
66 plt.show()
67
```

Calcula o numero do TASK ID:

```
1  # Parâmetros
2  N_inicial = 1000
3  N_final = 20000
4  passo = 200
5  n_pontos = int(N_final/passo)
6  print(f'n_pontos: {n_pontos}')
7  print(f'Passo: {passo}')
8  rhos = [0.1,0.3]  # Valores de rho
9  vetorials = [0, 1]  # Valores de vetorial
10 Nr_max = 5
11 fator_nr = 0.75
12 passo_nr = int(fator_nr*(N_final/Nr_max))
13 print(f'passo_nr: {passo_nr}')
14
15 # Criação do dicionário
16 task_data = {}
```

```
task_counter = 1

# Gerar as combinações
for rho in rhos:  # Itera sobre valores de rho
    for v in vetorials:  # Itera sobre valores de vetorial
        for N in range(N_inicial, N_final + 1, passo):  # Itera sobre
        ↪  valores de N
            # Determina Nr com base no valor de N
            aux = int((N - N_inicial)/(passo_nr))
            #Nr_values = list(range((Nr_max), 0, -1)) #Quantidade de
            ↪  repetições iguais para cada N
            Nr_values = list(range((Nr_max-aux), 0, -1))
            if len(Nr_values) == 0:
                # Preenche com 1 se for menor
                Nr_values.extend([1])
            for nr in Nr_values:  # Itera sobre valores de Nr ajustados
                task_data[task_counter] = {'vetorial': v, 'rho': rho,
                ↪  'N': N, 'Nr': nr}
                task_counter += 1

# Exibe o resultado
Quan_chaves = len(task_data)
print(f'Quan_chaves: {Quan_chaves}')

```

Compilador de dados

```
import os
import time
import numpy as np
from collections import Counter
from datetime import datetime
import pandas as pd

def listar_informacoes_dos_arquivos(pasta):
    """Lista e processa dados de arquivos na pasta especificada e
    ↪  captura o cabeçalho."""
    if not os.path.exists(pasta):
        print(f"A pasta '{pasta}' não existe.")
        return [], []

```

```
    matriz_dados = []
    cabecalhos = []

    with os.scandir(pasta) as it:
        for entry in it:
            if entry.is_file() and entry.name.endswith('.txt'):
                caminho_arquivo = entry.path
                try:
                    with open(caminho_arquivo, 'r', encoding='utf-8')
                    ↪  as file:
                        linhas = file.readlines()
                        cabecalho = linhas[:19]
                        dados = [
                            float(linha.strip())
                            for linha in linhas[19:]
                            if linha.strip()
                        ]
                        cabecalhos.append(cabecalho)
                        matriz_dados.append(dados)
                except Exception as e:
                    print(f"Erro ao processar o arquivo {entry.name}:
                    ↪  {e}")

    print("Processamento completo. Dados e cabeçalhos extraídos.")
    return matriz_dados, cabecalhos



def processar_cabecalhos(cabecalhos_individuais):
    resultados = {}

    # Função auxiliar para processar as linhas
    def processar_linha(cabecalhos, indice, prefixo):
        valores4 = []
        for cabecalho in cabecalhos:
            if len(cabecalho) > indice:  # Verifica se o índice existe
            ↪  no cabeçalho
                linha = cabecalho[indice]
                depaco = linha.replace(prefixo, "").replace("-",
                ↪  "").replace("horas", "").split()
                valores4.append(depaco)
```

