INSTITUTO TECNOLÓGICO DE COSTA RICA

ÁREA ACADÉMICA DE INGENIERÍA MECATRÓNICA

MT 8008
Inteligencia Artificial

Tarea 2 – Parte 1
Computación Evolutiva

Jose Fabio Navarro Naranjo – 2019049626

César Argüello Salas – 2019047699

Profesor: Juan Luis Crespo Mariño

Semestre II - 2022

Análisis del problema

Se solicita diseñar un filtro activo paso banda, con topología Sallen-Key, como el que se muestra en la figura 1. Para ello, se cuenta con varias restricciones que se mencionan a continuación:

- Únicamente se pueden emplear valores de resistencias y capacitores comerciales.
- La aplicación requiere una frecuencia central crítica de 10 kHz.
- La calidad del filtro debe ser cercana a Q=4, pero este parámetro no es crítico y puede variar.

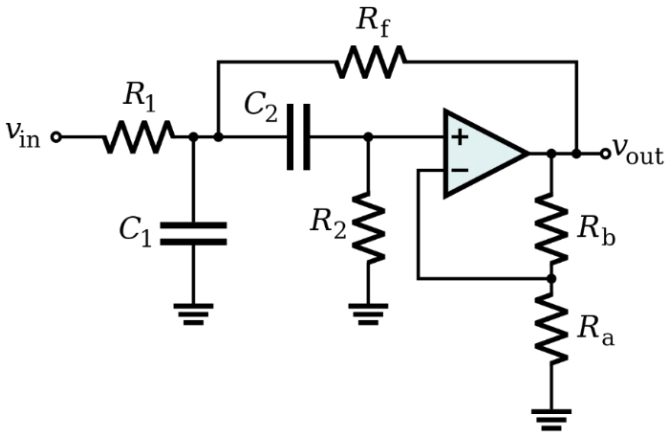

Figura 1. Topología del filtro paso banda a diseñar.

El objetivo del problema es encontrar una combinación de resistencias y capacitores disponibles comercialmente que logre minimizar el error asociado a la frecuencia central del filtro. La naturaleza del problema permite obtener un modelo matemático que lo describa, sin embargo, resulta muy dificultoso obtener una solución de manera analítica, debido a la gran cantidad de variables que deben tomarse en cuenta.

En consecuencia de lo anterior, resulta apropiado utilizar un algoritmo evolutivo para hallar una solución al problema, puesto que puede evaluarse la calidad de una solución en función de qué tanto se acerque la frecuencia central y la calidad del filtro a los valores deseados. Esta técnica computacional facilita la toma de decisiones ingenieriles, ya que permite aproximar una o varias posibles soluciones a un problema que es complicado de resolver por otros medios.

A continuación, se brinda una breve explicación de los conceptos teóricos más relevantes para el desarrollo de esta tarea.

i. Fundamento teórico

La computación evolutiva se refiere a una serie de métodos computacionales, inspirados por los procesos de evolución natural, que permiten resolver problemas de naturalezas muy variadas. Emplea algoritmos que simulan procesos biológicos como la selección natural, la reproducción, la

mutación, entre otros. Esta característica hace que la computación evolutiva sea útil para encontrar soluciones que se adapten bien a un problema en particular, de manera análoga a como sucede con los seres vivos en un entorno natural [1].

Un algoritmo evolutivo se compone de los siguientes elementos básicos:

- Una codificación de la solución: se refiere a la manera en la que el sistema computacional representa cada una de las posibles soluciones del problema [1]. Generalmente se usa una estructura llamada cromosoma (que puede ser un vector), la cual contiene los distintos genes o componentes de la codificación. Además, debe definirse un espacio de alelos, que corresponde a todas las posibles variantes de un gen (como, por ejemplo, el rango de valores que puede tomar) [2].
- Una función de calidad: es una función que permite evaluar la calidad de una solución, es decir, da una medida de qué tan apta es una solución para el problema dado [2].
- Mecanismo de selección: brinda un criterio para escoger a los individuos que serán recombinados para generar nuevas soluciones, o bien, para seleccionar a los individuos que serán reemplazados por una nueva generación [3].
- Operador de recombinación: se usa para recombinar los genes de dos individuos y así generar nuevas soluciones (hijos) a partir de soluciones existentes (padres) [3].
- Operador de mutación: define la aparición de mutaciones en el material genético de los nuevos individuos. Estas mutaciones son cambios aleatorios en el genoma, que permiten explorar nuevas soluciones [3].

Aparte de estos componentes principales, un algoritmo evolutivo también debe contemplar la manera de generar la población inicial de individuos, así como un criterio de parada que permita definir cuándo se ha alcanzado una solución suficientemente óptima [1].

Descripción de la solución

El primer paso para plantear la solución del problema fue el de obtener las ecuaciones de la frecuencia central y la calidad del filtro, en términos de los valores de sus componentes. Para ello, se investigó en la literatura de Franco [4], de donde se extrajeron las ecuaciones descritas a continuación.

