Instituto Politécnico Nacional
Escuela Superior de Cómputo
Evolutionary Computing
6 Particle Swarm Optimization
Martínez Coronel Brayan Yosafat
Rosas Trigueros Jorge Luis
15/10/2021
22/10/2021

Theorical framework

In this practice, we try to find a function minimum using particle swarm optimization, the functions we are talking about are Ackley and Rastrigin, both are used for optimizations test. But let us talk about them deeper.

Used widely in problem optimization, Ackley function is characterized by a nearly flat outer region, and a large hole at the center. The function poses a risk for optimization algorithms, particularly hill climbing algorithms, to be trapped in one of its many local minima [1].

$$f(x) = -Ae^{-B\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}} - e^{\frac{1}{d}\sum_{i=1}^{d}\cos(Cx_i)}A + e \qquad (1)$$

On the other hand, proposed by Rastrigin in 1974 as a 2-D function for optimization problems and generalized later by Rudolph and then popularized by Hoffmeister & Bäck. This function is hard to find a minimum for two reasons, the domain, and the number of local minima [2].

$$f(x) = An + \sum_{i=1}^{n}[x_i^2 - A\cos(2\pi x_i)] \qquad (2)$$

Now, talking about particle swarm optimization, it is a computational method that optimizes a problem by iteratively trying to improve a candidate solution about a given measure of quality. It solves a problem by having a population of candidate

solutions, here dubbed particles, and moving these particles around in the search-space according to simple mathematical formula over the particle's position and velocity. Each particle's movement is influenced by its local best-known position but is also guided toward the best-known positions in the search-space, which are updated as better positions are found by other particles. This is expected to move the swarm toward the best solutions [3].

Material and equipment

In this practice, DataSpell with Jupiter Notebooks, Conda and Python were used for development, all of them were run in a personal computer which has an Intel Core i7 with Windows 10.

Practice development

Following the framework developed over these practices, problems are solved using Oriented-Object Programming. Now, every optimization contains two class definitions, Swarm and Particle, a function definition, and, of course, iteration and resetting code.

This code is specially made for acting as a wrapper of a given function desired to be minimized. We must observe that, as a metaheuristic, we find some parameters (which we could call them hyperparameters) as follows:

- K_DIMENSION: Number of dimensions
- K_LIMITE_SUPERIOR: Real number, upper limit
- K_LIMITE_INFERIOR: Real number, lower limit
- K_COEFICIENTE_PESO: Weight for inertia force
- K_COEFICIENTE_MEMORIA: Weight for memory of the particle
- K_COEFICIENTE_LIDER: Weight for following lead position
- K_NUM_PARTICULAS: Number of particles

So, we just need to look at the code one time, and understand both optimizations, taking our time talking about both classes in this section. Now this code is intended to be used inside a Jupiter Notebook but putting it in a package would be a lot more organized. The code showed is taken from Ackley optimization.

```
K_LIMITE_SUPERIOR = 33
K_LIMITE_INFERIOR = -33

class Particle:
    def __init__(self):
        dominio = K_LIMITE_SUPERIOR - K_LIMITE_INFERIOR

        self._dimension = K_DIMENSION
        self.coordenadas = K_LIMITE_INFERIOR +
dominio*np.random.random(size=self._dimension)
        self.memoria = np.copy(self.coordenadas)
        self.velocidad = -dominio/2 +
dominio*np.random.random(size=self._dimension)

    def __str__(self):
        return str(self.coordenadas)

    def to_evaluate(self, use_coors=True):
        if use_coors:
            return [np.array(self.coordenadas[dim]) for dim in
range(self._dimension)]
        else:
            return [np.array(self.memoria[dim]) for dim in
range(self._dimension)]

    def get(self, index: int):
        return self.coordenadas[index]
```

