revised-ising-gap.py

```
import bootstrap
import matplotlib.pyplot as plt
import time
import datetime
import numpy as np
from matplotlib.backends.backend_pdf import PdfPages

class Grid(object):
    def __init__(self, dim, kmax, lmax, mmax, nmax, allowed_points, disallowed_points):
        self.dim = dim
        self.kmax = kmax
        self.lmax = lmax
        self.mmax = mmax
        self.nmax = nmax
        self.allowed_points = allowed_points
        self.disallowed_points = disallowed_points

class IsingGap(object):
  bootstrap.cutoff=1e-10
    def __init__(self, from_file = False, file_name = 'name', gap = 3, sig_values = np.arange
        (0.5,0.85,0.05).tolist(), eps_values = np.arange(1.0,2.2,0.2).tolist()):
        if from_file == True:
            self.recover_table(file_name)
        else:
            self.default_inputs = {'dim': 3, 'kmax': 7, 'lmax': 7, 'mmax': 2, 'nmax': 4}
            self.inputs = self.default_inputs
            self.gap = gap
            self.sig_values = sig_values
            self.eps_values = eps_values
            self.table = []

    def determine_grid(self):
        #key = [self.inputs['dim'], self.inputs['kmax'], self.inputs['lmax'], self.inputs['mmax
            '], self.inputs['nmax']]
        key = list(self.inputs.values())
        tab1 = bootstrap.ConformalBlockTable(*key)
        tab2 = bootstrap.ConvolvedBlockTable(tab1)

        # Instantiate a Grid object with appropriate input values.
        grid=Grid(*key, [], [])

        for sig in self.sig_values:
            for eps in self.eps_values:

                sdp = bootstrap.SDP(sig,tab2)
                sdp.set_bound(0,float(self.gap))
                sdp.add_point(0,eps)
                result = sdp.iterate()

                if result:
                    grid.allowed_points.append((sig, eps))
                else:
                    grid.disallowed_points.append((sig,eps))

```

```
        # Now append this grid object to the IsingGap table.
        # Note we will need to implement a look up table to retrieve desired data.
        self.table.append(grid)

    def iterate_parameter(self, par, par_range):
        if type(par_range) == int:
            par_range = [par_range]
        for x in par_range:
            self.inputs[par] = x
            if self.get_grid_index(*list(self.inputs.values())) != -1:
              continue
            self.determine_grid()
        self.inputs = self.default_inputs

    def save_table_to_file(self, name):
        with open(name + ".py", 'w') as file:
            file.write("self.default_inputs = " + self.default_inputs.__str__() + "\n")
            file.write("self.inputs = " + self.inputs.__str__() + "\n")
            file.write("self.gap = " + self.gap.__str__() + "\n")
            file.write("self.sig_values = " + self.sig_values.__str__() + "\n")
            file.write("self.eps_values = " + self.eps_values.__str__() + "\n")
            file.write("self.table = []\n")
            for grid in self.table:
              file.write("dim = " + str(grid.dim) + "\n")
              file.write("kmax = " + str(grid.kmax) + "\n")
              file.write("lmax = " + str(grid.lmax) + "\n")
              file.write("mmax = " + str(grid.mmax) + "\n")
              file.write("nmax = " + str(grid.nmax) + "\n")
              file.write("allowed_points = " + str(grid.allowed_points) + "\n")
              file.write("disallowed_points = " + str(grid.disallowed_points) + "\n")
              file.write("self.table.append(Grid(dim, kmax, lmax, mmax, nmax, allowed_points,
                  disallowed_points))" + "\n")
#             file.write("self.table = table")

    def recover_table(self, file_name):
      exec(open(file_name + ".py").read())

    # Searches table of grids for index matching input parameters. Returns -1 if not found.
    def get_grid_index(self, dim, kmax, lmax, mmax, nmax):
      for i in range(0, len(self.table)):
        if self.table[i].dim == dim and self.table[i].kmax == kmax and self.table[i].lmax == lmax
             and self.table[i].mmax == mmax and self.table[i].nmax == nmax:
          return i
      return -1
```