IA.C2.A1.Estrategias.Cruza::Listado031.35Demostracion del código; insertadolstlisting.1.3

Universidad Politécnica de Chiapas

Actividad C1.A1 "Algoritmo genético"

Inteligencia Artificial

2024.C2.08 IA

P R E S E N T A :

Matrícula Nombres Apellidos

Docente:

Carlos Alberto Díaz Hernández

Chiapas, cuatrimestre mayo - agosto de 2024

Índice general

Índice de figuras

Índice de tablas

Lista de ecuaciones

$$1.1).x = a + i \cdot \Delta x \tag{1}$$

Listings

Capítulo 1

2024.C2.08X.C1.A1 Modelado de un problema de optimización a través de un Algoritmo Genético con un Data set

Periodo:	Mayo - agosto de 2024
Grupo:	8A
Asignatura:	Inteligencia Artificial
Corte:	2
Actividad:	2024.C2.08X.C1.A1b
Fecha de entrega:	2024.06.03
Matrícula	**Nombre completo**
221213	Marco Darinel Ortiz Díaz
221225	Jesús Ignacio Velasquez Hernandez
221197	David Antonio Gomez Gonzalez

1.1. Problema a resolver

El problema a resolver consiste en encontrar los valores óptimos de las constantes (a, b, c, d, e) para una fórmula lineal que prediga con precisión la variable dependiente Y en función de las variables independientes $x1, x2, x3, x4$. Como entradas se proporcionan los valores de $x1, x2, x3, x4$ y los valores reales de Y.

El algoritmo genético genera una población inicial de posibles soluciones, cada una representando un conjunto de constantes (a, b, c, d, e). A través de múltiples generaciones, la población evoluciona mediante selección, cruce y mutación para optimizar una función de ajuste que minimiza el error entre las predicciones y los valores reales de Y. Como salida, se presenta la mejor solución en una tabla con el fitness (error) y los valores de las constantes para cada generación. Adicionalmente, se generan gráficos que muestran la evolución del error y las constantes a lo largo de las generaciones, así como comparaciones entre los valores predichos y los valores reales de Y.

1.2. Modelado del problema

El objetivo es encontrar los valores óptimos de las constantes a, b, c, d, e para una fórmula lineal que prediga con precisión la variable dependiente Y en función de las variables independientes x_1, x_2, x_3, x_4. El algoritmo genético busca minimizar el error entre las predicciones de la fórmula $y = a + b \cdot x_1 + c \cdot x_2 + d \cdot x_3 + e \cdot x_4$ y los valores reales de Y.

1.2.1. Distribución Discreta de Materia

La representación de posibles soluciones se realiza en un espacio discreto definido por los valores proporcionados por el usuario. Cada solución potencial se considera como una "masa puntual.[en] esta distribución discreta, lo cual facilita la aplicación de un algoritmo genético para identificar la mejor solución.

Para determinar el valor de x, se siguen los siguientes pasos:

- Se obtiene el número en binario.
- El número binario se convierte a un número decimal.
- A partir del número decimal se calcula x utilizando la siguiente función (ver Ecuación 1.1).

$$x = a + i \cdot \Delta x \tag{1.1}$$

Esta metodología permite explorar exhaustivamente el espacio de soluciones discretas, donde cada configuración de los valores de a, b, c, d, e se representa como una masa puntual que contribuye a la estructura general del problema. El empleo de algoritmos genéticos en este contexto facilita la optimización de las soluciones mediante una combinación de selección, cruce y mutación, garantizando una búsqueda efectiva de los valores óptimos.

1.2.2. Estrategias

- **Estrategia A1**: Para cada individuo, se genera un número aleatorio m entre $[0, n]$, donde m representa la cantidad de individuos con los que se cruzará. Luego, se generan m números aleatorios que referencia a los individuos seleccionados para el cruce. Se omite que un individuo se cruce consigo mismo. El valor de n es un parámetro que define el tamaño máximo de la población.
- **Estrategia C1**: Se emplea un punto de cruce aleatorio para cada pareja de individuos. De los posibles puntos de cruce en las listas de constantes, se selecciona aleatoriamente una posición para realizar la cruza.
- **Estrategia M2**: En caso de mutación, se realiza un intercambio de posiciones de bits. Se elige aleatoriamente una posición en la lista de constantes de un individuo y se intercambia su valor con otro valor aleatorio dentro del rango permitido.
- **Estrategia P3**: Basándose en los valores de fitness, se generan clases de individuos. Luego, dentro de cada clase, se mantienen algunos individuos de manera aleatoria para asegurar diversidad genética en la población.

1.2.3. Formación de parejas

La **estrategia A1**, formación de parejas. Nos plantea en generar un numero aleatorio llamado m entre $[0, n]$, por lo cual, este valor m representa la cantidad de veces, que nuestro individuo principal debe formar parejas, es decir representa la cantidad de individuos con lo cual deberá cruzarce nuestro individuo. Después de determinar la cantidad de individuos con el cual se va a formar pareja o cruzar nuestro individuo principal, este se guardara con el individuo y sus parejas. Este proceso se repetirá para todos los individuos. $n <= |P_k| P_k$ es la población actual.

Nota: Para la estrategia se puede omitir que sea pareja de sí mismo.

Código implementado 1.1

```
    def generar_parejas(poblacion):
    parejas_cruce = []
    poblacion_indices = list(range(len(poblacion)))

        for i in range(len(poblacion)):
            cantidad_parejas = random.randint(0, len(poblacion)-1)
            parejas = random.sample(poblacion_indices[:i] + poblacion_indices[i + 1:],
    cantidad_parejas)
            parejas_cruce.append((poblacion[i], [poblacion[j] for j in parejas]))

    return parejas_cruce
```

Listing 1.1: Ejemplo de código insertado

1.2.4. Cruza

La **estrategia C1**, un puntos de cruza, plantea determinar un punto de cruza, es decir, que la pareja en cuestión se va a realizar una combinación de genes para generar los hijos.

