06/01/21 02:55:06 /home/ana/Desktop/proj/code/Bender/Bender.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, 04/04/2021

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 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]


## TEST
def complex_fsolve(func, E0, **kwargs):
    # root finding algorithm. FINDS: Energy values from error() function
    # call: complex_fsolve(error, E0, args=(ϵ, n))
    def real_func(*args):
        return np.real(func(*args))

    real_root = fsolve(real_func, E0, **kwargs)
    #     def imag_func(*args):
    #         return np.imag(func(*args))
    #     imag_root = fsolve(imag_func, E0, **kwargs)
    # Check that the imaginary part of func is also zero
    value = func(real_root[0], *kwargs['args'])
    assert abs(np.imag(value)) < 1e-10, "Imaginary part wasn't zero"
    # print(f"E = {Energies[0]:.04f}")
    return real_root[0]


def integrand(x_prime, tp_minus, E, ϵ):
    # Change of variables integrand
    x = x_prime + 1j * np.imag(tp_minus)
    return np.sqrt(E - x ** 2 * (1j * x) ** ϵ)


def LHS(n):
    # Quantization condition
    return (n + 1 / 2) * np.pi


def RHS(E, ϵ):
    # Integral defining E
    # integration LIMITS
    tp_minus = E ** (1 / (ϵ + 2)) * np.exp(1j * np.pi * (3 / 2 - (1 / (ϵ + 2))))
    tp_plus = E ** (1 / (ϵ + 2)) * np.exp(-1j * np.pi * (1 / 2 - (1 / (ϵ + 2))))
    tp_minus_prime = np.real(
        E ** (1 / (ϵ + 2)) * np.exp(1j * np.pi * (3 / 2 - (1 / (ϵ + 2))))
        - 1j * np.imag(tp_minus)
    )
    tp_plus_prime = np.real(
        E ** (1 / (ϵ + 2)) * np.exp(-1j * np.pi * (1 / 2 - (1 / (ϵ + 2))))
        - 1j * np.imag(tp_minus)
    )
    # print(tp_minus_prime)
    # print(tp_plus_prime)
    return complex_quad(integrand, tp_minus_prime, tp_plus_prime, args=(tp_minus, E, ϵ))


def error(E, ϵ, n):
    return RHS(E, ϵ) - LHS(n)


def compare():
    # comparison to WKB results reported for  (ϵ, n) = (1, 0) using IC: E0 = E0
    E_RK = E0
    E_WKB = 1.0943
    diff_RK_WKB = E_RK - E_WKB
    diff_RK_mine = E_RK - complex_fsolve(error, E0, args=(1, 0))
    how_many_sigmas_theory = diff_RK_WKB / E_RK
    how_many_sigmas_mine = diff_RK_mine / E_RK
    print(
        f"\nnumber of σ away is WKB from exact result: {abs(how_many_sigmas_theory):.3f}"
    )
    print(f"number of σ away am I from exact result: {abs(how_many_sigmas_mine):.3f}\n")


def analytic_E(ϵ, n):
    # Bender equation (34) pg. 960
    top = gamma(3 / 2 + 1 / (ϵ + 2)) * np.sqrt(np.pi) * (n + 1 / 2)
    bottom = np.sin(np.pi / (ϵ + 2)) * gamma(1 + 1 / (ϵ + 2))
    return (top / bottom) ** ((2 * ϵ + 4) / (ϵ + 4))


def brute_force(func, E, ϵ):
    # BRUTE INTEGRAL
    def real_func(*args):
        return np.real(func(*args))

    tp_minus = E ** (1 / (ϵ + 2)) * np.exp(1j * np.pi * (3 / 2 - (1 / (ϵ + 2))))
    tp_plus = E ** (1 / (ϵ + 2)) * np.exp(-1j * np.pi * (1 / 2 - (1 / (ϵ + 2))))
    tp_minus_prime = E ** (1 / (ϵ + 2)) * np.exp(
        1j * np.pi * (3 / 2 - (1 / (ϵ + 2)))
    ) - 1j * np.imag(tp_minus)
    tp_plus_prime = E ** (1 / (ϵ + 2)) * np.exp(
        -1j * np.pi * (1 / 2 - (1 / (ϵ + 2)))
    ) - 1j * np.imag(tp_minus)
    # domain & differential (infinitesimal)
    x_prime = np.linspace(tp_minus_prime, tp_plus_prime, 50000)
    dx_prime = x_prime[1] - x_prime[0]
    return np.sum(real_func(x_prime, E, ϵ) * dx_prime)


