
revised-ising-gap.py

```
1 import bootstrap
2 import matplotlib.pyplot as plt
3 import time
4 import datetime
5 import numpy as np
6 from matplotlib.backends.backend_pdf import PdfPages
7
8 class Grid(object):
9     def __init__(self, kmax, lmax, mmax, nmax, allowed_points, disallowed_points):
10         self.kmax = kmax
11         self.lmax = lmax
12         self.mmax = mmax
13         self.nmax = nmax
14         self.allowed_points = allowed_points
15         self.disallowed_points = disallowed_points
16
17 # We define a class with imposes a gap in the Z2-even operator sector.
18 # The continuum starts at a specified value, and we add an operator between this and unitarity
19 # bound.
20 class IsingGap(object):
21     bootstrap.cutoff=1e-10
22     def __init__(self, from_file = False, file_name = 'name', dim = 3, gap = 3, sig_values = np.
23         arange(0.5,0.85,0.05).tolist(), eps_values = np.arange(1.0,2.2,0.2).tolist()):
24         self.dim = dim
25         self.gap = gap
26         self.sig_values = sig_values
27         self.eps_values = eps_values
28         if from_file == True:
29             self.recover_table(file_name)
30         else:
31             self.table = []
32
33 # Determines allowed and disallowed scaling dimensions for whatever the parameters are.
34 def determine_grid(self, key):
35     if self.get_grid_index(key) != -1:
36         tab1 = bootstrap.ConformalBlockTable(self.dim, *key)
37         tab2 = bootstrap.ConvolvedBlockTable(tab1)
38
39         # Instantiate a Grid object with appropriate input values.
40         grid=Grid(*key, [], [])
41
42         for sig in self.sig_values:
43             for eps in self.eps_values:
44                 sdp = bootstrap.SDP(sig, tab2)
45                 sdp.set_bound(0, float(self.gap))
46                 sdp.add_point(0, eps)
47                 result = sdp.iterate()
48                 if result:
49                     grid.allowed_points.append((sig, eps))
50                 else:
51                     grid.disallowed_points.append((sig, eps))
52
53 # Now append this grid object to the IsingGap table.
54 # Note we will need to implement a look up table to retrieve desired data.
```

```

53     self.table.append(grid)
54
55     # For a given set of conformal blocks, set by kmax and lmax, generate a grids for a specified
56     range of mmax and nmax.
57
58     # If we obtain a grid of entirely disallowed points, fill in the rest of the grids for that
59     kmax and lmax.
60
61     def iterate_parameters(self, kmax_range, lmax_range, mmax_range, nmax_range):
62         keys = self.generate_keys(kmax_range, lmax_range, mmax_range, nmax_range)
63
64         while len(keys) > 0:
65             # Used keys will store the keys for which there is already a grid in table.
66             used_keys = []
67             null_keys = []
68
69             for key in keys:
70                 #if self.get_grid_index(key) != -1:
71                 # used_keys.append(key)
72                 # continue
73                 print("Trying kmax = " + str(key[0]) + ", lmax = " + str(key[1]) + ", mmax = " + str(key
74                     [2]) + ", nmax = " + str(key[3]))
75                 self.determine_grid(key)
76                 used_keys.append(key)
77
78                 # If the grid has only disallowed points...
79                 if self.table[self.get_grid_index(key)].allowed_points == []:
80                     print("In the if statement.")
81                     k = key[0]
82                     l = key[1]
83                     m = key[2]
84                     n = key[3]
85
86                     null_keys = [key for key in keys if key[0] == k and key[1] == l and key[2] >= m and key
87                         [3] >= n]
88
89                     for key in null_keys:
90                         grid = Grid(*key, [], [])
91
92                         for sig in self.sig_values:
93                             for eps in self.eps_values:
94                                 grid.disallowed_points.append((sig, eps))
95
96                         self.table.append(grid)
97
98                     break
99
100                 # We remove all keys from the list that we are done with.
101                 keys = [key for key in keys if key not in null_keys and key not in used_keys]
102
103     # Saves the data as an executable file that will repopulate the table attribute.
104     def save_to_file(self, name):
105         with open(name + ".py", 'a') as file:
106             file.write("self.table = []\n")
107             for grid in self.table:
108                 file.write("kmax = " + str(grid.kmax) + "\n")
109                 file.write("lmax = " + str(grid.lmax) + "\n")

