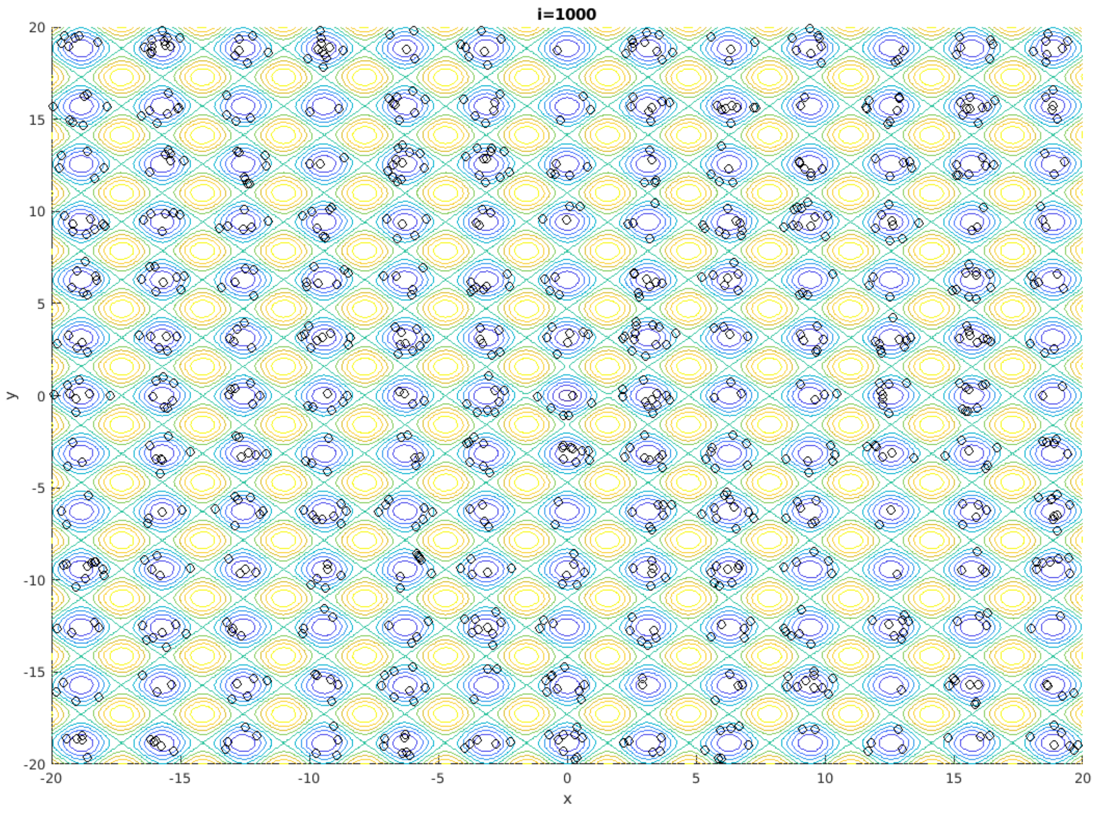
