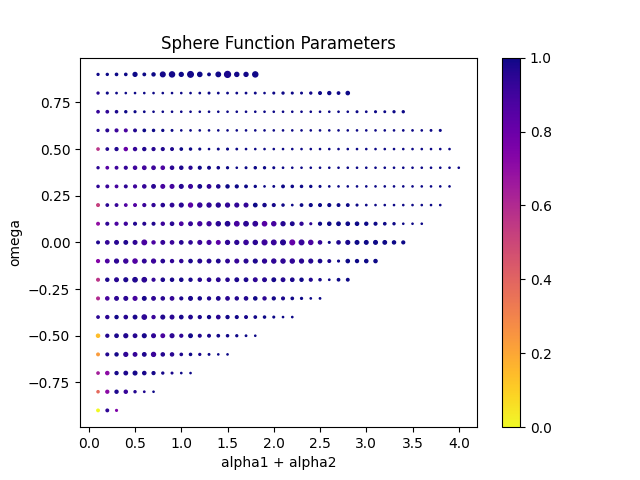
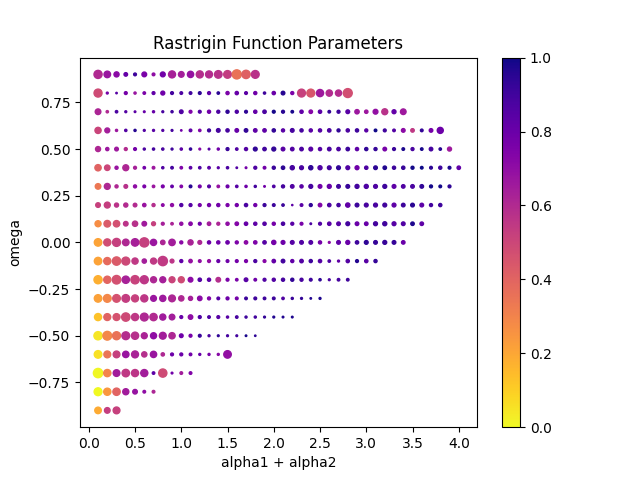
**Analysis of Particle Swarm Optimization**

The algorithm is modified based on the tutorial solution template, it uses an asynchronous way to update the global best, and it memorize the best solution in the history, although the best solution ever does not affect particles’ velocity. Same as the very first PSO experiment, I have chosen 20 as the particle population. To terminate the search, I have used two conditions, one is to detect the convergence, another is the maximum iteration. If the best solution ever is not updated for several iterations, the algorithm will terminate the search, as it might have converged, and the default number of iteration is 10 for question 1. Moreover, the algorithm will be forced to stop after iterated for too many times, even it is not converged, as otherwise the cost for parameter search would be too expensive. The dimensionality I have chosen is 6, as I found it helps with the analysis (the algorithm always succeeds or fails when the dimensionality is too low or too high).

For sphere function, any parameter should be feasible, as the function has only one local optimum, that is the global optimum. As long as particles have ability to exploit, the algorithm will find a good solution. To speed up the search, we let particles move more aggressively towards a better solution, so should be small but and are greater and should smaller than 1, otherwise it will overshoot. For the Rastrigin function, there are many local optima, therefore we want the algorithm to have a good balance between exploration and exploitation, in other words, and we want or and to be large to overcome the local optima.

We want the algorithm to converge, so we want to try parameters that are inside the parabolic region [tutorial 2, Poli‘s analysis]. I have set =. To run the experiment, I have selected parameters within the parabolic curve, run the algorithm with these parameters, repeated and recorded the average number of iterations (which is equivalent to the running time given a fixed swarm size and dimensionality) and the average fitness. The plots for experiment result are placed in the appendix, each dot in the plot is a choice of parameter, **the size/area of the dot is inverse proportional to the algorithm’s running time with that parameter configuration, and the color of the dot is related to the fitness. Large, dark blue dots indicate good parameters.** Examples are for the sphere function, .2 for the Rastrigin function. My experiment result is consistent with the argument in the previous paragraph, better parameters for the Rastrigin function is clustered closer to the top right side (more zigzag behavior) compared to good parameters for sphere function (more exploit).

