**Marco Teórico.**

En esta práctica se implementa un algoritmo genético para la optimización de funciones matemáticas complejas, utilizando dos funciones de prueba clásicas: la función de Ackley y la función de Rastrigin. Estas funciones son comúnmente empleadas para evaluar el rendimiento de los algoritmos de optimización debido a sus características de multimodalidad, es decir, la presencia de múltiples mínimos locales que dificultan la convergencia hacia el óptimo global.

Un algoritmo genético es un algoritmo de optimización inspirado en los principios de la evolución natural, como la selección, el cruce y la mutación. En este enfoque, una población de posibles soluciones (cromosomas) evoluciona con el tiempo para mejorar sus características según una función de aptitud (fitness). Cada cromosoma está representado por una cadena binaria que codifica una solución potencial al problema.

En el contexto de esta práctica, se utilizan las siguientes funciones de prueba:

* Función de Ackley: Esta función tiene múltiples mínimos locales y es utilizada para probar la capacidad de un algoritmo para escapar de estos puntos y encontrar el mínimo global. La función se define en dos dimensiones y su expresión matemática incluye un término exponencial y uno trigonométrico, lo que la hace especialmente desafiante para algoritmos de búsqueda local.
* Función de Rastrigin: Esta función es conocida por su gran cantidad de oscilaciones y mínimos locales, lo que dificulta la optimización. Se define en tres dimensiones y combina componentes cuadráticos con funciones coseno, lo cual aumenta su complejidad y hace que los algoritmos de optimización tiendan a quedarse atrapados en mínimos locales.

Los principales operadores utilizados en el algoritmo genético de esta práctica son:

1. Selección: Se emplea el método de selección por ruleta para elegir los cromosomas que participarán en el proceso de reproducción. Este método favorece a los cromosomas con mejor fitness, pero también da oportunidades a los menos aptos para mantener la diversidad en la población.
2. Cruce (Crossover): Se realiza un cruce de un solo punto, donde se selecciona aleatoriamente un punto en los cromosomas de los padres y se intercambian segmentos para generar dos descendientes. Esto permite la combinación de características de los padres, promoviendo la exploración del espacio de búsqueda.
3. Mutación: Con una probabilidad del 10%, se mutan algunos bits de los cromosomas. La mutación introduce variabilidad aleatoria en la población, ayudando al algoritmo a evitar la convergencia prematura hacia óptimos locales.

El objetivo principal de esta práctica es explorar la capacidad del algoritmo genético para encontrar soluciones cercanas al óptimo global en funciones que presentan retos significativos debido a su estructura compleja. Se busca demostrar la eficacia de la evolución basada en selección, cruce y mutación para espacios de búsqueda con mínimos locales.

**Material y Equipo**

* **Hardware:** Computadora con capacidad para ejecutar Python.
* **Software:**
* Python 3.x
* Google Colab.

**Desarrollo de la practica**

En esta práctica se resolverán dos funciones de prueba clásicas en el ámbito de la optimización: la función de Ackley y la función de Rastrigin. La función de Ackley se caracteriza por tener múltiples mínimos locales que representan un desafío para encontrar el óptimo global, mientras que la función de Rastrigin tiene una gran cantidad de oscilaciones y mínimos locales que dificultan la optimización.

