Centro Universitario de Ciencias Exactas e Ingenierías

Licenciatura en Ingeniería en Computación

Materia: Seminario de Solución de Problemas de Inteligencia Artificial I. Clave: I7039.

Profesor: Sencion Echauri Felipe

Estudiante: Silva Moya José Alejandro. Código: 213546894.

Actividad 9: Differential Evolution

Instrucciones: Implementar y evaluar el rendimiento del algoritmo de optimización por evolución diferencial (differential evolution) para las siguientes funciones:

- Sphere
- Rosenbrock
- Rastrigin
- Quartic

Para cada función realizar 5 ejecuciones con 2, 4, 8 y 16 dimensiones, cada ejecución se detendrá a las 2000 generaciones.

Se deberá graficar el comportamiento del algoritmo; para ello se deberá promediar el valor del mejor fitness de las 5 ejecuciones en la generación 0, 100, 200, ... 2000. Se deberá generar una gráfica para cada dimensión y además una gráfica en la que se incluyan las ejecuciones para 2, 4, 8 y 16 dimensiones, es decir un total de 5 gráficas por función.

Desarrollo

```
21 #The actual Differential Evolution Algorithm.
22 def differentialEvolution(ecuation, dimensions, individuals, F, c, generations):
23     graphArray = np.array([])   #For graphing the results.
24 
25     #We begin by generatin a population. Each individual has a vector with
26     population = []     #as much spaces as dimensions for the ecuation, with values between the ecuation bounds.
27     for i in range(0, individuals):
28         individual = []
29         for j in range(dimensions):
30             individual.append(random.uniform(ecuation.MIN_VALUE, ecuation.MAX_VALUE))
31         population.append(individual)
```

Comenzamos por generar una población de individuos, que tendrán un vector de tamaño igual a la cantidad de dimensiones del problema, con valores aleatorios entre los límites de búsqueda de solución del problema.

```
33     for i in range(0, generations):
34         print('GENERATION:',i)
35         fitness_scores = [] #We catch the best fitness here.
36 
37         for j in range(0, individuals):
38             #We start with the actual mutation.
39             #We begin by selecting 3 random individuals that are also different from the iteration individual.
40             candidates = list(range(0, individuals))
41             candidates.remove(j)
42             random_candidate = random.sample(candidates, 3)
43 
44             x1 = population[random_candidate[0]]
45             x2 = population[random_candidate[1]]
46             x3 = population[random_candidate[2]]
47             actual_individual = population[j]
```

Ahora procedemos para cada individuo existente de la población:

- Tomamos 3 individuos aleatorios que sean únicos entre sí y también diferentes al individuo correspondiente a la iteración. Esto lo hacemos para poder crear el vector nuevo mutado.

```
49            #We get the difference between two vectors to create the mutant vector.
50            difference = [x2_i - x3_i for x2_i, x3_i in zip(x2, x3)]
51
52            #We multiply the difference by the mutation factor (F) and add to x1
53            mutant_vector = [x1_i + F * difference_i for x1_i, difference_i in zip(x1, difference)]
54
55            #We make sure the vector won't go out of the solving problem bounds.
56            mutant_vector = ensure_bounds(mutant_vector, ecuation)
```

En cada iteración obtenemos las diferencias por posición del vector 2 con respecto del 3, y luego generamos el nuevo vector mutado, sumando a cada posición del vector tomado en el punto anterior y sumándole nuestro valor F multiplicado por las diferencias obtenidas también en el paso anterior.

```
1  import random
2  import numpy as np
3
4  #We make sure that the individual won't go out of the ecuation solution bounds.
5  def ensure_bounds(vector, ecuation):
6      new_vector = []
7
8      for i in range(len(vector)):
9          if vector[i] < ecuation.MIN_VALUE:      #If the value goes below the minimum value,
10             new_vector.append(ecuation.MIN_VALUE)   #we re-write it to the actual minimum.
11
12         if vector[i] > ecuation.MAX_VALUE:      #If the value goes above the maximum value,
13             new_vector.append(ecuation.MAX_VALUE)   #we re-write it to the actual maximum.
14
15         if ecuation.MIN_VALUE <= vector[i] <= ecuation.MAX_VALUE:   #If the value is acceptable,
16             new_vector.append(vector[i])                            #we just let it as it is.
17
18     return new_vector
```

Dado que necesitamos revisar que las soluciones no se salgan del espacio de solución de ecuación, si encontramos que se salen en positivo o negativo, los regresamos a los límites, y en cualquier otro caso los dejamos igual, ya que serían aceptables.

```
58            #Recombination of the vectors. If a random value goes below our "c", we change the value.
59            trial_vector = []
60            for k in range(len(actual_individual)):
61                crossover = random.random()
62                if crossover <= c:
63                    trial_vector.append(mutant_vector[k])
64
65                else:
66                    trial_vector.append(actual_individual[k])
```

Iniciamos con la recombinación. Generamos un número aleatorio correspondiente al rango de dimensiones, un vector donde haremos la recombinación, y con un número aleatorio entre 0.1 y 0.9, si se cumple que sea menor que nuestro valor aleatorio c, intercambiamos el valor del vector original con el del vector mutado; en caso contrario, lo dejamos como está.

```
68              #Selection of the best individual.
69              trial_vector_fitness  = ecuation.fitness(trial_vector)
70              actual_individual_fitness = ecuation.fitness(actual_individual)
71
72              if trial_vector_fitness < actual_individual_fitness:
73                  population[j] = trial_vector
74                  fitness_scores.append(trial_vector_fitness)
75
76              else:
77                  fitness_scores.append(actual_individual_fitness)
```

Finalmente evaluamos el fitness de nuestros resultados. Si el fitness del vector resultante es mejor que el actual, reemplazamos, y en caso contrario, lo dejamos como está.

```
79          generation_best = population[fitness_scores.index(min(fitness_scores))]     # solution of best individual
80          print(generation_best, ecuation.fitness(generation_best), '\n')
81          if i % 100 == 0 :
82              graphArray = np.append(graphArray, [ecuation.fitness(generation_best)])
83
84      return graphArray
```

Imprimimos el mejor resultado de cada generación, y cada 100 generaciones obtenemos el mejor fitness para poder graficarlo posteriormente.