Según [4], la frecuencia central de un filtro Sallen-Key paso banda está dada por:

$$\omega_0 = \frac{\sqrt{1 + R_1/R_f}}{\sqrt{R_1 R_2 C_1 C_2}}$$

Y puesto que $\omega_0 = 2\pi f_0$, se puede despejar la frecuencia expresada en Hz (f_0), obteniéndose la siguiente ecuación:

$$f_0 = \frac{1}{2\pi}\sqrt{\frac{R_1 + R_f}{R_1 R_2 R_f C_1 C_2}}$$

Por otro lado, el autor [4] establece la siguiente ecuación para determinar la calidad del filtro:

$$Q = \frac{\sqrt{1 + R_1/R_f}}{\left[1 + (1-K)\frac{R_1}{R_f}\right]\sqrt{\frac{R_2 C_2}{R_1 C_1}} + \sqrt{\frac{R_1 C_2}{R_2 C_1}} + \sqrt{\frac{R_1 C_1}{R_2 C_2}}}$$

Donde K es la ganancia y está dada por: $K = 1 + R_b/R_a$. Sustituyendo y simplificando se obtiene lo siguiente:

$$Q = \frac{\sqrt{(R_1 + R_f)R_1 R_2 R_f C_1 C_2}}{R_1 R_f (C_1 + C_2) + R_2 C_2 \left(R_f - \frac{R_b}{R_a} R_1\right)}$$

Una vez que se tuvieron las fórmulas para calcular f_0 y Q en términos de R_1, R_2, R_f, R_a, R_b, C_1 y C_2, se procedió con la respectiva codificación de la solución, la cual se explica en la siguiente sección.

i. Codificación

Para definir la codificación de la solución, primero se decidió representar el valor de cada componente por medio de dos genes, uno para su mantisa (o cifras significativas) y otro para su orden de magnitud, de manera similar a la notación científica. Los valores de las resistencias del filtro se separaron en dos genes nombrados r_i (para la mantisa) y x_i (para el orden de magnitud), mientras que los capacitores se dividieron en los genes c_i (para la mantisa) y y_i (para el orden de magnitud), según las relaciones mostradas a continuación:

$$R_i = r_i \times 10^{x_i - 2}\ [\Omega] \quad \text{donde} \quad i = 1, 2, f, a, b$$

$$C_i = c_i \times 10^{y_i - 12}\ [\text{F}] \quad \text{donde} \quad i = 1, 2$$

Se puede notar que a los exponentes x_i y y_i se les resta un número para corregir la magnitud del número resultante. Esto se hace para que ambos genes puedan tener el mismo rango de variación (de 0 a 9) y además resulten en valores de resistencia y capacitancia disponibles comercialmente.

En cuanto a los genes de la mantisa, r_i y c_i, estos solo pueden adoptar magnitudes estándar (comerciales). Para simplificar la codificación, se planteó que estos genes pueden tomar solo los valores enteros de la siguiente lista, que corresponden a las cifras significativas de las resistencias y los capacitores más comunes en el mercado.

$$r_i, c_i \in \{10, 12, 22, 33, 47, 56, 68, 82\}$$

A partir de lo anterior, se definió el cromosoma como un vector de 7 tuplas, donde cada tupla contiene los dos genes que representan cada componente, tal como se muestra a continuación:

$$\text{Cromosoma:} \quad [(r_1, x_1) \quad (r_2, x_2) \quad (r_f, x_f) \quad (r_a, x_a) \quad (r_b, x_b) \quad (c_1, y_1) \quad (c_2, y_2)]$$

Cuyo espacio de alelos está dado por:

$$\begin{cases} r_i, c_i \in \{10, 12, 22, 33, 47, 56, 68, 82\} \\ 0 \leq x_i, y_i \leq 9 \quad \text{donde} \quad x_i, y_i \in \mathbb{N} \end{cases}$$

Esta codificación permite representar componentes dentro de los siguientes rangos:

- Rango máximo del valor de resistencias: 0.10 Ω a 820 MΩ
- Rango máximo del valor de capacitores: 10 pF a 82000 μF

Se considera que la codificación de la solución es idónea puesto que abarca un rango sumamente amplio de valores de componentes comerciales, y esto lo logra por medio de genes con valores enteros y de magnitud relativamente baja que los hace fácilmente interpretables. Asimismo, se tiene que el espacio de alelos para r_i y c_i es el mismo, igual que con x_i y y_i, lo cual facilita la implementación del algoritmo en código, ya que el cromosoma se compone de 7 tuplas con exactamente el mismo espacio de alelos.

Seguidamente, se dará una explicación de la función de calidad definida.

ii. Función de calidad

Como ya se mencionó, el objetivo del problema es el de minimizar el error asociado a la frecuencia central y a la calidad del filtro propuesto, con respecto a los requerimientos de $f_0 = 10$ kHz y $Q = 4$. De esto se deduce que mientras menor sea el error de una solución, mejor será su calidad. Por lo tanto, la solución se plantea de manera que se busque minimizar la función de calidad, la cual será directamente proporcional a los errores de ambos parámetros.

