

06/01/21 02:55:06 /home/ana/Desktop/proj/code/Bender/Bender.py

```

1  # PHS3350
2  # Week 5 - Energy levels of a family of non-Hermitian Hamiltonians
3  # "what I cannot create I do not understand" - R. Feynman.
4  # Ana Fabela Hinojosa, 04/04/2021
5
6  import numpy as np
7  import matplotlib
8  import matplotlib.pyplot as plt
9  from scipy.integrate import quad
10 from scipy.optimize import fsolve
11 from scipy.special import gamma
12 import scipy.special as sc
13 from scipy import linalg
14 from tqdm import tqdm
15 from odhobs import psi as cpsi_blank
16
17 plt.rcParams['figure.dpi'] = 200
18 np.set_printoptions(linewidth=200)
19
20
21 def complex_quad(func, a, b, **kwargs):
22     # Integration using scipy.integrate.quad() for a complex function
23     def real_func(*args):
24         return np.real(func(*args))
25
26     def imag_func(*args):
27         return np.imag(func(*args))
28
29     real_integral = quad(real_func, a, b, **kwargs)
30     imag_integral = quad(imag_func, a, b, **kwargs)
31     return real_integral[0] + 1j * imag_integral[0]
32
33
34 ## TEST
35 def complex_fsolve(func, E0, **kwargs):
36     # root finding algorithm. FINDS: Energy values from error() function
37     # call: complex_fsolve(error, E0, args=(epsilon, n))
38     def real_func(*args):
39         return np.real(func(*args))
40
41     real_root = fsolve(real_func, E0, **kwargs)
42     #     def imag_func(*args):
43     #         return np.imag(func(*args))
44     #     imag_root = fsolve(imag_func, E0, **kwargs)
45     # Check that the imaginary part of func is also zero
46     value = func(real_root[0], *kwargs['args'])
47     assert abs(np.imag(value)) < 1e-10, "Imaginary part wasn't zero"
48     # print(f"E = {Energies[0]:.04f}")
49     return real_root[0]
50
51
52 def integrand(x_prime, tp_minus, E, epsilon):
53     # Change of variables integrand
54     x = x_prime + 1j * np.imag(tp_minus)

```

```

55     return np.sqrt(E - x ** 2 * (1j * x) ** ε)
56
57
58 def LHS(n):
59     # Quantization condition
60     return (n + 1 / 2) * np.pi
61
62
63 def RHS(E, ε):
64     # Integral defining E
65     # integration LIMITS
66     tp_minus = E ** (1 / (ε + 2)) * np.exp(1j * np.pi * (3 / 2 - (1 / (ε + 2))))
67     tp_plus = E ** (1 / (ε + 2)) * np.exp(-1j * np.pi * (1 / 2 - (1 / (ε + 2))))
68     tp_minus_prime = np.real(
69         E ** (1 / (ε + 2)) * np.exp(1j * np.pi * (3 / 2 - (1 / (ε + 2))))
70         - 1j * np.imag(tp_minus)
71     )
72     tp_plus_prime = np.real(
73         E ** (1 / (ε + 2)) * np.exp(-1j * np.pi * (1 / 2 - (1 / (ε + 2))))
74         - 1j * np.imag(tp_minus)
75     )
76     # print(tp_minus_prime)
77     # print(tp_plus_prime)
78     return complex_quad(integrand, tp_minus_prime, tp_plus_prime,
79 args=(tp_minus, E, ε))
80
81 def error(E, ε, n):
82     return RHS(E, ε) - LHS(n)
83
84
85 def compare():
86     # comparison to WKB results reported for (ε, n) = (1, 0) using IC: E0 = E0
87     E_RK = E0
88     E_WKB = 1.0943
89     diff_RK_WKB = E_RK - E_WKB
90     diff_RK_mine = E_RK - complex_fsolve(error, E0, args=(1, 0))
91     how_many_sigmas_theory = diff_RK_WKB / E_RK
92     how_many_sigmas_mine = diff_RK_mine / E_RK
93     print(
94         f"\nnumber of σ away is WKB from exact result:
95 {abs(how_many_sigmas_theory):.3f}"
96     )
97     print(f"number of σ away am I from exact result:
98 {abs(how_many_sigmas_mine):.3f}\n")
99
100 def analytic_E(ε, n):
101     # Bender equation (34) pg. 960
102     top = gamma(3 / 2 + 1 / (ε + 2)) * np.sqrt(np.pi) * (n + 1 / 2)
103     bottom = np.sin(np.pi / (ε + 2)) * gamma(1 + 1 / (ε + 2))
104     return (top / bottom) ** ((2 * ε + 4) / (ε + 4))
105
106 def brute_force(func, E, ε):
107     # BRUTE INTEGRAL
108     def real_func(*args):