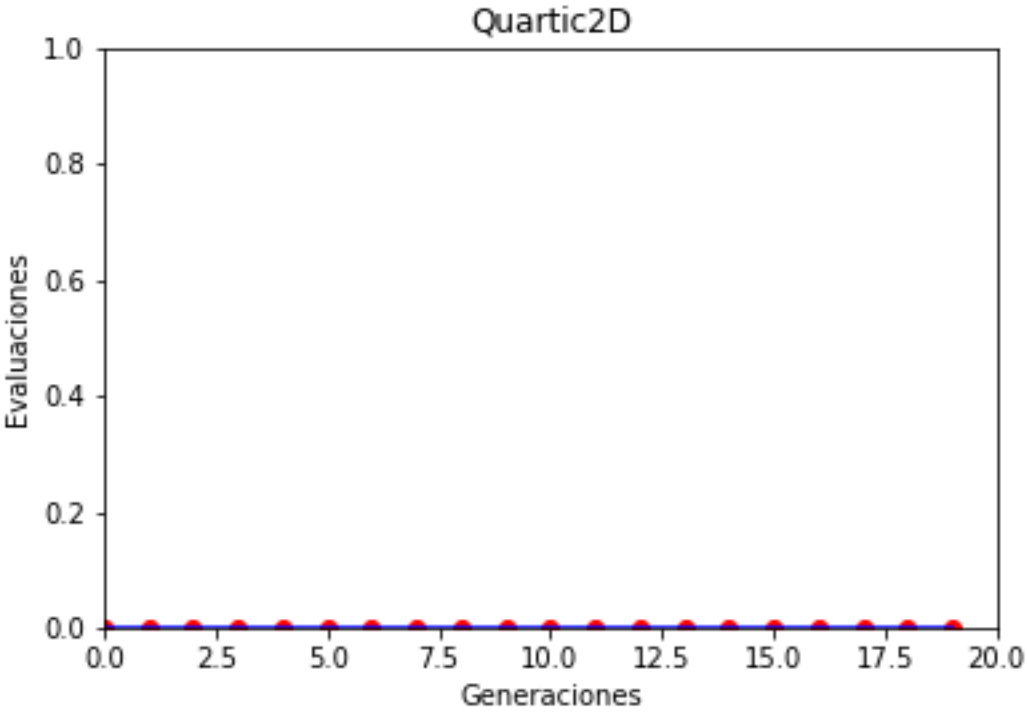
```python
import DE
import sphere
import rosenbrock
import quartic
import rastrigin
import drawer
import numpy as np

def main():
    sph = sphere.sphere()
    ros = rosenbrock.rosenbrock()
    qua = quartic.quartic()
    ras = rastrigin.rastrigin()
    draw = drawer.Drawer()

    aux = np.array([])
    graph = np.array([])
    executions = 5
    stepSize_parameter = 0.5    #[0.4, 0.9] (F in DE)
    crossover_rate = 0.7        #[0.1, 1.0] (c in DE)
    individuals = 30
    generations = 2000
    dimensions = 16

    graphName = "Quartic" + str(dimensions) + "D"
```

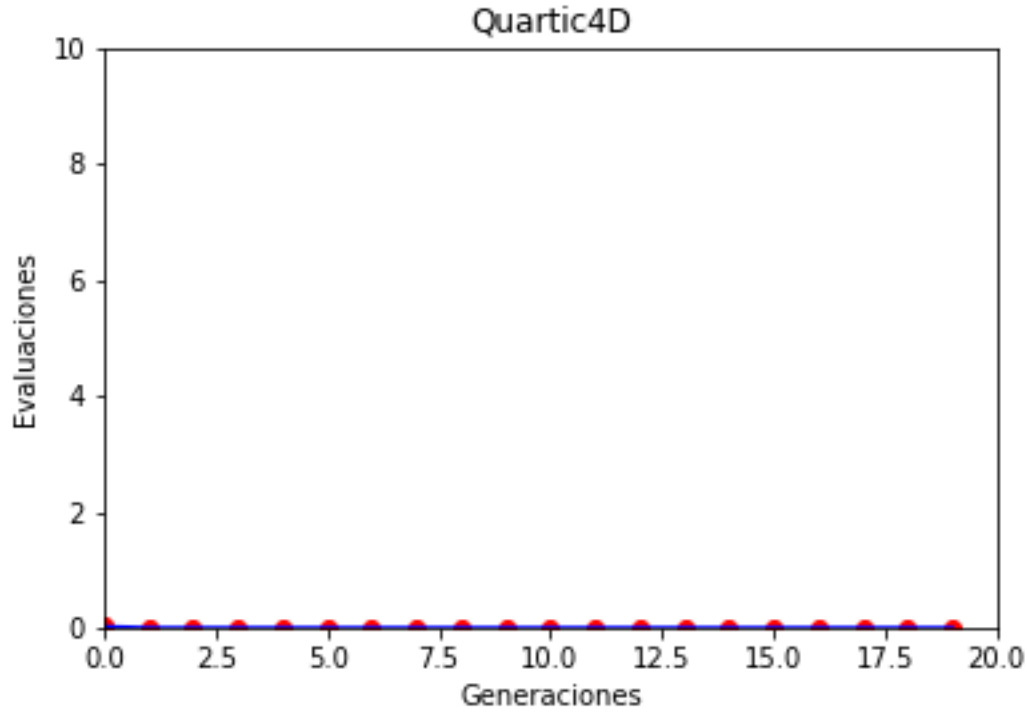
En el main tenemos objetos de cada ecuación a resolver, un objeto para graficar individual y en grupo, y los parámetros específicos que necesita el algoritmo para funcionar.

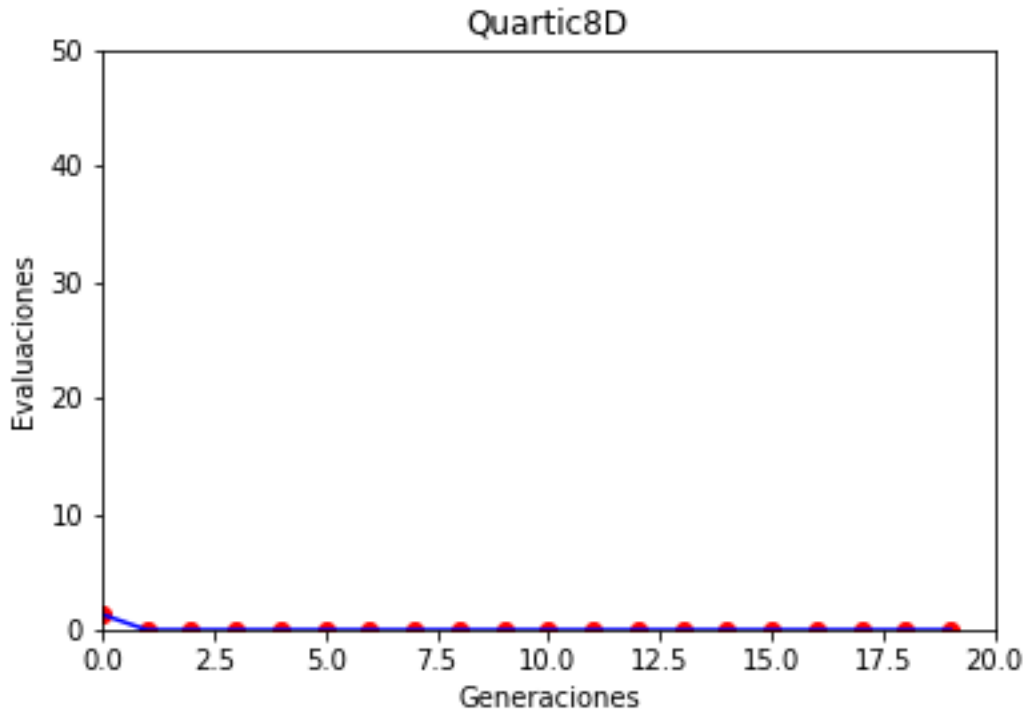
```python
    for i in range(executions):
        aux = DE.differentialEvolution(sph, dimensions, individuals, stepSize_parameter,
                                       crossover_rate, generations)

        if i == 0:      #For the first vector, we just make a copy to the one we'll be adding to.
            graph = aux
        else:           #For the rest, we just add.
            for a in range(len(aux)):
                graph[a] = graph[a] + aux[a]

    for x in range(len(graph)):
        graph[x] = graph[x]/executions    #We obtain the average for each vector position.

    draw.drawIndividual(graph, graphName)

    file = open(graphName + ".txt", "w")
    for y in range(len(graph)):
        file.write(str(graph[y]) + "\n")
    file.close()
#    draw.drawGroup("Sphere")

if __name__ == '__main__':
    main()
```

