PHY441 Final Project

Charles Knox

December 10, 2015

Intro

Using the Metropolis algorithm, an Einsten-model solid is simulated. A series of N quantum harmonic oscillators coupled with a heat bath at temperature T will cause the average energy $\langle E(T) \rangle$ to eventually reach a stable, converging value.

The probability of a given energy level being populated in a canonical ensemble is given by

$$P(E_n) = \frac{e^{-n\beta h\nu}}{Z}$$

In our system we have set $h\nu = 1$, and the thermodynamic beta $\beta = 1/T$ (we are working in units such that the Boltzmann constant is 1):

$$P(E_n) = \frac{e^{-n\beta}}{Z}$$

The partition function is

$$\begin{aligned} Z &= \sum_{n=0}^{\infty} e^{-n\beta} \\ &= \frac{1}{1-e^{-\beta}} \end{aligned}$$

And the average energy is

$$\begin{aligned} \langle E \rangle &= \sum_{n=0}^{\infty} e^{-n\beta} \\ &= \frac{e^{-\beta}}{1-e^{-\beta}} \end{aligned}$$

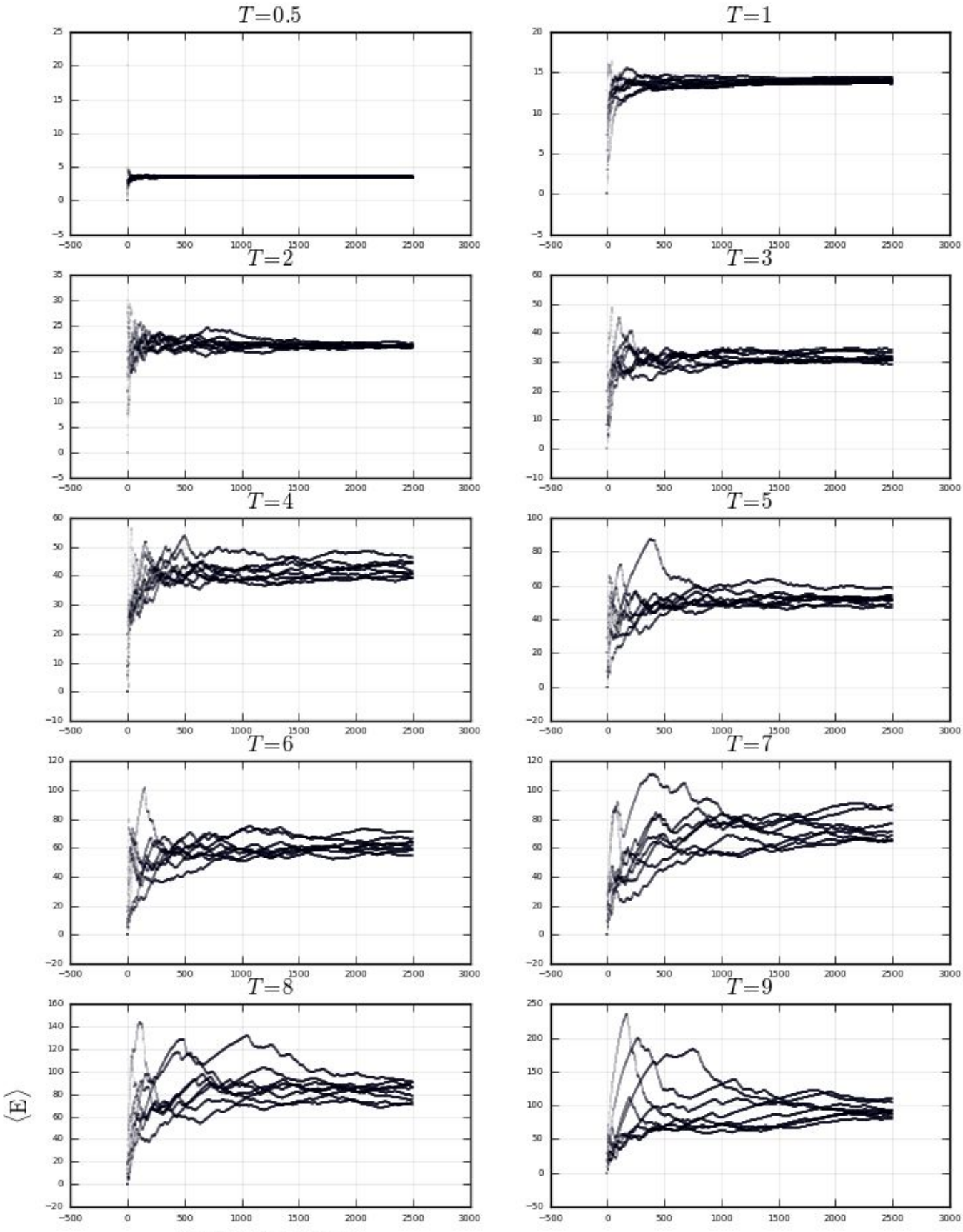
The Metropolis algorithm has been implemented as follows, where we have pre-selected fixed T, n, N (N = number of Monte Carlo steps/iterations to perform):