```
51          return valores4
52
53      for i in range(12):  # Itera até o índice máximo necessário
54          if i == 1:  # "Comecou em"
55              valores4 = processar_linha(cabecalhos_individuais, i,
                ↪  "Comecou   em:")
56              valores_achatados = [''.join(item) for item in valores4]
57              data_str = str(min(valores_achatados))
58              data_hora = datetime.strptime(data_str, "%Y%m%d%H%M")
59              resultados['time'] =
                ↪  f"{data_hora.strftime('%Y%m%d-%H%M')}"
60              resultados['inicio'] = f"{data_hora.strftime('%Y %m %d - %H
                ↪  %M')}"
61
62          elif i == 2:  # "Finalizou em"
63              valores4 = processar_linha(cabecalhos_individuais, i,
                ↪  "Finalizou em:")
64              valores_achatados = [''.join(item) for item in valores4]
65              data_str = str(max(valores_achatados))
66              data_hora = datetime.strptime(data_str, "%Y%m%d%H%M")
67              resultados['fim'] = f"{data_hora.strftime('%Y %m %d - %H
                ↪  %M')}"
68
69          elif i == 4:  # "Densidades"
70              valores4 = processar_linha(cabecalhos_individuais, i,
                ↪  "Densidades:")
71              valores_achatados = [item for sublist in valores4 for item
                ↪  in sublist]
72              contador = Counter(valores_achatados)
73              resultados['densidade_mais_frequente'] = max(contador)
74              resultados['densidade_menos_frequente'] = min(contador)
75
76          elif i == 5:  # "Vetorial"
77              valores4 = processar_linha(cabecalhos_individuais, i,
                ↪  "Vetorial:")
78              valores_achatados = [item for sublist in valores4 for item
                ↪  in sublist]
79              contador = Counter(valores_achatados)
80              resultados['vetorial_mais_frequente'] = max(contador)
81              resultados['vetorial_menos_frequente'] = min(contador)
82
```

```
83          elif i == 6:  # "Realizações"
84              valores4 = processar_linha(cabecalhos_individuais, i,
                ↪  "Realizações:")
85              valores_achatados = [item for sublist in valores4 for item
                ↪  in sublist]
86              contador = Counter(valores_achatados)
87              resultados['realizacoes_mais_frequente'] = max(contador,
                ↪  key=contador.get)
88              resultados['realizacoes_menos_frequente'] = min(contador,
                ↪  key=contador.get)
89
90          elif i == 10:  # "NAtomos"
91              valores4 = processar_linha(cabecalhos_individuais, i,
                ↪  "NAtomos:")
92              valores_achatados = [item for sublist in valores4 for item
                ↪  in sublist]
93              contador = Counter(valores_achatados)
94              resultados['natomos_mais_frequente'] = max(contador,
                ↪  key=contador.get)
95              resultados['natomos_menos_frequente'] = min(contador,
                ↪  key=contador.get)
96              print(resultados['natomos_mais_frequente'] )
97
98          elif i == 11:  # "Tempo de realizacao"
99              valores4 = processar_linha(cabecalhos_individuais, i,
                ↪  "Tempo de realizacao:")
100             valores_achatados = [item for sublist in valores4 for item
                ↪  in sublist]
101             contador = Counter(valores_achatados)
102             resultados['tempo_mais_frequente'] = max(contador)
103
104     return resultados
105
106 def salvar_matriz_com_cabecalho(matriz, cabecalho_geral,
    ↪  cabecalhos_individuais, diretorio, nome_arquivo):
107     """Salva a matriz com cabeçalho geral e específicos por arquivo."""
108     os.makedirs(diretorio, exist_ok=True)  # Cria o diretório se não
        ↪  existir
109     file_path = os.path.join(diretorio, f'{nome_arquivo}.txt')
110
111     with open(file_path, 'w', encoding='utf-8') as f:
```

```
112         # Escreve o cabeçalho geral
113         f.write(cabecalho_geral)
114         np.savetxt(f, matriz, delimiter=', ', fmt='%.17e')
115 
116     print(f"Arquivo '{nome_arquivo}.txt' criado com sucesso em
    ↪  {file_path}!")
117 
118 # Função para agrupar e calcular a média para valores com Col9 iguais,
    ↪  preservando a ordem das colunas
119 def agrupar_e_calcular_media(grupo):
120     if not grupo.empty:
121         # Realiza o agrupamento e calcula a média
122         grupo_media = grupo.groupby("Col9", as_index=False).mean()
123         # Garante que a ordem das colunas seja preservada
124         return grupo_media.reindex(columns=grupo.columns)
125     return grupo
126 
127 # Função para converter DataFrame em matriz numpy com colunas ordenadas
128 def salvar_como_matriz(grupo, colunas_ordem):
129     if not grupo.empty:
130         # Reorganizando as colunas para garantir a ordem
131         grupo = grupo[colunas_ordem]
132         return grupo.to_numpy()
133     return np.array([])  # Retorna uma matriz vazia caso o grupo esteja
    ↪  vazio
134 
135 
136 def salvando_txt_com_mean(pasta,nome,directory,spread):
137 
138     # Processa e salva os dados
139     matriz_dados, cabecalhos_individuais =
    ↪  listar_informacoes_dos_arquivos(pasta)
140     resultados = processar_cabecalhos(cabecalhos_individuais)
141     if spread == 'on':
142         media = 'sem_media'
143     elif spread == 'off':
144         media = 'media'
145 
146     # Cabeçalho a ser adicionado
147     cabecalho_geral = f"""Simulação de Cluster
148 Comecou   em: {resultados['inicio']}
```

