Practica 1. Algoritmo genético para espacios continuos

Algoritmos Bioinspirados

Diego Castillo Reyes
Marthon Leobardo Yañez Martinez
Aldo Escamilla Resendiz

20 de marzo de 2024

Introducción

En esta practica hemos implementado un algoritmo genético para resolver un problema de optimización en un espacio continuo. El problema consiste en encontrar el mínimo de las funciones de Rosenbrock y Ackley, representadas por las siguientes ecuaciones respectivamente:

$$f(x)_{Rosenbrock} = \sum_{i=1}^{n-1}[100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2] \tag{1}$$

$$f(x)_{Ackley} = -20exp(-0{,}2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) - exp(\frac{1}{n}\sum_{i=1}^{n} cos(2\pi x_i)) + 20 + e \tag{2}$$

Para llevar a cabo esta tarea, se optó por el lenguaje de programación Python debido a su versatilidad, facilidad de uso y a la gran cantidad de librerías que ofrece para el manejo de arreglos y operaciones matemáticas. Además, se utilizó la librería *matplotlib* para graficar las funciones y el comportamiento del algoritmo genético.

Desarrollo

El algoritmo se dividió en 6 funciones principales:

Función de inicialización de población

```python
def start_poblation(f, poblation_size, dimension, lower_limit, upper_limit):
    poblation = []

    for _ in range(poblation_size):
        individual = []
        for _ in range(dimension):
            individual.append(random.uniform(lower_limit, upper_limit))

        if f == "rosenbrock":
            evaluation = rosenbrock(individual)
        elif f == "ackley":
            evaluation = ackley(individual)

        poblation.append([individual, evaluation])

    return poblation

```

Listing 1: Funcion poblacion

Esta función se utiliza para inicializar una población de individuos con valores aleatorios en un rango específico dado por la función. La función toma cinco parámetros: el número de individuos, el número de variables, f, tamaño de la población, dimensión, limite inferior y limite superior.

- f: Es una cadena que especifica la función de aptitud que se utilizará, ya sea Rosenbrock o Ackley.
- tamaño de la población: Es el número de individuos que se generarán.
- dimensión: Es el número de variables que tendrá cada individuo.
- limite inferior: Es el valor mínimo que pueden tomar las variables de cada individuo.
- limite superior: Es el valor máximo que pueden tomar las variables de cada individuo.

La función comienza inicializando una población de lista vacía para contener la población de individuos. Luego ingresa a un bucle que ejecuta las veces de individuos seleccionados. En cada iteración de este ciclo, se crea un nuevo individuo.

Un individuo se representa como una lista de números de dimensión. Cada número es un flotante aleatorio generado por la función random.uniform, que devuelve un flotante aleatorio dentro del rango especificado por el límite inferior y límite superior. Una vez que se crea un individuo, su aptitud se evalúa utilizando la función de aptitud especificada. Si f es Rosenbrock, la función Rosenbrock se llama con el individuo como argumento. Si f es Ackley, en su lugar se llama a la función Ackley. La evaluación de aptitud se almacena en la evaluación de variables.

Finalmente, el individuo y su evaluación de aptitud se agregan como un par a la lista de población. Este proceso se repite hasta que se hayan creado y evaluado los individuos del tamaño de la población. La función devuelve la lista de población, que ahora contiene pares del tamaño de la población, cada uno de los cuales consta de un individuo y su evaluación de aptitud.

Función que define las parejas

```
    def define_couples(poblacion, poblacion_size):
        fitness = [poblacion[i][1] for i in range(poblacion_size)]
        selected = []

        # Calcular la suma total de fitness
        total_fitness = sum(fitness)

        while len(selected) < poblacion_size:
            # Generar un numero aleatorio entre 0 y la suma total de fitness
            selection = random.uniform(0, total_fitness)

            acumulated = 0
            for i, fit in enumerate(fitness):
                acumulated += fit
                # Verificar si el acumulado supera el numero aleatorio y si el indice no ha sido
    seleccionado previamente
                if acumulated >= selection:
                    if i not in selected:
                        selected.append(i)
                        break
                    else:
                        # Si el indice ya fue seleccionado, generar un nuevo numero aleatorio y
    reiniciar el proceso
                        selection = random.uniform(0, total_fitness)
                        acumulated = 0

        # Asegurar que la cantidad de indices seleccionados sea par para formar parejas
        if len(selected) % 2 != 0:
            selected.pop()

        return selected

```

Listing 2: Funcion parejas

La función recibe como parámetros la población y el número de parejas que se seleccionarán.