Code 1. Particle class definition

Particle class provides utilities for Swarm class, making it easier to manage a population of N particles and M dimensions.

```
from typing import Callable

K_COEFICIENTE_PESO = 0.5
K_COEFICIENTE_MEMORIA = 0.3
K_COEFICIENTE_LIDER = 0.2

class Swarm:
    def __init__(self, num_particulas: int, function: Callable[...,
np.ndarray]):
        self._num_particulas = num_particulas
        self._poblacion = [Particle() for _ in range(num_particulas)]
```

```
        self._coef_peso = 0.5
        self._num_generacion = 0
        self._function = function
        self._init_best()

    def _init_best(self):
        self._mejor : Particle
        salida = self._evaluar()
        index = salida.argmin()
        self._mejor = self._poblacion[index]

    def _evaluar(self) -> np.ndarray:
        coordenadas = self._get_coordenadas()
        return self._function(*coordenadas)

    def _get_coordenadas(self) -> list[np.ndarray]:
        return [np.array([particula.get(dim) for particula in
self._poblacion]) for dim in range(K_DIMENSION)]

    def _actualizar_particula(self, particula: Particle):
        actual =
self._function(*particula.to_evaluate(use_coors=True))
        ultimo_mejor =
self._function(particula.to_evaluate(use_coors=False))

        if actual < ultimo_mejor:
            particula.memoria = np.copy(particula.coordenadas)
            mejor_del_enjambre =
self._function(self._mejor.to_evaluate(use_coors=False))

            if ultimo_mejor < mejor_del_enjambre:
                self._mejor = particula

    def iterar(self):
        self._num_generacion += 1
        self._print_generation()

        for particula_index in range(self._num_particulas):
            particula: Particle = self._poblacion[particula_index]

            for dim in range(K_DIMENSION):
                coef_inercia = K_COEFICIENTE_PESO *
particula.velocidad[dim]
                coef_memoria = K_COEFICIENTE_MEMORIA *
np.random.random() * (particula.memoria[dim] -
particula.coordenadas[dim])
                coef_lider = K_COEFICIENTE_LIDER * np.random.random()
* (self._mejor.memoria[dim] - particula.coordenadas[dim])

                particula.velocidad[dim] = coef_inercia + coef_memoria
+ coef_lider
                particula.coordenadas[dim] =
particula.coordenadas[dim] + particula.velocidad[dim]
```

```
            self._actualizar_particula(particula)

    def _print_generation(self):
        valor =
self._function(self._mejor.to_evaluate(use_coors=False))

        print(f"Generation {self._num_generacion}")
        print("Best so far:")
        print(f"Coordenates: {self._mejor}")
        print(f"Function value: {valor}")
```

Code 2. Swarm class definition

On the other hand, Swarm class could be called as a wrapper, it takes a higher-order function and uses it to minimize results over iterations. One important detail is the use of Callable class as hint annotation for function parameter, it could take any number of dimensions, but it is expected to just return one dimension.

```
K_NUM_PARTICULAS = 10
enjambre = Swarm(K_NUM_PARTICULAS, ackley)
```

```
enjambre.iterar()
```

Code 3 and 4. Reset and iterate codes

And these lines are the cherry on the cake, we just define a function put it as a param and iterate. Define hyperparameters, and just have fun (or maybe try to minimize the function).

Ackley function

```
import numpy as np

K_DIMENSION = 2
K_A = 20
K_B = 0.2
K_C = 2 * np.pi

def ackley(*args: np.ndarray) -> np.ndarray:
    valores = np.array([args[i] for i in range(0, len(args))])
    valores = valores.transpose() if valores.shape[0] > 1 else valores
    radicando = 1/K_DIMENSION * np.sum(np.power(valores, 2), axis=-1)
    exponente = 1/K_DIMENSION * np.sum(np.cos(K_C * valores))

    return -K_A * np.exp(-K_B * np.sqrt(radicando)) -
np.exp(exponente) + K_A + np.exp(1)
```

Code 5. Ackley function definition

Taking the fact that Swarm and Particle classes handle N dimensions, why not to make Ackley function to do so, so it receives N arguments and calculates M points given in form of numpy arrays. Now, if we want to change to upper or lower number of dimensions, we just change the hyperparameter, easy as that.

Hyperparameter	Value
K_DIMENSION	2
K_LIMITE_SUPERIOR	33
K_LIMITE_INFERIOR	-33
K_COEFICIENTE_PESO	0.5
K_COEFICIENTE_MEMORIA	0.2
K_COEFICIENTE_LIDER	0.3
K_NUM_PARTICULAS	10

Table 1. Ackley-hyperparameters settings

Ackley is easier to minimize compared to Rastrigin. The common domain for this function is from -32.5 to 32.5, so, we just take a close integer from them. The results are show below.

```
Generation 1
Best so far:
Coordenates: [-10.37485523  10.52567961]
Function value: [19.81599429]
```

```
Generation 2
Best so far:
Coordenates: [  3.35023289 -11.38653863]
Function value: [18.47509247]
```

Fig. 1 and 2: Generations 1 and 2

```
Generation 6
Best so far:
Coordenates: [1.92438813 3.36856054]
Function value: [10.05217063]
```

```
Generation 8
Best so far:
Coordenates: [0.25332848 0.38722676]
Function value: [3.30818895]
```

Fig. 3 and 4: Generations 6 and 8

```
Generation 15
Best so far:
Coordenates: [ 0.22466237 -0.01465558]
Function value: [1.56409375]
```

```
Generation 20
Best so far:
Coordenates: [0.08924809 0.12048369]
Function value: [0.78820438]
```