```

```

107         file.write("mmax = " + str(grid.mmax) + "\n")
108         file.write("nmax = " + str(grid.nmax) + "\n")
109         file.write("allowed_points = " + str(grid.allowed_points) + "\n")
110         file.write("disallowed_points = " + str(grid.disallowed_points) + "\n")
111         file.write("self.table.append(Grid(kmax, lmax, mmax, nmax, allowed_points,
        disallowed_points))" + "\n")
112
113     # Recovers a table stored to a file.
114     def recover_table(self, file_name):
115         exec(open(file_name + ".py").read())
116
117
118     # Searches table of grids for index matching the input key. Returns -1 if not found.
119     def get_grid_index(self, key):
120         for i in range(0, len(self.table)):
121             if self.table[i].kmax == key[0] and self.table[i].lmax == key[1] and self.table[i].mmax ==
        key[2] and self.table[i].nmax == key[3]:
122                 return i
123         return -1
124
125     # Plots and saves a series of grids to an output PDF file.
126     # Takes as input parameter values for which we want plotted grids, and the desired PDF file
        name.
127     def plot_grids(self, keys, file_name):
128         table = self.generate_table(keys)
129         pdf_pages = PdfPages(file_name + ".pdf")
130
131         # Define the number of plots per page and the size of the grid board.
132         nb_plots = len(table)
133         nb_plots_per_page = 6
134         nb_pages = int(np.ceil(nb_plots / float(nb_plots_per_page)))
135         grid_size=(3,2)
136
137         # This will define which row of the grid we are on.
138         row_index = 0
139
140         # We go through each 'grid' in 'table', generating a plot for each.
141         for i in range(nb_plots):
142             # To begin, declare a new figure / page if we have exceeded limit of the last page.
143             if i % nb_plots_per_page == 0:
144                 fig = plt.figure(figsize=(8.27, 11.69), dpi=100)
145
146             # Now, add a plot for the current grid on the grid board.
147             plt.subplot2grid(grid_size, (row_index, i % grid_size[1]))
148             if i % grid_size[1] == 1:
149                 row_index += 1
150
151             # Handle our data. Retrieve isolated points for plotting from out input table of Grid
                objects.
152             allowed_sig = [points[0] for points in table[i].allowed_points]
153             allowed_eps = [points[1] for points in table[i].allowed_points]
154             disallowed_sig = [points[0] for points in table[i].disallowed_points]
155             disallowed_eps = [points[1] for points in table[i].disallowed_points]
156
157             # Plot a grid.
158             plt.plot(allowed_sig, allowed_eps, 'r+')
159             plt.plot(disallowed_sig, disallowed_eps, 'b+')
160             plt.title('kmax : ' + table[i].kmax.__str__() + " " +

```

```

161         'lmax : ' + table[i].lmax.__str__() + " " +
162         'mmax : ' + table[i].mmax.__str__() + " " +
163         'nmax : ' + table[i].nmax.__str__())
164
165     # If we have filled a page, or have reached the end of our plots, tight-pack and save the
166     # page.
167     if (i + 1) % nb_plots_per_page == 0 or (i + 1) == nb_plots:
168         plt.tight_layout()
169         pdf_pages.savefig(fig)
170         row_index = 0
171
172     pdf_pages.close()
173
174 # Returns a key or list of keys generated by the input parameter ranges.
175 def generate_keys(self, kmax_range, lmax_range, mmax_range, nmax_range):
176     if type(kmax_range) == int:
177         kmax_range = [kmax_range]
178     if type(lmax_range) == int:
179         lmax_range = [lmax_range]
180     if type(mmax_range) == int:
181         mmax_range = [mmax_range]
182     if type(nmax_range) == int:
183         nmax_range = [nmax_range]
184     keys = []
185     for kmax in kmax_range:
186         for lmax in lmax_range:
187             for mmax in mmax_range:
188                 for nmax in nmax_range:
189                     key = [kmax, lmax, mmax, nmax]
190                     keys.append(key)
191     return keys
192
193 # Generates a subtable table of desired, already determined grids from main table.
194 # Gives a warning message if a grid isn't found.
195 def generate_table(self, keys):
196     # table to store the resulting grids.
197     table = []
198     for key in keys:
199         if self.get_grid_index(key) == -1:
200             print("Grid at kmax = " + str(key[0]) + ", " +
201                   "lmax = " + str(key[1]) + ", " +
202                   "mmax = " + str(key[2]) + ", " +
203                   "nmax = " + str(key[3]) + ", " + "does not exist.")
204         else:
205             table.append(self.table[self.get_grid_index(key)])
206
207     return table
208
209 # Takes two keys and returns a dictionary with the direction of every point.
210 def changes(self, key1, key2):
211     changes = {}
212     allowed_one = self.table[self.get_grid_index(key1)].allowed_points
213     allowed_two = self.table[self.get_grid_index(key2)].allowed_points
214
215     for sig in self.sig_values:
216         for eps in self.eps_values:
217             if (sig, eps) in allowed_one and (sig, eps) in allowed_two:
218                 changes[(sig, eps)] = 0