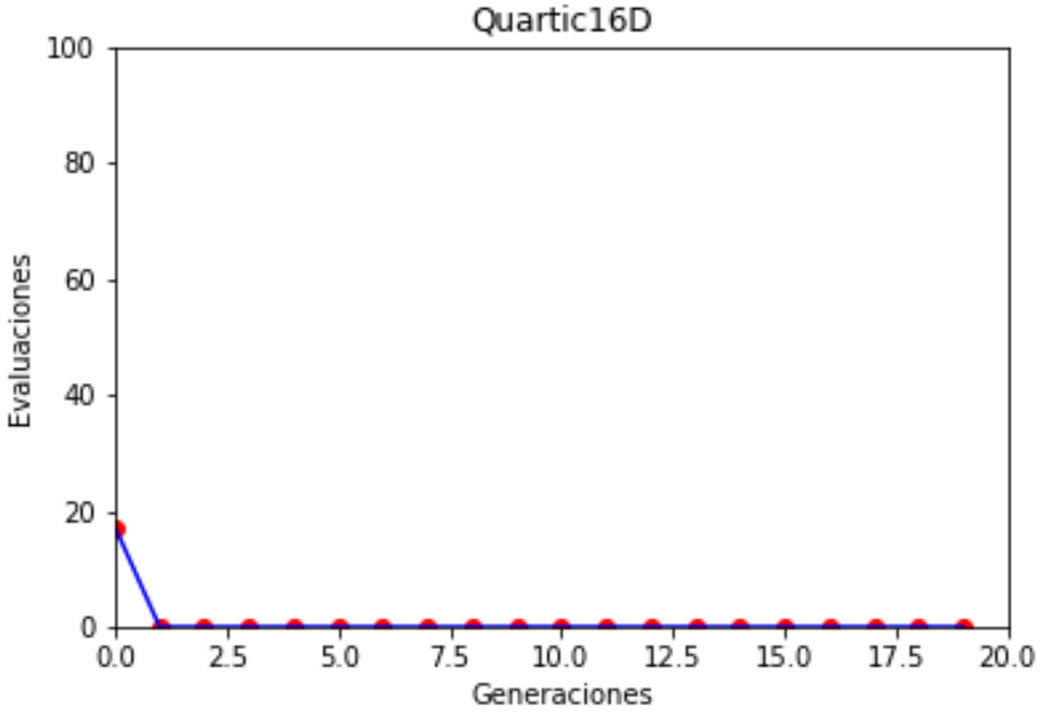
El resto del algoritmo (así como el fitness de las ecuaciones), son exactamente las mismas que en las entregas anteriores.

