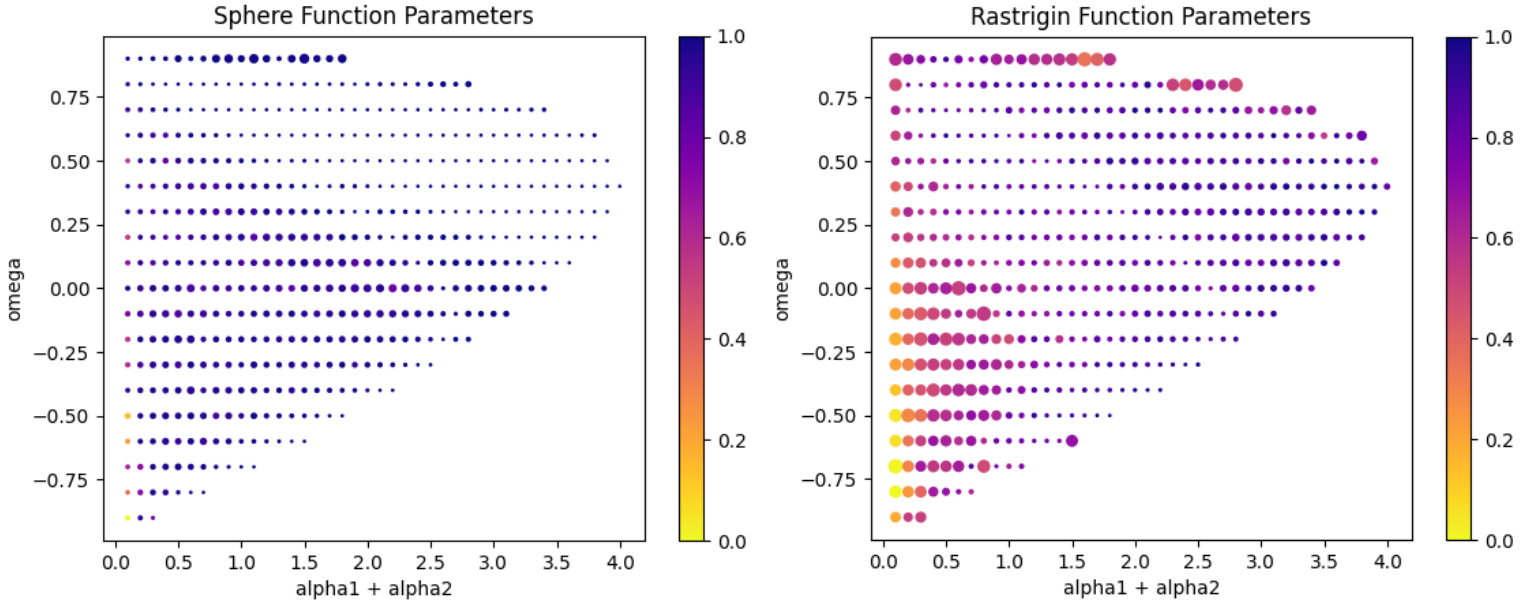
Analysis of Particle Swarm Optimization

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] **[i]**, I have chosen 20 as the swarm size. 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 **the sphere function**, most parameters should be feasible, as the function has only one local optimum. As long as particles have ability to exploit, the algorithm will find a good solution. **To speed up the search, we can let particles move more aggressively towards a better solution, so ω should be small but $\alpha 1$ and $\alpha 2$ are greater. $\alpha 1$ and $\alpha 2$ 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 ω or $\alpha 1$ and $\alpha 2$ to be large to overcome the local optima.**

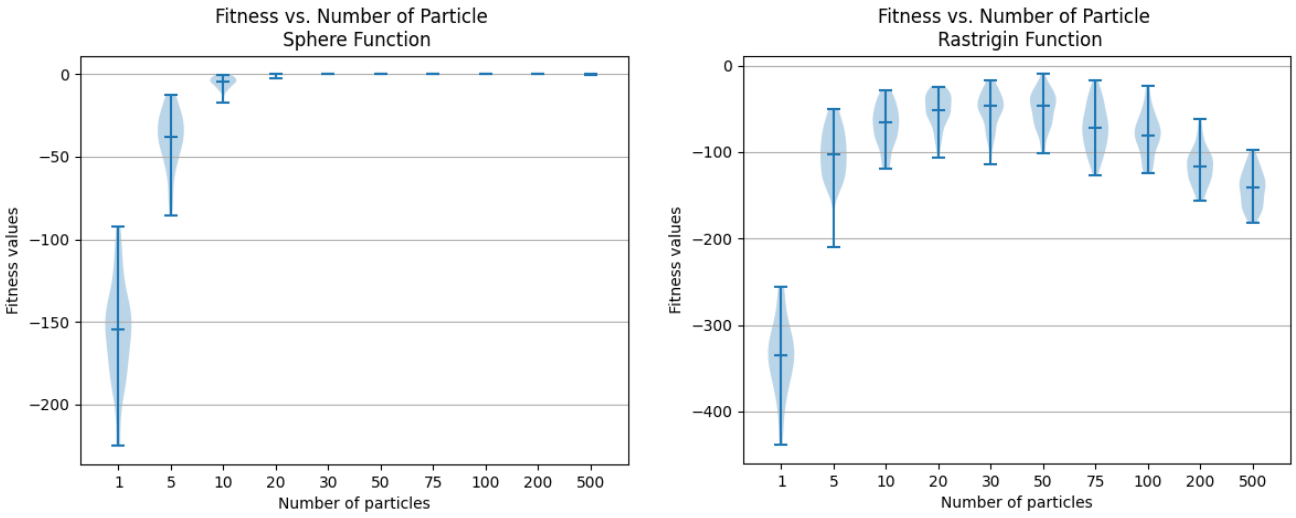
We want the algorithm to converge, so we want to try parameters that are inside the parabolic region [Poli's analysis][ii] **[ii]**. I have set $\alpha 1$=$\alpha 2$. To run the experiment, I have selected parameters within the parabolic curve, run the algorithm with these parameters, repeated and recorded the average number of iterations (which is equivalent to the running time given a fixed swarm size and dimensionality) and the average fitness. 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 $\omega = 0, \alpha 1 = \alpha 2 = 1$ for the sphere function, $\omega = 0.4, \alpha 1 = \alpha 2 = 1.2$ for the Rastrigin function. My experiment result is consistent with the argument in the previous paragraph, better parameters for the Rastrigin function is clustered at to the top right part of the graph, but good parameters for sphere function is clustered at the center of the graph.



