INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO
INTELIGENCIA ARTIFICIAL
ING. EN SISTEMAS COMPUTACIONALES

Laboratorio 1: Búsqueda aleatoria

Integrantes:

Hurtado Morales Emiliano - 2021630390

Maestro: Andrés García Floriano

Grupo: 6CV3

Fecha de entrega: 09/09/2024

Ciclo Escolar: 2024 – 2

Introducción

La presente práctica tiene como objetivo principal el desarrollo y análisis de dos aplicaciones utilizando el lenguaje de programación Python. El primero de estos problemas aborda la optimización de funciones matemáticas mediante un enfoque estocástico, específicamente, la búsqueda aleatoria. Este método se emplea para encontrar el valor mínimo de una función no lineal en un espacio de búsqueda dado. La optimización de funciones es un tema fundamental en muchas áreas de las matemáticas aplicadas, la ingeniería y la ciencia de datos, donde se busca maximizar o minimizar funciones objetivo para tomar decisiones óptimas en escenarios complejos.

El segundo problema explora la creación de un juego de gato (también conocido como tic-tac-toe) en un tablero de 4x4, implementado con una interfaz gráfica de usuario (GUI) utilizando la biblioteca tkinter de Python. El desarrollo de interfaces gráficas es un aspecto crucial en la ingeniería de software, ya que permite a los usuarios interactuar de manera intuitiva con aplicaciones complejas. El juego de gato es un clásico problema de lógica y estrategia, y su extensión a un tablero de 4x4 añade un nivel adicional de dificultad y requiere una adaptación de las reglas y verificaciones tradicionales.

Ambos problemas se abordan con un enfoque práctico, permitiendo a los estudiantes aplicar conocimientos teóricos en la resolución de problemas reales. El uso de Python para la optimización y el desarrollo de interfaces gráficas pone de manifiesto la versatilidad de este lenguaje en el ámbito académico y profesional. La práctica no solo refuerza conceptos de programación, sino que también permite explorar técnicas de optimización, diseño de algoritmos y creación de experiencias de usuario interactivas.

Explicación de los Algoritmos

1. Optimización de Funciones mediante Búsqueda Aleatoria

El primer programa se centra en la optimización de la función:

$$f(x,y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2$$

Con

$$-4.5 \leq x, y \leq 4.5$$

Búsqueda Aleatoria: La búsqueda aleatoria es un algoritmo de optimización estocástico que consiste en evaluar la función objetivo en un conjunto de puntos generados aleatoriamente dentro del espacio de búsqueda. Este método es simple, pero puede ser efectivo para problemas donde la función objetivo es complicada y no se dispone de un gradiente claro o métodos determinísticos eficientes.

Pasos del algoritmo:

1. **Inicialización:** Se establece un número de iteraciones (10,000 en este caso) y un rango de búsqueda para *x* y *y*.

```
# Definimos la función objetivo
def funcion_objetivo(x, y):
    return (1.5 - x + x * y)**2 + (2.25 - x + x * y**2)**2 + (2.625 - x + x *
y**3)**2

# Parámetros de la búsqueda aleatoria
iteraciones = 10000
rango_min = -4.5
rango_max = 4.5

# Inicialización
mejor_valor = float('inf')
mejores_variables = None
```

2. **Generación Aleatoria:** En cada iteración, se generan valores aleatorios para *x* y *y* dentro del rango especificado.

```
# Búsqueda aleatoria
for i in range(iteraciones):
    # Generar valores aleatorios para x y y
    x = np.random.uniform(rango_min, rango_max)
    y = np.random.uniform(rango_min, rango_max)
```

3. **Evaluación:** Se calcula el valor de la función objetivo *f (x,y)* para cada par *(x,y)*.

```
    valor = funcion_objetivo(x, y)
```

4. **Selección:** Se guarda el menor valor encontrado junto con los valores de *x* y *y* que lo produjeron.

```
    # Actualizar si encontramos un valor mejor
    if valor < mejor_valor:
        mejor_valor = valor
        mejores_variables = (x, y)
```

5. **Resultado:** Al final de las iteraciones, se reporta el valor mínimo encontrado y los valores correspondientes de *x* y *y*.

```
print(f"Mejor valor encontrado: {mejor_valor}")
print(f"Valores de x y y: {mejores_variables}")
```

Ventajas y Limitaciones:

- **Ventajas:** Es fácil de implementar y no requiere información adicional sobre la función (como su derivada).
- **Limitaciones:** Puede ser ineficiente para funciones con muchos picos o valles debido a la naturaleza aleatoria del algoritmo y la alta posibilidad de no encontrar el mínimo global.