- población: Es la lista de individuos que se cruzarán.
- número de parejas: Es el número de parejas que se seleccionarán.

Esta función se utiliza para seleccionar las parejas de individuos que se cruzarán para generar una nueva población. La función toma dos parámetros: la población y el número de parejas que se seleccionarán. La función empieza creando la lista fitness, que contiene los valores de ajuste de la población. Posteriormente crea la lista de seleccionados donde se almacenarán los indices de los individuos seleccionados. Luego, se ingresa a un bucle que se ejecuta el número de veces que se seleccionarán parejas. En cada iteración de este ciclo, se selecciona una pareja de individuos y se agrega a la lista de acumulados. Después de seleccionar una pareja, comprueba que la lista de las parejas que se cruzarán sea un número par y si no lo es, se elimina al ultimo individuo de la lista de seleccionados. Finalmente, la función devuelve la lista de seleccionados, que contiene los índices de los individuos seleccionados para cruzarse.

Función de cruza

```
    def crossover(parent1, parent2):
        # Seleccionar un punto de cruce aleatorio evitando extremos
        crossover_point = random.randint(1, len(parent1) - 1)

        # Sumamos las listas padre para obtener las listas hijas
        child1 = parent1[:crossover_point] + parent2[crossover_point:]
        child2 = parent2[:crossover_point] + parent1[crossover_point:]

        return child1, child2

```

Listing 3: Funcion cruza

La función cruza los individuos seleccionados para formar una nueva población.

- padre 1: Es el primer individuo seleccionado para cruzarse.
- padre 2: Es el segundo individuo seleccionado para cruzarse.

La función esta diseñada para realizar una cruza entre dos individuos y generar dos hijos. Esta empieza seleccionando un punto de cruza aleatorio, excluyendo el primero y el último índice de los individuos. Luego, crea dos hijos, el hijo 1 se crea utilizando la primera parte del padre 1 y la segunda parte del padre 2, mientras que el hijo 2 se crea utilizando la primera parte del padre 2 y la segunda parte del padre 1. Finalmente, la función devuelve una lista con los dos hijos generados.

Función de mutación

```
    def mutation(child, delta, mutation_percentage):
        mutated_child = []

        for gen in child:
            if random.uniform(0, 1) <= mutation_percentage:
                if random.uniform(0, 1) <= 0.5:
                    gen += delta
                else:
                    gen -= delta

            mutated_child.append(gen)

        return mutated_child

```

Listing 4: Funcion mutacion

La función muta a los individuos de la población para mantener la diversidad en la población.

- hijo: Es el individuo que se mutará.
- delta: Es el valor que se sumará o restará a cada variable del individuo.
- porcentaje de mutación: Es el porcentaje de individuos que se mutarán.

La función toma tres argumentos como entrada: el hijo, el delta y el porcentaje de mutación. El hijo es un individuo que se mutará, el delta es el valor que se sumará o restará a cada variable del individuo y el porcentaje de mutación es el porcentaje de individuos que se mutarán.

La función comienza creando una lista donde se guardarán los hijos mutados. Luego, se itera sobre cada gen del hijo y se decide si se muta o no. La mutación en si misma esta definida aleatoriamente por un valor entre 0 y 1, si este valor es menor al porcentaje de mutación. Aunque el gen este mutado o no este se guardara en la lista de hijos mutados. Finalmente, la función devuelve la lista de hijos mutados.

Función de creación de hijos

```
    def create_children(couples, poblation, crossover_percentage):
        children = []
        i = 0

        while i < len(couples):
            if random.uniform(0, 1) <= crossover_percentage:
                parent1 = poblation[couples[i]][0]
                parent2 = poblation[couples[i + 1]][0]
                child1, child2 = crossover(parent1, parent2)
                children.append(child1)
                children.append(child2)

            i += 2

        return children

```

Listing 5: Funcion hijos

La función crea hijos a partir de los individuos seleccionados la población y el porcentaje de cruza.

- población: Es la lista de individuos que se cruzarán.
- parejas: Es la lista de índices de los individuos seleccionados.
- porcentaje de cruza: Es el porcentaje de individuos que se cruzarán.

La función esta diseñada de tal forma que crea una nueva generación de individuos a partir de la población actual. Esta toma tres argumentos como entrada: la población, las parejas y el porcentaje de cruza.