```
Finalizou em: {resultados['fim']}
{resultados['time']}
Densidades:[{resultados['densidade_menos_frequente']},{resultados['densidade_mais_frequente']}]
Vetorial:
 :[{resultados['vetorial_menos_frequente']},{resultados['vetorial_mais_frequente']}]
Realizações:
 [{resultados['realizacoes_mais_frequente']},{resultados['realizacoes_menos_frequente']}]
Num de Pontos: 0
Corte: {cabecalhos_individuais[0][8]}Raio: ... Faz as contas ai
NAtomos:
 [{resultados['natomos_mais_frequente']},{resultados['natomos_menos_frequente']}]
Tempo de realizacao: {resultados['tempo_mais_frequente']}
Numero de Threads:
Titulo da Figura:
{media}
Descrição: Esta simulacao ...Os dados estão no formato:
(gm,R,rho,vetorial,elapsed_time,tm1,tm2,tm3,NAtoms,cortes,compa,r2)
Autores: Alexandre J. Fuzita
Notas adicionais: PC:Cluster; Linguagem: Python; e nome do codigo que
 gerou os dados:0912_cODIGO_PES e texto80G e fast .
    """

    if spread == 'on':
        if matriz_dados:
            matriz = np.array(matriz_dados)
            salvar_matriz_com_cabecalho(matriz, cabecalho_geral,
             cabecalhos_individuais, directory, nome)
    elif spread == 'off':
        # Convertendo para DataFrame
        df = pd.DataFrame(matriz_dados, columns=[f"Col{i+1}" for i in
         range(12)])

        # Passo 1: Organizar pela coluna 4 (0 ou 1)
        df = df.sort_values(by="Col4")

        # Passo 2: Dividir em subgrupos por valores distintos na Col4 e
         Col2 (0.1 e 0.3)
        grupo_1 = df[(df["Col4"] == 0) & (df["Col3"] == 0.1)]
        grupo_2 = df[(df["Col4"] == 0) & (df["Col3"] == 0.3)]
        grupo_3 = df[(df["Col4"] == 1) & (df["Col3"] == 0.1)]
        grupo_4 = df[(df["Col4"] == 1) & (df["Col3"] == 0.3)]
```

```
        grupo_5 = df[(df["Col4"] == 1) & (df["Col3"] == 1)]
        grupo_6 = df[(df["Col4"] == 0) & (df["Col3"] == 1)]

        # Passo 3: Organizar cada subgrupo pela coluna 9
        grupo_1 = grupo_1.sort_values(by="Col9")
        grupo_2 = grupo_2.sort_values(by="Col9")
        grupo_3 = grupo_3.sort_values(by="Col9")
        grupo_4 = grupo_4.sort_values(by="Col9")
        grupo_5 = grupo_5.sort_values(by="Col9")
        grupo_6 = grupo_6.sort_values(by="Col9")

        # Aplicando a função aos grupos
        grupo_1_media = agrupar_e_calcular_media(grupo_1)
        grupo_2_media = agrupar_e_calcular_media(grupo_2)
        grupo_3_media = agrupar_e_calcular_media(grupo_3)
        grupo_4_media = agrupar_e_calcular_media(grupo_4)
        grupo_5_media = agrupar_e_calcular_media(grupo_5)
        grupo_6_media = agrupar_e_calcular_media(grupo_6)

        # Garantindo que todas as colunas estão na mesma ordem
        colunas_ordem = grupo_1_media.columns  # Pegamos a ordem de
        ↪  colunas do primeiro grupo como referência

        # Convertendo os grupos para matrizes numpy
        matriz_grupo_1 = salvar_como_matriz(grupo_1_media,
        ↪  colunas_ordem)
        matriz_grupo_2 = salvar_como_matriz(grupo_2_media,
        ↪  colunas_ordem)
        matriz_grupo_3 = salvar_como_matriz(grupo_3_media,
        ↪  colunas_ordem)
        matriz_grupo_4 = salvar_como_matriz(grupo_4_media,
        ↪  colunas_ordem)
        matriz_grupo_5 = salvar_como_matriz(grupo_5_media,
        ↪  colunas_ordem)
        matriz_grupo_6 = salvar_como_matriz(grupo_6_media,
        ↪  colunas_ordem)