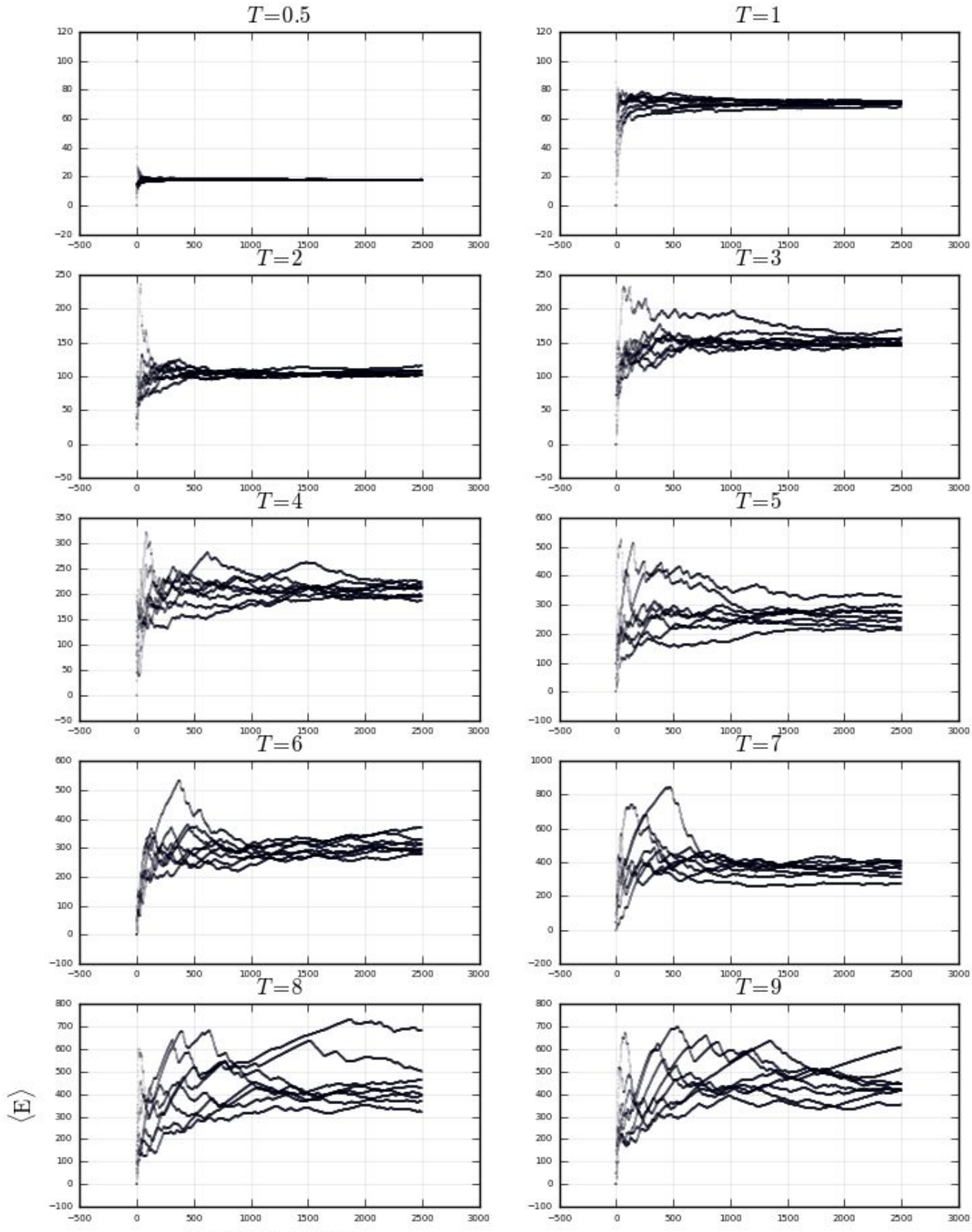
1. For n particles, create a microstate with corresponding $E_0 = n$ energy.
2. For the first step, set $E_i = E_0 = N$.
3. Set the value of $E_j = E_i \pm 1$, where the sign is randomly picked. Ensure that it can never be negative.
4. For all sequential steps, calculate the value of $\Delta E = E_j - E_i$.
   (a) If $\Delta E < 0$, accept the change, and set $E_i = E_j$ and proceed to the next iteration.
   (b) If $\Delta E > 0$, choose a random number between $r = (0, 1)$. Then, choose another number $w = e^{-\beta \Delta E}$:
      i. If $r \leq w$, accept the change.
      ii. If $r > w$, $E_i = E_j \cdot w = E_j \cdot e^{-\beta \Delta E}$. *This is the key part of the algorithm!*