```

```

109         return np.real(func(*args))
110
111     tp_minus = E ** (1 / (ε + 2)) * np.exp(1j * np.pi * (3 / 2 - (1 / (ε + 2))))
112     tp_plus = E ** (1 / (ε + 2)) * np.exp(-1j * np.pi * (1 / 2 - (1 / (ε + 2))))
113     tp_minus_prime = E ** (1 / (ε + 2)) * np.exp(
114         1j * np.pi * (3 / 2 - (1 / (ε + 2)))
115     ) - 1j * np.imag(tp_minus)
116     tp_plus_prime = E ** (1 / (ε + 2)) * np.exp(
117         -1j * np.pi * (1 / 2 - (1 / (ε + 2)))
118     ) - 1j * np.imag(tp_minus)
119     # domain & differential (infinitesimal)
120     x_prime = np.linspace(tp_minus_prime, tp_plus_prime, 50000)
121     dx_prime = x_prime[1] - x_prime[0]
122     return np.sum(real_func(x_prime, E, ε) * dx_prime)
123
124
125 ## Runge-Kutta, finding IC!
126 def find_k(x, ε, E):
127     return np.sqrt(x ** 2 * (1j * x) ** ε - E)
128
129
130 # Schrödinger equation
131 def Schrodinger_eqn(x, Ψ):
132     psi, psi_prime = Ψ
133     psi_primeprime = (x ** 2 * (1j * x) ** ε - E) * psi
134     Ψ_prime = np.array([psi_prime, psi_primeprime])
135     return Ψ_prime
136
137
138 def Runge_Kutta(x, delta_x, Ψ):
139     k1 = Schrodinger_eqn(x, Ψ)
140     k2 = Schrodinger_eqn(x + delta_x / 2, Ψ + k1 * delta_x / 2)
141     k3 = Schrodinger_eqn(x + delta_x / 2, Ψ + k2 * delta_x / 2)
142     k4 = Schrodinger_eqn(x + delta_x, Ψ + k3 * delta_x)
143     return Ψ + (delta_x / 6) * (k1 + 2 * k2 + 2 * k3 + k4)
144
145
146 #####function
147 calls#####
148
149 # WKB approximation
150 # IC based on RK results given (ε, n) = (1, 0)
151 ε = 1
152 E1 = 1.1563
153 E = E1
154
155 N = 100
156 epsilons = np.linspace(-1.0, 0, N)
157
158 tp_minus = E ** (1 / (ε + 2)) * np.exp(1j * np.pi * (3 / 2 - (1 / (ε + 2))))
159 tp_plus = E ** (1 / (ε + 2)) * np.exp(-1j * np.pi * (1 / 2 - (1 / (ε + 2))))
160
161 tp_minus_prime = E ** (1 / (ε + 2)) * np.exp(
162     1j * np.pi * (3 / 2 - (1 / (ε + 2)))
163 ) - 1j * np.imag(tp_minus)
164 tp_plus_prime = E ** (1 / (ε + 2)) * np.exp(
165     -1j * np.pi * (1 / 2 - (1 / (ε + 2)))