Para definir la función objetivo, se empezó por obtener los porcentajes de error de f_0 y Q, los cuales están dados por las siguientes expresiones:

$$error_{f_0} = \frac{\left|\frac{1}{2\pi}\sqrt{\frac{R_1+R_f}{R_1R_2R_fC_1C_2}} - 10000\right|}{10000} = \left|\frac{1}{20000\pi}\sqrt{\frac{R_1+R_f}{R_1R_2R_fC_1C_2}} - 1\right|$$

$$error_Q = \frac{\left|\frac{\sqrt{(R_1+R_f)R_1R_2R_fC_1C_2}}{R_1R_f(C_1+C_2)+R_2C_2\left(R_f-\frac{R_b}{R_a}R_1\right)} - 4\right|}{4} = \left|\frac{\frac{1}{4}\sqrt{(R_1+R_f)R_1R_2R_fC_1C_2}}{R_1R_f(C_1+C_2)+R_2C_2\left(R_f-\frac{R_b}{R_a}R_1\right)} - 1\right|$$

Ahora, la función de calidad se puede definir como la suma algebraica de ambos errores. Sin embargo, como se sabe que la frecuencia central del filtro es un parámetro crítico, se le dará un mayor peso a su error que al asociado a la calidad Q, dado que esta puede presentar mayor variación. Se utilizó una ponderación de 10:1 para ambos errores, dando como resultado la función de calidad mostrada a continuación:

$$f_{calidad} = 10 \cdot error_{f_0} + error_Q$$

Se puede ver que es necesario construir la solución para poder evaluar su calidad. Esto porque en los porcentajes de error se contempla el cálculo de f_0 y Q a partir de los valores propuestos de los componentes. De este modo, se tiene que evaluación de la calidad es fenotípica.

Se considera que la función de calidad es apropiada porque puntúa las soluciones con respecto a su porcentaje de error, el cual se busca minimizar en todo caso. Además, brinda un mayor peso al error de la frecuencia central, lo cual es necesario porque este parámetro es más crítico que el otro.

Luego de definir la codificación de la solución y la función de calidad, se procedió con la implementación del algoritmo en Python y el estudio de hiperparámetros respectivo, lo cual se discute en la siguiente sección.

iii. Estudio de hiperparámetros

Para la implementación del código se utilizó la librería DEAP de Python, que es especializada en algoritmos evolutivos. El código realizado se puede consultar en el anexo A. Este se basó principalmente en el ejemplo del problema “One Max” de DEAP Project [5].

Inicialmente, se definieron los siguientes hiperparámetros:

- Tamaño de la población: 200 individuos
- Número de generaciones: 200 generaciones hasta que se detenga el algoritmo
- Número de hijos por generación: 50 hijos
- Método de selección para recombinación: torneo con parejas de individuos

- Operador de recombinación: Cruzamiento uniforme: cada tupla de genes puede intercambiarse con la de otro individuo, con una probabilidad del 50%
- Operador de mutación: Una tupla de genes del individuo puede ser reemplazada por una nueva generada aleatoriamente
- Probabilidad de mutación: 10%
- Método de selección para reemplazo: se reemplazan los padres por sus hijos

Al haber implementado el algoritmo y definido los hiperparámetros iniciales, se procedió con el estudio respectivo. A continuación, se presentan los resultados obtenidos en cada iteración.

Primera iteración

Se ejecutó el algoritmo dos veces, usando los hiperparámetros definidos inicialmente, y se obtuvieron las curvas de calidad y desviación estándar mostrados en la figura 2.

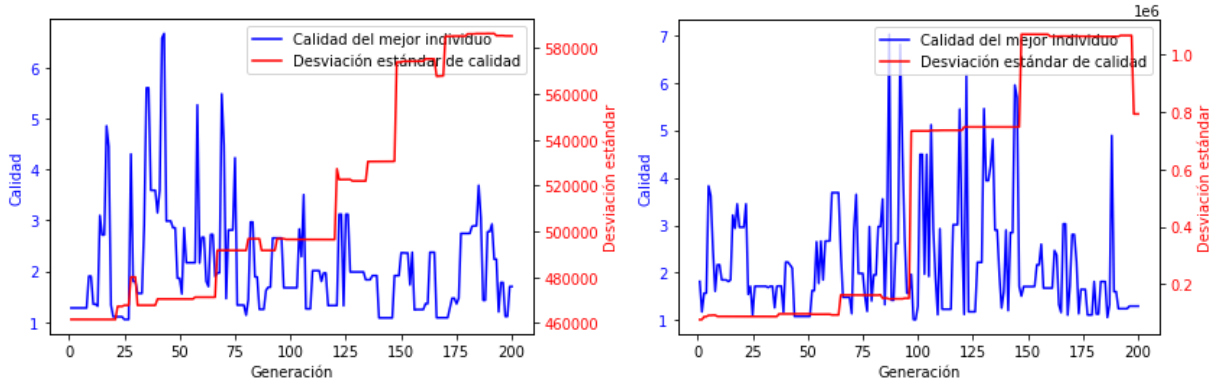

Figura 2. Curvas de calidad y desviación estándar en dos corridas de la primera iteración.

La curva de color azul representa la calidad del mejor individuo obtenido en cada generación, y es deseable que esta vaya disminuyendo dado que se trata de un problema de minimización. Por otro lado, la curva roja es la desviación estándar de la calidad de los individuos de la población, y esta da una medida de cuán amplio es el reservorio genético de la población.