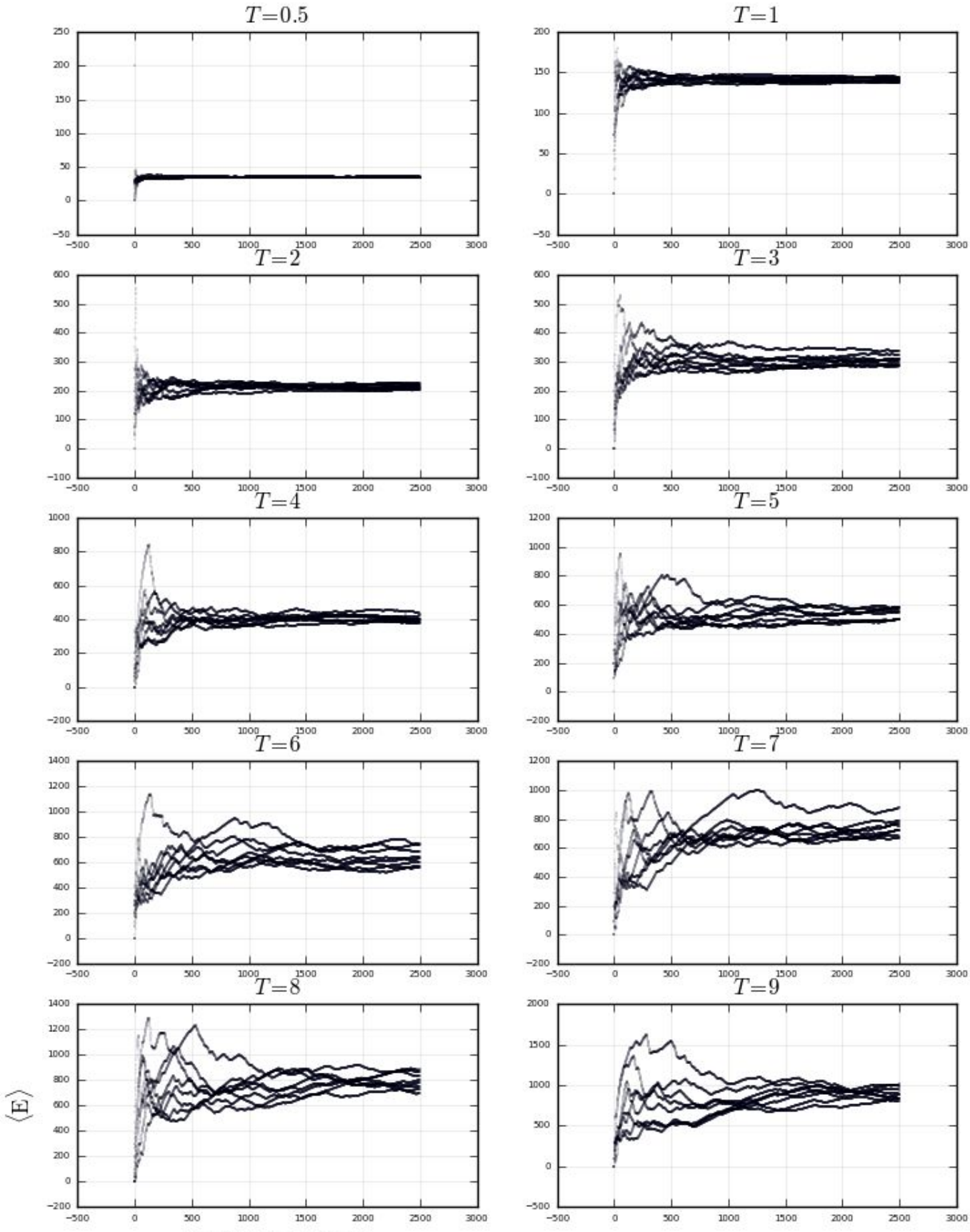
Results

Note that in these plots we have done 2500 iterations in the Monte Carlo simulation. The results of higher iterations do not seem to differ much at these temperatures, so it is safe to assume that these results have allowed a sufficient progression for us to distinguish divergent and convergent behaviors.

