06/02/21 09:08:39 /home/ana/Desktop/proj/code/Bender/Bending.py

```python
# PHS3350
# Week 7 - Energy levels of a family of non-Hermitian Hamiltonians
# "what I cannot create I do not understand" - R. Feynman.
# Ana Fabela Hinojosa, 18/04/2021

import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from scipy.integrate import quad
import scipy.special as sc
from scipy import linalg
from tqdm import tqdm
from odhobs import psi as cpsi_blank


plt.rcParams['figure.dpi'] = 200
np.set_printoptions(linewidth=200)


def complex_quad(func, a, b, **kwargs):
    # Integration using scipy.integratequad() for a complex function
    def real_func(*args):
        return np.real(func(*args))

    def imag_func(*args):
        return np.imag(func(*args))

    real_integral = quad(real_func, a, b, **kwargs)
    imag_integral = quad(imag_func, a, b, **kwargs)
    return real_integral[0] + 1j * imag_integral[0]

#################################### Matrix SOLVING
###########################################         )

def Hamiltonian(x, ϵ, n):
    x = np.array(x)
    x[x == 0] = 1e-200
    h = 1e-6
    psi_n = cpsi_blank(n, x)
    d2Ψdx2 = (cpsi_blank(n, x + h) - 2 * psi_n + cpsi_blank(n, x - h)) / h ** 2
    return -d2Ψdx2 + (x ** 2 * (1j * x) ** ϵ) * psi_n


def element_integrand(x, ϵ, m, n):
    # CHECK THESE IF mass = 1 instead of 1/2
    psi_m = cpsi_blank(m, x)
    return np.conj(psi_m) * Hamiltonian(x, ϵ, n)


# NxN MATRIX
def Matrix(x, N):
    M = np.zeros((N, N), dtype="complex")
    for m in tqdm(range(N)):
        for n in tqdm(range(N)):
```

```python
            b = np.abs(np.sqrt(4 * min(m, n) + 2)) + 2
            element = complex_quad(
                element_integrand, -b, b, args=(ϵ, m, n), epsabs=1.49e-08, limit=1000
            )
            # print(element)
            M[m][n] = element
    return M


####################################function calls#################################################
# GLOBALS
epsilons = np.linspace(-1, 0, 100)
k = 1 / 2
x = 2
N = 100

# NxN MATRIX
for i, ϵ in enumerate(epsilons):
    print(f"{ϵ = }")
    matrix = Matrix(x, N)
    np.save(f'matrices/matrix_{i:03d}.npy', matrix)

##############plots################
# m = 300
# n = 300
# b = np.abs(np.sqrt(4 * min(m, n) + 2)) + 2
# xs = np.linspace(-40, 40, 2048 * 10, endpoint=False)
# plt.plot(xs, cpsi_blank(300, xs), linewidth=1)
# plt.plot(xs, np.real(Hamiltonian(xs, ϵ, n)), label="Real part", linewidth=1)
# plt.plot(xs, np.imag(Hamiltonian(xs, ϵ, n)), label="Imaginary part", linewidth=1)
# plt.plot(xs, np.real(element_integrand(xs, ϵ, m, n)), label="Real part", linewidth=1)
# plt.plot(xs, np.imag(element_integrand(xs, ϵ, m, n)), label="Imaginary part", linewidth=1)
# plt.axvline(b, color='grey' , linestyle=":", label="Turning points")
# plt.axvline(-b, color='grey' , linestyle=":")
# plt.legend()
# plt.show()
# ##############plots################
```