Nota: El punto de cruza se elige aleatoriamente.

Para poder comprender con detalle se presenta el código implementado **??**

```
    def cruza(pareja1, pareja2):
        posicion = random.randint(1, len(pareja1) - 1)
        hijo1 = pareja1[:posicion] + pareja2[posicion:]
        hijo2 = pareja2[:posicion] + pareja1[posicion:]
        return hijo1, hijo2
```

Listing 1.2: Demostracion del código; insertado

Ahora el siguiente código, visualizara como es el proceso de guardado, ya que este se encuentra en la funcion principal **??**:

```
    for pareja1, parejas in cruces:
        for pareja2 in parejas:
            hijo1, hijo2 = cruza(pareja1, pareja2)
            poblacion.append(definir_mutacion(hijo1, pmutacion, pmutaciong))
            poblacion.append(definir_mutacion(hijo2, pmutacion, pmutaciong))
```

Listing 1.3: Demostracion del código; insertado

1.2.5. Mutación

Para esta **estrategia de Mutación**, esta realiza una evaluación de las constantes o del "gen", si muta este generara un numero aleatorio entre -1 hasta 1, este numero decimal se sumara con un numero entero $1 + n$ **aleatoria -1 a 1**, ahora después de sumar estos dos números se multiplicara la constante por este valor. El valor que nos de es el valor que sustituirá la constante anterior, este proceso se repite con todos los hijos.

Nota: Este proceso genera un numero aleatorio que sea menor que la probabilidad de mutación dada por nosotros, pasara por cada constante y en caso que ni uno mute este proceso se repetirá hasta que alguna constante mute.

Para poder comprender con detalle se presenta el código implementado 1.4

```
    def mutacion(individuo, pmutacion):
        nuevo = individuo
        muto = False
        while not muto:
            for i, constante in enumerate(nuevo):
                if random.randint(1, 99) / 100 < pmutacion:
                    nuevo[i] = round(constante * (1 + (np.random.normal(0, 0.4))), 2)
                    muto = True
        return nuevo
```

Listing 1.4: Demostracion del código; insertado

1.2.6. Poda

Para la **estrategia P2**, eliminación, primero reseteamos la población para guardar el mejor individuo, este al resetear la población , el mejor individuo se posiciona en el índice 0 para guardarlo.

Para comprender mejor agregamos el código donde realizamos el guardado del mejor individuo **??**:

```
    mejor_individuo = poblacion[fitnes.index(mejor_fitnes)]
    poblacion=[]
    poblacion.append(mejor_individuo)
```

Listing 1.5: Demostracion del código; insertado

Después realizamos la poda para mantener la población al máximo.

Nota: En dado caso que el tamaño de la población no haya llegado al máximo, este solo guarda la población actual, y repite el proceso hasta que haya llegado al máximo.

Para poder comprender con detalle se presenta el código implementado **??**

```
    def podar(poblacion, max_individuos):
        nueva_poblacion = []
        i = 0
        while len(nueva_poblacion) < max_individuos and i < len(poblacion):
            nueva_poblacion.append(poblacion[i])
            i += 1
    return nueva_poblacion
```

Listing 1.6: Demostracion del código; insertado

1.3. Resultados

1.3.1. Configuración 1

Indicar los valores de los parámetros insertados en la interfaz y que significado tienen.