En las curvas de la figura 2 se puede apreciar que, en este caso, el algoritmo no converge adecuadamente a una solución, pues la calidad del mejor individuo es sumamente variable y la desviación estándar presenta una tendencia creciente en general, lo que indica que la variabilidad aumenta en cada generación. Este comportamiento no es deseable, y puede ser causado porque los padres, que en su mayoría son individuos de muy buena calidad, están siendo reemplazados totalmente por sus hijos, los cuales pueden tener calidades muy variables a causa del cruzamiento.

Entonces, para la siguiente iteración, se modificará el método de selección de individuos que serán reemplazados. En lugar de eliminar totalmente a los padres, se usará una selección por ruleta, que favorezca a los individuos de peor calidad para ser sustituidos por la nueva generación de hijos.

Segunda iteración

En la figura 3 se muestran los resultados obtenidos en esta iteración.

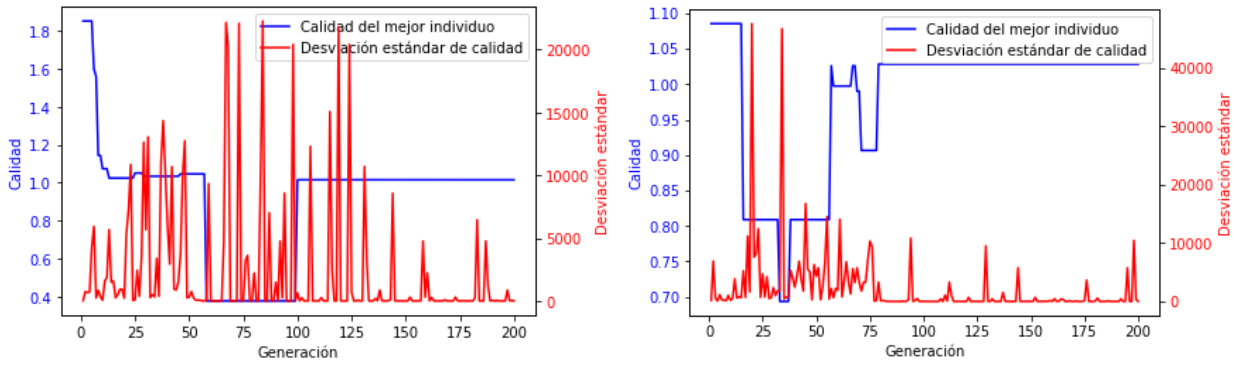

Figura 3. Curvas de calidad y desviación estándar en dos corridas de la segunda iteración.

Se puede apreciar que en esta nueva iteración el algoritmo sí converge a una solución en particular, puesto que la curva de calidad se llega a mantener en un valor constante. Sin embargo, se puede ver que ocurre la aparición de super individuos, que son individuos con muy buena calidad pero que son eliminados de manera temprana por el algoritmo. Este efecto es una consecuencia del método de selección de ruleta para reemplazar individuos, porque incluso los mejores tienen una probabilidad de ser eliminados.

Para la tercera iteración se probará con un criterio totalmente elitista para reemplazar individuos, lo que quiere decir que se eliminarán solo los de peor calidad.

Tercera iteración

Las curvas resultantes en esta iteración se presentan en la figura 4.

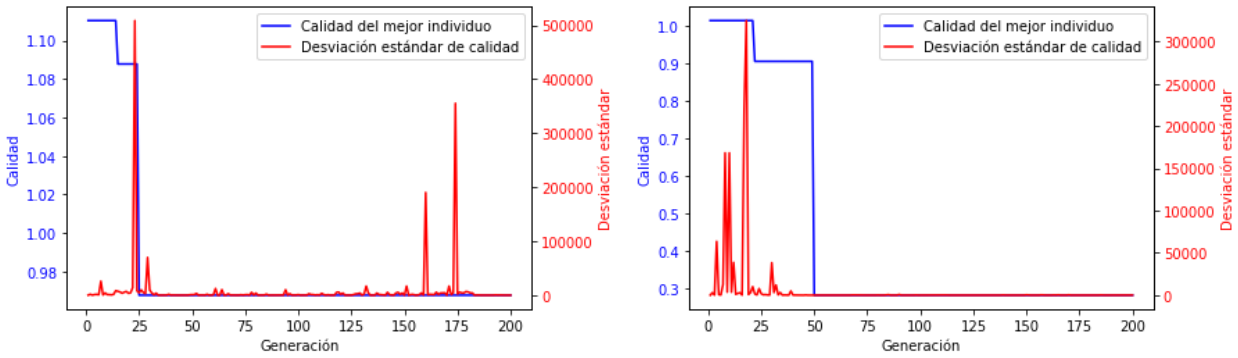

Figura 4. Curvas de calidad y desviación estándar en dos corridas de la tercera iteración.