2. Juego de Gato 4x4 con Interfaz Gráfica

El segundo programa consiste en la implementación de un juego interactivo de gato en un tablero de 4x4, utilizando la biblioteca tkinter para la GUI.

Descripción del Juego: El gato (o tic-tac-toe) es un juego clásico donde dos jugadores se turnan para colocar su símbolo (X u O) en un tablero, intentando formar una línea de tres símbolos iguales en horizontal, vertical o diagonal. En este caso, se extiende el tablero a 4x4, aumentando la dificultad y el espacio de juego.

Interfaz Gráfica:

- **Tablero:** Se crea un tablero de 4x4 donde cada celda es un botón que los jugadores pueden presionar para realizar su movimiento.

```
class JuegoGato:
    def __init__(self, root):
        self.root = root
        self.root.title("Juego de Gato 4x4")
        self.crear_widgets()
        self.iniciar_juego()

    def crear_widgets(self):
        self.tablero_frame = tk.Frame(self.root)
        self.tablero_frame.grid(row=0, column=0, columnspan=4)

        self.botones = [[None for _ in range(4)] for _ in range(4)]
        for i in range(4):
            for j in range(4):
                boton = tk.Button(self.tablero_frame, text=" ", font=("Arial",
20), width=5, height=2,
                                  command=lambda i=i, j=j:
self.movimiento_jugador(i, j))
                boton.grid(row=i, column=j)
                self.botones[i][j] = boton
```

- **Movimientos de la Computadora:** Los movimientos de la computadora se generan aleatoriamente, lo que añade un nivel de desafío para el jugador humano.

```
   def movimiento_computadora(self):
        while True:
            fila = random.randint(0, 3)
            columna = random.randint(0, 3)
            if self.tablero[fila][columna] == " ":
                self.tablero[fila][columna] = "O"
                self.botones[fila][columna].config(text="O")
```

```
                break
        if self.verificar_ganador("O"):
            self.finalizar_juego("¡La computadora ganó!")
        elif self.tablero_lleno():
            self.finalizar_juego("¡Es un empate!")
        self.turno_jugador = True
```

- **Verificación de Ganador:** Después de cada movimiento, el programa verifica si hay un ganador o si el tablero está lleno, indicando el final del juego.

```
    def verificar_ganador(self, jugador):
        # Verificar filas y columnas
        for i in range(4):
            if all(self.tablero[i][j] == jugador for j in range(4)) or \
               all(self.tablero[j][i] == jugador for j in range(4)):
                return True
        # Verificar diagonales
        if all(self.tablero[i][i] == jugador for i in range(4)) or \
           all(self.tablero[i][3 - i] == jugador for i in range(4)):
            return True
        return False
```

- **Botón de Reinicio:** Se incluye un botón de "Reiniciar" para permitir a los jugadores comenzar una nueva partida sin necesidad de cerrar la aplicación.

```
    def reiniciar_juego(self):
        # Limpiar el mensaje final si existe
        for widget in self.root.grid_slaves(row=2):
            widget.destroy()
        # Reiniciar el juego
        self.iniciar_juego()
```

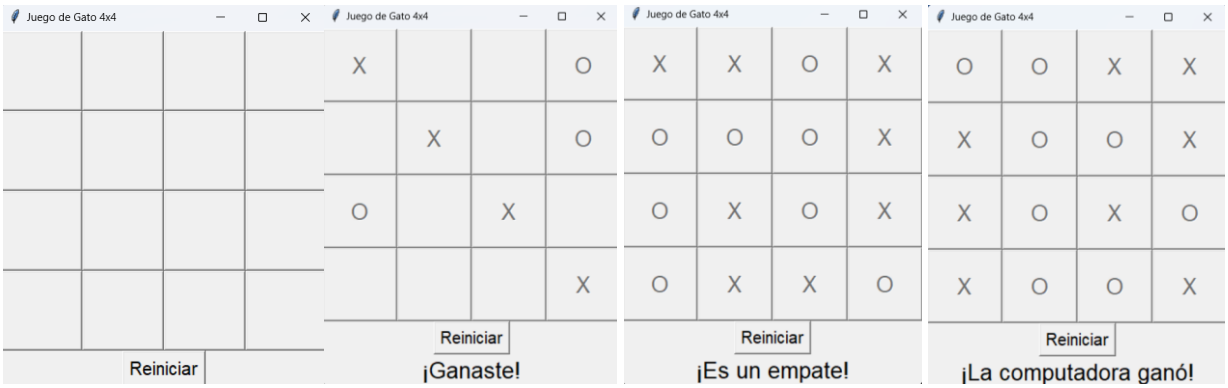
Expansión del Algoritmo: El algoritmo se amplía para manejar un tablero de mayor tamaño (4x4) y para verificar el estado del juego después de cada turno. La implementación de una interfaz gráfica mejora la experiencia del usuario al permitir una interacción más intuitiva y visual con el juego.