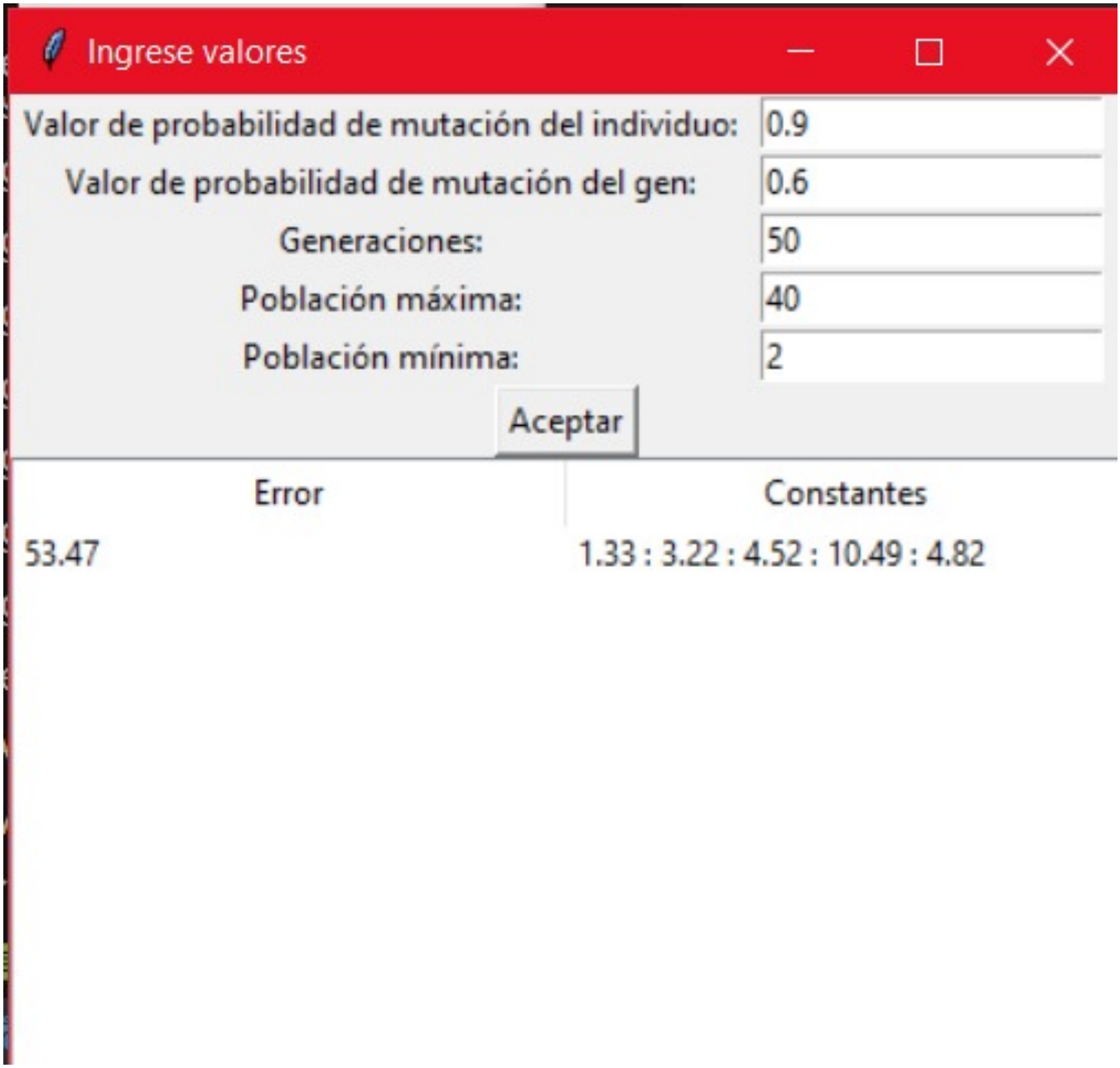

Figura 1.1: Imagen Valores 1. Realizada por Velazquez Hernandez, Ortiz Díaz y Gómez González.

Evolución de la aptitud

Insertar la gráfica de evolución de la aptitud.

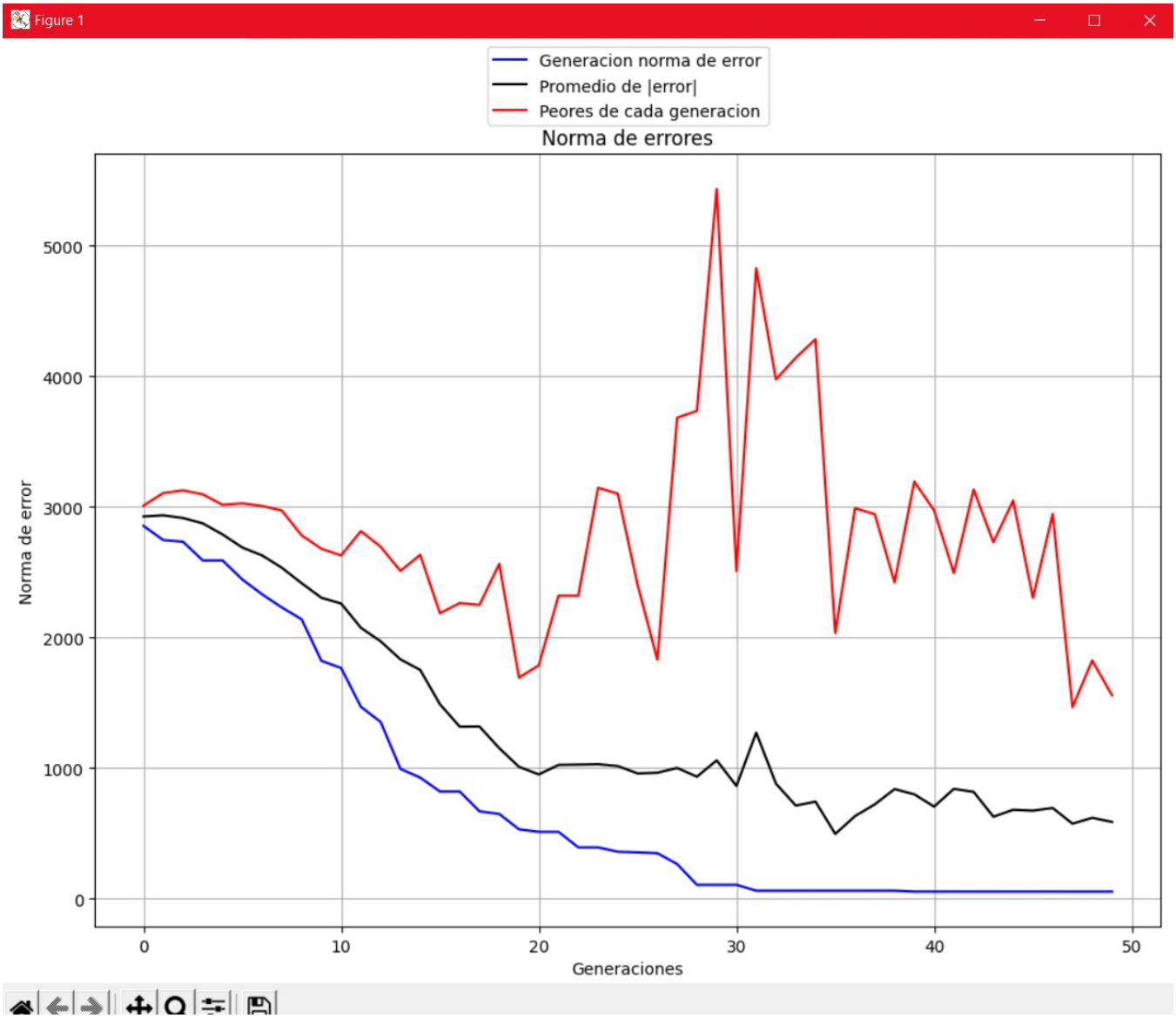

Figura 1.2: Imagen Evolución 1. Realizada por Velazquez Hernandez, Ortiz Díaz y Gómez González.

Mejor individuo del run

Inserta la tabla con el mejor individuo.

Constantes	Error
1.33 : 3.22 : 4.52 : 10.49 : 4.82	53.47

Evolución de la población

Inserta la gráfica de la población inicial, dos o tres de la población en generaciones intermedias, y la población final.