La función inicia creando una lista de hijos vacía, donde se almacenarán los hijos generados. Después, la función entra en un ciclo while que se ejecuta hasta que el iterador sea menor que el número de parejas. Dentro del ciclo genera números aleatorios entre 0 y 1, si el número es menor al porcentaje de cruza, si el numero es menor o igual al porcentaje de cruza, se selecciona a los individuos de la población que se cruzarán y se cruzan.

Los padres seleccionados desde la población usando los indices proporcionados por la función de selección de parejas se cruzarán y se generaran dos hijos. Los nuevos individuos serás indexados a la lista de hijos, se incrementará el iterador por dos y se repetirá el proceso hasta que se hayan terminado los indices de selección.

Finalmente, después de que se hayan procesado toda la lista de parejas, la función devuelve la lista de hijos generados.

Función del algoritmo genético

```
    def genetic(f, dimension, lower_limit, upper_limit, poblation_size, crossover_percentage,
    mutation_percentage, generations):
        # Inicializar la poblacion
        poblation = start_poblation(f, poblation_size, dimension, lower_limit, upper_limit)
        print(f"Poblacion inicial: {poblation}")
        # Listas para la grafica de convergencia
        best, worse, average = [], [], []

        for generation in range(generations):
            print(f"Generacion {generation + 1}")
            # Seleccionar parejas
            couples = define_couples(poblation, poblation_size)
            print(f"Parejas seleccionadas: {couples}")

            # Generar hijos
            children = create_children(couples, poblation, crossover_percentage)
            print(f"Hijos generados: {children}")

            # Generar mutaciones
            mutated_children = []
            for child in children:
                mutated = mutation(child, 0.1, mutation_percentage)
                if f == "rosenbrock":
```

```
                    mutated_children.append([mutated, rosenbrock(mutated)])
                elif f == "ackley":
                    mutated_children.append([mutated, ackley(mutated)])

            print(f"Hijos mutados: {mutated_children}")

            # Unir la poblacion con los hijos
            new_poblation = poblation + mutated_children
            print(f"Nueva poblacion: {new_poblation}")

            # Ordenar la poblacion
            sorted_poblation = sorted(new_poblation, key=itemgetter(1))
            print(f"Poblacion ordenada: {sorted_poblation}")

            # Seleccionar los mejores individuos
            poblation = sorted_poblation[:poblation_size]
            print(f"Mejores individuos: {poblation}")

            # Calcular el mejor, peor y promedio de la poblacion
            best.append(poblation[0][1])
            worse.append(poblation[-1][1])
            average.append(sum([ind[1] for ind in poblation]) / poblation_size)

        print(f"Mejor individuo: {poblation[0]}")

        return best, worse, average

```

Listing 6: Funcion genetico

La función principal del algoritmo genético, esta función se encarga de ejecutar el algoritmo genético.

- f: Es una cadena que especifica la función de aptitud que se utilizará, ya sea Rosenbrock o Ackley.
- tamaño de la población: Es el número de individuos que se generarán.
- dimensión: Es el número de variables que tendrá cada individuo.
- limite inferior: Es el valor mínimo que pueden tomar las variables de cada individuo.
- limite superior: Es el valor máximo que pueden tomar las variables de cada individuo.
- porcentaje de cruza: Es el porcentaje de individuos que se cruzarán.
- porcentaje de mutación: Es el porcentaje de individuos que se mutarán.
- generaciones: Es el número de generaciones que se ejecutarán.

La función comienza inicializando una población de individuos utilizando la función de inicialización de población. Cada individuo de la población se evalúa utilizando la función de aptitud especificada. La función de aptitud devuelve un valor que representa la aptitud del individuo.

Posteriormente, para cada generación, evalúa cada generación con los siguientes pasos:

- Selecciona las parejas de individuos que se cruzarán utilizando la función de selección de acuerdo a su fitness, mediante el algoritmo de la ruleta.
- Crea una nueva generación de individuos a partir de la población actual utilizando la función de creación de hijos.
- Muta a los individuos de la nueva generación utilizando la función de mutación.
- Evalúa la nueva generación de individuos utilizando la función de aptitud.
- Ordena los individuos según su aptitud.
- Selecciona los mejores individuos para la siguiente generación.
- Guarda el mejor, el peor y la aptitud media de la generación para su graficación posterior.

