

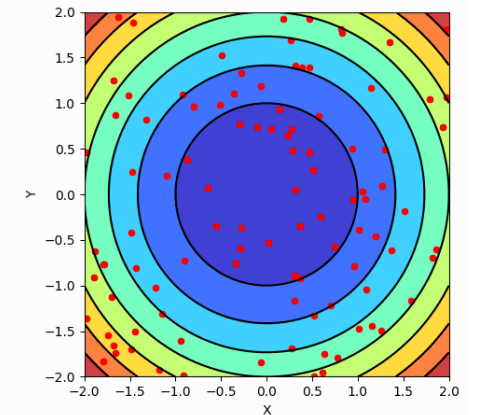
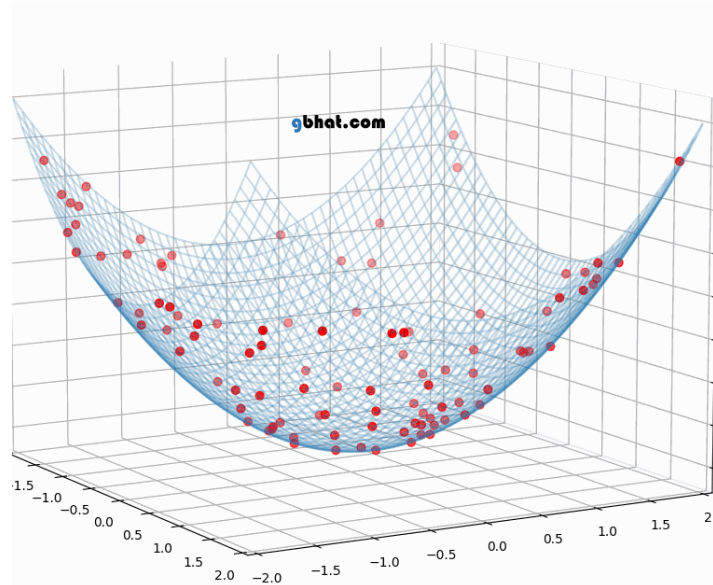
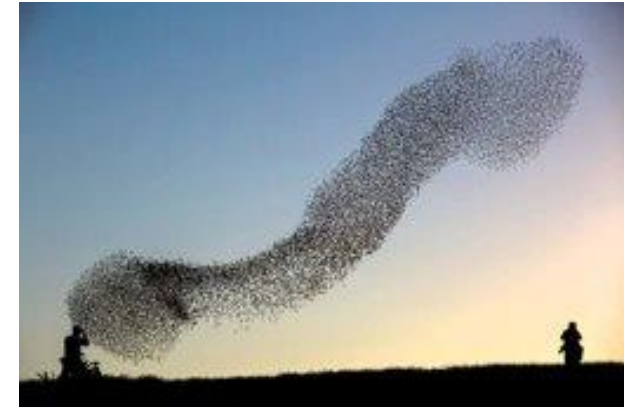
Particle Swarm Optimization

Aya Arabiat



Particle Swarm Optimization

- PSO is a global optimization technique inspired by the social behavior of bird flocking or fish schooling.
- PSO operates by simulating the behavior of a swarm of particles moving in a multi-dimensional search space.



Particle Swarm Optimization

- Unlike gradient-based methods, PSO and similar evolutionary algorithms do not require derivatives of the objective function, making them suitable for:
 - Highly nonlinear functions.
 - Global extremum finding.
 - Black-box functions, where internal workings are not accessible.

Particle Swarm Optimization

- We initialize particle positions at random.
- Each particle has a position in the search space and a velocity, as well as a fitness value.
- We have two nested loops, one goes through all particles, and then the outer one moves the iteration.

$$P_i^{t+1} = P_i^t + V_i^{t+1}$$

$$V_i^{t+1} = \underbrace{wV_i^t}_{\text{Inertia}} + \underbrace{c_1 r_1 (P_{best(i)}^t - P_i^t)}_{\text{Cognitive (Personal)}} + \underbrace{c_2 r_2 (P_{bestglobal}^t - P_i^t)}_{\text{Social (Global)}}$$

- c_1 is the cognitive (personal) coefficient and c_2 is the social (global) coefficient.
- r_1 and r_2 are random values uniformly distributed between 0 and 1, so they can explore the space
- P_{tbest} is the best position found by the particle up to iteration t .
- $P_{bestglobal}^t$ is the global best position found by any particle up to iteration t .