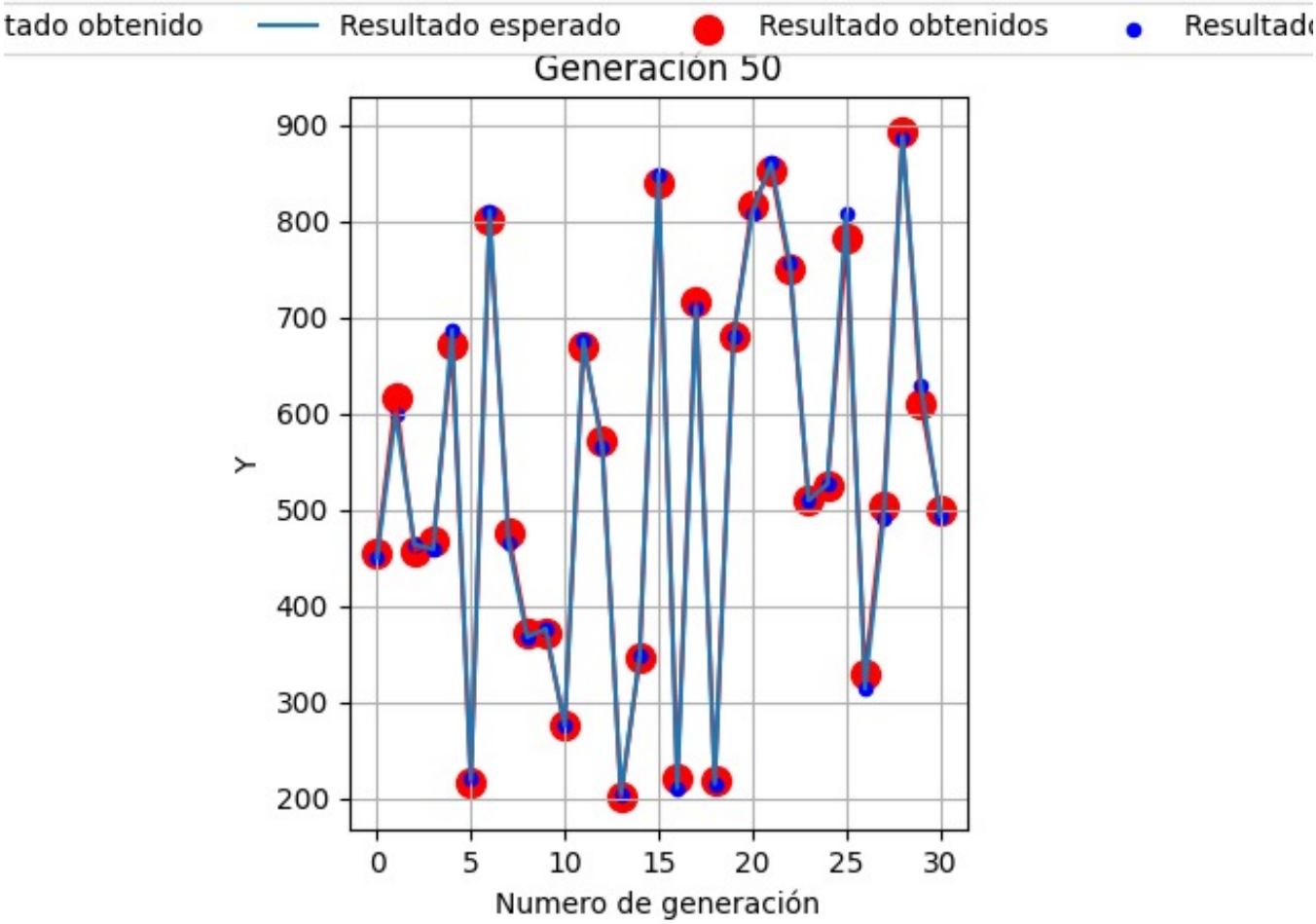

Figura 1.3: Imagen Población 1.Realizada por Velazquez Hernandez, Ortiz Díaz y Gómez González.

1.3.2. Configuración 2

Indicar los valores de los parámetros insertados en la interfaz y que significado tienen.

