Project #2 – Particle Swarm Optimization Example

Name: Samuel Weems

Course: CMPS 5463 Fall 2018

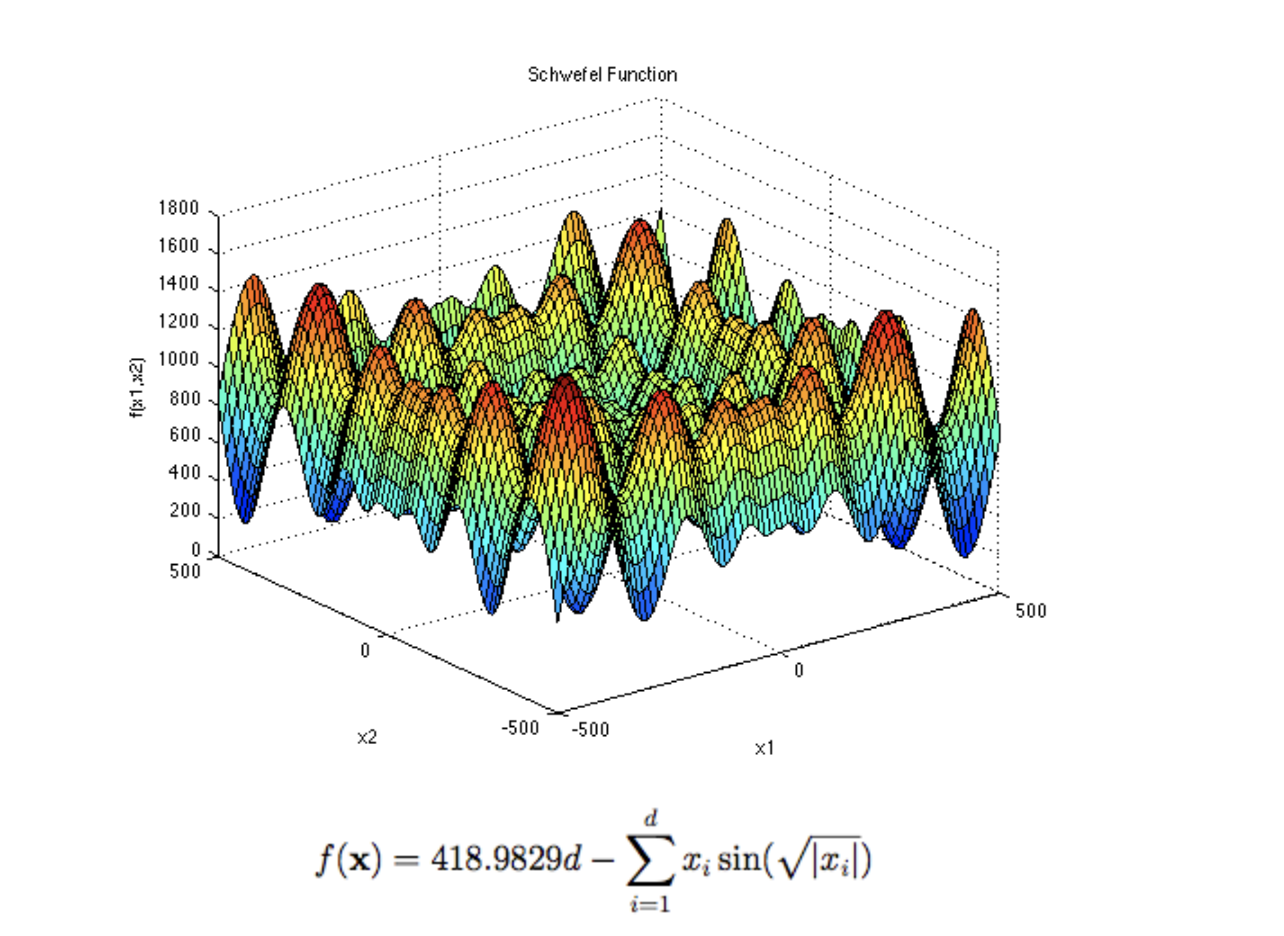
Professor: Dr. Richard P. Simpson

Date: October 08, 2018

1. Introduction to program

In this project we wrote a particle swarm algorithm to optimize the Schwefel function. A full description of the function can be found at https://www.sfu.ca/~ssurjano/schwef.html and the function is shown below in figure 1. The answer to this complex function such that f(x) = 0 is located at ~420.9687 for each dimension. For example, using 2 dimensions the value for each dimension x and y would be 420.9687 or (420.9687, 420.9687) that the function evaluates to 0. The program creates particles for the set number of dimensions with random location values ranging from -500 to 500 and random velocities in the range of -1 to 1. These particles are updated each iteration or generation by calculating new velocities and locations based off a particle swarm algorithm that weights the particle’s best location or fitness and the population’s overall best location or fitness. The program sets a maximum and minimum velocity range (20) and a location range (+/- 500) to keep particles within an acceptable range. The random values for this program are created using new C++ engines and distribution methods. The best fitnesses or locations are defined as those that yield a solution to the function closest to 0.

Figure 1: Schwefel Function



The program’s overall complexity is based on the most complex section of code which is the reoccurring nested looping that goes through each generation, particle, and dimensions of those particles. The more generations, particles, and dimensions of those particles, the more time complexity increases. If one of those factors is increased, for example increasing the number of particles each run, then the complexity would only increase by O(n). If two of these factors are increased each time then the complexity would increase by O(n2). If all three factors are increased each time then the program has a complexity of O(n3).

1. Discussion

After running this program a number of times with different initial conditions, I found that for 1,000 iterations/ generations and 2 dimensions that setting the number of particles to 40 and the other assumptions described in the introduction, was sufficient for most runs to come close to the solution. I ran the program on 1 dimension and found that only 20 particles are sufficient for finding the solution rather quickly. The graphs and charts below show the results of the global best fitness and locations for each of the 1000 iterations/generations. The global fitness improves over time in each run, some runs come closer than others to the optimal fitness or values based on particle initialization.

In the first run the global location quickly narrows down and finds values very close to the desired dimensional values. The majority of the iterations are smaller increments getting closer to the optimal fitness values. The corresponding graph of locations for run 1 shows the first best locations found to be close to the appropriate X1 value but far from the appropriate X2(or Y) value but that most of the iterations/generations are spent finding locations very close to one another and the final optimal location.

Runs 2 and 3 performed very similarly only the initial global best fitnesses began at different values.

In run 4, the algorithm focused in on a local minimum near (426.686, -278.777). As seen in the function description in the introduction section. There are a number of local minima. In this run the particles found this local minimum rather than any particle find the global minimum. Even with more iterations this particular run would return the same result. Increasing the number of particles and/or adjusting the velocity factors could overcome this to allow for more exploration.

Run 5 had a similar result as run 4, this time finding a different local minimum at (-268.239, 423.079).