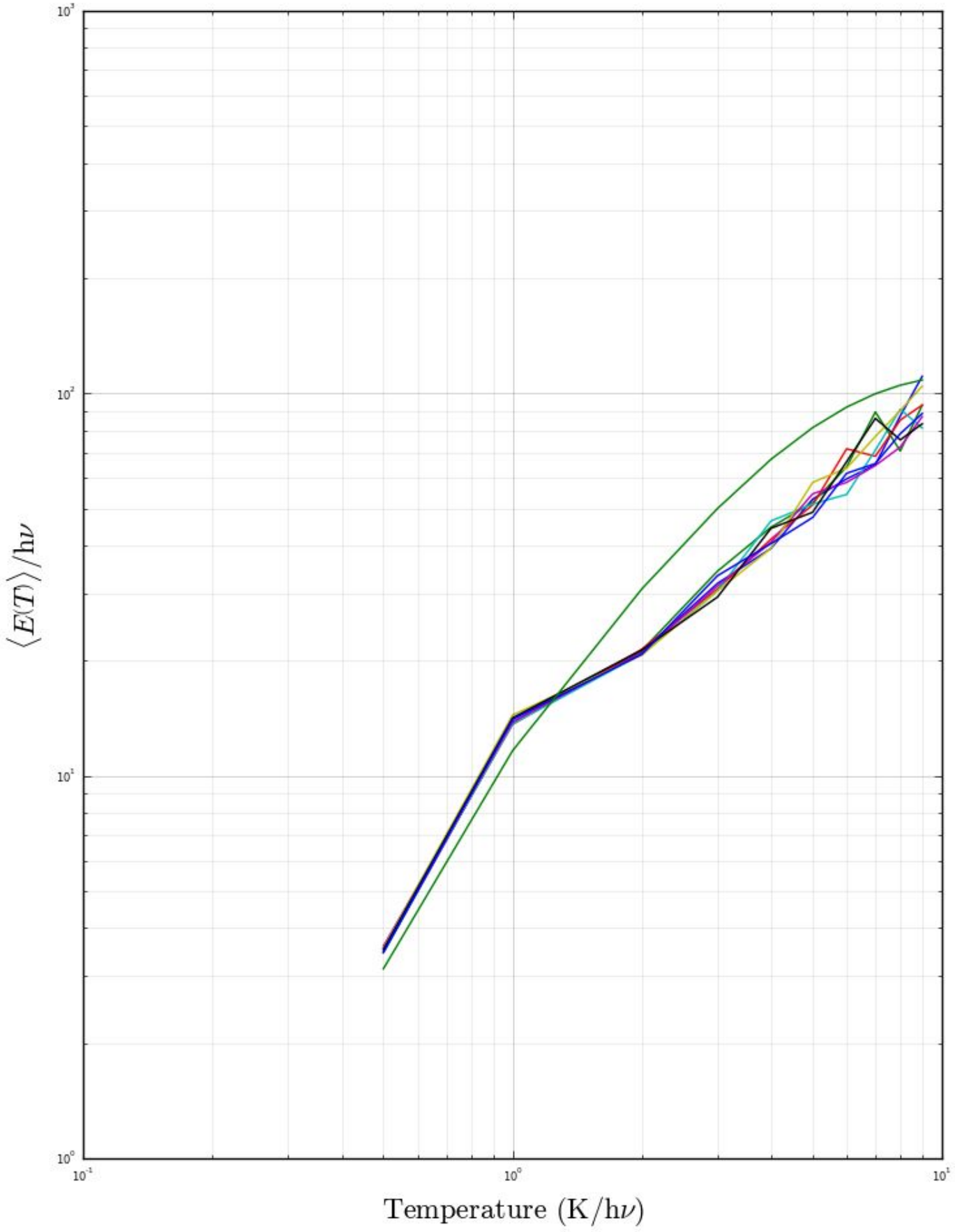
- In the first category of the attached figures, we see the plots for both $n = 20, 100, 200$ and $0.5 < T < 9$. There are multiple Monte Carlo simulations composited in each plot.
  - We can see pretty clearly that the $\langle E \rangle$ values converge eventually.
  - For smaller temperature values, the behavior converges quickly; for larger temperature values, the behavior is still convergent but takes longer.
- In the second category of the attached figures, again we see the plots for $n = 20, 100, 200$ particles for $\langle E(T) \rangle$ values.
  - The analytical $\langle E(T) \rangle$ trend is easily distinguished, colored in turquoise.
  - We can see that for larger T, the $\langle E(T) \rangle$ values begin to diverge but still stay relatively near each other.
  - The analytical and simulated results show a similar trend in general, but there are noticeable differences between the two.


