revised-ising-gap.py

```
import bootstrap
1
    import matplotlib.pyplot as plt
2
    import time
3
    import datetime
4
5
    import numpy as np
    from matplotlib.backends.backend_pdf import PdfPages
6
7
8
    class Grid(object):
      def __init__(self, kmax, lmax, mmax, nmax, allowed_points, disallowed_points):
9
10
        self.kmax = kmax
        self.lmax = lmax
11
12
        self.mmax = mmax
        self.nmax = nmax
13
14
        self.allowed_points = allowed_points
        self.disallowed_points = disallowed_points
15
16
    # We define a class with imposes a gap in the Z_2-even operator sector.
17
    # The continuum starts at a specified value, and we add an operator between this and unitarity
18
        bound.
    class IsingGap(object):
19
20
      bootstrap.cutoff=1e-10
21
      def __init__(self, from_file = False, file_name = 'name', dim = 3, gap = 3, sig_values = np.
          arange(0.5,0.85,0.05).tolist(), eps_values = np.arange(1.0,2.2,0.2).tolist()):
        self.dim = dim
22
23
        self.gap = gap
24
        self.sig_values = sig_values
25
        self.eps_values = eps_values
        if from_file == True:
26
27
          self.recover_table(file_name)
        else:
28
29
          self.table = []
30
      # Determines allowed and disallowed scaling dimensions for whatever the parameters are.
31
      def determine_grid(self, key):
32
        if self.get_grid_index(key) != -1:
33
          tab1 = bootstrap.ConformalBlockTable(self.dim, *key)
34
35
          tab2 = bootstrap.ConvolvedBlockTable(tab1)
36
          # Instantiate a Grid object with appropriate input values.
37
          grid=Grid(*key, [], [])
38
39
40
          for sig in self.sig_values:
             for eps in self.eps_values:
41
               sdp = bootstrap.SDP(sig, tab2)
42
               sdp.set_bound(0, float(self.gap))
43
               sdp.add_point(0, eps)
44
               result = sdp.iterate()
45
               if result:
46
                 grid.allowed_points.append((sig, eps))
47
               else:
48
                 grid.disallowed_points.append((sig, eps))
49
50
        # Now append this grid object to the IsingGap table.
51
        # Note we will need to implement a look up table to retrieve desired data.
52
```

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

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

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

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

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