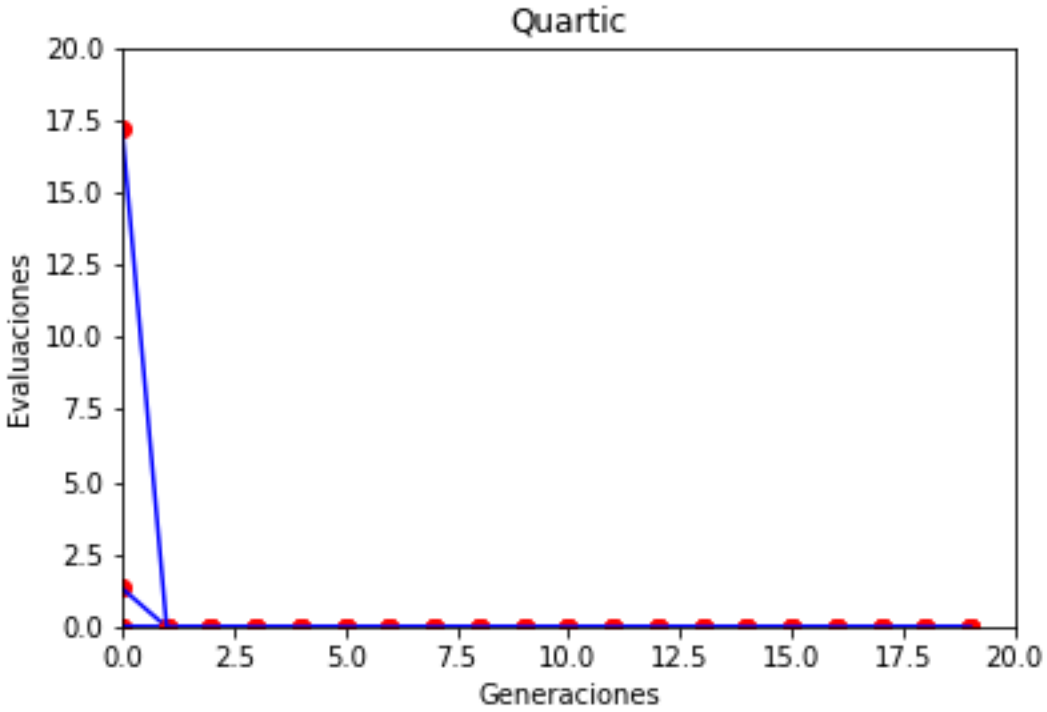
Resultados obtenidos

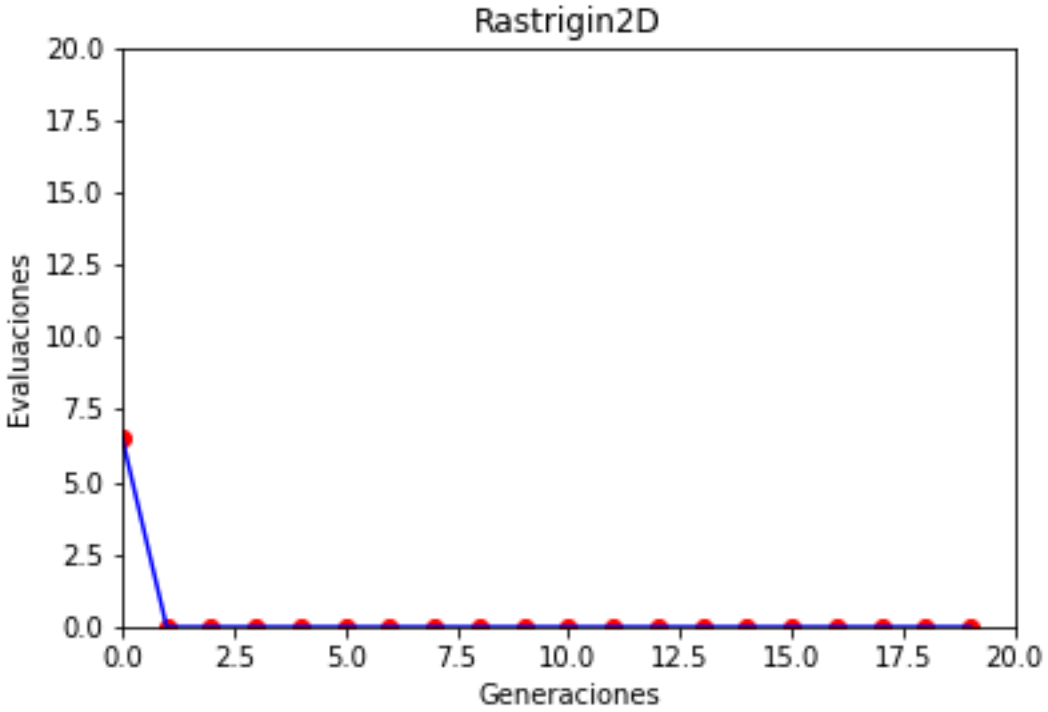


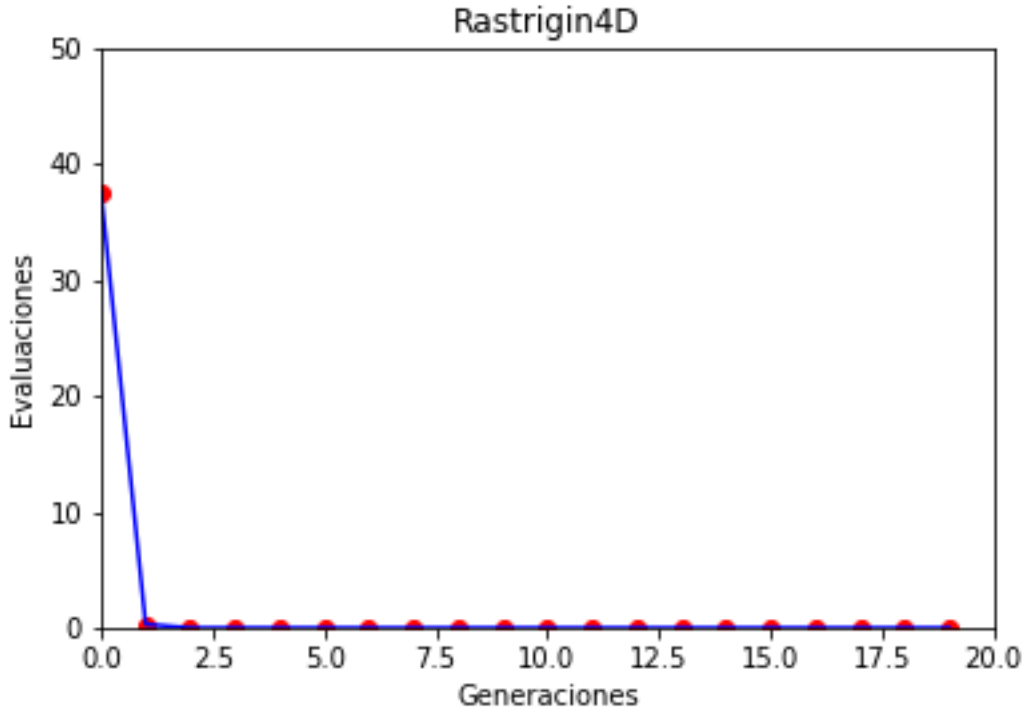


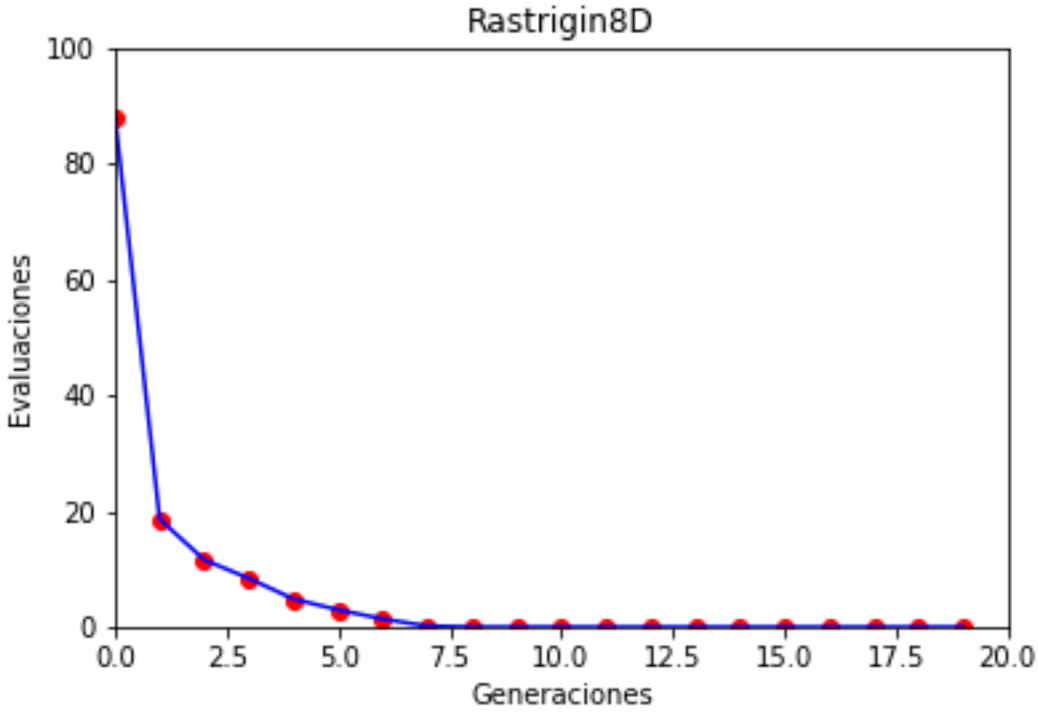

Quartic8D
Evaluaciones
Generaciones
0
10
20
30
40
50
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

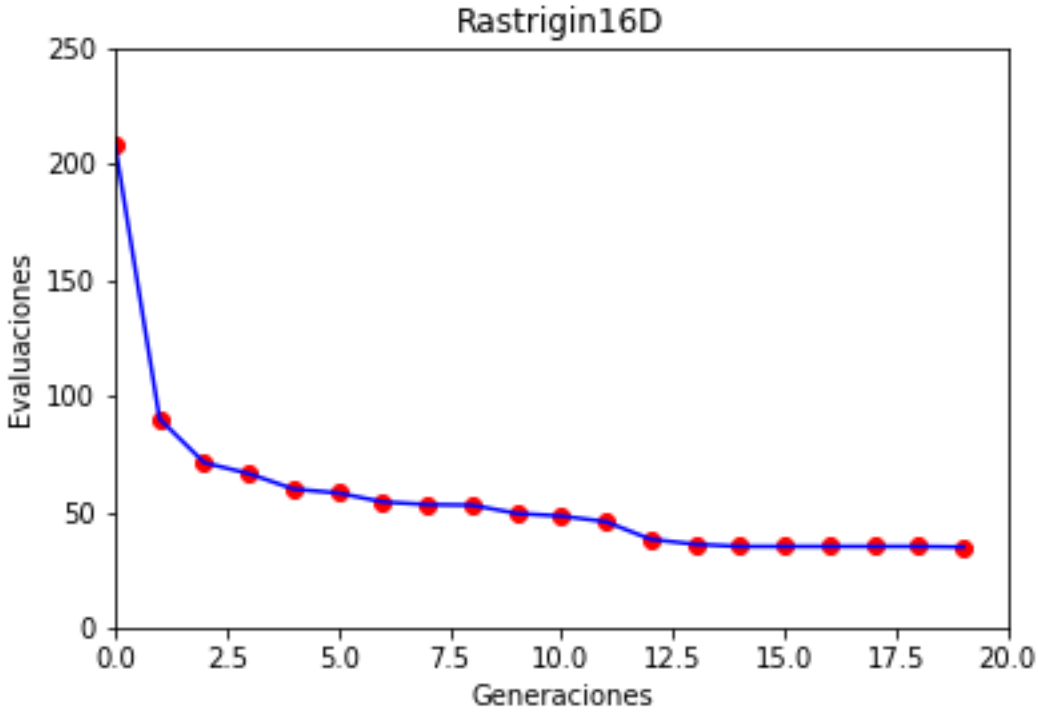

Quartic16D
Evaluaciones
Generaciones
0
20
40
60
80
100
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