        # Filtrando grupos não vazios
        matrizes = [matriz for matriz in [matriz_grupo_1,
        ↪  matriz_grupo_2, matriz_grupo_3,
        ↪  matriz_grupo_4,matriz_grupo_5, matriz_grupo_6] if
        ↪  matriz.size > 0]
```

```
        print(matrizes)
        # Juntando as matrizes em uma única
        matriz_completa = np.vstack(matrizes)

        # Verificando a matriz resultante
        nome = nome + '_mean'
        if matriz_dados:
            matriz = np.array(matriz_completa)
            salvar_matriz_com_cabecalho(matriz, cabecalho_geral,
            ↪  cabecalhos_individuais, directory, nome)
    return

# Configurações iniciais
nome_salvar = 'resultados_1701_img_20k_Nr_5_rhos_nf'
pasta =
↪  r"\\wsl$\Ubuntu-20.04\root\resultados_1701_img_20k_Nr_5_rhos_nf"
directory = r'C:\Users\xandy\OneDrive\Área de Trabalho\Dados'
spread = 'off'
salvando_txt_com_mean(pasta,nome_salvar,directory,spread)

```

Kmeans:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random
import time

from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from kneed import KneeLocator
from sympy import *
from scipy.special import hankel1
from matplotlib.patches import Circle

# Number of atoms.
NAtoms = 2000

# System density.
rho = 1
```

```
vetorial = 0

# Maximum radius for the cloud.
Rx = np.sqrt((NAtoms / rho) / (np.pi))
Ry = Rx

# Create random positions for each atom, between -r to r.
x = 2 * Rx * (np.random.rand(NAtoms) - 0.5)
y = 2 * Ry * (np.random.rand(NAtoms) - 0.5)


# Verify which atoms are within the maximum radius, and create more
↪  atoms until all are within.

for j in range(NAtoms):
    while np.sqrt(x[j]**2 / Rx**2 + y[j]**2 / Ry**2) > 1:
        x[j] = 2 * Rx * (np.random.rand() - 0.5)
        y[j] = 2 * Ry * (np.random.rand() - 0.5)

# Distance between each pair of atoms.
xij = np.outer(x, np.ones(NAtoms)) - np.outer(np.ones(NAtoms), x) +
↪  np.eye(NAtoms)
yij = np.outer(y, np.ones(NAtoms)) - np.outer(np.ones(NAtoms), y) +
↪  np.eye(NAtoms)

# Distance between each pair of atoms in r.
dij = np.sqrt(xij**2 + yij**2)
phiij = np.arctan2(yij,xij)

# Auxiliary matrix.
aux = np.ones((NAtoms, NAtoms)) - np.eye(NAtoms)

# Scalar kernel matrix.
hankel0 = hankel1(0, dij)
gamma00 = np.eye(NAtoms) + aux * hankel0


if vetorial == 1:
    # Vector kernel matrix

    hankel2 = aux*(hankel1(2,dij))
```

```
    # gammaMM = gamma00
    gammaMP = -np.exp(2*1j*phiij)*hankel2

    gammaPM = -np.exp(-2*1j*phiij)*hankel2
    # gammaPP = gamma00
    Gamma00 = np.block([[gamma00, gammaMP], [gammaPM, gamma00]])
else:
    Gamma00 = gamma00

# Diagonalization.
e_vec_0, v_vec0 = np.linalg.eig(Gamma00)

# Naming gamma and omega.
Gamma = np.real(e_vec_0)
Omega = np.imag(e_vec_0)

# Filtering modes less than 1e-14.

ind_14 = []
gamma_14 = []
omega_14 = []
autovec_14 = []
corte = 1e-14

for indice, numero in enumerate(Gamma):
    if numero > corte:
        ind_14.append(indice)
        gamma_14.append(numero)
        if indice < len(Omega):
            omega_14.append(Omega[indice])
        if indice < len(v_vec0):
            autovec_14.append(v_vec0[:,indice])

ind_14_np = np.array(ind_14)
Gamma_14_np = np.array(gamma_14)
omega_14_np = np.array(omega_14)
autovec_14_np = np.array(autovec_14)

# Defining function to calculate IPR.
def ipr(autovetor):
    ipr = sum(np.power(abs(autovetor), 4))/
    ↪  (sum(np.power(abs(autovetor), 2))**2)
```

```
98
99      return ipr
100
101 # Calculate IPR for all modes.
102 ipr_n = np.empty(len(autovec_14_np))
103 for i in range(len(autovec_14_np)):
104     ipr_n[i] = ipr(autovec_14_np[i, :])
105
106 # Calculate Psi**2 for all modes.
107 prob = (abs(autovec_14_np))**2
108
109 # Calculate center of mass (x, y, r) for all modes.
110 xcm = np.dot(x,
    ↪  np.transpose(abs(autovec_14_np)))/np.sum(abs(autovec_14_np),
    ↪  axis=1)
111 ycm = np.dot(y,
    ↪  np.transpose(abs(autovec_14_np)))/np.sum(abs(autovec_14_np),
    ↪  axis=1)
112 rcm = np.sqrt(xcm**2 + ycm**2)
113
114 # Calculate |r - rcm|, vector distance between the center of mass and
    ↪  the atoms, for all.
115 rd = []
116 for i in range(len(autovec_14_np)):
117     rd.append(np.sqrt((x-xcm[i])**2 + (y-ycm[i])**2 ))
118
119 i_ipr_max=np.argsort(Gamma_14_np)[0]
120
121 # Criando uma figura e subplots
122 fig, axs = plt.subplots(2, 2, figsize=(12, 8))
123
124 # Plotando os gráficos
125
126 scatter = axs[1, 0].scatter(x, y, c=np.log10(prob[i_ipr_max]),
    ↪  cmap='jet')
127 axs[1, 0].scatter(xcm[i_ipr_max], ycm[i_ipr_max], c='black',
    ↪  marker='+', linewidth=20, label='Centro de Massa')
128 axs[1, 0].add_patch(Circle((0, 0), Rx, color='black', fill=False))  #
    ↪  Adicionando o círculo
129 axs[1, 0].axis('equal')
130
```

```
131
132 axs[1, 0].set_xlabel('Eixo X')
133 axs[1, 0].set_ylabel('Eixo Y')
134 axs[1, 0].set_title(f'Visualização do modo: {i_ipr_max}')