Trials = 2500, n = 20 Particles, 8 Runs
T=0.5
T=1
T=2
T=3
T=4
T=5
T=6
T=7
T=8
T=9
⟨E⟩
Monte Carlo Steps

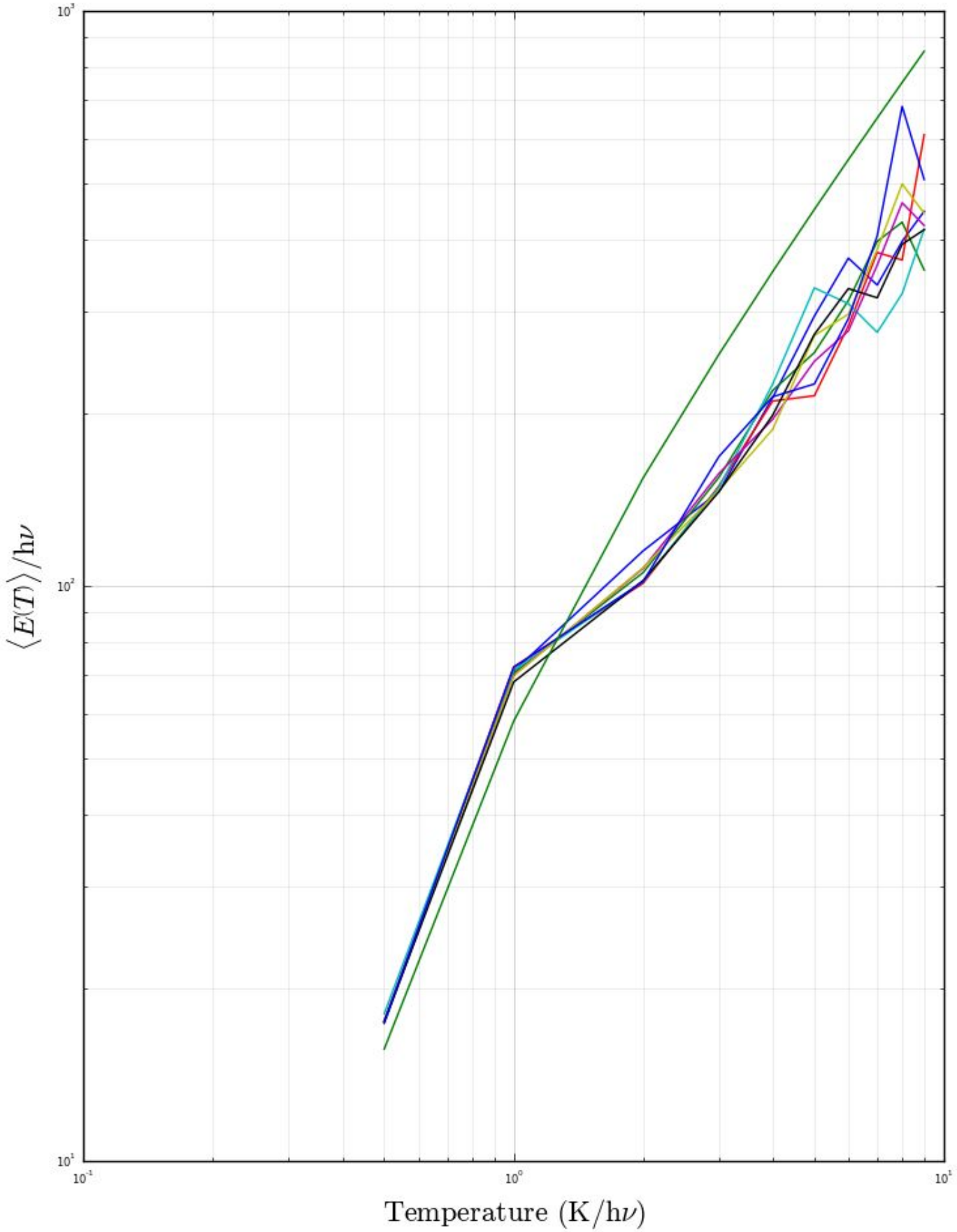

Trials = 2500, n = 100 Particles, 8 Runs
T=0.5
T=1
T=2
T=3
T=4
T=5
T=6
T=7
T=8
T=9
⟨E⟩
Monte Carlo Steps


Trials = 2500, n = 200 Particles, 8 Runs
T=0.5
T=1
T=2
T=3
T=4
T=5
T=6
T=7
T=8
T=9
⟨E⟩
Monte Carlo Steps