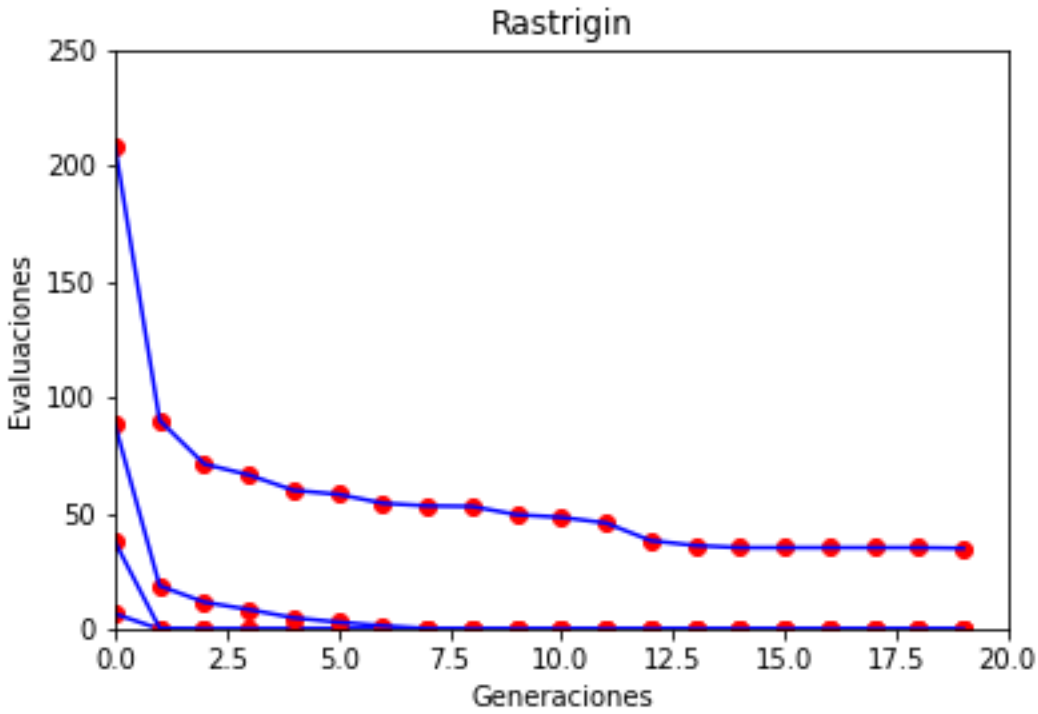

Quartic
Evaluaciones
Generaciones
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

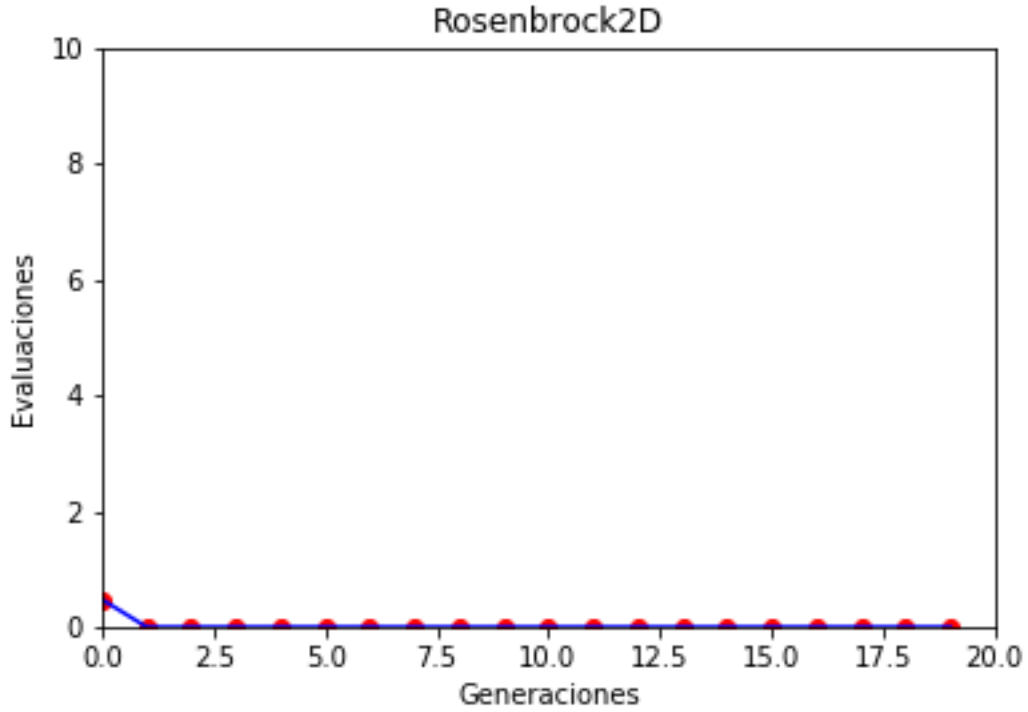

Rastrigin2D
Evaluaciones
Generaciones
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

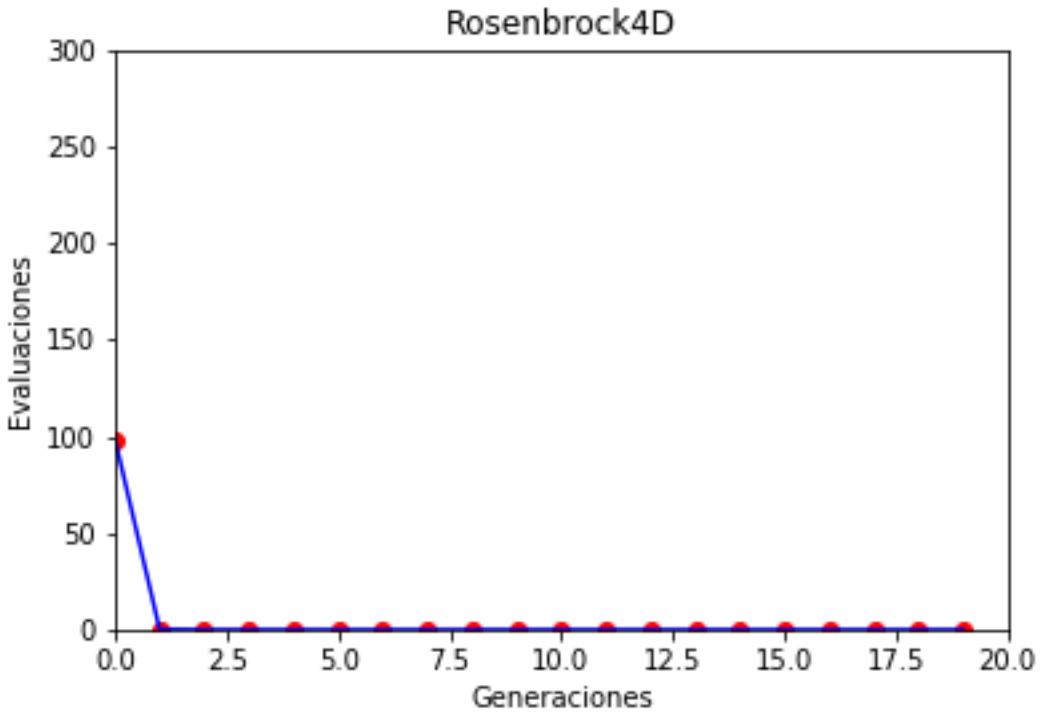

Rastrigin4D
Evaluaciones
Generaciones
0
10
20
30
40
50
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