```

```

165 ) - 1j * np.imag(tp_minus)
166
167 ##### FINAL plot of eigenvalues
168 #####
169 # Energies_2 = []
170 # for n in range(10):
171 #     E_s = []
172 #     for e in np.linspace(0, 3, 30):
173 #         E = complex_fsolve(error, E1, args=(e, n))
174 #         E_s.append(E)
175 #         ## print(f" {e = }, {n = }, {E = }")
176 #     Energies_2.append(E_s)
177 # np.save("Energies_unbroken.npy", Energies_2)
178 Energies_2 = np.load("Energies_unbroken.npy")
179
180
181 def figure_final_form():
182     for E_es in Energies_2:
183         # print(E_es)
184         e = np.linspace(0, 3, 30)
185         # plt.plot(e, E_es, "o-", color='k', markersize=1) # MARK OR UNMARK
186
187         eigenvectors_list = []
188         for i, e in enumerate(epsilons):
189             matrix = np.load(f'matrices/matrix_{i:03d}.npy')
190             eigenvalues, eigenvectors = linalg.eig(matrix)
191             eigenvectors_list.append(eigenvectors)
192
193             # # full figure # MARK OR UNMARK
194             # positive_evals = [
195             #     i for i in eigenvalues if 0 < np.real(i) < 20 and abs(np.imag(i))
196             #     < 0.3
197             # ]
198
199             # # broken region
200             positive_evals = [
201                 i for i in eigenvalues if 0 < np.real(i) < 20 and abs(np.imag(i)) <
202                 6
203             ]
204
205             sorted_eigenvalues = sorted(positive_evals, key=lambda x: np.real(x))
206             sorted_eigenvalues = sorted_eigenvalues[:11]
207
208             e_list = np.full(len(sorted_eigenvalues), e)
209
210             mask_imag = 1e-6 < abs(np.imag(sorted_eigenvalues))
211             plt.plot(
212                 e_list[mask_imag],
213                 np.imag(sorted_eigenvalues)[mask_imag],
214                 marker='.',
215                 linestyle='None',
216                 color='r',
217                 markersize=1,
218             )
219
220             plt.plot(

```

```

219         e_list,
220         np.real(sorted_eigenvalues),
221         marker='.',
222         linestyle='None',
223         color='k',
224         markersize=2.5,
225     )
226
227     mask_real = 1e-6 < abs(np.imag(sorted_eigenvalues))
228     plt.plot(
229         e_list[mask_real],
230         np.real(sorted_eigenvalues)[mask_real],
231         marker='.',
232         linestyle='None',
233         color='xkcd:azure',
234         markersize=2,
235     )
236
237     # # full figure # MARK OR UNMARK
238     # plt.axis(xmin=-1, xmax=3, ymin=0, ymax=20)
239     # plt.axvline(0, color='purple', linestyle=':', label="PT-symmetry
breaking")
240     # plt.legend()
241
242     # # only broken symmetry region
243     plt.axis(xmin=-1, xmax=0, ymin=-2, ymax=12)
244     plt.axhline(0, color='grey', linestyle='--')
245
246     plt.xlabel("ε")
247     plt.ylabel("E")
248     plt.savefig("NHH_eigenvalues.png")
249     plt.show()
250     return np.array(eigenvectors_list)
251
252 coefficients = figure_final_form()
253
254 ##### Eigenvectors plot #####
255
256 def spatial_wavefunctions(N, x, epsilons):
257     x[x == 0] = 1e-200
258     PSI_ns = []
259     for n in range(N):
260         psi_n = cpsi_blank(n, x)
261         PSI_ns.append(psi_n)
262     PSI_ns = np.array(PSI_ns)
263     np.save(f"PSI_ns.npy", PSI_ns)
264
265     eigenstates = []
266     for i, ε in enumerate(epsilons):
267         c = coefficients[i]
268         for j in range(N): # for each eigenvector
269             d = c[:, j]
270             psi_jx = np.zeros(x.shape, complex)
271             for n in range(N): # for each H.0. basis vector
272                 psi_jx += d[n] * PSI_ns[n]
273             plt.plot(x, abs(psi_jx) ** 2)