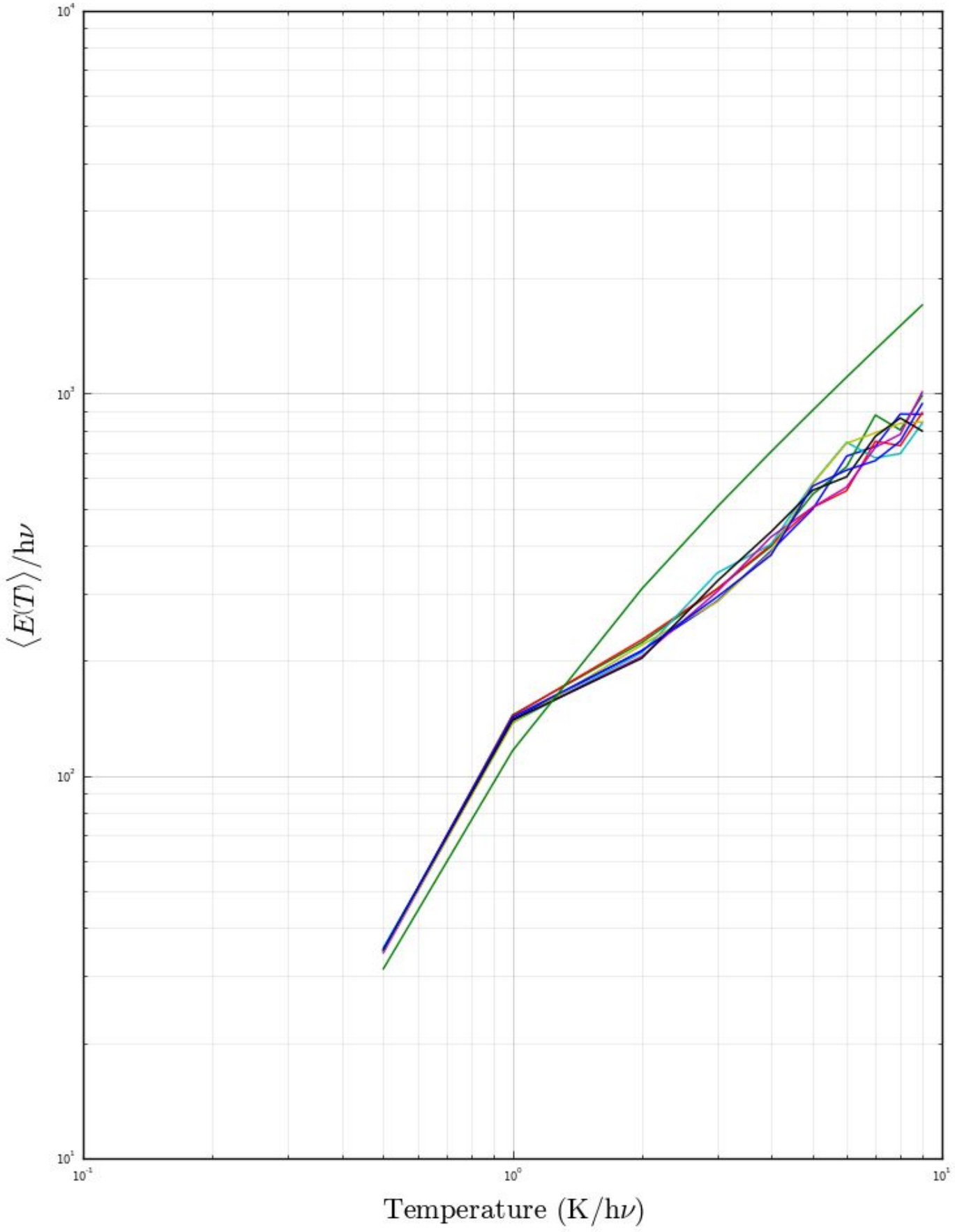
$\langle E(T) \rangle$ values : $n = 20$, Trials $= 2500$