En este caso, analizando la figura 4 se puede apreciar que el algoritmo converge adecuadamente y que, además, no se presentan super individuos como en el caso anterior. Con esto se logra que el

algoritmo converja a una solución con mejor calidad. También, se puede ver que se presentan algunos picos en la desviación estándar de la calidad. Esto puede ocurrir debido a que en la recombinación o mutación pueden surgir nuevos individuos con calidades muy distintas a la del resto de la población, lo que eleva abruptamente la desviación estándar de los datos. Sin embargo, la variabilidad también llega a estabilizarse, lo que demuestra una correcta convergencia.

Seguidamente, se cambiará el operador de recombinación por un cruzamiento en un punto variable, para ver si presenta un cambio significativo en el algoritmo.

Cuarta iteración

Los resultados de esta iteración se muestran en la figura 5.

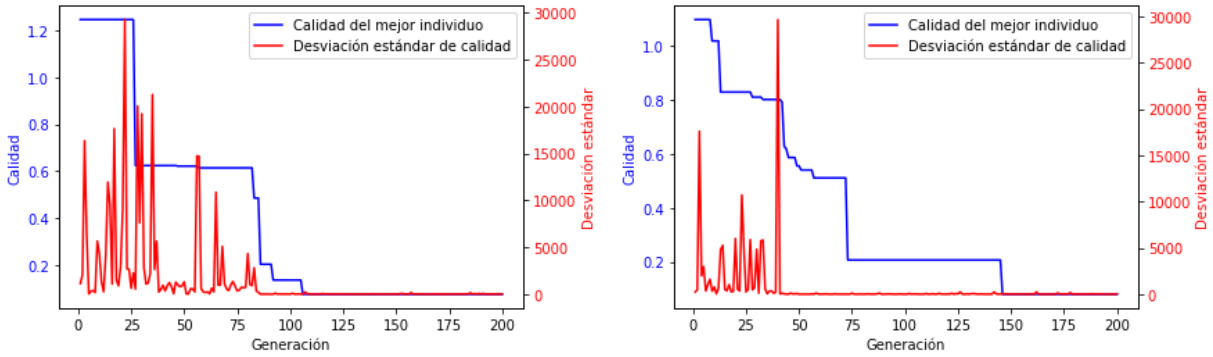

Figura 5. Curvas de calidad y desviación estándar en dos corridas de la cuarta iteración.

En las curvas de la figura 5 se puede apreciar que el algoritmo convergió a soluciones con mejor calidad que en los casos anteriores. Se puede ver que la calidad de las nuevas soluciones fue menor a 0.2, mientras que en las iteraciones anteriores no se consiguió una puntuación tan baja.

Además, se puede notar que, con un cruzamiento en un punto variable, la curva de calidad del mejor individuo disminuyó de manera más progresiva que en las demás iteraciones. Es decir, tomó un mayor número de generaciones para converger. Esto indica que se redujo la presión selectiva del algoritmo, logrando explorar más combinaciones del reservorio genético antes de converger a una solución, por lo que hay más probabilidades de encontrar una de mayor calidad.

Ahora, se modificará el número de hijos en cada generación. Se probará con generar 100 hijos dentro de una población de 200 individuos, para apreciar los cambios que esto produce.

Quinta iteración

En la figura 6 se pueden observar las curvas obtenidas en esta iteración.

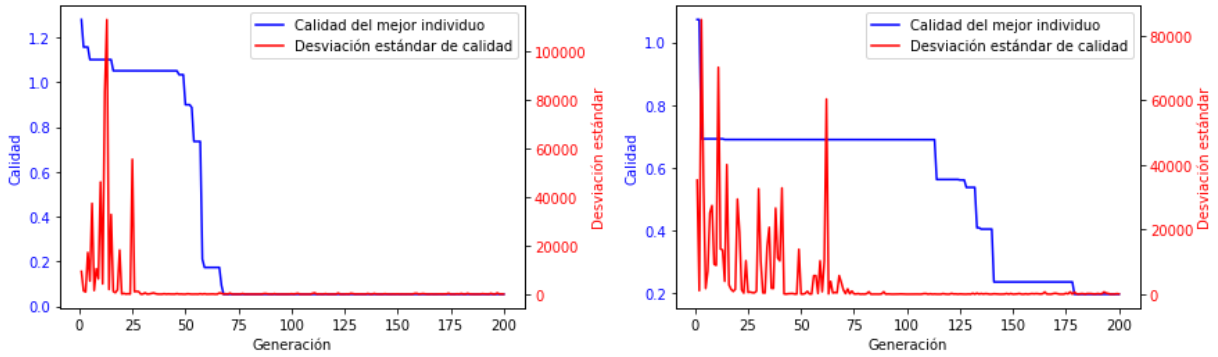

Figura 6. Curvas de calidad y desviación estándar en dos corridas de la quinta iteración.

Al comparar las curvas de la figura 6 con las de la figura 5, no se pudieron apreciar cambios realmente significativos, puesto que presentan la misma tendencia general que en el caso anterior. Para la siguiente iteración se cambiará el número de hijos a 20, para observar las posibles diferencias.

Sexta iteración

Los resultados de esta iteración se presentan en la figura 7.