135 axs[1, 0].grid(True)
136
137 scatter = axs[1, 0].scatter(x, y, c=np.log10(prob[i_ipr_max]),
    ↪  cmap='jet')
138 axs[1, 0].scatter(xcm[i_ipr_max], ycm[i_ipr_max], c='black',
    ↪  marker='+', linewidth=20, label='Centro de Massa')
139 axs[1, 0].add_patch(Circle((0, 0), Rx, color='black', fill=False))  #
    ↪  Adicionando o círculo
140 axs[1, 0].axis('equal')
141
142
143 axs[1, 0].set_xlabel('Eixo X')
144 axs[1, 0].set_ylabel('Eixo Y')
145 axs[1, 0].set_title(f'Visualização do modo: {i_ipr_max}')
146 axs[1, 0].grid(True)
147
148 # Adicionando o colorbar ao gráfico 1
149 cbar = fig.colorbar(scatter, ax=axs[1,0])
150 cbar.set_label('Log10(|ψ^2 )')
151
152 # Ajustando o layout
153 plt.tight_layout()
154
155 # Adicionando a legenda ao gráfico 1
156 axs[0,0].legend()
157 axs[0,1].legend()
158
159 # Exibindo o gráfico
160 plt.show()
161
162 # Create a matrix with the information that will be used in Kmeans.
163 np_array = np.column_stack((omega_14, gamma_14,ipr_n, rcm))
164
165 # Use pandas and give a title to each column of the Data.
166 df = pd.DataFrame(np_array, columns=['omega', 'Gamma','IPR','Rcm'])
167
168 # Drop rows with NA values in any columns.
```

```
df = df.dropna()

# Create scaled DataFrame where each variable has mean of 0 and
↪  standard deviation of 1.
scaled_df = StandardScaler().fit_transform(df)

# Initialize kmeans parameters.
kmeans_kwargs = {
"init": "random",
"n_init": 10,
"random_state": 1,
}

# Create list to hold SSE values for each k.
sse = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
    kmeans.fit(scaled_df)
    sse.append(kmeans.inertia_)

#Find Knee.
kl = KneeLocator(range(1, 11), sse, curve="convex",
↪  direction="decreasing")
cotovelo = kl.elbow
print(cotovelo)

# instantiate the k-means class, using optimal number of clusters.
kmeans = KMeans(init="random", n_clusters=cotovelo, n_init=10,
↪  random_state=1)

# fit k-means algorithm to data.
kmeans.fit(scaled_df)

# append cluster assingments to original DataFrame.
df['cluster'] = kmeans.labels_

# Make a copy of the Data in matrix format.
np_array = df.values

# Define each column as variables.
k_omega = np_array[:,0]
```

```
k_gamma = np_array[:,1]
k_ipr = np_array[:,2]
k_cluster = np_array[:,-1]

# Draw random indices in each cluster.
def Sort_indices(M,N_graf):
    Coluna_cl = df.query(f'cluster == {M}')
    Np_Coluna_cl = Coluna_cl.values
    mod_rand=np.random.choice(Np_Coluna_cl[:,1], N_graf, replace=False)
    indices_lista = []
    for valor in mod_rand:
        indices = df.index[df['Gamma'] == valor].tolist()
        indices_lista.extend(indices)
    # Convert the list of indices into a NumPy array.
    indices_matriz = np.array(indices_lista)
    return indices_matriz

# Count the number of modes in each cluster.
def contar_numeros(vetor):
    # Create a dictionary to count the frequency of each number.
    frequencia = {}

    # Count the frequency of each number in the cluster.
    for num in vetor:
        if num in frequencia:
            frequencia[num] += 1
        else:
            frequencia[num] = 1

    # Convert the dictionary into a NumPy array.
    resultado = np.array([[freq, num] for num, freq in
    ↪  frequencia.items()])

     # Sort by Cluster.
    num_em_clust = resultado[resultado[:, 1].argsort()]

    return num_em_clust

# Count the number of modes in each cluster.
num_em_clust = contar_numeros(k_cluster)
```

```
# Sorting the data matrix by cluster.
sorted_matrix = np_array[np_array[:, -1].argsort()]

# Separating states by cluster...
matrizes_separadas = []

# Variable for the operation of 'for', starts with the first state and saves all.
temp_row = [sorted_matrix[0]]

for i in range(1, sorted_matrix.shape[0]): # Will run all states. 'sorted_matrix.shape[0] = len(sorted_matrix)'
    if sorted_matrix[i][-1] == temp_row[-1][-1]:
    # 'sorted_matrix[i][-1]': checks the cluster of line i, if it is equal to the cluster of the state.
        temp_row.append(sorted_matrix[i]) # Adicionar o estado na matrix 'temp_row'
    else:
        matrizes_separadas.append(np.array(temp_row))
        # If not equal, save states with the same cluster ('temp_row') in a matrix.
        temp_row = [sorted_matrix[i]] # Update the status you are seeing.
matrizes_separadas.append(np.array(temp_row))

# The previous loop in a matrix of several matrices, changing to a list.
lista_media_cluster = []
for i in range(0,cotovelo):
    # Calculate the mean for each column.
    lista_media_cluster.append(np.mean(matrizes_separadas[i], axis=0))

# Create a list to be the percentage column.
lista_media_cluster_p = np.column_stack((lista_media_cluster,num_em_clust[:,0]*100/len(autovec_14)))


# Create DataFrame with data from the list.
d_cluster = pd.DataFrame(lista_media_cluster_p, columns=['omega', 'Gamma', 'IPR','Rcm', 'Cluster', 'Porcentagem'])

