Programa 4: Algoritmo Genético

Gil Ramírez Bruno
Moguel Silva José Miguel
Ramirez Santamaria Isaí
Zacarias Daniel Luis Alberto

29 de Mayo de 2023

Contents

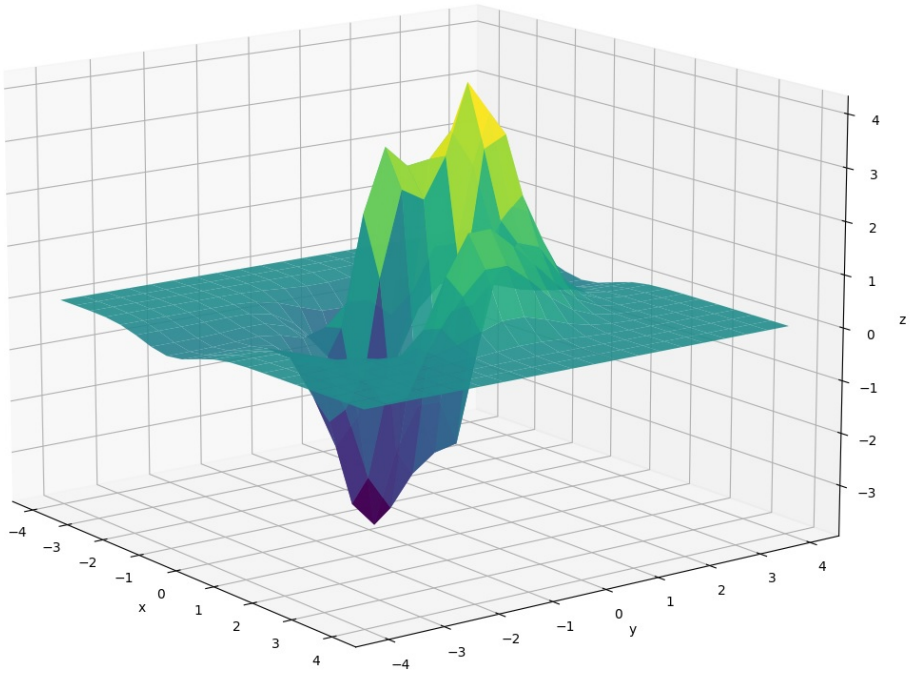
---

1 Planteamiento del problema

Diseñar un algoritmo genético que explore el espacio de búsqueda en la región determinada por los puntos (-4,-4),(-4,4),(4,4)y(4,-4) de la siguiente función:

$$z = e^{-(y+1)^2-x^2}(x-1)^2 - \frac{e^{-(x+1)^2}}{3} + e^{-x^2-y^2}(10x^3 - 10x + 10y^3)$$

1. Generar una población de 10 individuos aleatorios dentro del espacio de búsqueda definido.
2. Diseñar mínimo dos operaciones de cruza. Y usar el método de selección de ruleta.
3. Diseñar una operación de mutación (que no se escape del espacio definido).
4. Usar elitismo de los mejores dos individuos.



5. Pintar con diferente color cada vez más intenso los hijos de la nueva generación dentro de la gráfica.

6. Solicitar n generaciones (el usuario las debe introducir).

7. Al final devolver la mejor de las soluciones de la última generación y pintarla de color rojo.

8. Consideraciones importantes:

   (a) La función de aptitud es la z=F(x,y).

   (b) Los individuos deben contener valores decimales, pues con valores enteros la búsqueda quedaría limitada.

2 Metodología

2.1 Diseño de las Funciones de Cruza

2.1.1 crossover_one

La función ***crossover_one*** implementa un método de cruce conocido como **cruza promedio**. Esta técnica de cruce se utiliza para combinar las características de dos individuos (denominados padres) y generar un nuevo individuo (denominado hijo) en la población. La cruza promedio se basa en el promedio de los valores de los genes de los padres para generar los genes del hijo. La razón de seleccionar este método de cruce, en este caso, se debe a que la cruza promedio tiende a generar descendencia que se encuentra cerca del centro geométrico de los padres. Esto puede ser útil en problemas de optimización cuando se busca explorar el espacio de búsqueda de manera equilibrada. Además, el uso de un método de cruce simple como este puede ayudar a mantener la diversidad genética en la población, evitando convergencia prematura hacia soluciones subóptimas.