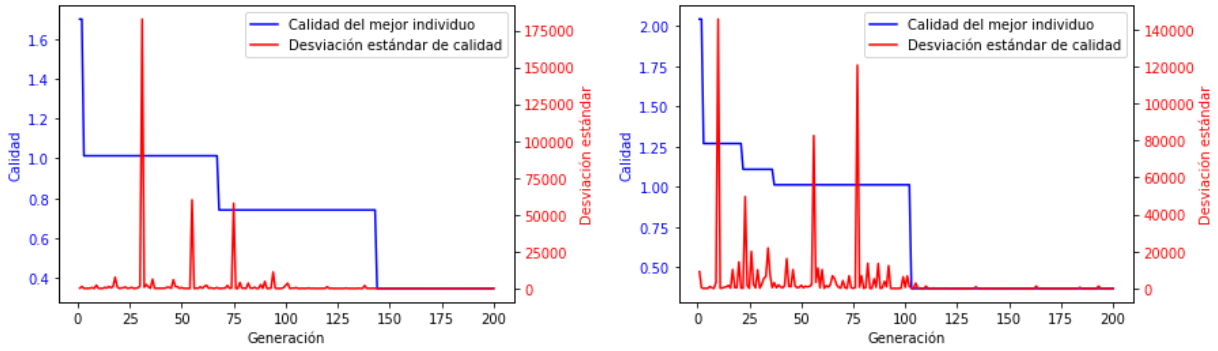

Figura 7. Curvas de calidad y desviación estándar en dos corridas de la sexta iteración.

En esta iteración, las curvas de calidad y desviación estándar mantienen el mismo comportamiento que el de las figuras 5 y 6, aunque se puede notar que las soluciones no convergen a una calidad tan baja como en los casos anteriores. Por lo tanto, se decide mantener el hiperparámetro del número de hijos por generación en 50, por ser un valor intermedio entre 20 y 100, y que se demostró que brinda buenos resultados.

En la siguiente iteración, se aumentará el tamaño de la población a 400 individuos, donde una cuarta parte de ellos será seleccionada para la recombinación (100 hijos), con tal de mantener constante la proporción original de 50 hijos en una población de 200 individuos.

Sétima iteración

Las curvas resultantes de esta iteración se muestran en la figura 8.

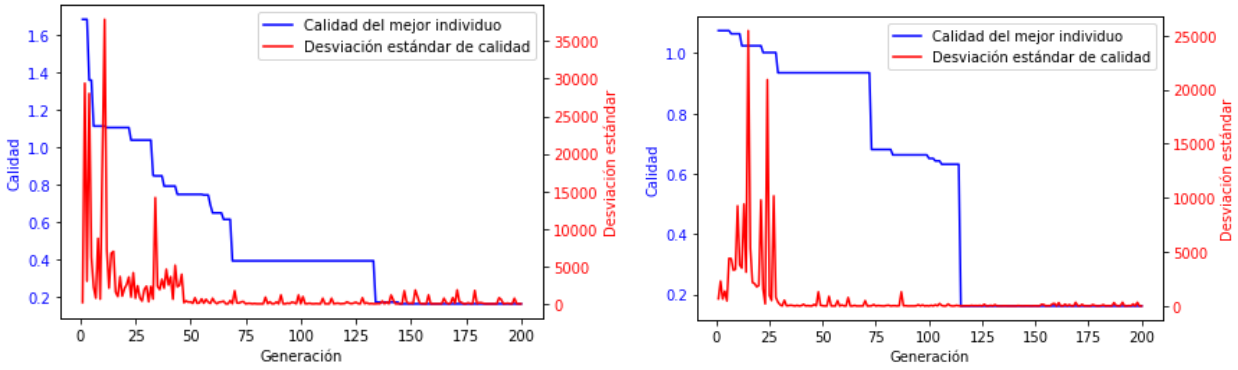

Figura 8. Curvas de calidad y desviación estándar en dos corridas de la sétima iteración.

Analizando las curvas de la figura 8 y comparándolas con la figura 5 (que corresponde a la iteración con 50 hijos en una población de 200 individuos), se observa que ambas tienen un comportamiento similar. Las dos convergen adecuadamente y la calidad de la solución final es menor a 0.2 en ambos casos.

Se elige continuar con la población de 400 individuos y 100 hijos, ya que al ser más grande ofrece la ventaja de tener un reservorio genético más amplio, por lo que existe mayor probabilidad de obtener una buena combinación de genes a través del mecanismo de recombinación.

Ahora, se realizará una última iteración donde se aumentará la probabilidad de mutación, del 10% al 20%, para observar si presenta alguna ventaja sobre el algoritmo.

Octava iteración

En la figura 9 se muestran los resultados de esta iteración.