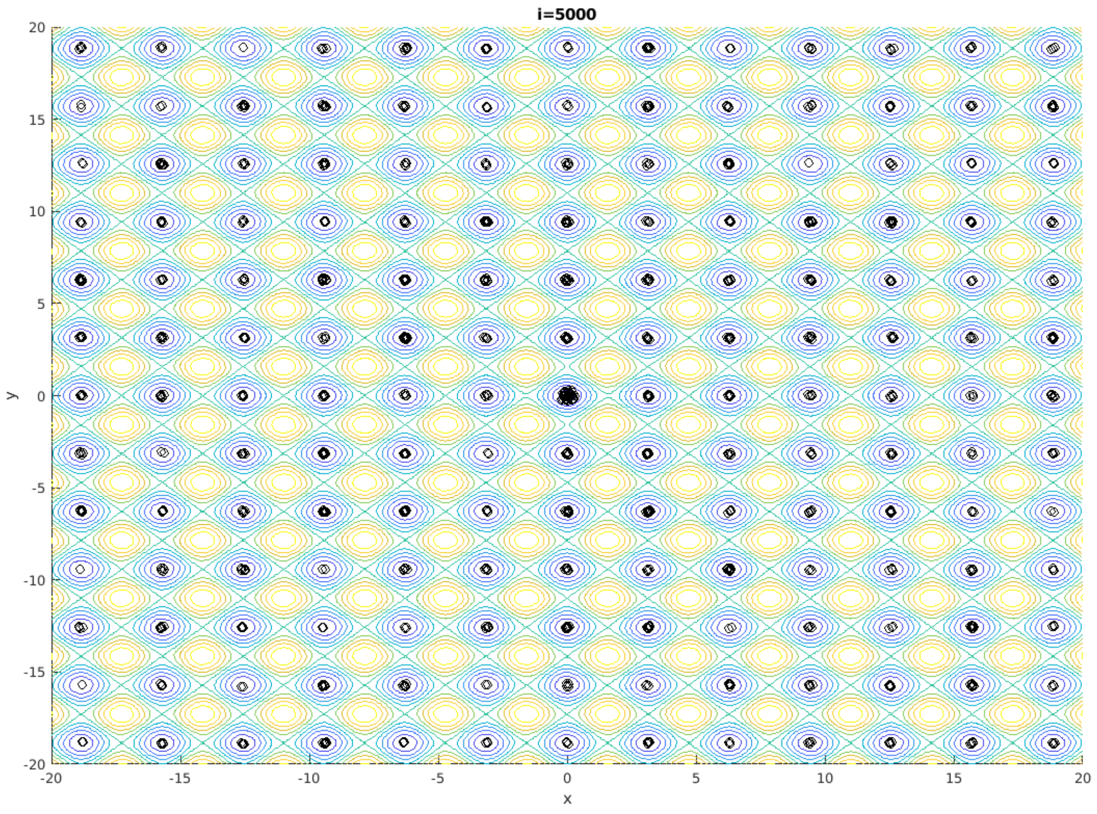
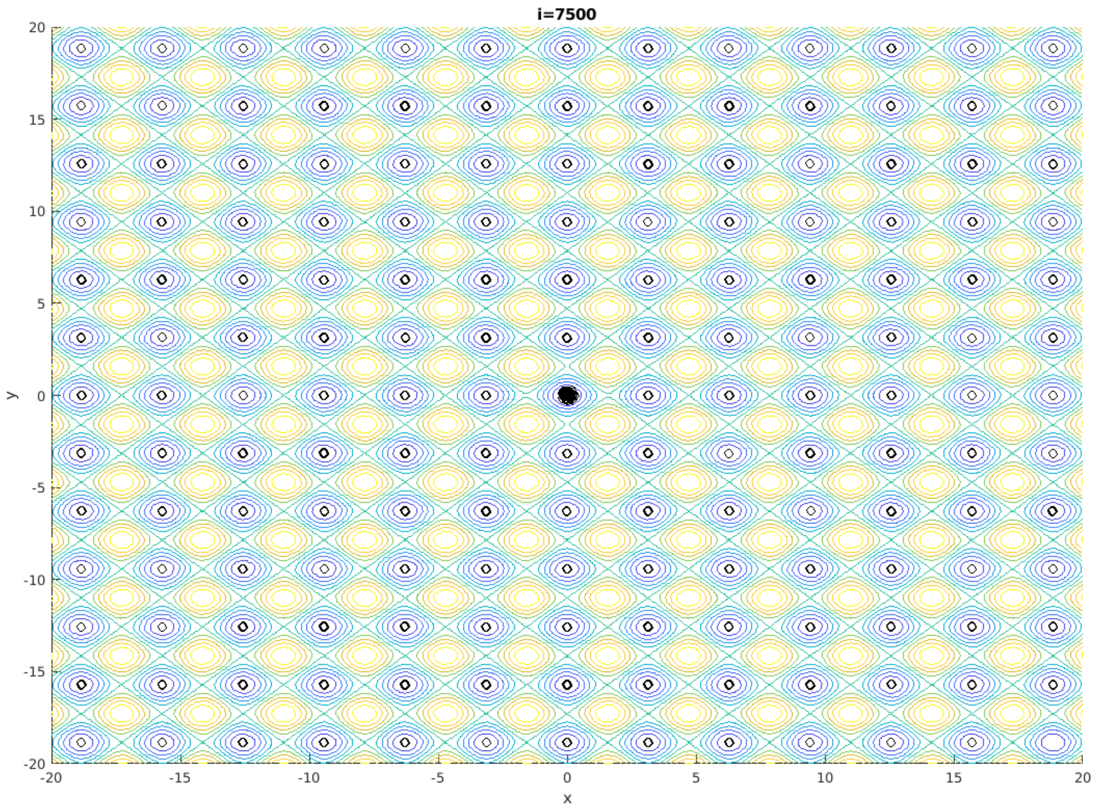