Fig. 5 and 6: Generations 15 and 20

```
Generation 35
Best so far:
Coordenates: [-0.00133571  0.06927346]
Function value: [0.31887602]
```

Fig. 7 Generation 35

Rastrigin function

```python
import numpy as np

K_DIMENSION = 3
K_A = 10

def rastrigin(*args: np.ndarray) -> np.ndarray:
    valores = np.array([args[i] for i in range(0, len(args))])
    valores = valores.transpose() if valores.shape[0] > 1 else valores
    sumando = np.sum(np.power(valores, 2) - K_A * np.cos(2*np.pi *
valores), axis=-1)

    return K_A * 3 + sumando
```

Code 6. Rastrigin function definition

Same as Ackley function, it can calculate M points given, with N dimensions predefined. But a lot harder to find the global optimal. The reason for this behavior can be found in the fact Rastrigin has a lot more local minima, now this function is planned to use three dimensions, making it even harder. Even if the domain is a lot smaller (from -5.22 to 5.22) it is harder to find it.

Again, we take an integer close to 5.22 and -5.22, it is, in fact, possible to use floating numbers, but just to keep all parameters as integers. Results are shown below the following table.

Hyperparameter	Value
K_DIMENSION	3
K_LIMITE_SUPERIOR	5
K_LIMITE_INFERIOR	-5
K_COEFICIENTE_PESO	0.8
K_COEFICIENTE_MEMORIA	0.4
K_COEFICIENTE_LIDER	0.8
K_NUM_PARTICULAS	20

Table 2. Rastrigin-hyperparameters settings

```
Generation 1
Best so far:
Coordenates: [ 2.94041359 -1.71521161 -1.08886296]
Function value: [27.15341616]
```

```
Generation 4
Best so far:
Coordenates: [ 0.0987413   0.19015431 -0.10437211]
Function value: [10.32250169]
```

Fig. 8 and 9: Generations 1 and 4

```
Generation 8
Best so far:
Coordenates: [ 0.11477809 -0.04106878  0.17144568]
Function value: [8.12727718]
```

```
Generation 17
Best so far:
Coordenates: [ 0.08201889 -0.10443737  0.04717294]
Function value: [3.83148666]
```

```
Generation 24
Best so far:
Coordenates: [-0.07502997 -0.05334407 -0.06726218]
Function value: [2.54007542]
```

```
Generation 33
Best so far:
Coordenates: [-0.07172042 -0.0052814  -0.05413935]
Function value: [1.58490335]
```

Fig. 10, 11, 12 and 13: Generations 8 and 17, 24 and 33

```
Generation 35
Best so far:
Coordenates: [0.02445157 0.03210558 0.02718308]
Function value: [0.46843144]
```

```
Generation 36
Best so far:
Coordenates: [ 3.19055622e-05  2.49847618e-02 -2
.08877276e-02]
Function value: [0.21002566]
```

```
Generation 45
Best so far:
Coordenates: [ 0.01285559 -0.01404492  0.00891359]
Function value: [0.08763787]
```

```
Generation 47
Best so far:
Coordenates: [ 0.00060032 -0.01104269  0.00066799]
Function value: [0.02434251]
```

Fig. 14, 15, 16 and 17: Generations 35, 36, 45 and 47

```
Generation 55
Best so far:
Coordenates: [-0.00948647 -0.03212104  0.0001016 ]
Function value: [0.00524025]
```

Fig. 18: Best result from Rastrigin optimization

Conclusions and recommendations

Talking about these problems, again I see Rastrigin is harder to find its global minimum, this approach looks a lot faster to develop, and, of course, knowing beforehand that it has a global minimum makes this easier. But what about a flat function with some minima in random locations. I think that wouldn't work for this approach. So far, it is looking good for both.

Now, talking personally, it is a pleasure to look at what I have done and what I have become, the first two weeks I was very scared of this kind of problems, and now I think it is not easy, but fun. And I compare it as taking a job, it is a responsibility, of course, but rather than taking it as that we should look at the beauty of our work, the pleasure it causes to us, and the satisfaction.

References

[1] S. Surjanovic, D. Bingham. *ACKLEY FUNCTION*. 2013. Virtual Library of Simulation Experiments. Accessed on: Oct. 29, 2021. [Online] Available: https://www.sfu.ca/~ssurjano/ackley.html

[2] Wikipedia. *Rastrigin function*. Apr. 20, 2021. Wikipedia. Accessed on: Oct. 29, 2021. [Online] Available: https://en.wikipedia.org/wiki/Rastrigin_function

[3] Wikipedia. *Particle Swarm Optimization.* Sep. 25, 2021. Wikipedia. Accessed on: Oct. 16, 2021. [Online] Available: https://en.wikipedia.org/wiki/Particle_swarm_optimization.