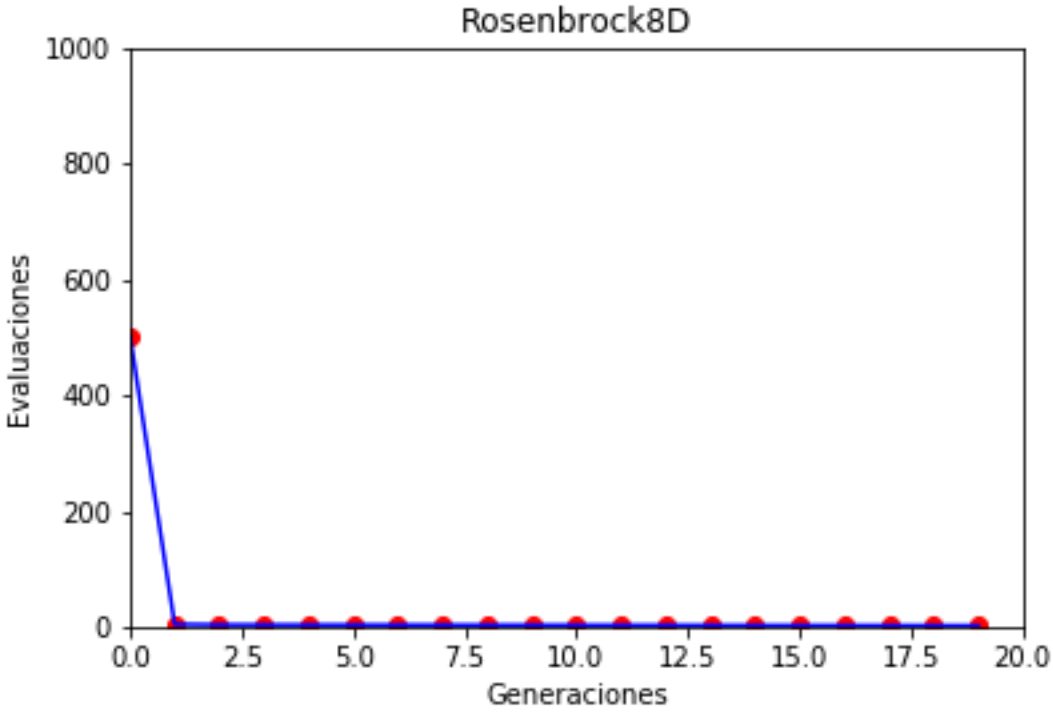

Rastrigin8D
Evaluaciones
Generaciones
0
20
40
60
80
100
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

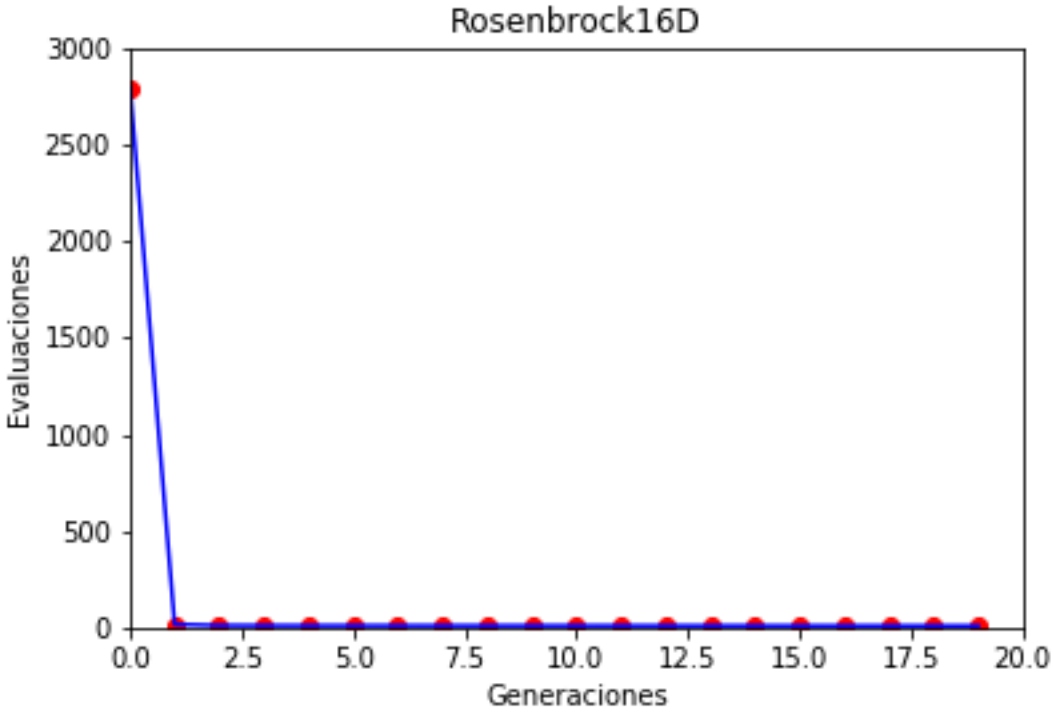

Rastrigin16D
Evaluaciones
Generaciones
0
50
100
150
200
250
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

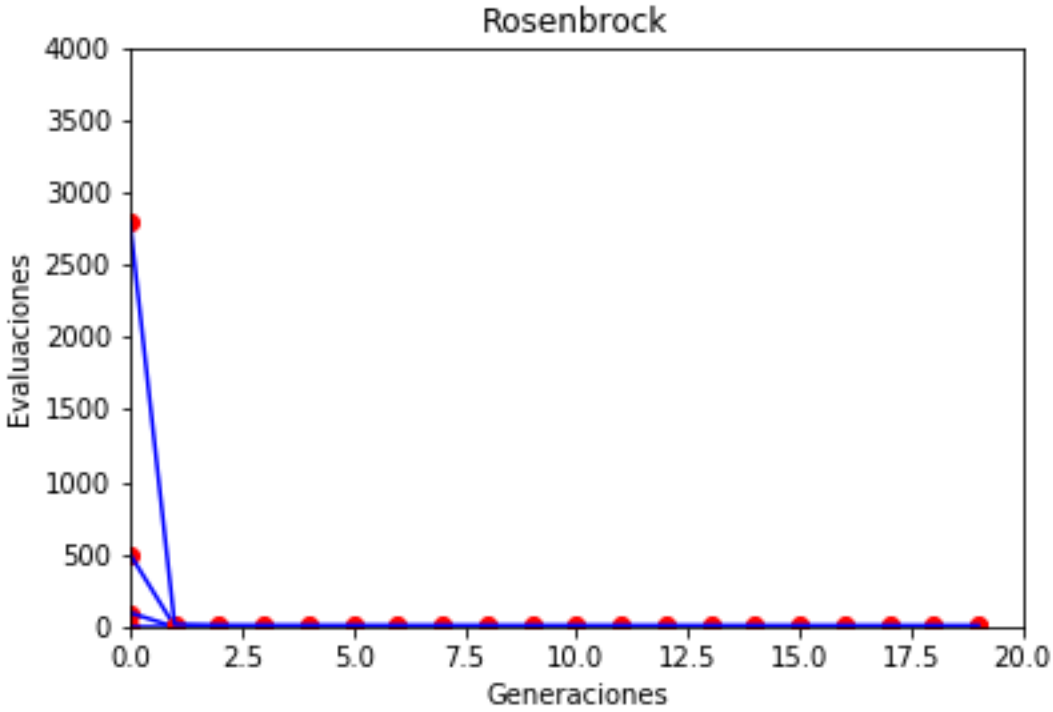

Rastrigin
250
200
150
100
50
0
Evaluaciones
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Generaciones