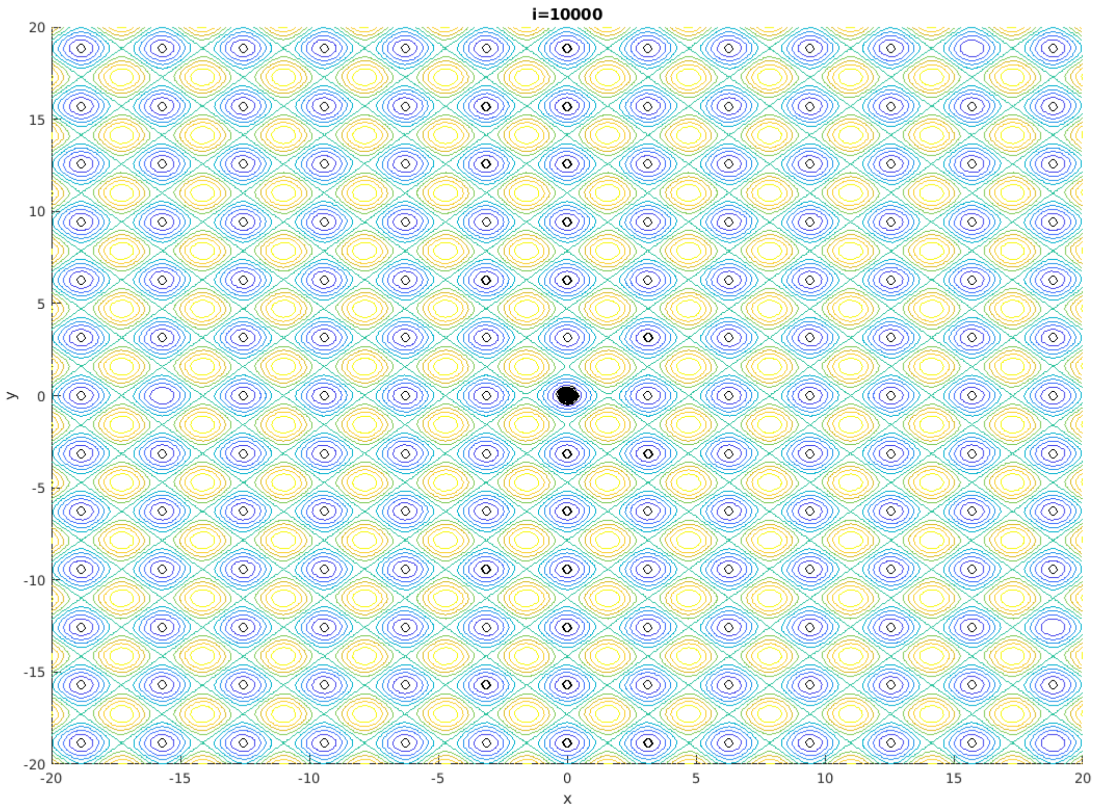
Lab VIII