## Runge-Kutta, finding IC!
def find_k(x, ϵ, E):
    return np.sqrt(x ** 2 * (1j * x) ** ϵ - E)


# Schrödinger equation
def Schrodinger_eqn(x, Ψ):
    psi, psi_prime = Ψ
    psi_primeprime = (x ** 2 * (1j * x) ** ϵ - E) * psi
    Ψ_prime = np.array([psi_prime, psi_primeprime])
    return Ψ_prime


def Runge_Kutta(x, delta_x, Ψ):
    k1 = Schrodinger_eqn(x, Ψ)
    k2 = Schrodinger_eqn(x + delta_x / 2, Ψ + k1 * delta_x / 2)
    k3 = Schrodinger_eqn(x + delta_x / 2, Ψ + k2 * delta_x / 2)
    k4 = Schrodinger_eqn(x + delta_x, Ψ + k3 * delta_x)
    return Ψ + (delta_x / 6) * (k1 + 2 * k2 + 2 * k3 + k4)


#######################################function calls############################################

# WKB approximation
# IC based on RK results given (ϵ, n) = (1, 0)
ϵ = 1
E1 = 1.1563
E = E1

N = 100
epsilons = np.linspace(-1.0, 0, N)

tp_minus = E ** (1 / (ϵ + 2)) * np.exp(1j * np.pi * (3 / 2 - (1 / (ϵ + 2))))
tp_plus = E ** (1 / (ϵ + 2)) * np.exp(-1j * np.pi * (1 / 2 - (1 / (ϵ + 2))))

tp_minus_prime = E ** (1 / (ϵ + 2)) * np.exp(
    1j * np.pi * (3 / 2 - (1 / (ϵ + 2)))
) - 1j * np.imag(tp_minus)
tp_plus_prime = E ** (1 / (ϵ + 2)) * np.exp(
    -1j * np.pi * (1 / 2 - (1 / (ϵ + 2)))
```

```python
) - 1j * np.imag(tp_minus)

######################## FINAL plot of eigenvalues ###################################

# Energies_2 = []
# for n in range(10):
#     E_s = []
#     for ϵ in np.linspace(0, 3, 30):
#         E = complex_fsolve(error, E1, args=(ϵ, n))
#         E_s.append(E)
#         ## print(f" {ϵ = }, {n = }, {E = }")
#     Energies_2.append(E_s)
# np.save("Energies_unbroken.npy", Energies_2)
Energies_2 = np.load("Energies_unbroken.npy")


def figure_final_form():
    for E_ϵs in Energies_2:
        # print(E_ϵs)
        ϵ = np.linspace(0, 3, 30)
        # plt.plot(ϵ, E_ϵs, "o-", color='k', markersize=1) # MARK OR UNMARK

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

        # # full figure # MARK OR UNMARK
        # positive_evals = [
        #     i for i in eigenvalues if 0 < np.real(i) < 20 and abs(np.imag(i)) < 0.3
        # ]

        # # broken region
        positive_evals = [
            i for i in eigenvalues if 0 < np.real(i) < 20 and abs(np.imag(i)) < 6
        ]

        sorted_eigenvalues = sorted(positive_evals, key=lambda x: np.real(x))
        sorted_eigenvalues = sorted_eigenvalues[:11]

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

        mask_imag = 1e-6 < abs(np.imag(sorted_eigenvalues))
        plt.plot(
            ϵ_list[mask_imag],
            np.imag(sorted_eigenvalues)[mask_imag],
            marker='.',
            linestyle='None',
            color='r',
            markersize=1,
        )

        plt.plot(
            ϵ_list,
            np.real(sorted_eigenvalues),
            marker='.',
            linestyle='None',
            color='k',
            markersize=2.5,
        )

        mask_real = 1e-6 < abs(np.imag(sorted_eigenvalues))
        plt.plot(
            ϵ_list[mask_real],
            np.real(sorted_eigenvalues)[mask_real],
            marker='.',
            linestyle='None',
            color='xkcd:azure',
            markersize=2,
        )

    # # full figure # MARK OR UNMARK
    # plt.axis(xmin=-1, xmax=3, ymin=0, ymax=20)
    # plt.axvline(0, color='purple', linestyle=':', label="PT-symmetry
breaking")
    # plt.legend()

    # # only broken symmetry region
    plt.axis(xmin=-1, xmax=0, ymin=-2, ymax=12)
    plt.axhline(0, color='grey', linestyle='-')

    plt.xlabel("ϵ")
    plt.ylabel("E")
    plt.savefig("NHH_eigenvalues.png")
    plt.show()
    return np.array(eigenvectors_list)

coefficients = figure_final_form()