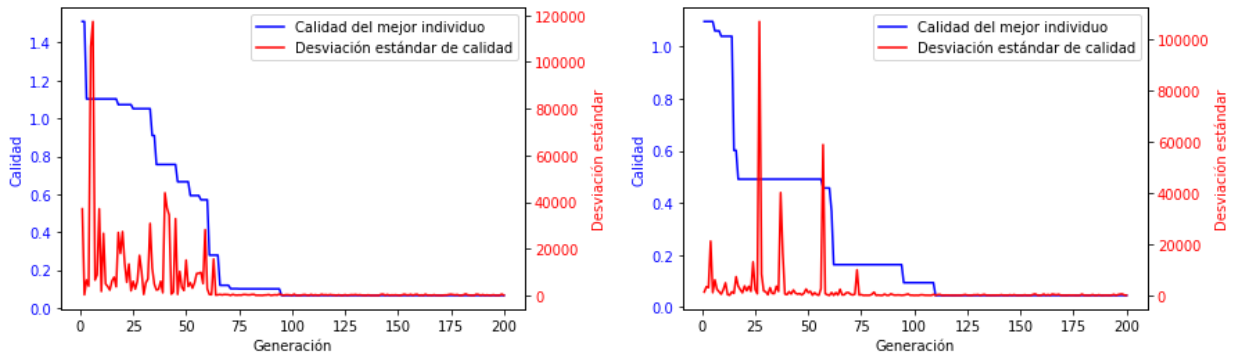

Figura 9. Curvas de calidad y desviación estándar en dos corridas de la octava iteración.

Comparando las curvas de las figuras 8 y 9, no se ha encontrado una diferencia significativa entre ambos resultados, por lo que tener una probabilidad de mutación más alta no genera ninguna ventaja apreciable sobre el evolutivo. En consecuencia, se decide mantener la tasa de mutación inicial del 10%.

Al haber efectuado este estudio, se completó la definición del algoritmo. Los hiperparámetros finales se describen a continuación:

- Tamaño de la población: 400 individuos
- Número de generaciones: 200 generaciones hasta que se detenga el algoritmo
- Número de hijos por generación: 100 hijos
- Método de selección para recombinación: torneo con parejas de individuos
- Operador de recombinación: Cruzamiento en un punto variable
- Operador de mutación: Una tupla de genes del individuo puede ser reemplazada por una nueva generada aleatoriamente
- Probabilidad de mutación: 10%
- Método de selección para reemplazo: Criterio totalmente elitista: se reemplazan los individuos de peor calidad

Resultados

Se utilizó el algoritmo evolutivo implementado en Python para obtener soluciones apropiadas al problema. Se buscó que la calidad de la solución fuese lo menor posible, para que así el error fuera mínimo. Dado que el algoritmo puede converger a soluciones distintas en cada corrida, se ejecutó el código varias veces, y de estas se extrajeron dos posibles soluciones con magnitud de calidad relativamente baja.

En el anexo B se muestran los resultados impresos por el código, donde se incluye el genoma, la calidad de la solución, los valores de cada componente del filtro, la frecuencia central y la calidad Q del filtro. Asimismo, en las figuras 10 y 11 se presentan las curvas del proceso de evolución de cada caso.

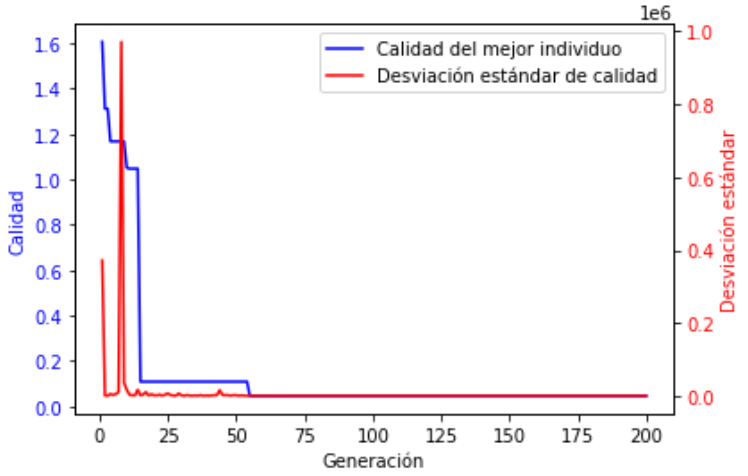

Figura 10. Curvas de calidad y desviación estándar de la solución #1.

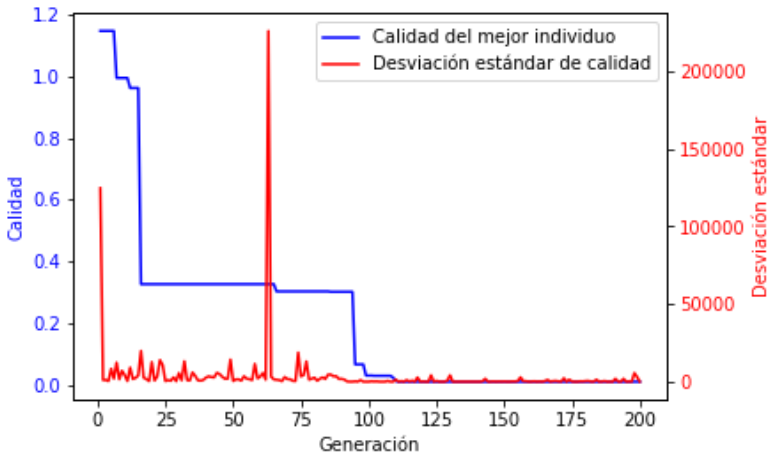

Figura 11. Curvas de calidad y desviación estándar de la solución #2.