1. Se obtienen las coordenadas **X** e **Y** de los padres ***parent1*** y ***parent2***. Luego, se realiza el cálculo del promedio de las coordenadas **X** y **Y** por separado, utilizando la fórmula:

$$x = \frac{x_1 + x_2}{2}$$
$$y = \frac{y_1 + y_2}{2}$$

   Donde **x1**, **y1** son las coordenadas **X** e **Y** del ***parent1***, y **x2**, **y2** son las coordenadas **X** e **Y** del ***parent2***.

2. Después de calcular las coordenadas del hijo, se calcula su aptitud utilizando una función de aptitud denominada ***fitness***. El valor de aptitud resultante se asigna a la variable **z**.

3. Finalmente, se crea un diccionario llamado ***individual*** que contiene las coordenadas **X** e **Y** del hijo, junto con su valor de aptitud **Z**. Este diccionario se devuelve como resultado de la función de cruce.

2.1.2 crossover_two

Esta fundió implementa un método de cruce conocido como **cruza en un punto**. La razón de seleccionar este método de cruce puede deberse a que la cruza en un punto permite la combinación de características de los padres en una región específica del espacio de búsqueda. Esto puede ser útil cuando se sospecha que ciertas características beneficiosas están concentradas en una determinada posición o rango de genes. Además, este método de cruce es simple de implementar y puede mantener cierta diversidad genética en la población al permitir la transferencia

directa de genes de un padre a un hijo.

1. Se obtienen las coordenadas **X** e **Y** de los padres ***parent1*** y ***parent2***. Luego, se genera un punto de corte aleatorio, representado por la variable ***cutoff***. El valor de ***cutoff*** se selecciona al azar entre 1 y 2, lo que indica que la cruza se realizará en el primer o segundo punto de corte.

2. Se realiza la cruza en el punto de corte determinado. Si ***cutoff*** es igual a **1**, se asigna el valor de **x1** a **x** y el valor de **y2** a **y**. Si ***cutoff*** es igual a **2**, se asigna el valor de **x2** a **x** y el valor de **y1** a **y**. En esencia, se intercambian los valores de las coordenadas **x** e **y** de los padres en el punto de corte.

3. Se calcula la aptitud del hijo utilizando una función de aptitud llamada ***fitness***. El valor de aptitud resultante se asigna a la variable **z**.

4. Finalmente, se crea un diccionario llamado ***individual*** que contiene las coordenadas **X** e **Y** del hijo, junto con su valor de aptitud **Z**. Este diccionario se devuelve como resultado de la función de cruce.

2.2 Diseño de la Función de Mutación

La función ***mutate*** implementa un método de mutación conocido como **mutación uniforme**. La mutación es una operación que introduce pequeñas modificaciones aleatorias en un individuo de la población para aumentar la diversidad genética y explorar nuevas soluciones en el espacio de búsqueda.

1. Se recibe un individuo y una tasa de mutación. El individuo está representado por un diccionario que contiene las coordenadas $\boldsymbol{X}$ e $\boldsymbol{Y}$ y su valor de aptitud $\boldsymbol{Z}$. Dentro de la función, se definen los límites del espacio de búsqueda para las coordenadas $\boldsymbol{X}$ e $\boldsymbol{Y}$. En este caso, los límites son -4 y 4 para ambas coordenadas.

2. Se obtienen las coordenadas $\boldsymbol{X}$ e $\boldsymbol{Y}$ del individuo. Si un número aleatorio entre 0 y 1 es menor que la tasa de mutación, se realiza la mutación en el individuo. Para cada coordenada, se suma un valor aleatorio uniforme en el rango [-1, 1]. Esto implica que el valor de $\boldsymbol{x}$ o $\boldsymbol{y}$ puede aumentar o disminuir en una pequeña cantidad aleatoria.