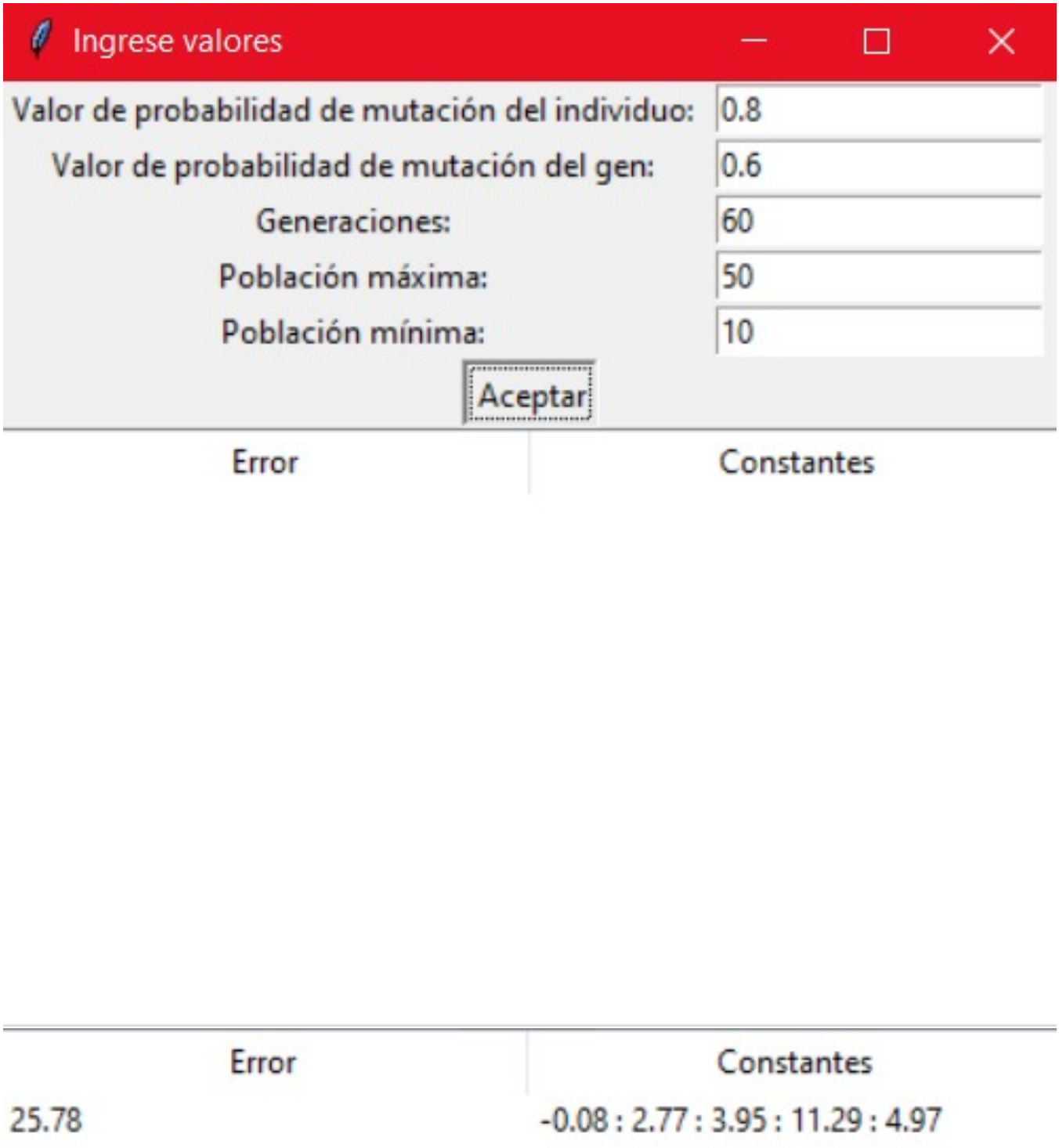

Figura 1.4: Imagen Valores 2. Realizada por Velazquez Hernandez, Ortiz Díaz y Gómez González.

Evolución de la aptitud Insertar la gráfica de evolución de la aptitud.

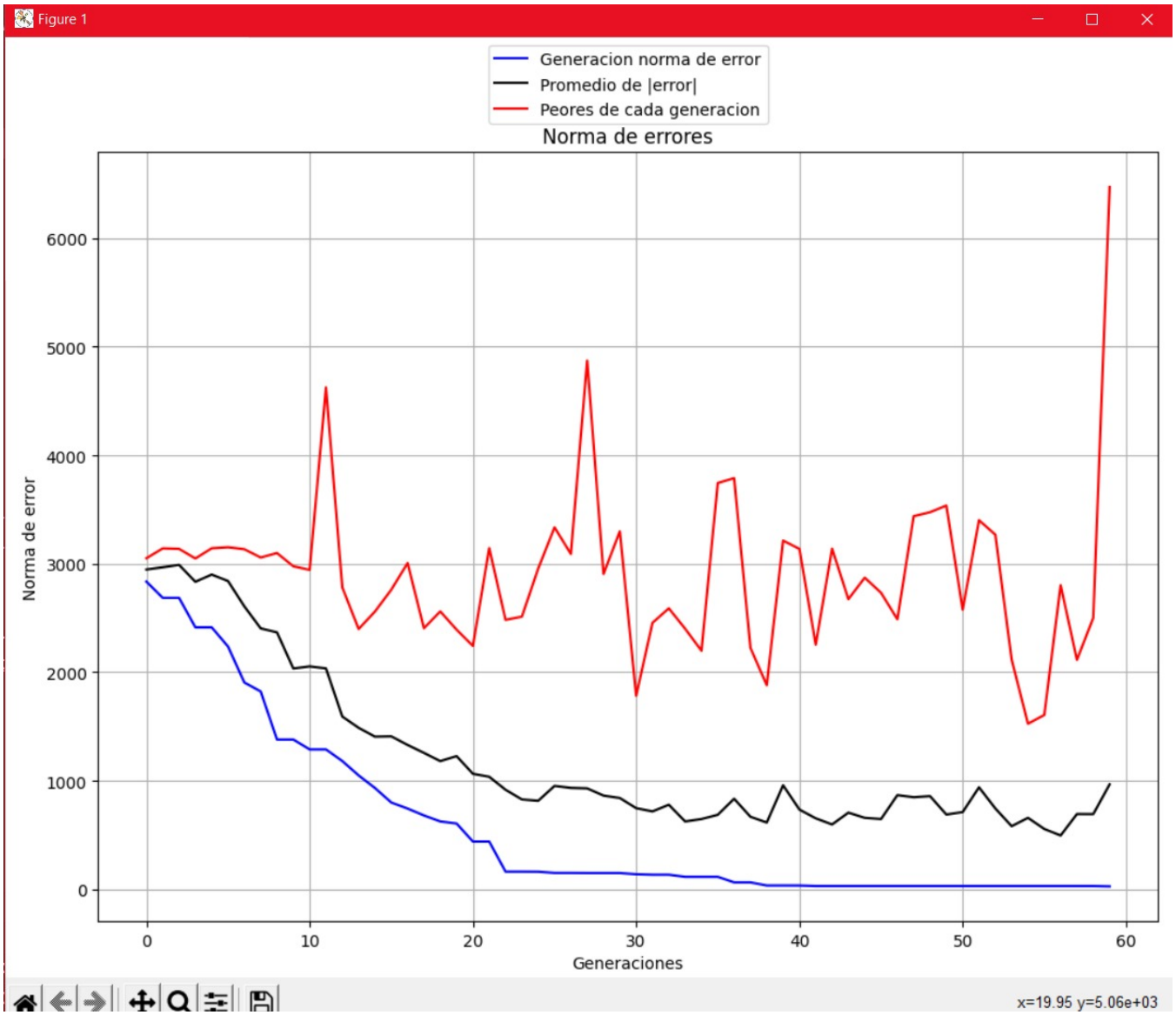

Figura 1.5: Imagen Evolucion 2. Realizada por Velazquez Hernandez, Ortiz Díaz y Gómez González.

