06/05/21 02:32:08 /home/ana/Desktop/proj/code/Bender/wavefunction_animation.py

```python
# PHS3350
# Week 5 - Energy levels of a family of non-Hermitian Hamiltonians
# "what I cannot create I do not understand" - R. Feynman.
# Ana Fabela Hinojosa, 16/05/2021

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

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

####################### eigenvalues & eigenvectors ############################

Energies_2 = np.load("Energies_unbroken.npy")

def linear_algebra(epsilons):

    eigenvalues_list = []
    eigenvectors_list = []
    for i, ϵ in enumerate(epsilons):
        matrix = np.load(f'matrices/matrix_{i:03d}.npy')
        eigenvalues, eigenvectors = linalg.eig(matrix)
        eigenvalues_list.append(eigenvalues)
        eigenvectors_list.append(eigenvectors)

        ϵ_list = np.full(len(eigenvalues), ϵ)

    return np.array(eigenvalues_list), np.array(eigenvectors_list)

####################### sorting Eigenvectors ##################################

def filtering_and_sorting(evals, evects):
    mask = (0 < evals.real) & (evals.real < 20)
    # print(mask)
    evals = evals[mask]
    evects = evects[:, mask]

    # sorting
    order = np.argsort(np.round(evals.real, 3) + np.round(evals.imag, 3) / 1e6)
    # print(order)
    evals = evals[order]
    evects = evects[:, order]

    # print(evals)

    return evals[1:3], evects[:, 1:3]

```

```
###################### Eigenvectors plot ###################################

def spatial_wavefunctions(N, x, epsilons, evals, evects):
    # calculating basis functions
    x[x == 0] = 1e-200
    PSI_ns = []
    for n in range(N):
        psi_n = cpsi_blank(n, x)
        PSI_ns.append(psi_n)
    PSI_ns = np.array(PSI_ns)

    # plt.plot(x, PSI_ns[1])
    # plt.show()
    np.save(f"PSI_ns.npy", PSI_ns)


    for i, ϵ in enumerate(epsilons):

        eigenvalues, c = filtering_and_sorting(evals[i], evects[i])

        if c.shape[1] < 2:
            print(i, "continuing")
            continue
        eigenstates = []
        for j in range(2):
            # print(len(evects))
            d = c[:, j]
            psi_jx = np.zeros(x.shape, complex)
            # for each H.O. basis vector relevant to the filtered and sorted
eigenvectors
            for n in range(N):
                psi_jx += d[n] * PSI_ns[n]
            # normalise
            psi_jx /= np.sqrt(np.sum(abs(psi_jx) ** 2 * delta_x))
            # impose phase convention at ~ x = 0
            psi_jx *= np.exp(-1j * np.angle(psi_jx[Nx // 2]))

            eigenstates.append(psi_jx)

        fig, ax = plt.subplots()
        # # probability density
        # plt.plot(x, abs(eigenstates[0])**2, "-", color='blue', linewidth=1,
label=fr"$|\psi_1|^2$") # first excited state
        # plt.plot(x, abs(eigenstates[1])**2, "-", color='orange', linewidth=1,
label=fr"$|\psi_2|^2$") # second excited state

        # spatial wavefunctions
        plt.plot(x, np.real(eigenstates[0]), "-", color='blue', linewidth=1,
label=fr"real $\psi_1$") # first excited state
        plt.plot(x, np.real(eigenstates[1]), "-", color='orange', linewidth=1,
label=fr"real $\psi_2$") # second excited state
        plt.plot(x, np.imag(eigenstates[0]), "--", color='blue', linewidth=0.4,
label=fr"imaginary $\psi_1$") # first excited state
        plt.plot(x, np.imag(eigenstates[1]), "--", color='orange',
linewidth=0.4, label=fr"imaginary $\psi_2$") # second excited state

        plt.legend(loc="upper right")
```

```python
        plt.xlabel(r'$x$')
        # plt.ylabel(r'$ |\psi_{n}|^2$')
        plt.ylabel(r'$ \psi_{n}$')
        textstr = '\n'.join((
            fr'$E_1 = {eigenvalues[0]:.03f}$',
            fr'$E_2 = {eigenvalues[1]:.03f}$',
            fr'$ϵ = {ϵ:.03f}$'
            ))

        # place a text box in upper left in axes coords
        ax.text(0.02, 1.15, textstr, transform=ax.transAxes, verticalalignment='top')
        plt.savefig(f"density_spatial_wavefunctions/wavefunction_{i:03d}.png")
        plt.savefig(f"spatial_wavefunctions/wavefunction_{i:03d}.png")
        plt.clf()
    return eigenvalues, eigenstates


####################### Function calls ##################################

N = 100
Nϵ = 100
Nx = 1024
epsilons = np.linspace(-1.0, 0, Nϵ)
xs = np.linspace(-10, 10, Nx)
delta_x = xs[1] - xs[0]

evals, evects = linear_algebra(epsilons)

eigenvalues, eigenstates = spatial_wavefunctions(N, xs, epsilons, evals, evects)
```