3. Se asegura de que las coordenadas del individuo se mantengan dentro de los límites del espacio de búsqueda. Esto se logra utilizando la función ***min*** y ***max*** para asegurarse de que $\boldsymbol{x}$ y $\boldsymbol{y}$ no excedan los límites definidos.

4. Se calcula la aptitud del individuo mutado utilizando la función de aptitud ***fitness*** y se actualiza el diccionario del individuo con las nuevas coordenadas $\boldsymbol{X}$ e $\boldsymbol{Y}$ y su valor de aptitud $\boldsymbol{Z}$.

5. Finalmente, el individuo mutado se devuelve como resultado de la función de mutación.

2.3 Descripción de la Función elitismo

Para llevar a cabo el elitismo, primero se ordenaron las soluciones de menor a mayor aptitud, para así, tener en los últimos dos índices las dos mejores soluciones de la generación generada. Dichas soluciones se almacenan en variables para que al momento de generar la siguiente generación, estas sean introducidas en la nueva población y se generen las 8 soluciones faltantes para completar esta nueva población.

2.4 Descripción del método de selección por ruleta

La función de selección por ruleta funciona de la siguiente manera, como parámetros se recibe la población y las aptitudes, donde la población son las soluciones generadas y las aptitudes solo se centra en los valores del eje z los cuales ya estarán ordenados de menor a mayor. Se procedió a hacer un valor absoluto a ellos para poder hacer la suma de todos los valores. Estos valores estarán en una variable llamada aptitudes_ajustadas. Una vez teniendo esta suma se procede a hacer un random.uniform, esto para retornar un número aleatorio entre 0 y el número total de la suma. Finalmente con un ciclo for y con la ayuda de un zip para que itere simultáneamente en población y aptitudes_ajustadas, con ayuda de una variable llamada rango_acumulado que esta irá incrementando cada iteración y bajo la condición de que si el valor de rango_acumulado es mayor o igual al valor que retorno random.uniform entonces el elemento o individuo que está en esa iteración será elegido como el padre. Finalmente, este elemento o individuo será el que se retorna.

3 Diseño y descripción del algoritmo

El objetivo del algoritmo consiste en generar una población inicial aleatoria de soluciones, conocida como generación inicial, para una función tridimensional $f(x, y) = z$. Esta generación inicial desempeñará el papel de padres en la generación posterior, y a su vez, esa generación será padres de la siguiente generación sucesivamente, con el fin de obtener soluciones más efectivas en cada generación subsiguiente.

Para la realización de este algoritmo se comenzó con la creación de la gráfica de la función de aptitud, en la cual se empezó por pintar en la gráfica y crear la primera población de posibles soluciones para la función. Después de generar esta primera generación, se ordenaron las soluciones de peor a mejor aptitud para así obtener fácilmente las dos elites que se consideraran dentro de la siguiente generación.