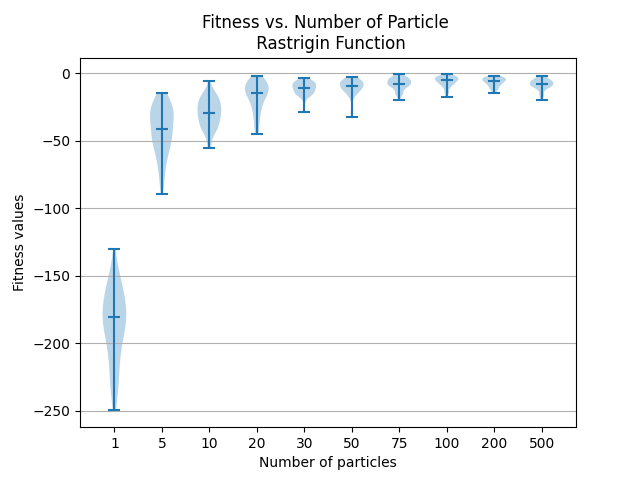
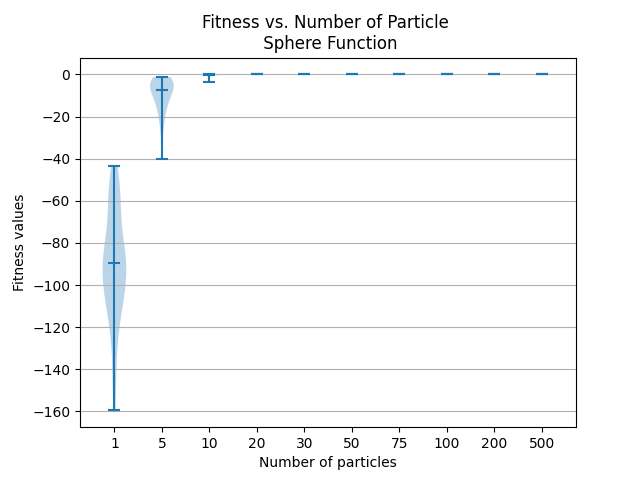


**Scaling**

In the first question, I have used 20 as the particle population size, as in the first PSO paper, authors have used 20 in their experiment. I have chosen the number of particles N as an additional parameter to investigate in this question. Some may argue that PSO with more particles tends to find better solution within less iterations, I think this way of evaluation is not fair, since with more particles, the algorithm will search more potential solutions in each iteration, and the computational cost of each iteration increases. In real world tasks, we usually have limited time to find answers, rather than limited iterations.

To ensure the fairness of the experiment and evaluation, I have added an additional PSO argument to restrict the time for each run, I have used 1 second for this section, as it is sufficient for PSO with 20 particles. The set of parameters is chosen based on the result of question 1, different from the previous question, I have selected parameters that yield the top 10 best results, ignoring the number iterations, since time is a hard limit in this section. The set of particle population sizes are {1, 5, 10, 20, 30, 50, 75, 100, 200, 500}. With each population size, and each choice of parameters, the algorithm run 5 times and the average fitness value of the final output was recorded.

The violin plot shows the fitness value vs. number of particles, each violin has 50 data samples (10 parameters times 5 repeats), the middle line in the violin plot is median, the upper and lower lines are the maximum and minimum fitness, no data point was rejected as outlier. According to the result, given a fixed amount of time, PSO algorithms with more than 20 particles tends to perform equally well. But if the particle size is too small, the algorithm is very likely to stuck in a random local optimum, that is also explains the variance in the fitness. The fitness is the negation of sphere and the Rastrigin functions, and 0 is the highest possible fitness for both functions. The experiment for the graph was carried out with dimensionality of 6. I have also tested the cases which the time is too short using a dimensionality of 18, time restriction of 0.2 second, that graph also shows the same trend except the average fitness is much lower. As a conclusion for this section, in practical if we have limited time, without any additional knowledge of the problem, choosing any number of particles greater than 20 would make the algorithm perform equally well.