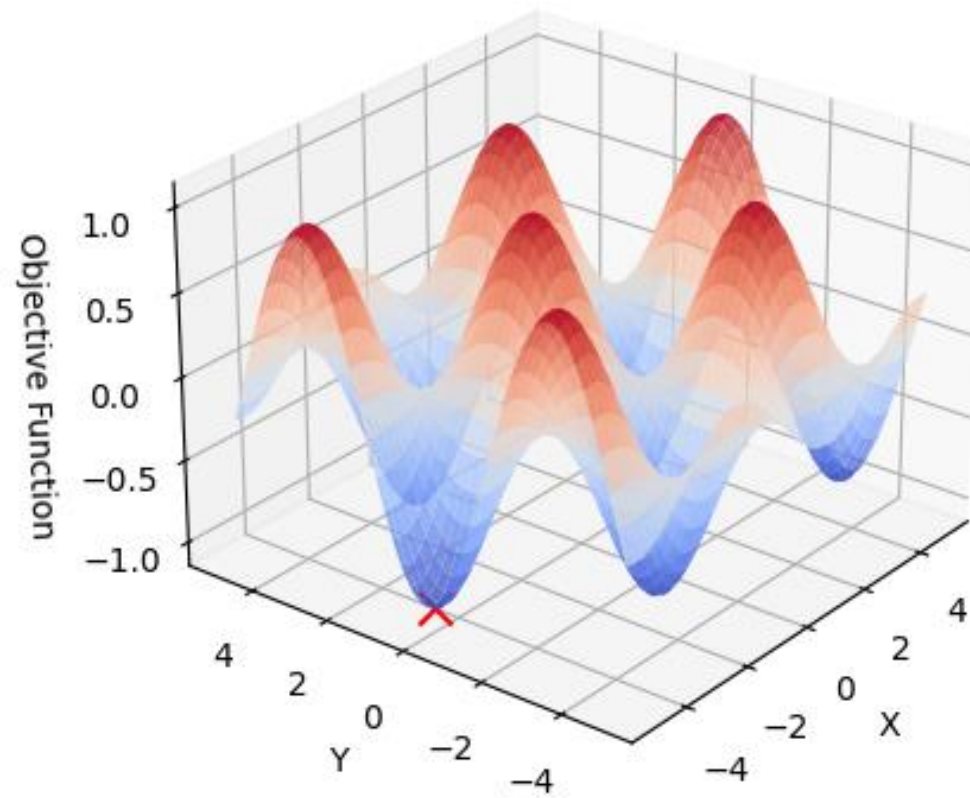
Updating the fitness

```
if particle_fitness < particle_best_fitness:  
    particle_best_fitness = particle_fitness  
    particle_best_position = particle_position  
return particle
```

- Then we update the state of the particle again according to new global best, and so on until convergence.