Mejor individuo del run

Inserta la tabla con el mejor individuo.

Constantes	Error
-0.08 : 2.77 : 3.95 : 11.29 : 4.97	25.78

Evolución de la población

Inserta la gráfica de la población inicial, dos o tres de la población en generaciones intermedias, y la población final.

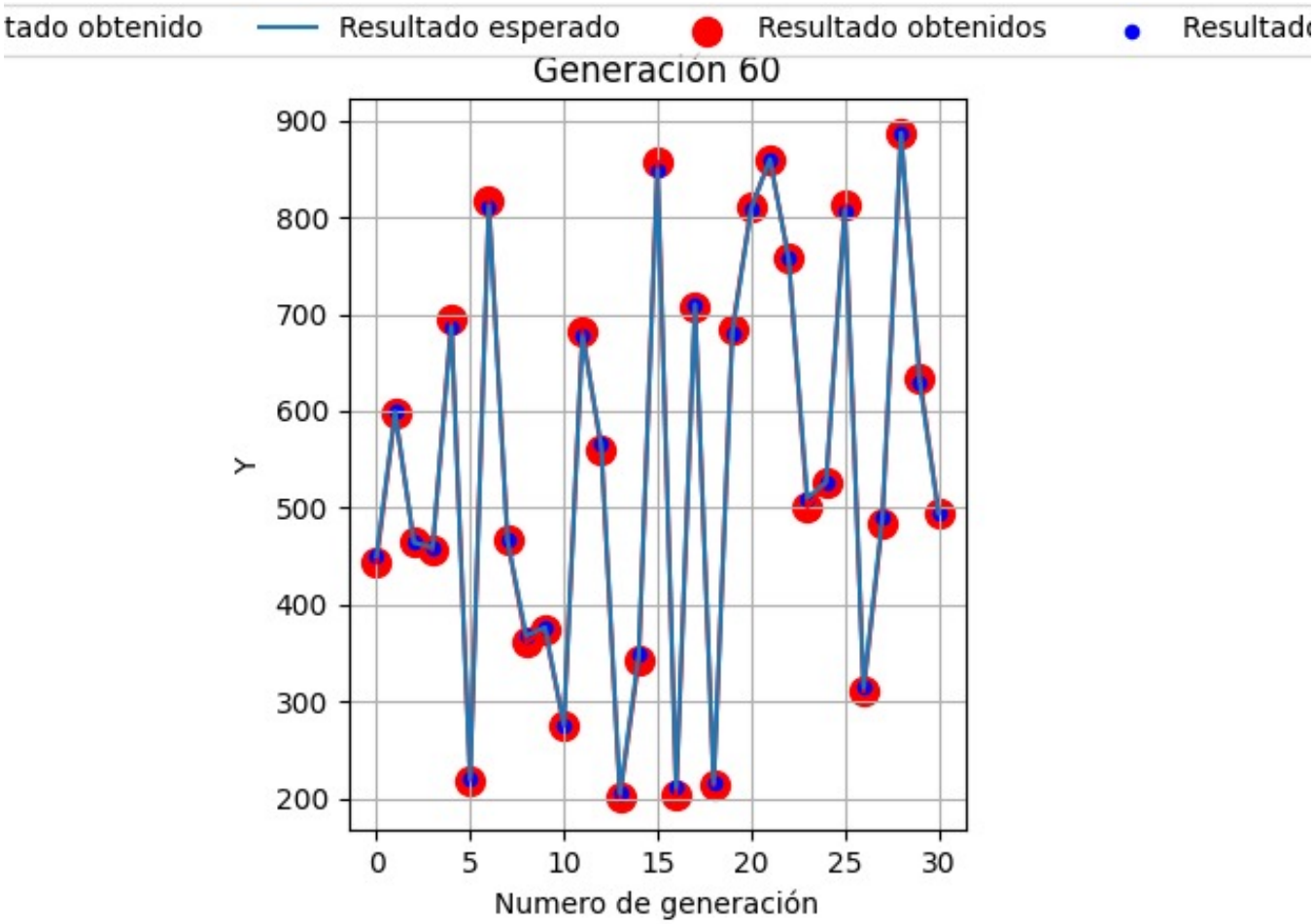

Figura 1.6: Imagen Población 2. Realizada por Velazquez Hernandez, Ortiz Díaz y Gómez González.

1.3.3. Configuración 3

Indicar los valores de los parámetros insertados en la interfaz y que significado tienen.