$\langle E(T) \rangle$ values: $n = 100$, Trials $= 2500$



$\langle E(T) \rangle$ values: $n = 200$, Trials $= 2500$



```
import numpy as np
import math
import random
import matplotlib.pyplot as plt

#Just the algorithm.

n_particles = 20
n_trials = 5000

temp = 2.0
beta = 1.0/temp

e_0 = 0.0
e_i = 0.0
e_j = 0.0

e_list = []
e_avg_list = []

def randi():
    return random.randint(0,1)

def rand():
    return random.random()

def get_p(dE):
    return math.exp(-1.0*beta*dE)

def get_p_list(e_vals):
    p_list = []
    for e_val in e_vals:
        p_list.append(get_p(e_val))
    return p_list
def mult(a, b):
    list_out = []
    if len(a) != len(b):
        print "got lists of different length"
    else:
        for i in range(0, len(a)):
            list_out.append(a[i] * b[i])
    return list_out

trials = range(0, n_trials)

for x in trials:
    if x is 0:
        e_i = e_0               #Initialize
    e_j = e_i + (1 if randi() is 1 else -1)
    if e_j < 0:
        e_j = e_i + (1 if randi() is 1 else -1)
    else:
        #calculate dE
        dE = e_j - e_i
        if dE <= 0:
            e_i = e_j
        else:
            w = get_p(dE)
```

```
59              if rand() <= w:
60                  e_i = e_j
61              else:
62                  e_i = e_j*w
63
64      e_list.append(e_i)
65
66      #calculate <E>
67      p_list = get_p_list(e_list)
68      Z = sum(p_list)
69      e_avg = n_particles * sum(mult(e_list, p_list)) / Z
70      e_avg_list.append(e_avg)
71
72  plt.plot(trials, e_avg_list)
73  #plt.savefig('algorithm_test.svg')
```

```
#
#   Metropolis Algorithm Implementation
#

import matplotlib.pyplot as plt
import argparse
import collections
import numpy as np
import math
import random
from time import strftime

def log(message):
    print message

def mult(a, b, scalar=1):
    list_out = []
    if len(a) != len(b):
        print "got lists of different length"
    else:
        for i in range(0, len(a)):
            list_out.append(scalar * (a[i]*b[i]))
    return list_out

def randi():
    return random.randint(0,1)

def rand():
    return random.random()

def get_p(beta, dE):
    return math.exp(-1.0*beta*dE)

def get_p_list(beta, e_vals):
    p_list = []
    for e_val in e_vals:
        p_list.append(get_p(beta, e_val))
    return p_list

def gen_mcs_dist(temp, num_particles, num_trials):
    n_particles = num_particles
    n_trials = num_trials

    temp = float(temp)
    beta = 1.0/temp

    e_0 = 0.0
    e_i = 0.0
    e_j = 0.0

    e_list = []
    e_avg_list = []

    trials = range(0, n_trials)

    for x in trials:
        if x is 0:
            e_i = e_0                                   #Initialize
        e_j = e_i + (1 if randi() is 1 else -1)         #Change by +/- 1
        if e_j < 0:
            e_j = e_i + (1 if randi() is 1 else -1) #Retry for negative
                                                        #energies
        else:
```

```
64              dE = e_j - e_i                          #calculate dE
65              if dE <= 0:
66                  e_i = e_j                           #Accept if -1
67              else:
68                  w = get_p(beta, dE)                 #Accept +1
69                  if rand() <= w:                     # if w <= r
70                      e_i = e_j                       #
71                  else:                               #Otherwise, accept
72                      e_i = e_j*w                     # value of (+1*w)
73
74          e_list.append(e_i)
75          p_list = get_p_list(beta, e_list)           #calculate <E> for each
76          Z = sum(p_list)                             # trial
77          #e_avg = n_particles * sum(mult(e_list, p_list)) / Z
78          e_avg = sum(mult(e_list, p_list, n_particles)) / Z
79          e_avg_list.append(e_avg)
80      return e_avg_list
81
82  def gen_complete_distributions(num_particles,
83                                 plots_temperatures,
84                                 num_trials,
85                                 run_number):
86      e_single_average_list = []
87      e_lists = []
88      for z in range(0, len(plots_temperatures)):
89          log('Starting: Monte Carlo run number ' + str(
90                  run_number) + ', temperature is ' + str(plots_temperatures[z]))
91          e_list = gen_mcs_dist(plots_temperatures[z],
92                  num_particles,
93                  num_trials)
94          e_avg_val = e_list[len(e_list) - 1]
95          e_single_average_list.append(e_avg_val) #Try last E_avg
96          e_lists.append(e_list)
97      return e_lists, e_single_average_list
98
99  def main(args_dict):
100     #-------------------------------------------------------------------------
101     num_particles = int(args_dict['num_particles'])
102     plots_temperatures = [0.5,1,2,3,4,5,6,7,8,9]
103     num_trials = int(args_dict['num_trials'])
104     trials = range(0, num_trials)
105     runs = int(args_dict['num_runs'])
106     log(strftime("%Y-%m-%d %H:%M:%S"))
107     log('Doing ' + str(runs) + ' total simulations with ' + str(
108             len(plots_temperatures)) + ' temperature variations per run')