*1. Optimización de la Función de Ackley.*

* Inicialización de Cromosomas: Se generaron cromosomas aleatorios con una longitud fija de 16 bits. La población inicial constó de 10 cromosomas.
* Decodificación de Cromosomas: Los cromosomas fueron decodificados para obtener valores reales que correspondieran al espacio de búsqueda de la función de Ackley. Se utilizó una escala lineal para mapear los bits a valores dentro del rango definido entre -20 y 20.
* Evaluación de la Población: Se evaluaron los cromosomas utilizando la función de Ackley. Para cada cromosoma, se calcularon los valores de fitness de acuerdo con la definición de la función.
* Selección por Ruleta: Se utilizó una estrategia de selección por ruleta para elegir a los padres que participarían en el proceso de cruce y mutación. La selección se basó en el fitness relativo de cada cromosoma.
* Cruce y Mutación: Se aplicó un operador de cruce de un solo punto aleatorio para generar dos nuevos descendientes. Además, se aplicó una probabilidad de mutación del 10% para introducir variación en la población.
* Generación de Nuevas Poblaciones: Se implementó un esquema de elitismo que garantizaba que los dos mejores cromosomas de cada generación se mantuvieran en la siguiente. Las generaciones subsecuentes fueron producidas mediante selección, cruce y mutación hasta alcanzar un número adecuado de iteraciones.
* Resultados Obtenidos: Se observó que el algoritmo logró minimizar la función de Ackley de manera exitosa después de varias generaciones, encontrando un óptimo global cercano al valor teórico.

*Optimización de la Función de Rastrigin.*

* Inicialización de Cromosomas: Se generaron cromosomas aleatorios con una longitud fija de 16 bits. La población inicial constó de 10 cromosomas.
* Decodificación de Cromosomas: Los cromosomas fueron decodificados para obtener valores reales que correspondieran al espacio de búsqueda de la función de Rastrigin. Se utilizó una escala lineal para mapear los bits a valores dentro del rango definido entre -20 y 20.
* Evaluación de la Población: Se evaluaron los cromosomas utilizando la función de Rastrigin. Para cada cromosoma, se calcularon los valores de fitness de acuerdo con la definición de la función.
* Selección por Ruleta: Se utilizó una estrategia de selección por ruleta para elegir a los padres que participarían en el proceso de cruce y mutación. La selección se basó en el fitness relativo de cada cromosoma.
* Cruce y Mutación: Se aplicó un operador de cruce de un solo punto aleatorio para generar dos nuevos descendientes. Además, se aplicó una probabilidad de mutación del 10% para introducir variación en la población.
* Generación de Nuevas Poblaciones: Se implementó un esquema de elitismo que garantizaba que los dos mejores cromosomas de cada generación se mantuvieran en la siguiente. Las generaciones subsecuentes fueron producidas mediante selección, cruce y mutación hasta alcanzar un número adecuado de iteraciones.
* Resultados Obtenidos: En la función de Rastrigin, debido a la gran cantidad de mínimos locales, se observó que el algoritmo a menudo convergía a soluciones subóptimas, demostrando la dificultad de esta función.

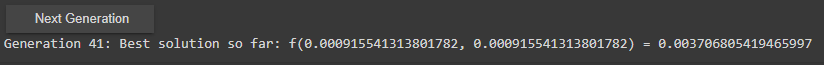
**Diagramas, gráficas y pantallas.**

**Figura 1**. Resultado de la primera generación en la optimización de la función de Ackley. Se observa que, en la primera generación, la solución está lejos del mínimo global esperado.



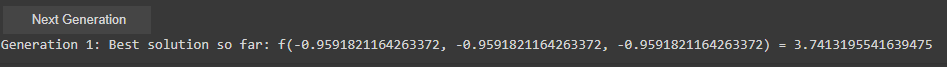
**Figura 1.**

**Figura 2**. Evolución en la generación 41 de la optimización de la función de Ackley. Se muestra que el algoritmo ha mejorado significativamente y se ha acercado al mínimo global



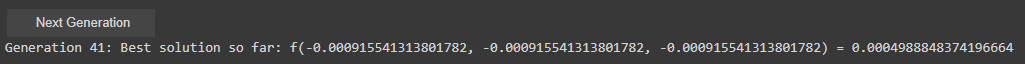
**Figura 2.**

**Figura 3**. Resultado de la primera generación en la optimización de la función de Rastrigin. Se observa que en la primera generación, la solución está lejos del mínimo global esperado.



**Figura 3.**

**Figura 4**. Evolución en la generación 41 de la optimización de la función de Rastrigin. Se muestra que el algoritmo ha mejorado significativamente y se ha acercado al mínimo global.