```

```

218         if (sig, eps) not in allowed_one and (sig, eps) not in allowed_two:
219             changes[(sig, eps)] = 0
220         if (sig, eps) in allowed_one and (sig, eps) not in allowed_two:
221             changes[(sig, eps)] = -1
222         if (sig, eps) not in allowed_one and (sig, eps) in allowed_two:
223             changes[(sig, eps)] = 1
224     return changes
225
226 def plot_changes(self, keys, file_name):
227     pdf_pages = PdfPages(file_name + ".pdf")
228
229     # Define the number of plots per page and the size of the grid board.
230     # We have one less plots than grids.
231     nb_plots = len(keys)
232     nb_plots_per_page = 6
233     nb_pages = int(np.ceil(nb_plots / float(nb_plots_per_page)))
234     grid_size=(3,2)
235
236     # This will define which row of the grid we are on.
237     row_index = 0
238
239     # We go through each 'grid' in 'table', generating a plot for each.
240     for i in range(nb_plots):
241         # To begin, declare a new figure / page if we have exceeded limit of the last page.
242         if i % nb_plots_per_page == 0:
243             fig = plt.figure(figsize=(8.27, 11.69), dpi=100)
244
245         # Now, add a plot for the current grid on the grid board.
246         plt.subplot2grid(grid_size, (row_index, i % grid_size[1]))
247         if i % grid_size[1] == 1:
248             row_index += 1
249
250         # We want the first grid to compare all changes to.
251         if i == 0:
252             grid = self.table[self.get_grid_index(keys[i])]
253             allowed_sig = [points[0] for points in grid.allowed_points]
254             allowed_eps = [points[1] for points in grid.allowed_points]
255             disallowed_sig = [points[0] for points in grid.disallowed_points]
256             disallowed_eps = [points[1] for points in grid.disallowed_points]
257
258             # Plot the grid.
259             plt.plot(allowed_sig, allowed_eps, 'r+')
260             plt.plot(disallowed_sig, disallowed_eps, 'b+')
261             plt.title('kmax : ' + grid.kmax.__str__() + " " +
262                     'lmax : ' + grid.lmax.__str__() + " " +
263                     'mmax : ' + grid.mmax.__str__() + " " +
264                     'nmax : ' + grid.nmax.__str__())
265
266             y_range = plt.ylim()
267             x_range = plt.xlim()
268
269         else:
270             changes = self.changes(keys[i-1], keys[i])
271             unchanged_points = []
272             to_allowed_points = []
273             to_disallowed_points = []
274             for point in changes:
275                 if changes[point] == 0:

```

```

276         unchanged_points.append(point)
277     if changes[point] == 1:
278         to_allowed_points.append(point)
279     if changes[point] == -1:
280         to_disallowed_points.append(point)
281
282     unchanged_sig = [points[0] for points in unchanged_points]
283     unchanged_eps = [points[1] for points in unchanged_points]
284     to_disallowed_sig = [points[0] for points in to_disallowed_points]
285     to_disallowed_eps = [points[1] for points in to_disallowed_points]
286     to_allowed_sig = [points[0] for points in to_allowed_points]
287     to_allowed_eps = [points[1] for points in to_allowed_points]
288
289     # Plot a grid.
290     plt.plot(to_allowed_sig, to_allowed_eps, 'r+')
291     plt.plot(to_disallowed_sig, to_disallowed_eps, 'b+')
292     plt.xlim(x_range)
293     plt.ylim(y_range)
294     plt.title('kmax : ' + self.table[self.get_grid_index(keys[i])].kmax.__str__() + " " +
295             'lmax : ' + self.table[self.get_grid_index(keys[i])].lmax.__str__() + " " +
296             'mmax : ' + self.table[self.get_grid_index(keys[i])].mmax.__str__() + " " +
297             'nmax : ' + self.table[self.get_grid_index(keys[i])].nmax.__str__())
298
299     # If we have filled a page, or have reached the end of our plots, tight-pack and save the
page.
300     if (i + 1) % nb_plots_per_page == 0 or (i + 1) == nb_plots:
301         plt.tight_layout()
302         pdf_pages.savefig(fig)
303         row_index = 0
304
305 pdf_pages.close()

```