**Heterogeneous Particle Swarm**

The algorithm that I have implemented is a static HPSO, which means each set of particles are assigned to a behavior which will never change during the running. Theoretically, HPSO can usually utilize the background knowledge better, since there are particles with two different behaviors. However, a drawback of the algorithm is the cost of grid search for finding the most optima parameter set, the search space for parameters is squared compared to standard PSO. Therefore, I am focusing on finding good parameters and trying to explain why they are good, in stead of finding the best set of parameters. All other function arguments are the same as those in the experiment for question 1, I have set = for both set of particles.

The parameter search is done by brute force, potentialvalues range from -0.9 to 0.9, with the interval of 0.3, and potential + values range from 0 to 3.5, with the interval of 0.5 between each two successive values. For each set of parameters, experiment is repeated 5 times and the average is calculated. Below is a table for example parameters with top results.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Parameters of Swarm 1  ( | Parameters of Swarm 2  ( | Average Fitness  (-1 \* Rastrigin Function) |
| 1 | (0.3, 1.0, 1.0) | (-0.3, 1.5, 1.5) | -1.393 |
| 2 | (0.9, 1.0, 1.0) | (-0.3, 1.25, 1.25) | -1.751 |
| 3 | (0.9, 0.75, 0.75) | (0, 1.5, 1.5) | -2.587 |
| 4 | (-0.3, 1.25, 1.25) | (0.9, 1.0, 1.0) | -2.660 |
| 5 | (-0.3, 1.5, 1.5) | (0, 1.25, 1.25) | -2.773 |
| 6 | (0, 1.5, 1.5) | (0.9, 1.0, 1.0) | -2.787 |
| 7 | (0.9, 0.75, 0.75) | (0.3, 1.25, 1.25) | -2.985 |
| 8 | (-0.3, 1.5, 1.5) | (0.3, 1.0, 1.0) | -3.050 |
| 9 | (0.9, 0.75, 0.75) | (0, 1.25, 1.25) | -3.385 |
| 10 | (-0.3, 1.5, 1.5) | (0.6, 0.5, 0.5) | -3.408 |

There are two types of combination: one is the attraction combined with repulsion; another a parameter set from top right part of the graph for question 1 combined with a parameter set that makes particles to move aggressively towards a better solution (small and large ). The first combination allows some particles to move away from the local optima, thus usually yield a better solution. There are other possible parameters like combination of larger , smaller with smaller , larger , if we allow and to be different. To compare with standard PSO, the best parameter yields an average fitness of -2.587, and the 10th best parameter yields an average fitness -3.573. For every two parameter sets with the same rank, static HPSO is always better (top 10 results are compared). We can say that static HPSO slightly outperform standard PSO on the Rastrigin function, when dimensionality is 6, swarm size is 20, maximum number of iterations is 1000. There is a study shows that standard PSO may outperform static HPSO on the Rastrigin function, when the dimensionality is high, but 6 is considered as a low dimensionality.

**Differential Evolution**

My implementation is modified based on the PSO for previous questions, thus they have the same termination strategy, the algorithm’s behavior should be similar to the code given on the lecture slide. DE is similar to the GA algorithm, as it treats each dimension separately. If we abstract the Rastrigin function’s ultimate goal, we want to set all dimensions of x equals to 0, which share a same characteristic as the all-ones problem (except it is all-zeros), DE is expected to perform better than the standard PSO on the Rastrigin function, as DE is likely to remember the best value for each dimensionality. Considering the case where a particle has one dimension equals 0, but all other dimensions equal to 5.12, and a better solution is found at a local optimum, which has no zero value for any dimension, the particle in a standard PSO is likely to be attracted towards the local optimum, and the value 0 in that dimension will be forgotten. However, in DE, that 0 is likely to be memorized, as it could be crossovered with the local optimum.

To ensure the fairness of evaluation, I have also restricted the maximum number of iterations to 1000, number of particles are set to 20, thus two algorithms could have searched the same number of positions before terminating, in other words, the total number of fitness evaluation T per run is fixed. The dimensionality of the space is 6 for both algorithms. Due to the nature of the DE algorithm, it is likely that the algorithm takes more iterations to find a better value, thus the number of iterations for convergence detections should be greater (I have tested 10, the algorithm yields very poor result, and I have also tested PSO with a greater number, but the result barely changes). Therefore, I decided to use 100, if no better solution is found in 100 iterations, the DE algorithm will be stoped.