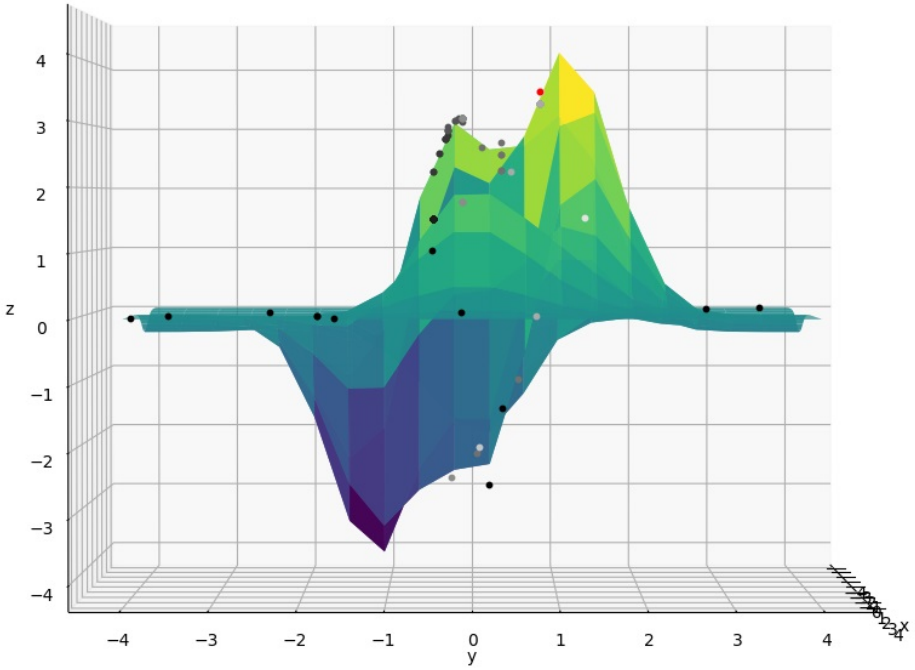
A partir de aquí, los pasos se repetirían en n iteraciones para crear las n generaciones, empezando por almacenar las dos elites en variables para poder incluirlas en la nueva generación. En estas nuevas generaciones se añaden los hijos creados de las soluciones padre que serán elegidas por la función de selección de ruleta, con la posibilidad de tener padres diferentes para cada hijo. La mitad de una nueva generación es resultado de una función de cruza que se aplica a los dos padres elegidos por ruleta, y la otra mitad es resultado de otra cruza, exceptuando las dos elites elegidas de la generación anterior que ya están consideradas como hijos de la nueva generación. Antes de acoplar los hijos como párate de la nueva población, se les aplica la función de mutación para que tengan una posibilidad de que uno de sus atributos sean alterados de manera aleatoria y pueda dar un resultado más interesante que las propiedades que heredaron de sus padres.

Al final de cada iteración, se incluye cada hijo a la nueva población y dicha es pintada y ordenada para poder encontrarás las elites fácilmente para la generación posterior, finalizando en una comparación para seguir tomando en cuenta la mejor solución encontrada en todas las generaciones juntas, así, dicha solución, pintarla de rojo intenso para marcarla en la gráfica y devolviendo los valores de x e y más adecuados para tener una aptitud alta.

4 Reporte de los individuos de la última generación y el mejor de dicha generación

El progreso de los mejores individuos en este ejemplo se vio de la siguiente forma:

```
Generación 0:
 Elite 1: {'X': -0.1932561361021028, 'Y': -0.4701413784493802, 'Z': 1.4962598745496871}
 Elite 2: {'X': 1.9279466501568656, 'Y': -1.8221903607099383, 'Z': 0.0034488173140108562}
Generación 1:
 Elite 1: {'X': -0.1932561361021028, 'Y': -0.4701413784493802, 'Z': 1.4962598745496871}
 Elite 2: {'X': -0.1932561361021028, 'Y': -0.4701413784493802, 'Z': 1.4962598745496871}
Generación 2:
 Elite 1: {'X': -0.3098087271230796, 'Y': -0.32453092496095715, 'Z': 2.7911710790304665}
 Elite 2: {'X': -0.3098087271230796, 'Y': -0.4701413784493802, 'Z': 2.253041029259805}
Generación 3:
 Elite 1: {'X': -0.3350902534482152, 'Y': -0.12655260980918548, 'Z': 3.127305425844941}
 Elite 2: {'X': -0.3224494902856474, 'Y': -0.29834699412928284, 'Z': 2.915948529121726}
Generación 4:
 Elite 1: {'X': -0.143877756510562, 'Y': 0.822352343884591, 'Z': 3.3579798022479226}
 Elite 2: {'X': -0.3350902534482152, 'Y': -0.12655260980918548, 'Z': 3.127305425844941}
Generación 5:
 Elite 1: {'X': -0.143877756510562, 'Y': 0.822352343884591, 'Z': 3.3579798022479226}
 Elite 2: {'X': -0.3350902534482152, 'Y': -0.12655260980918548, 'Z': 3.127305425844941}
Generación 6:
 Elite 1: {'X': -0.143877756510562, 'Y': 0.822352343884591, 'Z': 3.3579798022479226}
 Elite 2: {'X': -0.143877756510562, 'Y': 0.822352343884591, 'Z': 3.3579798022479226}
Generación 7:
 Elite 1: {'X': -0.20293938394270472, 'Y': 0.822352343884591, 'Z': 3.5369920448616705}
 Elite 2: {'X': -0.20293938394270472, 'Y': 0.822352343884591, 'Z': 3.5369920448616705}
Generación 8:
 Elite 1: {'X': -0.20293938394270472, 'Y': 0.822352343884591, 'Z': 3.5369920448616705}
 Elite 2: {'X': -0.20293938394270472, 'Y': 0.822352343884591, 'Z': 3.5369920448616705}
Generación 9:
 Elite 1: {'X': -0.20293938394270472, 'Y': 0.822352343884591, 'Z': 3.5369920448616705}
 Elite 2: {'X': -0.20293938394270472, 'Y': 0.822352343884591, 'Z': 3.5369920448616705}
La mejor solución: {'X': -0.20293938394270472, 'Y': 0.822352343884591, 'Z': 3.5369920448616705}
```