```

```
# Remover a coluna 'Cluster' do DataFrame
d_cluster_sem_cluster = d_cluster.drop(columns=['Cluster'])
# Showing average values of each Cluster.
print(f"---------------- Tabela para N={NAtoms} e densidade={rho}
↪  ------------------\n")
print("------------------ Valores médios de cada Cluster
↪  ------------------\n")
print(d_cluster_sem_cluster.head())

# Making the energy graph by cluster.


# Defining colors and names for the Clusters.
colors = ['r', 'g', 'b', 'm', 'orange']
group_labels = ['Cluster 0', 'Cluster 1', 'Cluster 2', 'Cluster 3',
↪  'Cluster 4','Cluster 5']

# Create the scatter plot.
plt.figure(figsize=(8, 6))
for i in range(cotovelo):
    plt.scatter(k_omega[k_cluster == i], k_gamma[k_cluster == i],
    ↪  color=colors[i], label=group_labels[i], alpha=0.7)
# Add labels.


plt.legend()
plt.yscale('log')
plt.xlabel('')
plt.ylabel('')

# Display the graph.
plt.grid(True)

# Save the graph.
plt.savefig('meu_grafico_de_dispersao.png')

# Show the graph.
plt.show()

# Making the spatial profile of modes for |r-rcm|.