For parameter selection, I have used brute force grid search, the amplification factor ranges from 0 to 2, and the crossover probability ranges from 0 to 1. The result for DE is surprisingly better than the result for PSO, almost all parameter settings have reached the average fitness of 0 (which is the highest possible fitness), even the parameter with the lowest fitness yields a fitness of -0.4, that is greater than the best performing PSO (-2.587). To further test the capability of the DE algorithm, I have increased the dimensionality to 10, the average result of all parameters is -3.676, which is already considered as a top result in PSO for dimension 6. For conclusion, the DE algorithm is indeed better than standard PSO on solving the Rastrigin function.

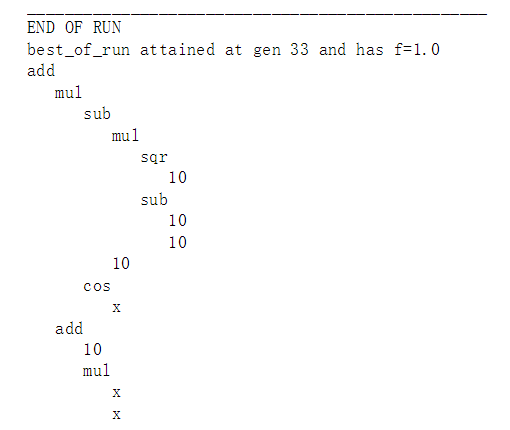
**Genetic Programming**

I have made minor change on the tutorial example GP algorithm and used it directly for this question. The change I have added some code to allow it to use non-terminals with only one child, since we need to compute the cos of a value. I think reproducing the Rastrigin function using a vanilla GP algorithm is nearly impossible, I have tried to write the function’s abstract syntax tree (AST), and an AST that is equivalent to the function has a minimum depth of 8, which could be too deeper for GP algorithm.

The fitness function is the normalized inverse mean absolute error, which is provided in the tutorial solution example. I have firstly tried to use a set of function consists of { add(x, y), subtract(x, y), square(x), cos(x), mul(x, y) }, set of terminals consists of {x, 10, 2, d, pi}. Any parameters remained unchanged. Since each run takes too long, it would be unfeasible to tune parameters with grid search. The database I have generated is x range from -5 to 5, with interval of 0.1, I have used a one-dimensional Rastrigin function. After running the algorithm for several times, I found the representation of the best fitted program is unreadable, for example, there are useless subtrees like +(-(-(2+2+2+2+2-10))), a more sophisticated GP algorithm should avoid and eliminate this kind of subtrees. The average best fitness in these runs is just above 0.1 (the fitness is normalized to [0,1]).

The result of the GP algorithm is heavily dependent on the dataset for computing fitness. If we use a dummy dataset, where x is always an integer value, and the dimension is 1. The Rastrigin function would be equal to square(x) in this case, the GP algorithm will find equations that are equivalent to square(x) within the first few generations and the fitness is 1.

To reduce the difficulty of the problem, I have decided to simplify the syntax. By replacing cos(x) with cos(2\*pi\*x), and remove 2, d, pi from the set of terminals, since they are no longer necessary. To reduce the running time, I have set the population size to 36, maximum number of generations to 50. The dataset is computed using x range from -5 to 5, with the interval of 0.6 (if x is multiplied of 0.5, cos(2\*pi\*x) will always be 1 or -1). I have run the algorithm with this configuration repeatedly, the fitness value of these functions is usually between 0.4 and 0.1, but in one run, the algorithm yields the desired function.



Since **cos x** in the output is **cos(2\*pi\*x)**, the node at the bottom is at the bottom right of the syntax tree. After simplification, the syntax tree is equal to the Rastrigin function.

For conclusion, reproducing a function using the GP algorithm is not a trivial task, this one-dimensional function that only takes one argument is already challenging to reproduce. Without knowing this function at the beginning, I would not know how to simplify it, or even cannot provide a representative dataset. In practice, the GP is usually used for unknown functions, with higher dimension and more input arguments.