**Figura 4.**

**Conclusiones**

En esta práctica se implementó un algoritmo genético para optimizar funciones matemáticas con características complejas y múltiples mínimos locales. Se observó que el algoritmo genético fue efectivo para encontrar soluciones cercanas al óptimo global en la función de Ackley, mostrando una buena capacidad para explorar el espacio de búsqueda y escapar de mínimos locales. Sin embargo, en la función de Rastrigin, el algoritmo presentó dificultades debido a la gran cantidad de oscilaciones y mínimos locales, lo que lo llevó a converger a soluciones subóptimas en varios casos.

**Bibliografía**

**Genetic Algorithm for Function Optimization - GeeksforGeeks**

* **Autor**: Equipo de GeeksforGeeks
* **Fecha**: 2024
* **Fuente**: <https://www.geeksforgeeks.org/genetic-algorithm/>

**Introduction to Genetic Algorithms - Tutorialspoint**

* **Autor**: Equipo de Tutorialspoint
* **Fecha**: 2024
* **Fuente**: https://www.tutorialspoint.com/genetic\_algorithms/index.htm

**Anexo:**

Código Ackley's and Rastrigin's Functions:

import ipywidgets as widgets

import math

import random

import numpy as np

from functools import cmp\_to\_key

from IPython import display as display

# Parámetros generales

L\_chromosome = 16  # Longitud del cromosoma (bits)

N\_chromosomes = 10  # Número de cromosomas

prob\_m = 0.1  # Probabilidad de mutación

a = -20  # Límite inferior del espacio de búsqueda

b = 20  # Límite superior del espacio de búsqueda

crossover\_point = L\_chromosome // 2  # Punto de cruce

Lwheel = N\_chromosomes \* 10  # Tamaño de la ruleta

n = 0  # Contador de generaciones

# Inicialización de cromosomas

def random\_chromosome*()*:

    return [random.randint(0, 1) for \_ in range(L\_chromosome)]

# Decodificación de cromosomas binarios a valores reales

def decode\_chromosome*(chromosome)*:

    value = 0

    for p in range(L\_chromosome):

        value += (2 \*\* p) \* chromosome[-1 - p]

    return a + (b - a) \* float(value) / (2\*\*L\_chromosome - 1)

# Función de Ackley en 2D

def ackley\_function*(x, y)*:

    term1 = -20 \* math.exp(-0.2 \* math.sqrt(0.5 \* (x\*\*2 + y\*\*2)))

    term2 = -math.exp(0.5 \* (math.cos(2 \* math.pi \* x) + math.cos(2 \* math.pi \* y)))

    return term1 + term2 + 20 + math.e

# Función de Rastrigin en 3D

def rastrigin\_function*(x, y, z)*:

    return 30 + (x\*\*2 - 10 \* math.cos(2 \* math.pi \* x)) + \

           (y\*\*2 - 10 \* math.cos(2 \* math.pi \* y)) + \

           (z\*\*2 - 10 \* math.cos(2 \* math.pi \* z))

# Evaluación de cromosomas para Ackley

def evaluate\_ackley\_population*(population)*:

    fitness\_values = []

    for chrom in population:

        x = decode\_chromosome(chrom)

        y = decode\_chromosome(chrom)

        fitness = ackley\_function(x, y)

        fitness\_values.append(fitness)

    return fitness\_values

# Evaluación de cromosomas para Rastrigin

def evaluate\_rastrigin\_population*(population)*:

    fitness\_values = []

    for chrom in population:

        x = decode\_chromosome(chrom)

        y = decode\_chromosome(chrom)

        z = decode\_chromosome(chrom)

        fitness = rastrigin\_function(x, y, z)

        fitness\_values.append(fitness)

    return fitness\_values

# Selección por ruleta

def create\_wheel*(fitness\_values)*:

    max\_fitness = max(fitness\_values)

    acc = 0

    for fv in fitness\_values:

        acc += max\_fitness - fv

    fraction = [(max\_fitness - fv) / acc for fv in fitness\_values]

    wheel = []

    pc = 0

    for f in fraction:

        Np = int(f \* Lwheel)

        for i in range(Np):

            wheel.append(pc)

        pc += 1

    return wheel

# Operador de cruce y mutación

def crossover\_and\_mutate*(parent1, parent2)*:

    crossover\_point = random.randint(1, L\_chromosome - 2)

    offspring1 = parent1[:crossover\_point] + parent2[crossover\_point:]

    offspring2 = parent2[:crossover\_point] + parent1[crossover\_point:]

    # Mutación

    for i in range(L\_chromosome):

        if random.random() < prob\_m:

            offspring1[i] ^= 1

        if random.random() < prob\_m:

            offspring2[i] ^= 1

    return offspring1, offspring2

# Creación de la nueva generación

def next\_generation*(population, fitness\_function)*:

    fitness\_values = fitness\_function(population)

    population.sort(key=lambda chrom: fitness\_function([chrom])[0])

    # Selección de los mejores

    new\_population = [population[0], population[1]]  # Elitismo

    wheel = create\_wheel(fitness\_values)

    while len(new\_population) < N\_chromosomes:

        p1 = random.choice(wheel)

        p2 = random.choice(wheel)

        offspring1, offspring2 = crossover\_and\_mutate(population[p1], population[p2])

        new\_population.append(offspring1)

        new\_population.append(offspring2)

    return new\_population[:N\_chromosomes]

# Botón para avanzar generaciones

def create\_button*()*:

    button = widgets.Button(

        description='Next Generation',

        disabled=False,

        button\_style='',  # 'success', 'info', 'warning', 'danger'

        tooltip='Next Generation',

        icon='check'  # Icono (FontAwesome)

    )

    return button

# Función para ejecutar la siguiente generación (Ackley)

def next\_generation\_ackley\_callback*(b)*:

    global population\_ackley, n

    display.clear\_output(wait=True)

    display.display(button\_ackley)

    population\_ackley = next\_generation(population\_ackley, evaluate\_ackley\_population)

    n += 1

    # Mostrar los mejores cromosomas y el valor de fitness

    best\_chrom = population\_ackley[0]

    x\_best = decode\_chromosome(best\_chrom)

    y\_best = decode\_chromosome(best\_chrom)

    print(f"Generation {n}: Best solution so far: f({x\_best}, {y\_best}) = {ackley\_function(x\_best, y\_best)}")

# Función para ejecutar la siguiente generación (Rastrigin)

def next\_generation\_rastrigin\_callback*(b)*:

    global population\_rastrigin, n

    display.clear\_output(wait=True)

    display.display(button\_rastrigin)

    population\_rastrigin = next\_generation(population\_rastrigin, evaluate\_rastrigin\_population)

    n += 1

    # Mostrar los mejores cromosomas y el valor de fitness

    best\_chrom = population\_rastrigin[0]

    x\_best = decode\_chromosome(best\_chrom)

    y\_best = decode\_chromosome(best\_chrom)

    z\_best = decode\_chromosome(best\_chrom)

    print(f"Generation {n}: Best solution so far: f({x\_best}, {y\_best}, {z\_best}) = {rastrigin\_function(x\_best, y\_best, z\_best)}")

# Crear el botón para Ackley

button\_ackley = create\_button()

button\_ackley.on\_click(next\_generation\_ackley\_callback)

# Crear el botón para Rastrigin

button\_rastrigin = create\_button()

button\_rastrigin.on\_click(next\_generation\_rastrigin\_callback)

# Inicialización de las poblaciones

population\_ackley = [random\_chromosome() for \_ in range(N\_chromosomes)]

population\_rastrigin = [random\_chromosome() for \_ in range(N\_chromosomes)]

# Mostrar el botón para Ackley

display.display(button\_ackley)

# Mostrar el botón para Rastrigin

display.display(button\_rastrigin)