Resultados Esperados

1. **Optimización mediante Búsqueda Aleatoria:**
    - Se espera encontrar valores de *x* y *y* que minimicen la función *f (x, y)* dentro del rango dado. Aunque la búsqueda aleatoria no garantiza encontrar el mínimo global, debería aproximarse a un valor bajo si se realizan suficientes iteraciones.
    - Los resultados incluirán el valor mínimo de la función y los valores correspondientes de *x* y *y*, demostrando la efectividad del algoritmo dentro de las limitaciones de la búsqueda aleatoria.

```
PS D:\ESCOM\9 Semestre\IA\Bender-IA\Lab_1> python Lab_1_1.py
Mejor valor encontrado: 0.003582539981698839
Valores de x y y: (3.118063384916815, 0.5200864593715622)
PS D:\ESCOM\9 Semestre\IA\Bender-IA\Lab_1> python Lab_1_1.py
Mejor valor encontrado: 0.0028733207279947465
Valores de x y y: (3.139885985426515, 0.5352834003441931)
PS D:\ESCOM\9 Semestre\IA\Bender-IA\Lab_1> python Lab_1_1.py
Mejor valor encontrado: 0.0044111136307662348
Valores de x y y: (3.1728153487130015, 0.5359911973974398)
PS D:\ESCOM\9 Semestre\IA\Bender-IA\Lab_1> python Lab_1_1.py
Mejor valor encontrado: 0.0017417814366398 67
Valores de x y y: (2.9394386749816546, 0.4773162350185123)
PS D:\ESCOM\9 Semestre\IA\Bender-IA\Lab_1> python Lab_1_1.py
Mejor valor encontrado: 4.086968304788804e-05
Valores de x y y: (2.9927293545034255, 0.4970004481069834)
```

2. **Juego de Gato 4x4:**
    - El juego debe funcionar correctamente, permitiendo al jugador humano y a la computadora realizar movimientos alternados en el tablero de 4x4.
    - El programa debe identificar correctamente al ganador (si lo hay) o indicar un empate cuando el tablero esté lleno.
    - La interfaz gráfica permitirá reiniciar el juego sin problemas, mejorando la usabilidad y la experiencia del usuario.



Conclusión

En esta práctica, se han logrado dos objetivos clave: la optimización de una función no lineal mediante un enfoque de búsqueda aleatoria y la implementación de un juego de gato en un tablero 4x4 utilizando una interfaz gráfica en Python.

La búsqueda aleatoria demostró ser un método sencillo pero efectivo para aproximar el valor mínimo de una función en un espacio de búsqueda definido. Aunque este método no garantiza encontrar el mínimo global, es útil en situaciones donde el espacio de búsqueda es complejo o no se dispone de un gradiente. Los resultados obtenidos confirman que, con un número suficiente de iteraciones, es posible acercarse a soluciones óptimas, lo que resalta la aplicabilidad del enfoque en problemas de optimización con características similares.

Por otro lado, la implementación del juego de gato 4x4 con tkinter no solo permitió poner en práctica conceptos de programación orientada a eventos y diseño de interfaces, sino que también brindó la oportunidad de explorar la lógica detrás de juegos de estrategia. La expansión del juego a un tablero de 4x4 introdujo desafíos adicionales en la verificación de las condiciones de victoria y en la gestión de la interfaz gráfica, los cuales fueron abordados con éxito. La adición de un botón de "Reset" mejoró significativamente la usabilidad del programa, ofreciendo una experiencia de usuario más completa y satisfactoria.

En resumen, la práctica no solo consolidó habilidades técnicas en la programación de algoritmos y diseño de interfaces, sino que también subrayó la importancia de la creatividad y la atención al detalle en el desarrollo de soluciones efectivas y amigables para el usuario. Estas habilidades son esenciales en el campo de la ingeniería de software y demuestran la capacidad de los estudiantes para abordar y resolver problemas complejos utilizando herramientas modernas.