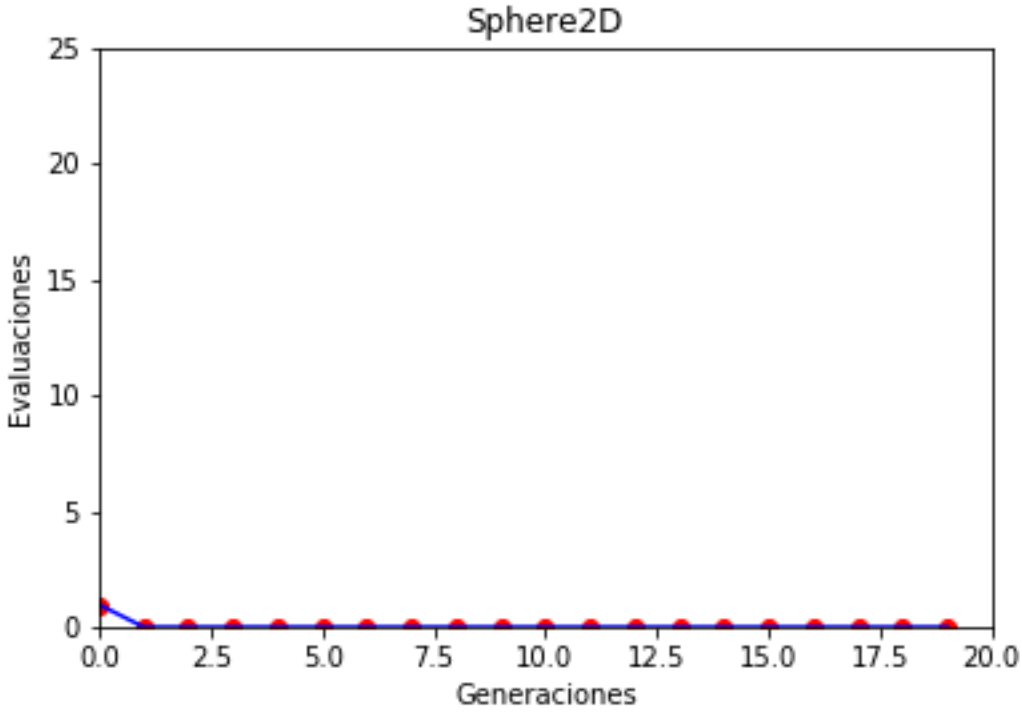

Rosenbrock2D
Evaluaciones
Generaciones
0
2
4
6
8
10
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

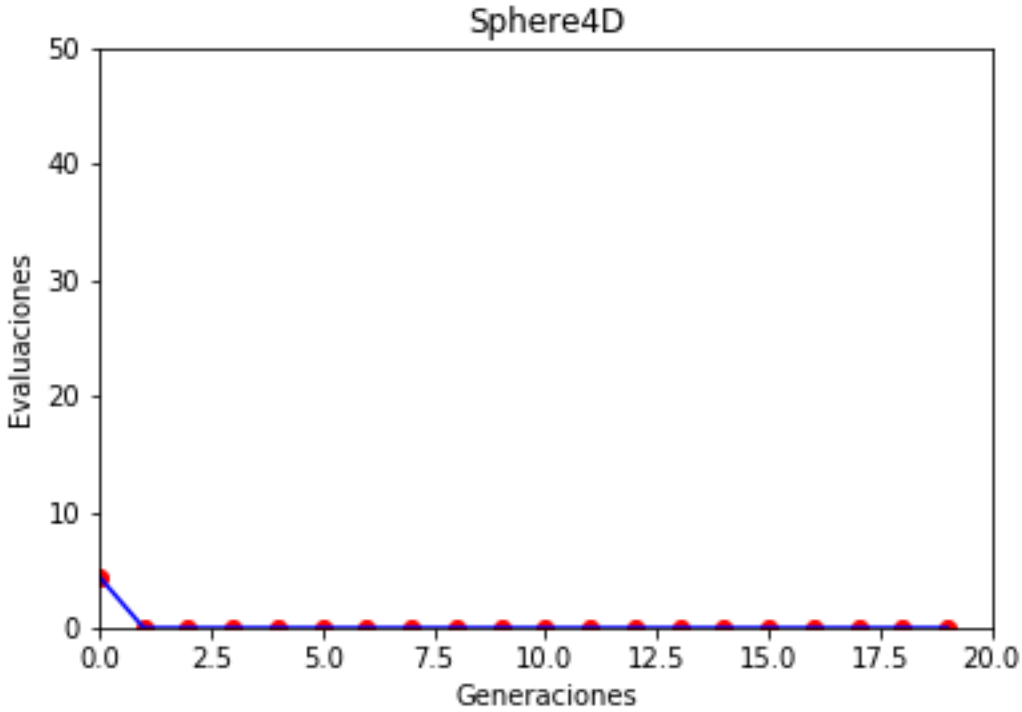

Rosenbrock4D
Evaluaciones
Generaciones
0
50
100
150
200
250
300
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