Las primeras generaciones empezaron con soluciones bastante bajas, lejos de la mejor solución, pero conforme las generaciones iban avanzando, las dos mejores aptitudes mejoraron, repitiéndose en algunas generaciones dado que su generación no creo nuevas soluciones que superaran a las elites de las poblaciones anteriores; más, sin embargo, en las últimas tres generaciones, logro crear muy buenas soluciones, casi alcanzando la mejor solución posible. Al final, eligiéndose como resultado, la mejor solución de la última generación, siendo también la mejor de todas las generaciones creadas anteriormente.

5 Apendices

5.1 Script algoritmo _genetico.py

5.1.1 Funciones generates_shades, getZ y fitness

```
# Generador de tonos
def generate_shades(n):
    if n <= 1:
        return None
    shades = []
    for i in range(n):
        shade_value = int((i / (n - 1)) * 255)  # Calcula el
            valor de sombra de 0 a 255
        hex_code = format(shade_value, '02x')  # Convierte el
            valor de la sombra a formato hexadecimal
        color = '#' + hex_code + hex_code + hex_code  # Crea el
            código hexadecimal del color repitiendo el valor del
            tono
        shades.append(color)
    return shades

# Herramienta para ordenar la población en base a la aptitud
def getZ(z):
  return z['Z']

# Función de aptitud
def fitness(x, y):
    return (np.exp(-(y + 1)**2 - x**2)) * ((x - 1)**2) - ((np.
        exp(-(x + 1)**2))/3) + (np.exp(-x**2 - y**2)) * (10*x**3
        - 10*x + 10*y**3)
```

5.1.2 Funciones punto3D y grafico3D

```
# Pinta los puntos en la gráfica
def punto3D(x, y, z, ax, color='#000000'):
    ax.scatter(x, y, z, c=color, s=10)

# Crea la grafíca 3D de la función
def grafica3D():
    # Generar datos para la gráfica
    x = np.linspace(-4, 4, 20)
    y = np.linspace(-4, 4, 20)
    X, Y = np.meshgrid(x, y)
    Z = fitness(X, Y)

    # Graficar la función
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d', computed_zorder=
        False)
```

```
        ax.plot_surface(X, Y, Z, cmap='viridis', alpha=0.9)
        ax.view_init(elev=11, azim=0)
        ax.set_xlabel('x')
        ax.set_ylabel('y')
        ax.set_zlabel('z')

        return ax
```