Finalmente, terminando todas las generaciones, la función muestra los resultados del mejor individuo, el peor individuo y la aptitud media de cada generación.

Resultados

Función de Rosenbrock Real

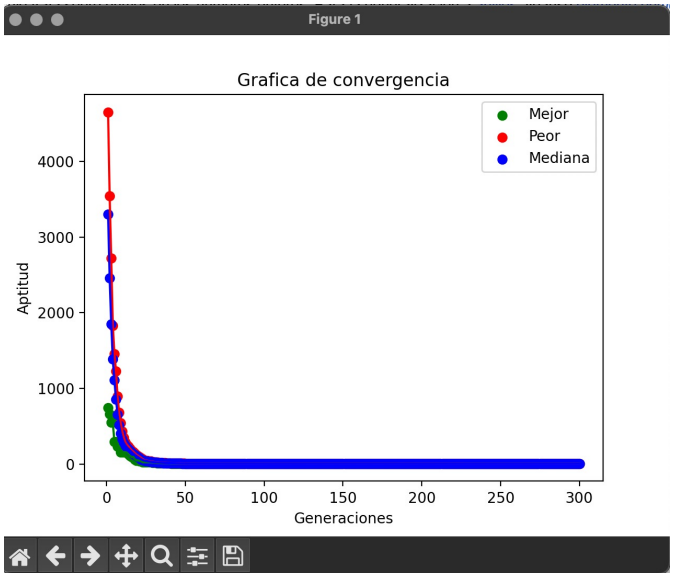

Figura 1: Rosenbrock, semilla= 7, Real

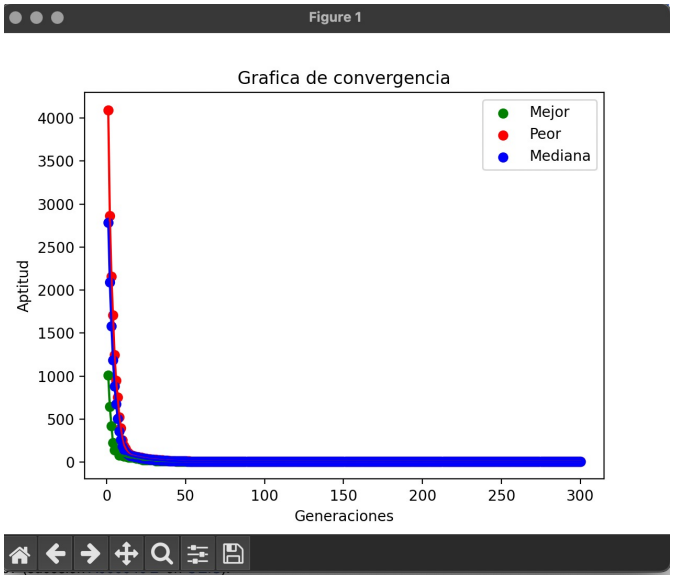

Figura 2: Rosenbrock, semilla= 11, Real

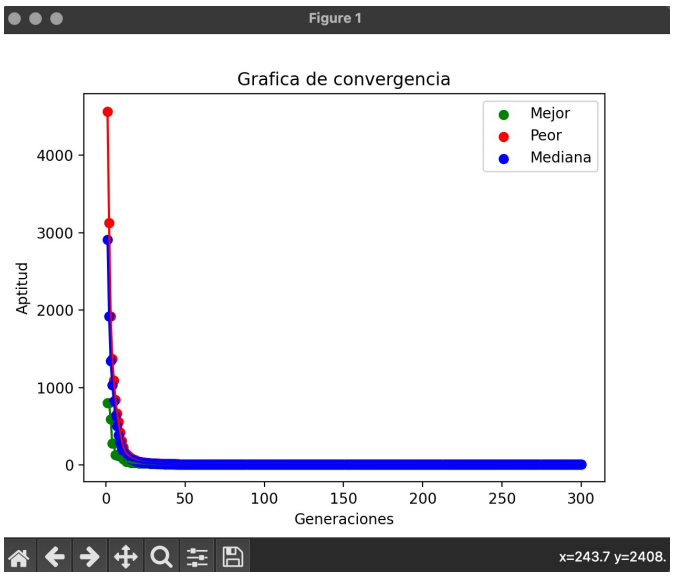

Figura 3: Rosenbrock, semilla= 61, Real

Función de Rosenbrock Binaria