109     #-------------------------------------------------------------------------
110     average_energies_lists = []                     #Later we will plot <E>/T
111     e_lists_lists = []                              #ex:e_lists_lists[run][temp]
112     #-----------------------------------------------#Do multiple simulations
113     for xx in range(0, runs):                       # and store the results
114         e_lists, e_single_average_list = gen_complete_distributions(
115                                             num_particles,
116                                             plots_temperatures,
117                                             num_trials,
118                                             xx)
119         average_energies_lists.append(e_single_average_list)
120         e_lists_lists.append(e_lists)
121     #-------------------------------------------------------------Figure1 config
122     fig = plt.figure(figsize=(8.5,11))
123     plot_index = 0
124     for z in range(0, len(plots_temperatures)):
125         plot_index += 1
126         #-----------------------------------------------------------Subplot config
```

```
        plt.subplot(5, 2, plot_index)
        plt.title(r"$T=" + str(plots_temperatures[z]) + r"$", fontsize=12)
        if plot_index == 9:
            plt.ylabel(r"$\mathrm{\langle E \rangle}$",
                       fontsize=12)
        if plot_index == 9:
            plt.xlabel(r"$\mathrm{Monte\,Carlo\,Steps}$", fontsize=12)
        plt.tick_params(axis='y', which='major', labelsize=5)
        plt.tick_params(axis='y', which='minor', labelsize=5)
        plt.tick_params(axis='x', which='major', labelsize=5)
        plt.tick_params(axis='x', which='minor', labelsize=5)
        plt.grid(axis="both", alpha=0.10, linestyle="-")
        for xx in range(0, runs):                               #Plot All Runs
            plt.scatter(trials,                                 #at Once!
                        e_lists_lists[xx][z],
                        s=0.25,
                        alpha=0.10)
        #-------------------------------------------------------Subplot Config
    fig.suptitle(r"$\mathrm{Trials}=" + str(num_trials) + r",\,n=" +
                 str(num_particles) + r"\mathrm{\,Particles,\,}" +
                 str(runs) + r"\mathrm{\, Runs}$", fontsize=18)
    plt.savefig("mcs_" + str(num_particles) + "_particles_" + str(num_trials) + "_trials.png")
    plt.savefig("mcs_" + str(num_particles) + "_particles_" + str(num_trials) + "_trials.svg")
    plt.savefig("mcs_" + str(num_particles) + "_particles_" + str(num_trials) + "_trials.pdf")
    #-------------------------------------------------------Figure1 End
    #
    #-------------------------------------------------------Figure2 Begin
    fig = plt.figure(figsize=(8.5,11))
    for average_energy_list in average_energies_lists:
        plt.loglog(plots_temperatures, average_energy_list)
    analytical_energy_averages = []                 #Calculate Analytical <E>
    for temp in plots_temperatures:
        beta = 1.0 / temp
        Z = 1.0 / (1.0 - get_p(beta, 1.0))
        analytical_energy_value = 0.0
        for n in range(0, num_particles):
            analytical_energy_value += n * get_p(beta, n)
        #exp_val = math.exp(-beta)
        #analytical_energy_value = exp_val / (1.0 - exp_val)
        analytical_energy_averages.append(num_particles * analytical_energy_value / Z)
    plt.loglog(plots_temperatures, analytical_energy_averages)
    #-------------------------------------------------------Figure2 config
    fig.suptitle(r"$\langle E\left(T\right)\rangle \mathrm{\,values:\,}n=" +
                 str(num_particles) + r"\mathrm{,\,Trials}=" +
                 str(num_trials) + r"$", fontsize=18)
    plt.ylabel(r"$\langle E\left( T\right) \rangle \mathrm{/h\nu }" +
                 r"$", fontsize=16)
    plt.xlabel(r"$\mathrm{Temperature\,(K/h\nu )}$", fontsize=16)
    plt.tick_params(axis='both', which='major', labelsize=6)
    plt.grid(axis="both", which='major', alpha=0.25, linestyle="-")
    plt.grid(axis="both", which='minor', alpha=0.10, linestyle="-")
    plt.savefig("e_averages_" + str(num_particles) + "_particles.svg")
    plt.savefig("e_averages_" + str(num_particles) + "_particles.png")
    plt.savefig("e_averages_" + str(num_particles) + "_particles.pdf")
    log("Finished")
    log(strftime("%Y-%m-%d %H:%M:%S"))
    #-------------------------------------------------------Figure2 config

if __name__ == "__main__":
    parser = argparse.ArgumentParser(
        description='Does the Metropolis algorithm and exports plots.')
    parser.add_argument("-n", "--num-trials", default=5000,
        help="Number of times to randomly alter energy")
```

```
190     parser.add_argument("-p", "--num-particles", default=20,
191         help="Number of particles in the macrostate (system)")
192     parser.add_argument("-R", "--num-runs", default=5,
193         help="Number of Monte Carlo simulations to do (recommend 3-5+")
194     args_dict = vars(parser.parse_args())
195     main(args_dict)
```