5.1.3 Funciones generate_population y roulette_selection

```
# Crea la primera generación de puntos
def generate_population(population_size):
    population = []
    for _ in range(population_size):
        x = random.uniform(-4, 4)
        y = random.uniform(-4, 4)
        z = fitness(x, y)
        individual = {'X':x, 'Y':y, 'Z':z}
        population.append(individual)
    return population

# Método de selección por ruleta
def roulette_selection(poblacion, aptitudes):
    min_aptitud = min(aptitudes)
    aptitudes_ajustadas = [aptitud + abs(min_aptitud) for aptitud in
        aptitudes] #Las aptitudes negativas las hace positivas para
       poder hacer la suma
    suma_aptitudes = sum(aptitudes_ajustadas)
    aleatorio = random.uniform(0, suma_aptitudes)

    rango_acumulado = 0
    for individuo, aptitud in zip(poblacion, aptitudes_ajustadas):
        rango_acumulado += aptitud
        if rango_acumulado >= aleatorio: #Entre mas alta sea la
           aptitud, mas probabilidad tiene de ser elegido
            individuo_seleccionado = individuo
            break

    return individuo_seleccionado
```

5.1.4 Funciones crossover_one y crossover_two

```
    # Operación de cruce (cruza promedio)
    def crossover_one(parent1, parent2):
        x1, y1 = parent1['X'], parent1['Y']
        x2, y2 = parent2['X'], parent2['Y']
        x = (x1 + x2) / 2
        y = (y1 + y2) / 2

        z = fitness(x, y)
        individual = {'X':x, 'Y':y, 'Z':z}
        return individual

    # Operación de cruce (cruza en un punto)
    def crossover_two(parent1, parent2):
```

```
    x1, y1 = parent1['X'], parent1['Y']
    x2, y2 = parent2['X'], parent2['Y']

    # Generar un punto de corte aleatorio
    cutoff = random.randint(1, 2)

    # Realizar la cruza en el punto de corte
    if cutoff == 1:
        x = x1
        y = y2
    else:
        x = x2
        y = y1

    z = fitness(x, y)
    individual = {'X':x, 'Y':y, 'Z':z}
    return individual
```

5.1.5 Función mutate

```
# Operación de mutación (mutación uniforme)
def mutate(individual, mutation_rate):
    # Definir los límites del espacio de búsqueda
    x_min, x_max = -4, 4
    y_min, y_max = -4, 4
    x, y = individual['X'], individual['Y']
    if random.uniform(0,1) < mutation_rate:
        x += random.uniform(-1, 1)
        y += random.uniform(-1, 1)
    # Asegurarse de que el individuo permanezca dentro del
       espacio de búsqueda
    x = max(min(x, x_max), x_min)
    y = max(min(y, y_max), y_min)

    z = fitness(x, y)
    individual = {'X':x, 'Y':y, 'Z':z}
    return individual
```

5.1.6 Función algoritmoGenetico

```
def algoritmoGenetico(mutation_rate, n):
    mejor = {}
    tonos = generate_shades(n) #Colores para los puntos
    ax = grafica3D()

    # Genera la primera generación
    poblacion = generate_population(10)
    poblacion.sort(key=getZ) # Ordena la población de menor a
       mayor aptitud
    for i in poblacion: # Dibuja la primera generación
        punto3D(i['X'], i['Y'], i['Z'], ax)
    mejor = poblacion[9] # Toma la mejor solución de la primera
       generación

    for j in range(0, n - 1):
```

```
            elite1, elite2 = poblacion[9], poblacion[8]
            poblacion = []
            # La siguiente generación incluye los dos mejores de la
                generación pasada
            poblacion.append(elite1)
            poblacion.append(elite2)
            # Genera la siguiente generación usando la selección por
                ruleta, dos cruzas y una posible mutación
            for i in range(8):
                padre1 = roulette_selection(poblacion, [i['Z'] for i
                    in poblacion])
                padre2 = roulette_selection(poblacion, [i['Z'] for i
                    in poblacion])
                if i <= 3: # A una mitad hace una cruza y a la otra
                    le aplica otra cruza
                    hijo = crossover_one(padre1, padre2)
                    hijo = mutate(hijo, mutation_rate)
                else:
                    hijo = crossover_two(padre1, padre2)
                    hijo = mutate(hijo, mutation_rate)
                poblacion.append(hijo)

            poblacion.sort(key=getZ) # Ordena la población de menor
                a mayor aptitud
            for i in poblacion: # Dibuja la nueva generación
                punto3D(i['X'], i['Y'], i['Z'], ax, tonos[j])

            if mejor['Z'] <= poblacion[9]['Z']: # Busca la mejor
                solución de todas las generaciones
                mejor = poblacion[9]

        punto3D(mejor['X'], mejor['Y'], mejor['Z'], ax, "#FF0000") #
            Al mejor lo pinta de rojo
        plt.show()
        return mejor
```