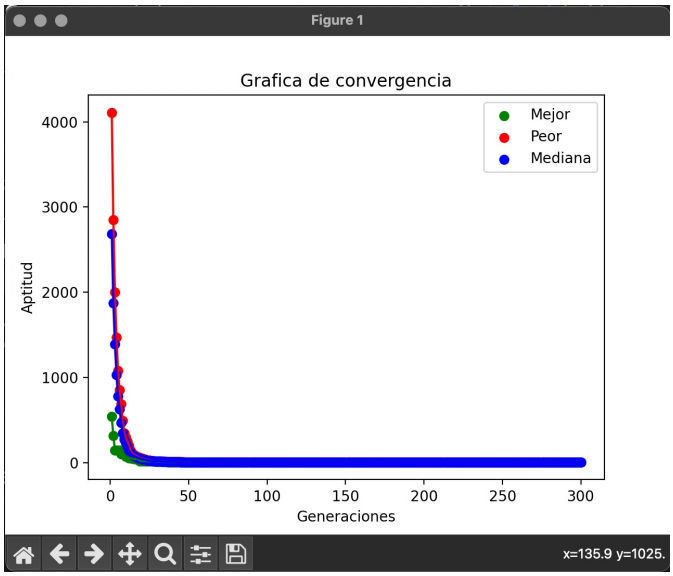

Figura 4: Rosenbrock, semilla= 59, Binaria

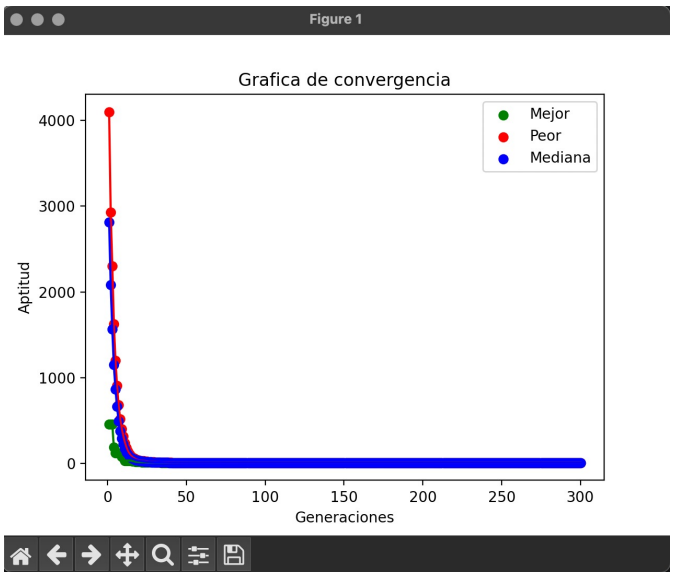

Figura 5: Rosenbrock, semilla= 79, Binaria

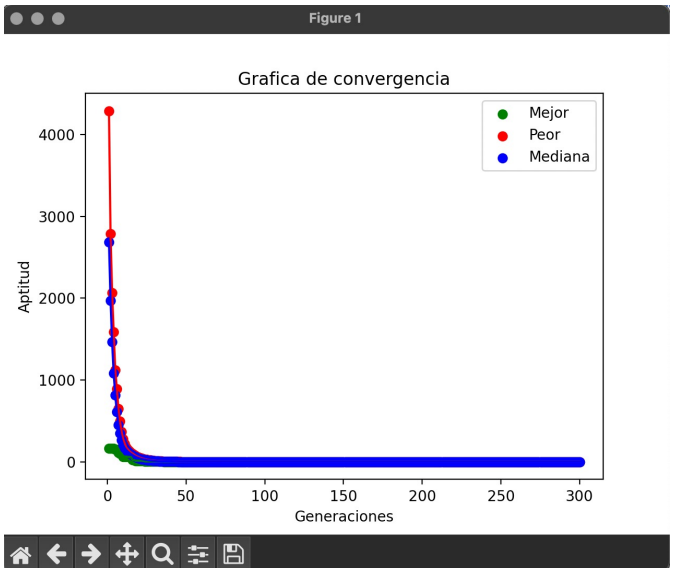

Figura 6: Rosenbrock, semilla= 97, Binaria

Función de Ackley Real

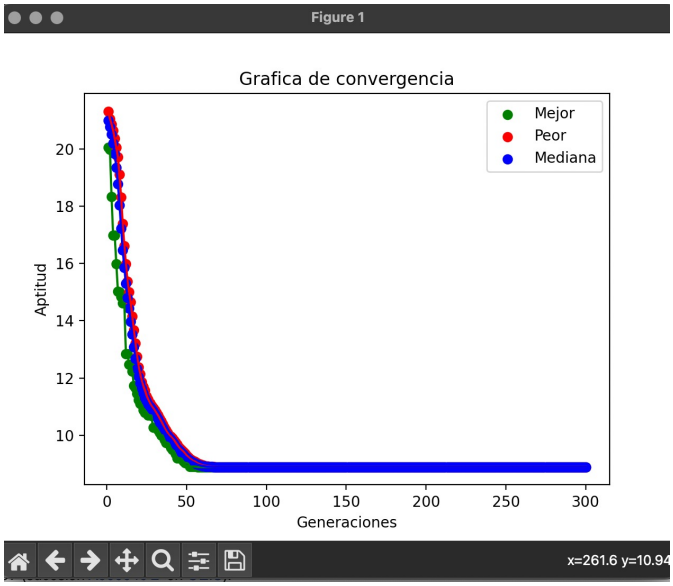

Figura 7: Ackley, semilla= 7, Real

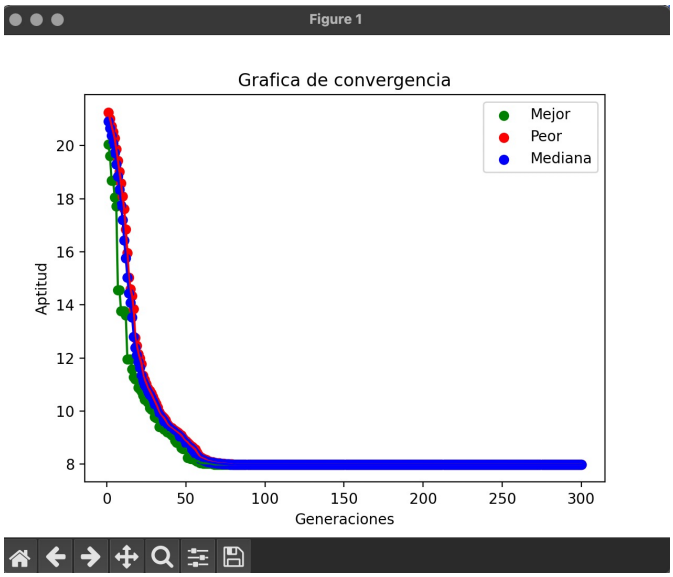

Figura 8: Ackley, semilla= 97, Real

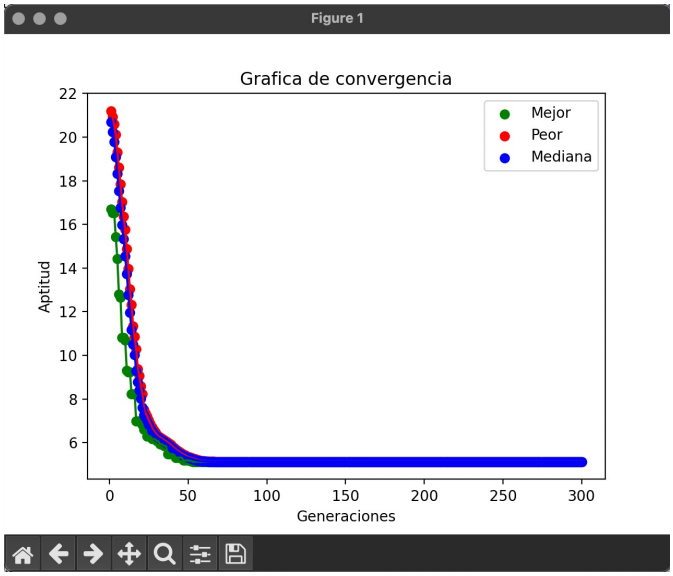