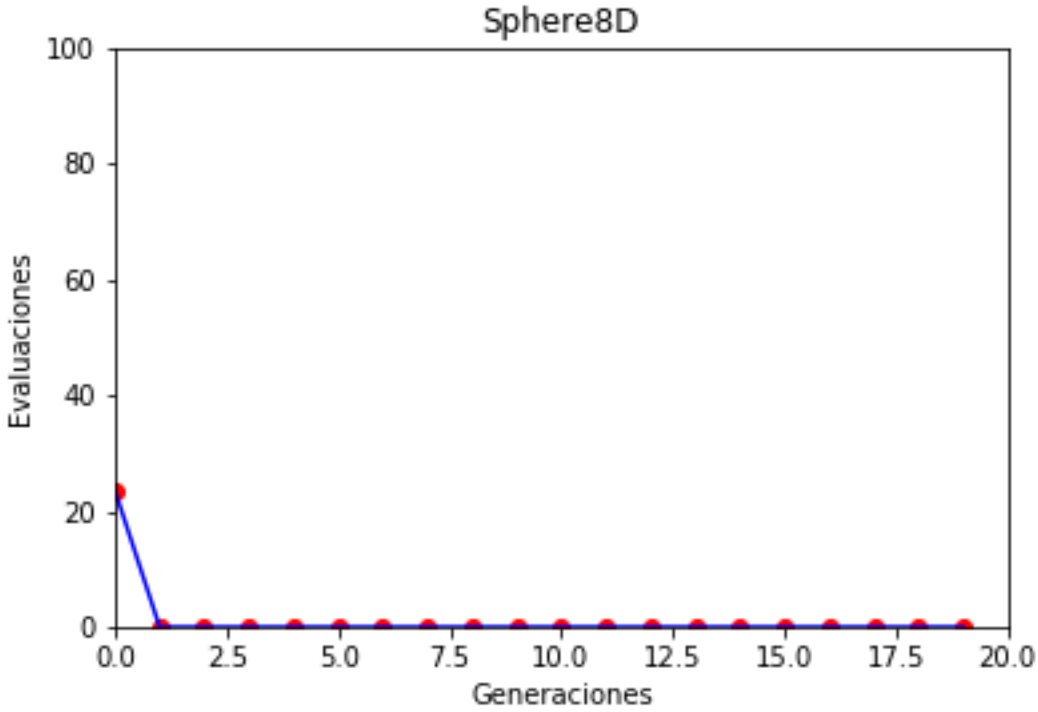

Rosenbrock8D
1000
800
600
400
200
0
Evaluaciones
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Generaciones

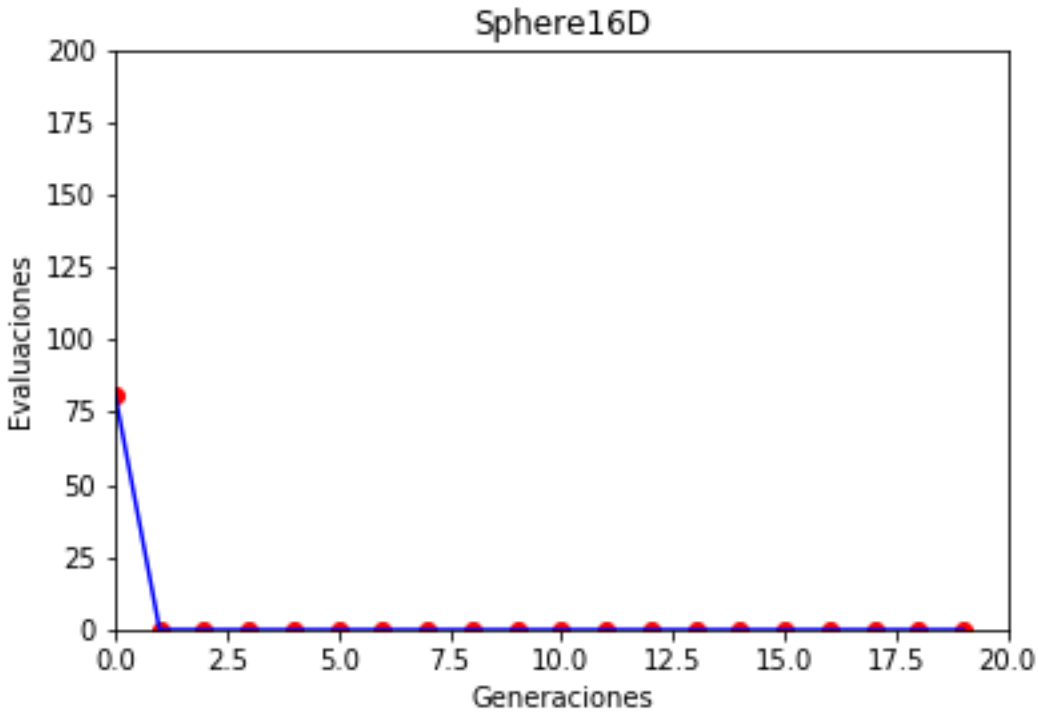

Rosenbrock16D
3000
2500
2000
1500
1000
500
0
Evaluaciones
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Generaciones

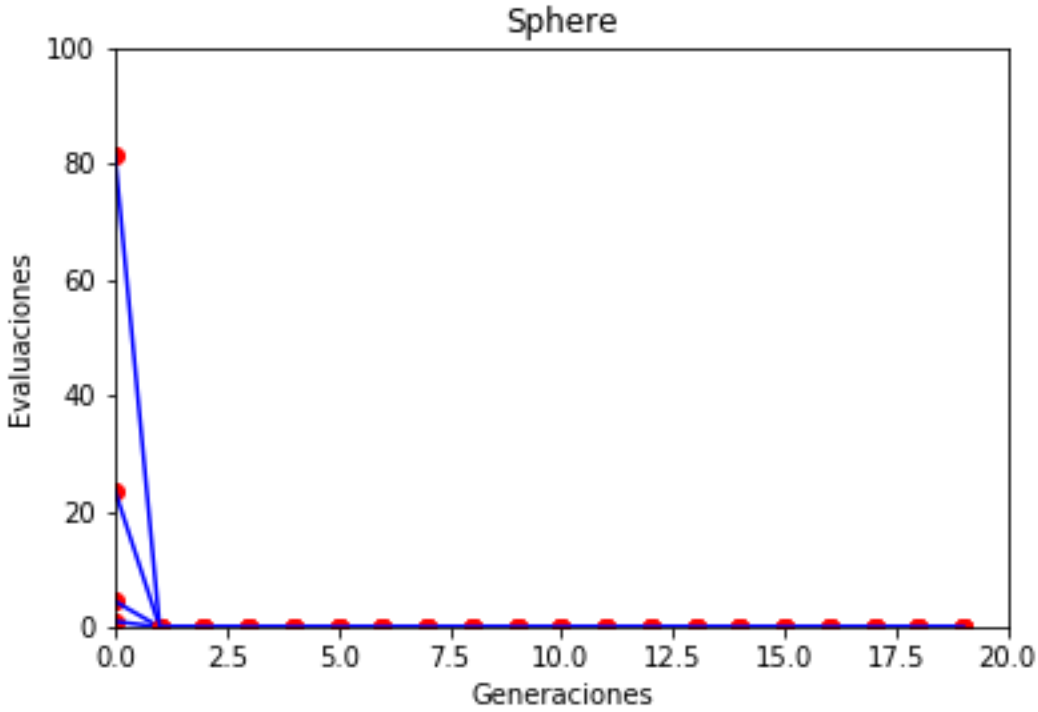

Rosenbrock
4000
3500
3000
2500
2000
1500
1000
500
0
Evaluaciones
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Generaciones


Sphere2D
Evaluaciones
Generaciones
0
5
10
15
20
25
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Sphere4D
Evaluaciones
Generaciones
0
10
20
30
40
50
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Sphere8D
Evaluaciones
Generaciones
0
20
40
60
80
100
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Sphere16D
Evaluaciones
Generaciones
0
25
50
75
100
125
150
175
200
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Sphere
100
80
60
40
20
0
Evaluaciones
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Generaciones

Conclusiones

Después de varios intentos y pruebas fallidas con errores demasiado extraños que no podía resolver, me di a la tarea de hacer el código desde cero una vez más (debido a que era bastante corto, así que no representaba una gran desventaja). Al final no sé realmente qué fue lo que cambié o arreglé, pero logré corregir los problemas, que me causaban que los fitness hicieran cosas raras y que las gráficas en lugar de solamente disminuir, de repente hicieran picos hacia arriba y luego volvieran a bajar.

Si bien esta implementación fue considerablemente sencilla, esos problemas me causaron un buen retraso, pero al final lo logré resolver.