5.1.7 Función main

```
if __name__ == '__main__':
    mutation_rate = 0.15 # Aumentar la mutación puede dar
        resultados mas interesantes
    generaciones = int(input("Introduce el número de
        generaciones: "))
    print(f'La mejor solución: {algoritmoGenetico(mutation_rate,
        generaciones)}')
```

5.2 Manual de Usuario

5.2.1 Instalación:

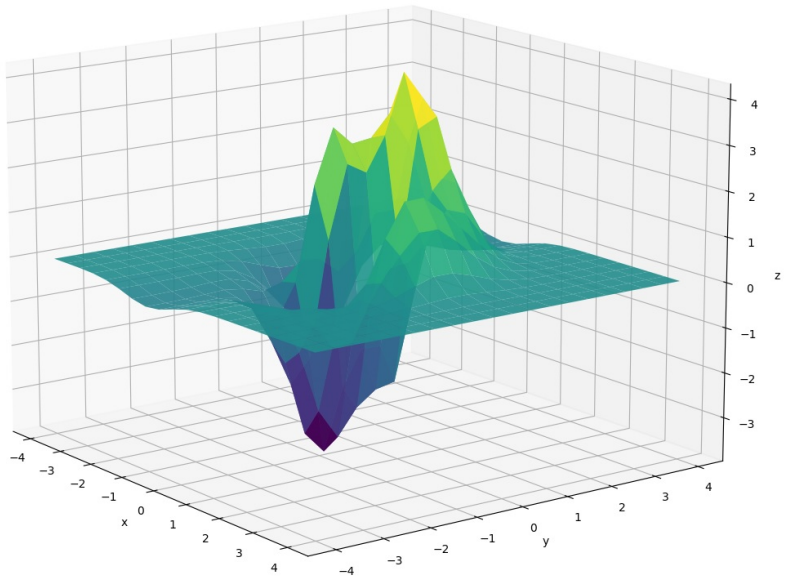


Figure 1: Z(x,y)

1. Instalar Python desde su sitio oficial:
   Descargar Python de Python.org

2. Instalar/Actualizar pip, setuptools y vitualenv:

   ```
   python -m pip install --upgrade pip setuptools virtualenv
   ```

3. Crear una carpeta donde guardar el programa y abrir terminal en ese directorio.

4. Descargar el programa en la carpeta usando Git:

   ```
   git clone https://github.com/CodeSwallow/TecnicasIA.git
   ```

   Si no se cuenta con Git entonces descargar archivo .zip del proyecto desde github.com, y descomprimirlo en la carpeta que creo anteriormente:
   Repositorio TecnicasIA de CodeSwallow

5. Crear un entorno virtual:

   ```
   python -m virtualenv kivy\_venv
   ```

6. Activar el entorno virtual:

```
kivy\_venv\textbackslash Scripts\textbackslash activate
```

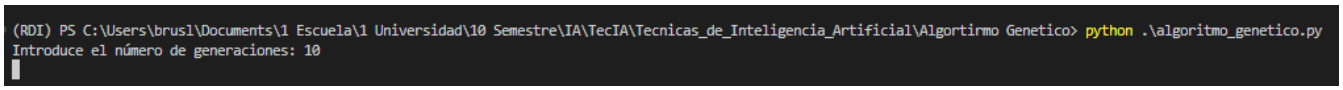
7. Instalar los requerimientos:

```
pip install -r requirements.txt
```