Figura 9: Ackley, semilla= 113, Real

Función de Ackley Binaria

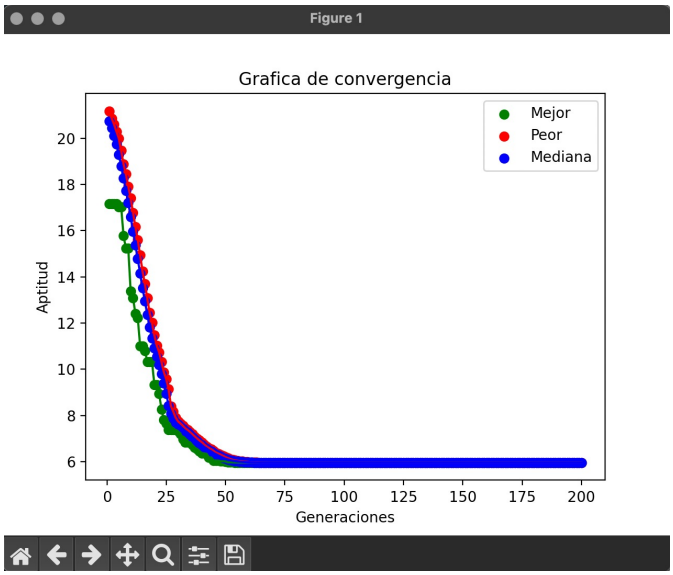

Figura 10: Ackley, semilla= 13, Binaria

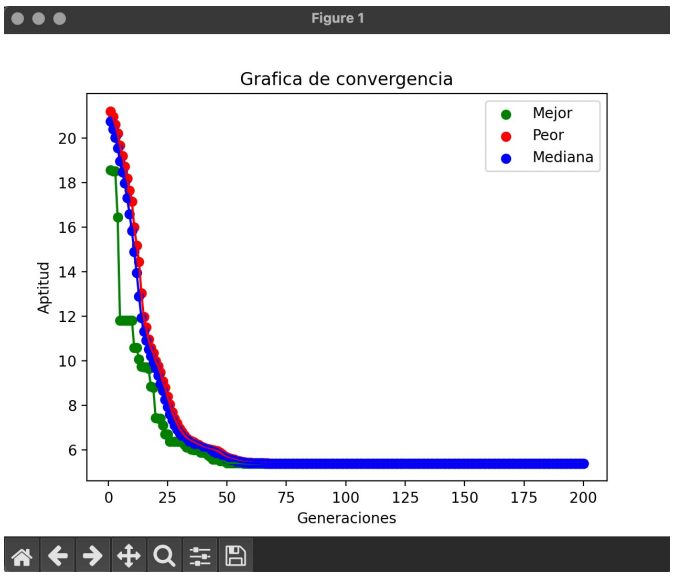

Figura 11: Ackley, semilla= 83, Binaria

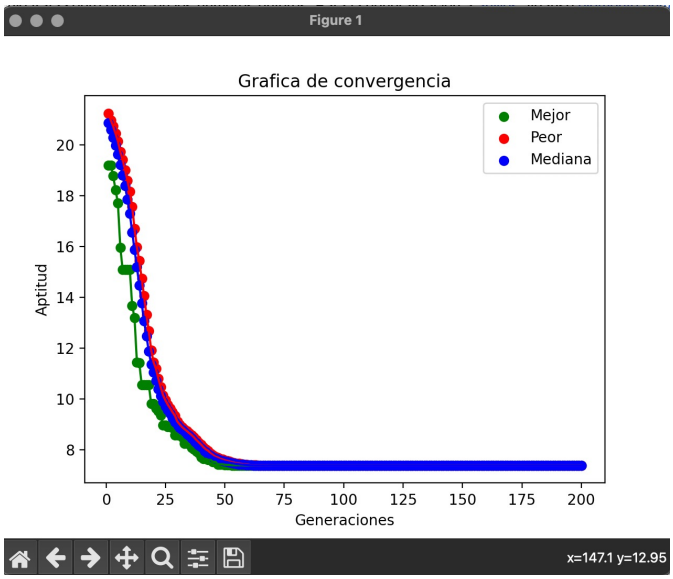

Figura 12: Ackley, semilla= 101, Binaria

Problema	Codificación	Mínimo	Máximo	Promedio	Desviación
Rosenbrock	Real	0.7570	1.1009	0.9826	0.0685
Rosenbrock	Binaria	-0.3456	1.1009	0.7826	0.0535
Ackley	Real	-10.3588	6.9370	2.87	0.0013
Ackley	Binaria	0.9357	17.8474	7.2671	0.045

Conclusiones

Esta practica nos permitió implementar un algoritmo genético para resolver problemas de optimización en espacios continuos, es muy interesante ver como el algoritmo genético es capaz de encontrar el mínimo de las funciones además de que podemos destacar el proceso de creación del código que nos ha sido muy educativo y nos ha permitido entender mejor el funcionamiento de estos algoritmos.