Scaling

In the first question, I have used 20 as the particle population size. I have chosen the number of particles N as the 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, the algorithm will not stop searching even it is converged until timeout.**

To ensure the fairness of the experiment and evaluation, I have used a timer to terminate each run, time I used is **0.2 second**, as it is insufficient for PSO with 20 particles to find good results for the Rastrigin function with. ω, $\alpha 1$ and $\alpha 2$ are 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 the hard limit in this section. The set of particle population sizes are {1, 5, 10, 20, 30, 50, 75, 100, 200, 500}. With each configuration parameters, the algorithm run 5 times and the average fitness was recorded.



The violin plot shows the fitness value vs. number of particles, each violin has 50 data samples (10 parameters, 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. The fitness is the negation of sphere and the Rastrigin functions, and 0 is the highest possible fitness for both functions. According to the result, given a very limited amount of time, if the algorithm with too many particles performs worse. And if the swarm size is too small, the algorithm is very likely to stuck in a random local optimum. **I have also tested the cases which the time sufficient, using dimension 10, time restriction of 1 second**, the graph (In Appendix) for both functions shows that the algorithm tends to perform equally well if the swarm size is greater than 20.

As a conclusion for this section, in practical **if we have a limited running time**, without any additional background knowledge of the problem, **we could try the algorithm with swarm size between 20-50 first**, with sufficient running time, there is also no need to use very big swarm size first.

Heterogeneous Particle Swarm

The algorithm that I have implemented is a static HPSO[iii] **[iii]**, 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 $\alpha 1$=$\alpha 2$ for both set of particles.

The parameter search is done by brute force, potential ω values range from -0.9 to 0.9, with the interval of 0.3, and potential $\alpha 1$+$\alpha 2$ 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 $(\omega, \alpha 1, \alpha 2)$	Parameters of Swarm 2 $(\omega, \alpha 1, \alpha 2)$	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**, as they allow some particles to move away from the local optima, thus usually yield a better solution; another is **parameters from the top right part of the parabolic region, combined with parameters with small ω and large α** (particles move aggressively towards a better solution), as they have balanced well between the exploration and the exploitation.

There are other possible parameters like combination of larger $\alpha 1$, smaller $\alpha 2$ with smaller $\alpha 1$, larger $\alpha 2$ (social behavior with cognitive behavior), if we allow $\alpha 1$ and $\alpha 2$ to be different. To compare with standard PSO, the best parameter of PSO 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 6-dimensional Rastrigin function, when the swarm size is 20, maximum number of iterations is 1000. The study[iv] **[iv]** 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 pseudocode 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 similar 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 dimension**. Considering the case where a particle has a dimension equal to 0, but all other dimensions equal to 5.12, when 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 usually 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 stopped.

For parameter selection, I have used the 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)**, the lowest average fitness is -0.3980, 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 two changes on the tutorial example GP algorithm and used it directly for this question. One change it adding some code to allow the algorithm to use non-terminals with only one child (for cosine and square functions), another change is adding the roulette selection. The tree depth of the one-dimensional Rastrigin function is 8, so it can be challenging to reproduce.

The fitness function is the normalized inverse mean absolute error. I have firstly tried to use a set of **function {add, subtract, square, cos, multiply}**, set of **terminals {x, 10, 2, d, pi}**. **Population size is 120**, **maximum number of generations is 100**. 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** (which is low), there is no significant difference between using the standard selection and the roulette selection.