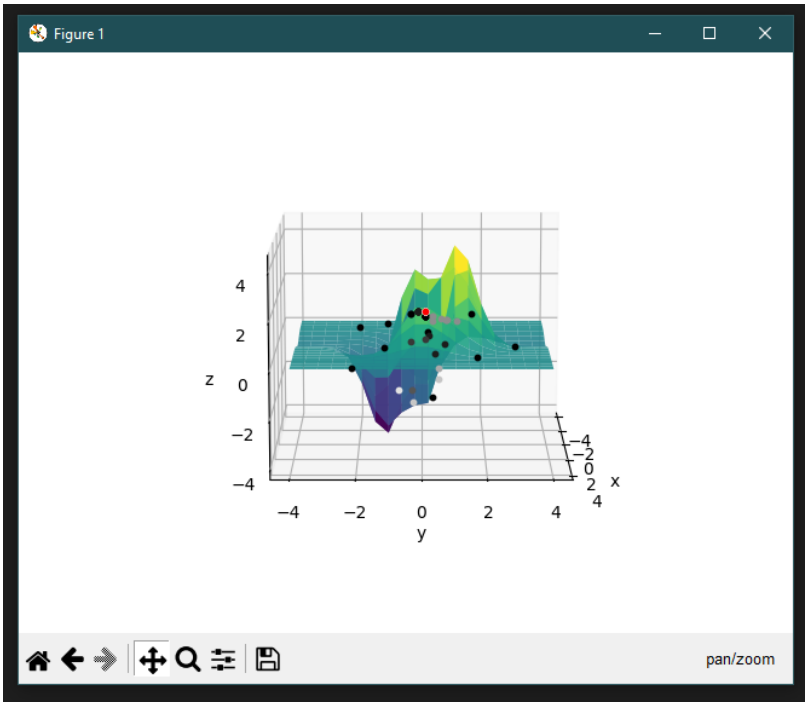
5.2.2 Instrucciones:

1. Ejecutar el programa con el comando:
   python algoritmo_genetico.py

2. Al ejecutarse se tendra:



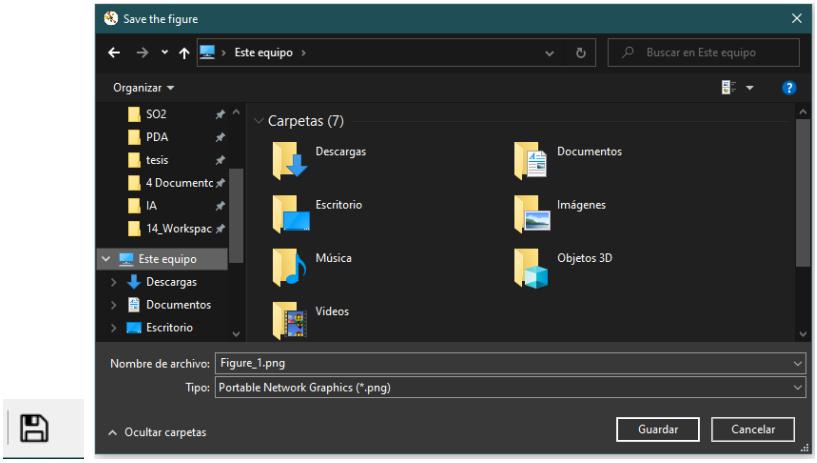
3. Despues de haber indicado el numero de cruzas, se abre una ventana mostrando las generaciones y como fueron explorando Z(x,y). Con el mouse sobre la imagen y haciendo click sobre ella, se puede manipular la posición del punto de vista.



4. En la parte inferior de la ventana se muestran los controles de la ventana para poder manipular la imagen y poder visualizarla al antojo.

5. Con el botón *home* se puede volver a la posicion original de la imagen.

6. Si se desea guardar la imagen tal y como esta, basta con presionar el botón *save*. Esto abrirá otra ventana que le permitirá seleccionar la ubicación del archivo donde sea conveniente.



7. Finalmente, al cerrar la ventana, se le indicara la mejor solución.

```
La mejor solución: {'X': 1.3210370781102712, 'Y': 0.11103102513926455, 'Z': 1.703903096424827}
```