Test Function

PSO Optimization of 2D Function



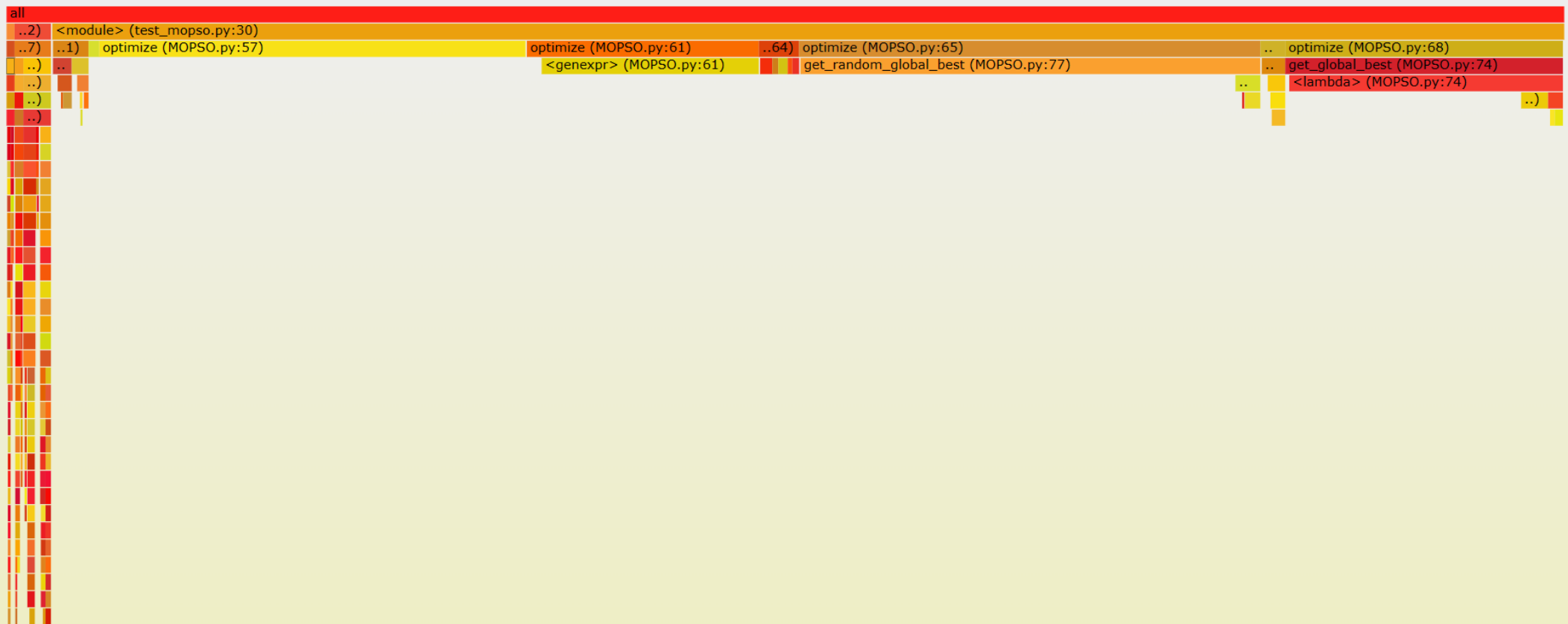
$$f(x_1, x_2) = -\sin(x_1)\cos(x_2)$$

Implementation

- I used the principles as follows; code should be:
 - Easy to develop: formatting, version control, automating tests, well-documented, modular, pythonic, etc.
 - Correct: testing suites like unittest and py-test.
 - Fast: After the code passes tests, we can move on to profiling then finally optimizing!

Profiling

- py-spy: optimize() function takes the most time




The Goal

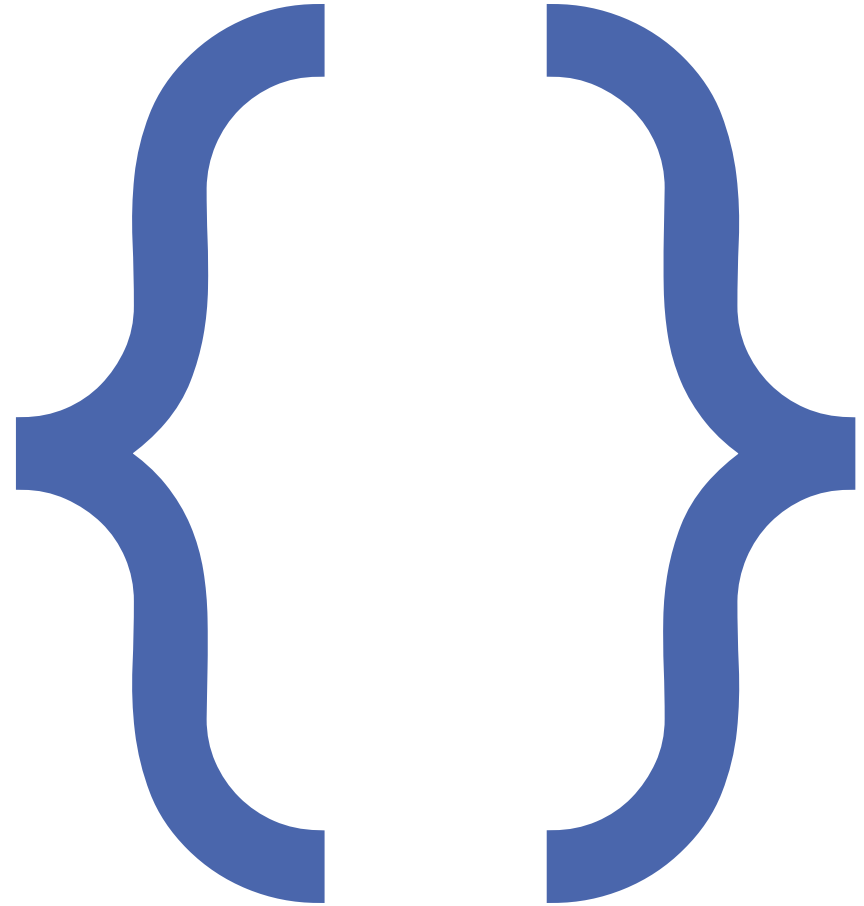
Parallelize the `optimize()` function as much as possible

Updating for each iteration

- The step of comparing all particles with the global best is hard to parallelize but may benefit from improving the data structure or using asynchronous methods.



Multiprocessing in Python (On Code)



Results

- Serial Execution Time: 1.3985
- CPU Usage: 3.3
- Memory Usage GB: 0.03725
- Parallel Execution Time: 5.76496
- CPU Usage: 33.5
- Memory Usage GB: 0.00225
- Async Execution Time: 32.81444
- CPU Usage: 6.6
- Memory Usage GB: 0.2291

Conclusions

- We did not benefit from any speed ups for the single objective function with two inputs case, due to significant overhead.
- But if the objective function is more complicated (CPU dependent) there might be still be a benefit.
- Furthermore, this algorithm is scalable to more objective functions, I would like to try parallelize the multi-objective case.
- More inputs can be added to test scalability.

Thank you!

https://github.com/Aya-42/PSO_HPC_CLASS