The result of the GP algorithm can be 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 equal to square(x) in terms of their outputs, the GP algorithm will find equations that are equivalent to square(x) within the first few generations and the fitness is 1. However, if we use a huge dataset, the GP algorithm may still perform poorly, as the fitness value for all trees except the correct one can be very low, and the algorithm cannot reliably select better parents to produce offspring. The fitness function matters as well, if I use the absolute error ratio (absolute (gp_output – expected_output) / expected_output) instead of the absolute error, the output function is usually square(x), as it has a similar general trend with the Rastrigin function, if we zoom out and observe the plot for both functions.

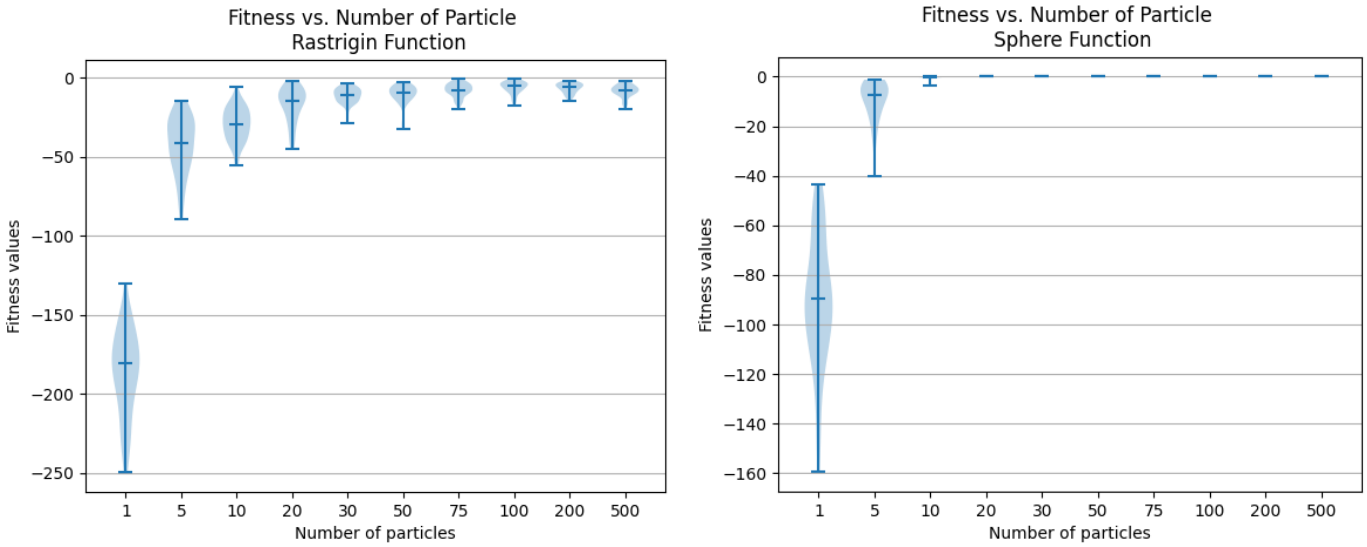
Inspired by the concept of sketching with formal synthesis[v] **[v]**, I decided to **simplify the problem by taking two steps**. The first goal of the algorithm is to find cos(2*pi*x), it repeats until the fitness reaches 1, and in the next step I replaced cos(x) with cos(2*pi*x), and remove 2, d, pi from the set of terminals. 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. I have run the algorithm with this configuration repeatedly, the algorithm yields the desired function once, even though the fitness of the output is usually between 0.1 and 0.4 in these repeated runs. Since **cos x** in the screenshot is **cos(2*pi*x)**, After simplification, the syntax tree in the figure is equal to the Rastrigin function.

```
------------------------------------------
END OF RUN
best_of_run attained at gen 21 and has f=1.0
add
   sqr
      sub
         x
         x
   add
      sub
         sqr
            x
         mul
            cos
               x
            10
      10
```

Screenshot of the Output When the Function is Found

For conclusion, 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, reproducing a function using the GP algorithm is not a trivial task.

Appendix



Fitness value vs. swarm size, when the dimension is 10, maximum running time is 1 second.

[i] J. Kennedy and R. Eberhart, "Particle swarm optimization," Proceedings of ICNN'95 - International Conference on Neural Networks, 1995, pp. 1942-1948 vol.4, doi: 10.1109/ICNN.1995.488968.

[ii] Harrison, Kyle & Ombuki-Berman, Beatrice & Engelbrecht, Andries. (2017). Optimal parameter regions for particle swarm optimization algorithms. 349-356. 10.1109/CEC.2017.7969333.

[iii] Engelbrecht, A.P. (2010). Heterogeneous Particle Swarm Optimization. In: , *et al.* Swarm Intelligence. ANTS 2010. Lecture Notes in Computer Science, vol 6234. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-15461-4_17

[iv] Engelbrecht, A.P. (2010). Heterogeneous Particle Swarm Optimization. In: , *et al.* Swarm Intelligence. ANTS 2010. Lecture Notes in Computer Science, vol 6234. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-15461-4_17

[v] Armando Solar-Lezama, Rodric Rabbah, Rastislav Bodík, and Kemal Ebcioğlu. 2005. Programming by sketching for bit-streaming programs. SIGPLAN Not. 40, 6 (June 2005), 281–294. https://doi.org/10.1145/1064978.1065045