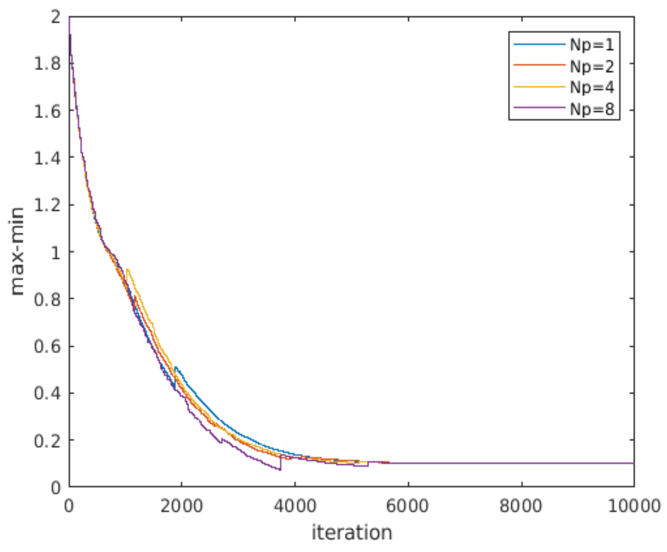


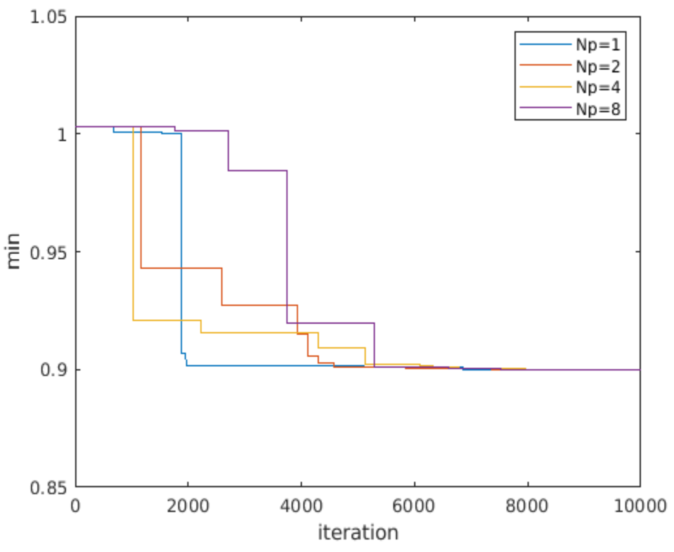



i=7500
y
x
20
15
10
5
0
-5
-10
-15
-20


i=10000
y
x
20
15
10
5
0
-5
-10
-15
-20

It is clear from the above graphs that the algorithm worked. At 1000 iteration, it can be seen that the points were forming clusters around local minimum in their proximity. By 5000 iteration, almost all the points have reached a minimum, be it local or global. The rest of the process was spent on “escaping” the local minimum and converging to the global minimum at (0, 0).





N_P	T	T_O	T_S
1	0.078		
2	0.062	0.046	1.26
4	0.053	0.134	1.51
8	0.154	1.178	0.48

Due to the stochastic nature of the algorithm, analysis of the effect of parallelization requires fixing the seed of random number generators. The above graphs were produced with the seed value of 100. The convergence rate of the difference between maximum and minimum of the cloud is quite similar between the given number of processors. The minimum by itself, however, behaved very differently when the algorithm was parallelized. According to the graph, the minimum took more steps to reach convergence or, in other words, the convergence is more gradual when parallelization is utilized. This, if valid, makes sense because more processes means more trial points will be generated based on similar (unimproved) cloud state.

Regardless, the final results were consistent among the given set of number of processes (and most likely any other). The expected difference lies in the computational performance. As can be seen from the above table, the runtime was reduced as more cores were used. The results for N_P=8 was an outlier with considerably longer runtime. This most likely stems from fact that the computer carrying out the process only have 6 processors. Perhaps more computationally intensive cost function would increase the reduction of runtime since the parallelization focuses on the evaluation of the points in the cloud.