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_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


  # Note, imputs will be a list of grid objects, as found in the table attribute.
  def plot_grids(dim_values, kmax_values, lmax_values, mmax_values, nmax_values):

    table = self.generate_table(dim_values, kmax_values, lmax_values, mmax_values, nmax_values)

    pdf_pages = PdfPages(’grids.pdf’)

    # Define the number of plots per page and the size of the grid board.
    nb_plots = len(table)
    nb_plots_per_page = 6
    nb_pages = int(np.ceil(nb_plots / float(nb_plots_per_page)))
    grid_size=(3,2)

    # This will define which row of the grid we are on.
    row_index = 0

    # We go through each ’grid’ in ’table’, generating a plot for each.
    for i in range(nb_plots):
      # To begin, declare a new figure / page if we have exceeded limit of the last page.
      if i % nb_plots_per_page == 0:
        fig = plt.figure(figsize=(8.27, 11.69), dpi=100)

      # Now, add a plot for the current grid on the grid board.
      plt.subplot2grid(grid_size, (row_index, i % grid_size[1]))
      if i % grid_size[1] == 1:
        row_index += 1

      # Handle our data. Retrieve isolated points for plotting from out input table of Grid objects.
      allowed_sig = [points[0] for points in table[i].allowed_points]
      allowed_eps = [points[1] for points in table[i].allowed_points]
      disallowed_sig = [points[0] for points in table[i].disallowed_points]
      disallowed_eps = [points[1] for points in table[i].disallowed_points]

      # Plot a grid.
      plt.plot(allowed_sig, allowed_eps, ’r+’)
      plt.plot(disallowed_sig, disallowed_eps, ’b+’)
      plt.title(’kmax : ’ + table[i].kmax.__str__() + " " +
          ’lmax : ’ + table[i].lmax.__str__() + " " +
          ’mmax : ’ + table[i].mmax.__str__() + " " +
          ’nmax : ’ + table[i].nmax.__str__())

      # If we have filled a page, or have reached the end of our plots, tight-pack and save the page.
      if (i + 1) % nb_plots_per_page == 0 or (i + 1) == nb_plots:
        plt.tight_layout()
        pdf_pages.savefig(fig)
        row_index = 0

    pdf_pages.close()


  # Generates a table of already determined grids, specified by lists of points of input parameters.
  def generate_table(dim_range, kmax_range, lmax_range, mmax_range, nmax_range):
    # table to store the resulting grids.
    table = []

    if type(dim_range) == int:
      dim_range = [dim_range]
    if type(kmax_range) == int:
      kmax_range = [kmax_range]
    if type(lmax_range) == int:
      lmax_range = [lmax_range]
    if type(mmax_range) == int:
      mmax_range = [mmax_range]
    if type(nmax_range) == int:
      nmax_range = [nmax_range]

    # Generates a list of unique keys, giving a warning message if a grid isn’t found.
    keys = []
    for dim in dim_range:
      for kmax in kmax_range:
        for lmax in lmax_range:
          for mmax in mmax_range:
            for nmax in nmax_range:
              key = [dim, kmax, lmax, mmax, nmax]
              if self.get_grid_index(*key) == -1:
                print("Grid at dim = " + str(key[0]) + ", " +
                  "kmax = " + str(key[1]) + ", " +
                  "lmax = " + str(key[2]) + ", " +
                  "mmax = " + str(key[3]) + ", " +
                  "nmax = " + str(key[4]) + " does not exist.")
              else:
                table.append(self.table[self.get_grid_index(*key)])

    return table
```