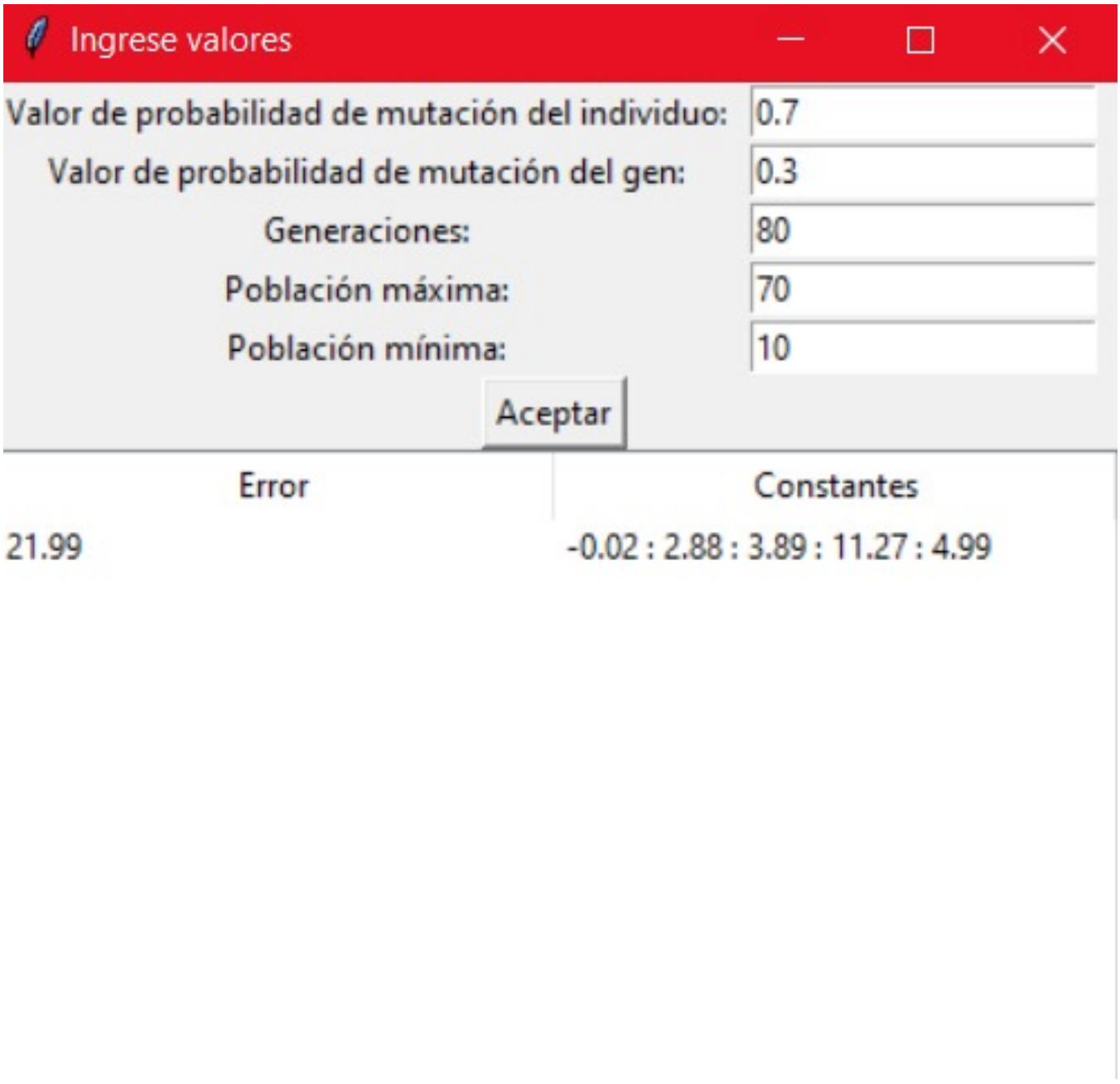

Figura 1.7: Imagen Valores 3. Realizada por Velazquez Hernandez, Ortiz Díaz y Gómez González.

Evolución de la aptitud Insertar la gráfica de evolución de la aptitud.

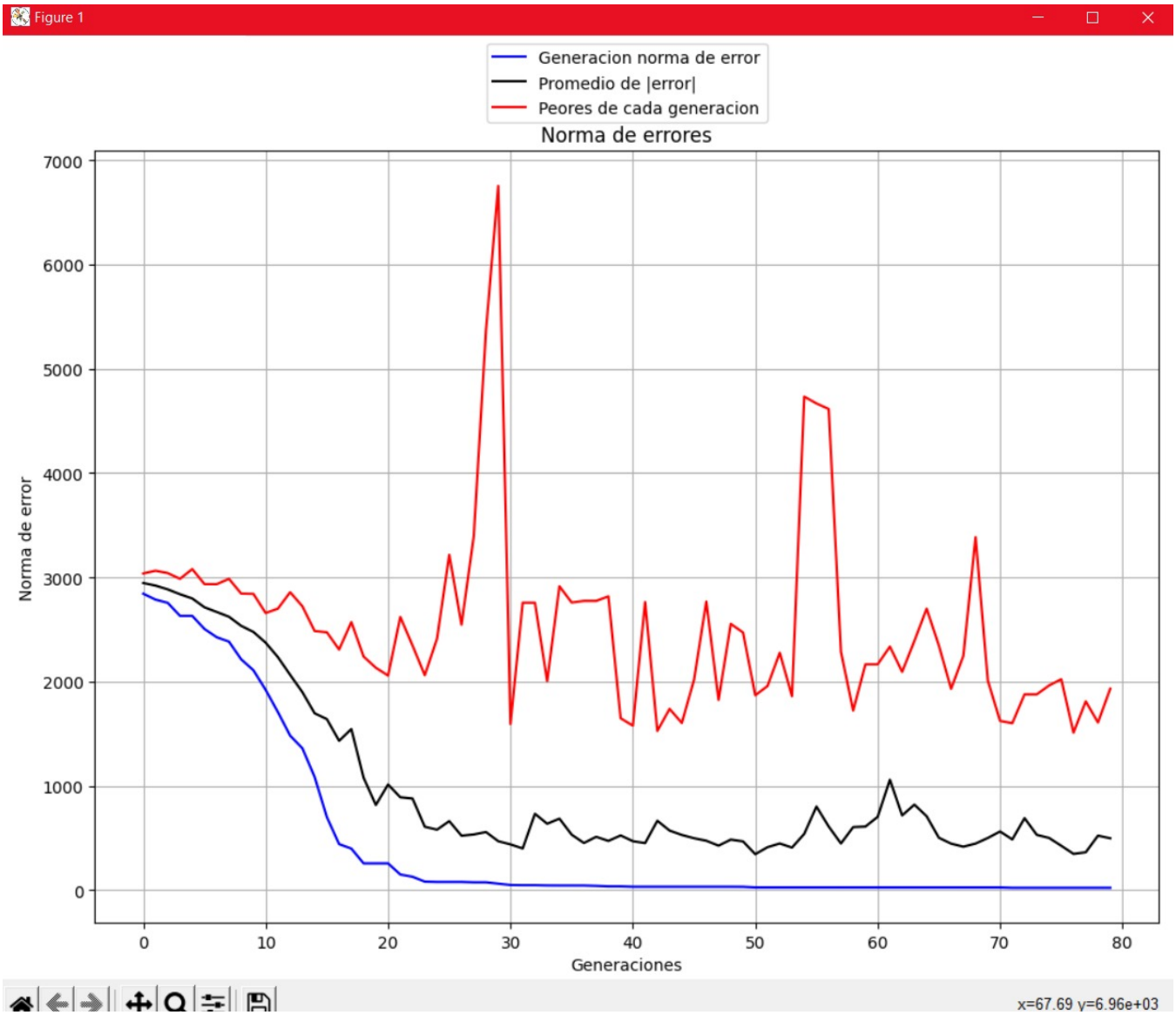

Figura 1.8: Imagen Evolución 3. Realizada por Velazquez Hernandez, Ortiz Díaz y Gómez González.