```

```

274         #
275         plt.savefig(f"spatial_wavefunctions/wavefunction_{\epsilon}_{n:03d}.png")
276         plt.clf()
277         eigenstates.append(psi_jx)
278     np.save(f'eigenstates.npy', np.array(eigenstates))
279
280     N = 100
281     epsilons = np.linspace(-1.0, 0, N)
282     xs = np.linspace(-20, 20, 1024)
283     spatial_wavefunctions(N, xs, epsilons)
284
285     ##### Eigenvectors plot #####
286
287
288     ##### Runge-Kutta test call #####
289
290     # psi = np.empty_like(xs, dtype = "complex_")
291     # for i in range(len(xs)):
292     #     psi[i] = Runge_Kutta(x, delta_x, \Psi0)[0]
293     #     # print(psi[i])
294     #     x += delta_x
295
296     ##### Runge-Kutta test call #####
297
298     # ##### WKB TEST 1 #####
299     # def whats_up_with_integrand(x_values, E, \epsilon):
300     #     # checking the integration path of the integrand in the x-complex plane
301     #     reals = []
302     #     imaginary = []
303     #     for x in x_values:
304     #         complex_num = np.sqrt(E - x**2 * (1j * x)**\epsilon)
305     #         reals.append(np.real(complex_num))
306     #         imaginary.append(np.imag(complex_num))
307     #     plt.plot(reals, imaginary, '-')
308     #     plt.plot(reals[0], imaginary[0], 'go', label='start here')
309     #     plt.plot(reals[-1], imaginary[-1], 'ro', markersize='1.2', label='finish
310     # here')
311     #     plt.legend()
312     #     plt.ylabel(r'$Im(\sqrt{E - x^2 (i x)^\epsilon})$')
313     #     plt.xlabel(r'$Re(\sqrt{E - x^2 (i x)^\epsilon})$')
314
315     #     # plt.title("Bender's integration contour")
316     #     # plt.title("change of variables integration contour")
317     #     plt.show()
318
319
320
321     # # Bender's integral
322     # x_values = np.linspace(tp_minus, tp_plus, 10000)
323
324     # # change of variables integral
325     # x_values = np.linspace(tp_minus_prime, tp_plus_prime, 10000) + 1j *
326     # np.imag(tp_minus)
327     # whats_up_with_integrand(x_values, E0, \epsilon)

```

```
328 # ##### WKB TEST 1 #####
329
330 ##### WKB TEST 2 #####
331 ## ITERATIVE approach 1 for  $\epsilon = 1$ 
332 # Energies_1 = []
333 # for n in range(10):
334 #     E = complex_fsolve(error, E1, args=(1, n))
335 #     Energies_1.append(E)
336
337 # # comparison to WKB and RK reported in Bender
338 # n = range(10)
339 # E_RK= [1.1563, 4.1093, 7.5623, 11.3144, 15.2916, 19.4515, 23.7667, 28.2175,
340 # 32.7891, 37.4698]
341 # E_WKB = [1.0943, 4.0895, 7.5489, 11.3043, 15.2832, 19.4444, 23.7603, 28.2120,
342 # 32.7841, 37.4653]
343 # plt.plot(n, Energies_1 , label="my calculated energies")
344 # plt.plot(n, E_RK , label=r"$E_{RK}$")
345 # plt.plot(n, E_WKB , label=r"$E_{WKB}$")
346 # plt.legend()
347 # plt.xlabel("n")
348 # plt.ylabel("E")
349 # plt.show()
350 ##### WKB TEST 2 #####
351
352 ##### WKB unbroken region #####
353
354 # E0 = 1.1563
355 # # # ITERATIVE
356 # Energies = []
357 # for n in range(10):
358 #     E_es = []
359 #     for  $\epsilon$  in np.linspace(0, 3, 30):
360 #         E_ε = complex_fsolve(error, E0, args=( $\epsilon$ , n))
361 #         E_es.append(E_ε)
362 #     Energies.append(E_es)
363
364 # #PLOTING ITERATIVE
365 # for E_es in Energies:
366 #      $\epsilon$  = np.linspace(0, 3, 30)
367 #     plt.plot( $\epsilon$ , E_es, "o-", markersize=2)
368 # plt.ylim(0, 20)
369 # plt.xlabel(" $\epsilon$ ")
370 # plt.ylabel("E")
371 # plt.show()
372
373 ##### WKB unbroken region #####
```