####################### Eigenvectors plot ##################################

def spatial_wavefunctions(N, x, epsilons):
    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)
    np.save(f"PSI_ns.npy", PSI_ns)

    eigenstates = []
    for i, ϵ in enumerate(epsilons):
        c = coefficients[i]
        for j in range(N):  # for each eigenvector
            d = c[:, j]
            psi_jx = np.zeros(x.shape, complex)
            for n in range(N):  # for each H.O. basis vector
                psi_jx += d[n] * PSI_ns[n]
                plt.plot(x, abs(psi_jx) ** 2)
                # plt.savefig(f"spatial_wavefunctions/wavefunction_{ϵ}_{n:03d}.png")
                plt.clf()
            eigenstates.append(psi_jx)

    np.save(f'eigenstates.npy', np.array(eigenstates))

N = 100
epsilons = np.linspace(-1.0, 0, N)
xs = np.linspace(-20, 20, 1024)
spatial_wavefunctions(N, xs, epsilons)

####################### Eigenvectors plot ##################################


###################### Runge-Kutta test call ###############################

# psi = np.empty_like(xs, dtype = "complex_")
# for i in range(len(xs)):
#     psi[i] = Runge_Kutta(x, delta_x, Ψ0)[0]
#     # print(psi[i])
#     x += delta_x

##################### Runge-Kutta test call ###############################

# ######################### WKB TEST 1 ####################################
# def whats_up_with_integrand(x_values, E, ϵ):
#     # checking the integration path of the integrand in the x-complex plane
#     reals = []
#     imaginary = []
#     for x in x_values:
#         complex_num = np.sqrt(E - x**2 * (1j * x)**ϵ)

#         reals.append(np.real(complex_num))
#         imaginary.append(np.imag(complex_num))

#     plt.plot(reals, imaginary, '-')
#     plt.plot(reals[0], imaginary[0],'go', label='start here')
#     plt.plot(reals[-1], imaginary[-1],'ro', markersize='1.2', label='finish here')
#     plt.legend()
#     plt.ylabel(r'$Im(\sqrt{E - x^2 (i x)^\epsilon})$')
#     plt.xlabel(r'$Re(\sqrt{E - x^2 (i x)^\epsilon})$')

#     # plt.title("Bender's integration contour")
#     plt.title("change or variables integration contour")
#     plt.show()


# # Bender's integral
# x_values = np.linspace(tp_minus, tp_plus, 10000)

# # change of variables integral
# x_values = np.linspace(tp_minus_prime, tp_plus_prime, 10000) + 1j * np.imag(tp_minus)

# whats_up_with_integrand(x_values, E0, ϵ)
# ######################## WKB TEST 1 #####################################

########################### WKB TEST 2 ####################################
## ITERATIVE approach 1 for ϵ = 1
# Energies_1 = []
# for n in range(10):
#   E = complex_fsolve(error, E1, args=(1, n))
#   Energies_1.append(E)

# # comparison to WKB and RK reported in Bender
# n = range(10)
# E_RK= [1.1563, 4.1093, 7.5623, 11.3144, 15.2916, 19.4515, 23.7667, 28.2175,
32.7891, 37.4698]
# E_WKB = [1.0943, 4.0895, 7.5489, 11.3043, 15.2832, 19.4444, 23.7603, 28.2120,
32.7841, 37.4653]
# plt.plot(n, Energies_1 , label="my calculated energies")
# plt.plot(n, E_RK , label=r"$E_{RK}$")
# plt.plot(n, E_WKB , label=r"$E_{WKB}$")
# plt.legend()
# plt.xlabel("n")
# plt.ylabel("E")
# plt.show()
########################### WKB TEST 2 ####################################

######################## WKB unbroken region ##############################

# E0 = 1.1563
# # # ITERATIVE
# Energies = []
# for n in range(10):
#     E_ϵs = []
#     for ϵ in np.linspace(0, 3, 30):
#         E_ϵ = complex_fsolve(error, E0, args=(ϵ, n))
#         E_ϵs.append(E_ϵ)
#     Energies.append(E_ϵs)

# #PLOTING ITERATIVE
# for E_ϵs in Energies:
#     ϵ = np.linspace(0, 3, 30)
#     plt.plot(ϵ, E_ϵs, "o-", markersize=2)
# plt.ylim(0, 20)
# plt.xlabel("ϵ")
# plt.ylabel("E")
# plt.show()

######################## WKB unbroken region ##############################
```