```

```
316 # Array with the number of clusters.
317 Cluster = np.arange(0, cotovelo, 1)
318
319 # Number of graphs.
320 N_graf=5
321
322 # Number of graphs per row.
323 N_graf_lin = 5
324
325 # Draw N random indices in each cluster.
326 indices = [Sort_indices(elemento,N_graf) for elemento in Cluster]
327
328 # Store the indices in a matrix.
329 matriz_indices = [element for row in indices for element in row]
330
331
332 # Creating a figure and axes for the subplots.
333 fig, axs = plt.subplots(int(cotovelo*(N_graf/N_graf_lin)),
    ↪  int(N_graf_lin), figsize=(20, 10))
334
335 # Counter for the current number of graphs.
336 a = 0
337
338 # Number of lines.
339 num_linha = int(cotovelo*(N_graf/N_graf_lin))
340
341 # Creating the graphs in each subplot.
342 for i in range(num_linha):
343     for j in range(N_graf_lin):
344         # Calculate the current subplot index.
345         index = i * 3 + j
346
347         # Plot the graph in the current subplot.
348         clusttt = int(a/N_graf)
349         axs[i, j].scatter(rd[matriz_indices[a]],
            ↪  prob[matriz_indices[a]], color=colors[clusttt])
350 #         axs[i, j].set_title(f'Cluster:{clusttt}-I:
    ↪  {matriz_indices[a]} ')
351 #         axs[i, j].set_title(f'Cluster: {clusttt}')
352         axs[i, j].set_yscale('log')
353
```

```
        axs[i, j].set_xlim(0, 1.5*Rx)  # Ajuste conforme necessário
        axs[i, j].set_ylim(10**-17, 1)  # Ajuste conforme necessário
        # Update the counter.
        a = a +1
      # Adjust layout to avoid overlap.
plt.tight_layout()
# Show the graphs.
plt.show()

import matplotlib.gridspec as gridspec

# Criando os colormaps para cada linha
colormaps = [ 'Reds', 'Greens','Blues', 'Purples', 'Oranges']
fig = plt.figure(figsize=(15, 10))
gs = gridspec.GridSpec(int(cotovelo*(N_graf/N_graf_lin)),
↪  int(N_graf_lin)+1, width_ratios=[1, 1, 1, 1, 1, 0.1])

a = 0

for i in range(num_linha):
    for j in range(N_graf_lin):
        ax = fig.add_subplot(gs[i, j])

        # Calcular o índice correto para os dados de exemplo
        idx = i * 3 + j

        clusttt = int(a/N_graf)

        # Plot the graph in the current subplot.
        sc = ax.scatter(x, y, c=np.log10(prob[matriz_indices[a]]),
        ↪  cmap=colormaps[clusttt], vmin=-20, vmax=1)

        # Show the center of mass.1
        ax.scatter(xcm[matriz_indices[a]], ycm[matriz_indices[a]],
        ↪  c='black', marker='+', linewidth=10, label='Centro de
        ↪  Massa')
        ax.set_xlim(-60, 60)  # Ajuste conforme necessário
        ax.set_ylim(-60, 60)  # Ajuste conforme necessário
        # Adding the cloud boundary circle.
        ax.add_patch(Circle((0, 0), Rx, color='black', fill=False))
        ax.axis('equal')
```

```
391
392         # Adicionar a colorbar à esquerda na primeira coluna
393         if j == 4:
394             cax = fig.add_subplot(gs[i, 5])
395             cbar = fig.colorbar(sc, cax=cax, orientation='vertical')
396             cbar.ax.yaxis.set_ticks_position('left')
397
398         a = a +1
399
400 # Ajustar espaçamento entre os subplots
401 plt.tight_layout()
402
403 # Mostrar o gráfico
404 plt.show()
```