Mejor individuo del run

Inserta la tabla con el mejor individuo.

Constantes	Error
-0.02 : 2.88 : 3.89 : 11.27 : 4.99	21.99

Evolución de la población

Inserta la gráfica de la población inicial, dos o tres de la población en generaciones intermedias, y la población final.

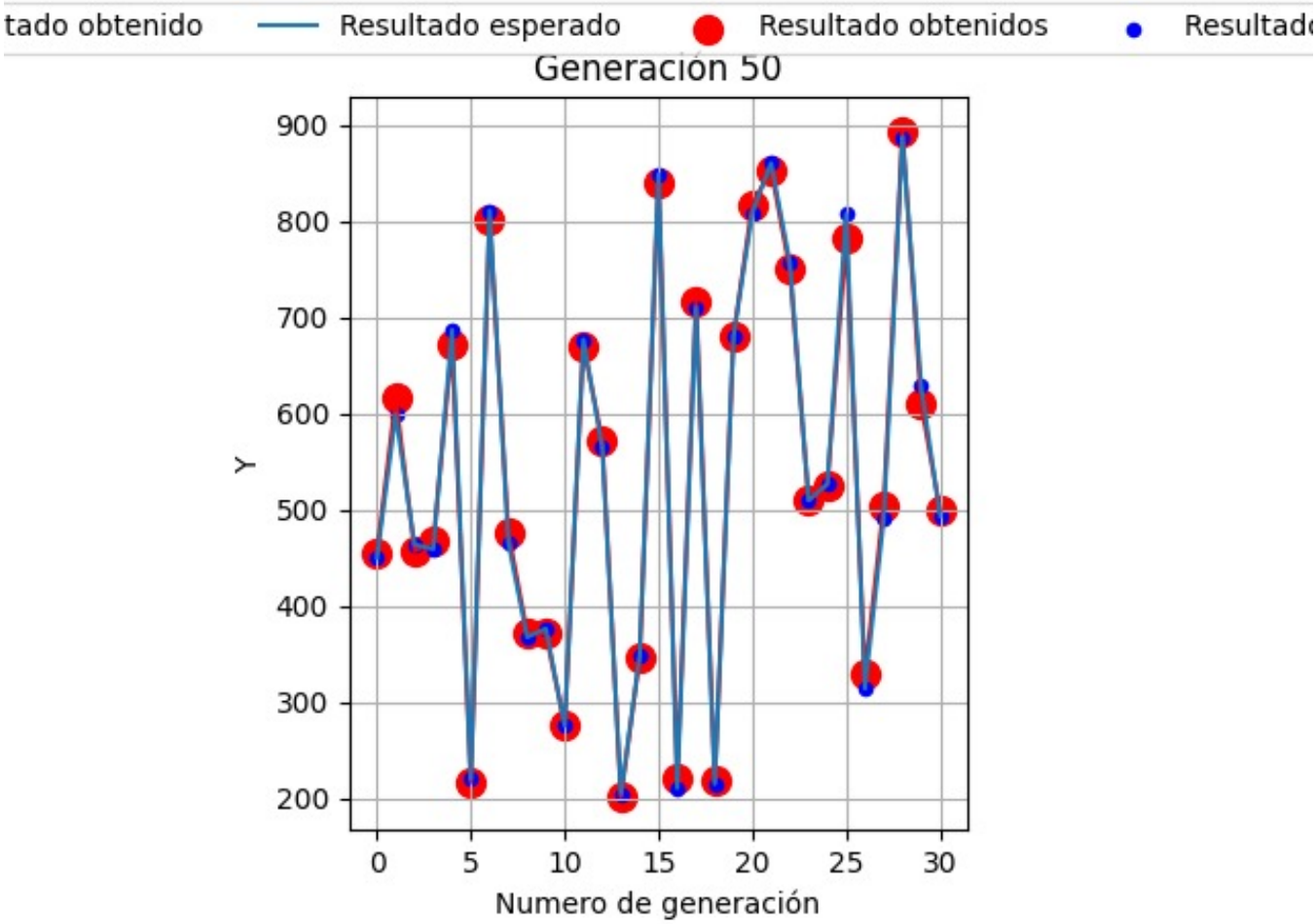

Figura 1.9: Imagen Población. Realizada por Velazquez Hernandez, Ortiz Díaz y Gómez González.

1.4. Comentarios finales

Al realizar varias ejecuciones del algoritmo genético con diferentes configuraciones de parámetros, se observó que modificar las probabilidades de mutación del individuo y del gen, así como el número de generaciones y el tamaño de la población, tuvo un impacto significativo en la reducción del error. En la primera ejecución, con una alta probabilidad de mutación del individuo (0.9) y una población mínima (2), el error fue bastante alto (53.47). Sin embargo, al mantener los mismos parámetros en la segunda ejecución, el error disminuyó a 25.78 debido a la naturaleza aleatoria del algoritmo genético. Posteriormente, en la tercera ejecución, al ajustar la probabilidad de mutación del individuo a 0.8 y aumentar las generaciones y el tamaño de la población, se esperaba una mejoría, aunque no se obtuvieron datos de error. Finalmente, en la cuarta ejecución, una reducción adicional en las probabilidades de mutación y un aumento en las generaciones y el tamaño de la población resultaron en el menor error registrado de 21.99. Estos resultados demuestran que la configuración óptima de los parámetros del algoritmo genético es crucial para minimizar el error y mejorar la precisión de las predicciones.