De acuerdo con los resultados presentados en el anexo B, se puede apreciar que la solución #1 posee una calidad de aproximadamente 0.046, mientras que la solución #2 tiene una calidad de 0.009. Este valor da una medida de qué tan apta es dicha solución para el problema planteado. Es claro que ambas soluciones son adecuadas porque tienen un valor de calidad relativamente pequeño, lo que se traduce en un porcentaje de error bajo.

Sin embargo, no es correcto juzgar una solución solo con base en su calidad, sino que esta debe evaluarse para conocer los parámetros de interés, los cuales en este caso son la frecuencia central y la calidad del filtro. Del anexo B se puede extraer lo siguiente:

- En la solución #1: $f_0 = 10.021$ kHz y $Q = 4.101$
- En la solución #2: $f_0 = 10.008$ kHz y $Q = 3.996$

En estos resultados se puede ver que los filtros diseñados cumplen con las especificaciones dadas, puesto que las frecuencias centrales son cercanas a 10 kHz y las calidades Q se aproximan a 4. El porcentaje de error para f_0 en la primera solución es de 0.21% y en la segunda es de 0.08%. En cuanto a los porcentajes de error de la calidad del filtro, en la solución #1 este es de 2.52% y en la solución #2 es de 0.10%. En todos los casos se obtuvieron porcentajes de error sumamente bajos, lo que demuestra la validez de las soluciones. Estos errores fueron menores al 1% en la mayoría de los casos, con excepción de la calidad Q del primer filtro, pero igualmente posee una desviación dentro de los límites tolerables, considerando que este parámetro no es crítico.

Con respecto a las curvas de las figuras 10 y 11, se puede ver que ambas corridas para encontrar las soluciones transcurrieron con normalidad. Las curvas de calidad del mejor individuo disminuyeron progresivamente hasta converger a la solución final; y las curvas de desviación estándar presentaron más irregularidades al inicio del proceso evolutivo, pero llegaron a estabilizarse con la convergencia del algoritmo. Una diferencia importante entre ambas figuras es que el proceso de evolución de la solución #1 convergió con un menor número de generaciones que la solución #2. De esto se deduce que el algoritmo pudo explorar más combinaciones de genes durante de búsqueda de la segunda solución que durante la primera.

Finalmente, se puede elegir uno de los dos filtros para ser implementado en la aplicación requerida. Para tomar esta decisión deben tomarse en cuenta múltiples aspectos, como, por ejemplo: la disponibilidad de componentes y el error permisible. En este caso, resulta prudente elegir la solución cuya frecuencia central tenga el menor porcentaje de error, ya que este es el parámetro más crítico de la aplicación. Por ende, se selecciona el filtro dado por la solución #2, y el diseño final de este sería el mostrado en la figura 12.

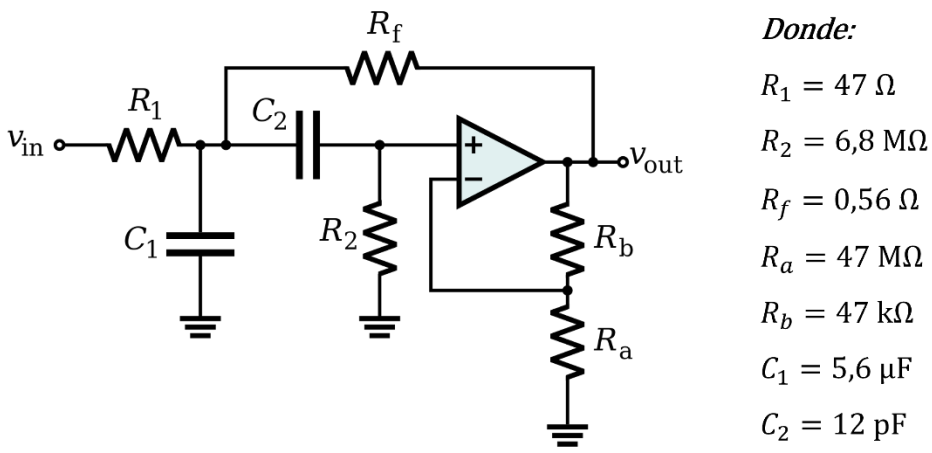

Figura 12. Filtro Sallen-Key paso banda diseñado.

Conclusiones y comentarios finales

Se logró implementar correctamente un algoritmo evolutivo en Python, para resolver un problema que sería muy complejo de abordar por otros medios más tradicionales. En este caso, se logró diseñar un filtro activo paso banda, con topología Sallen-Key, y usando únicamente valores de resistencias y capacitores comerciales.

Se lograron obtener porcentajes de error sumamente bajos: del 0.08% para la frecuencia central del filtro y del 0.1% para su calidad, lo que demuestra la utilidad de la computación evolutiva para obtener muy buenas soluciones a un problema, sin la necesidad de realizar análisis complejos o que requieran numerosas iteraciones para obtener una solución adecuada.

Este tipo de técnicas computacionales son herramientas muy poderosas para la solución de problemas, sin embargo, siempre se requiere de un análisis ingenieril posterior, que permita tomar decisiones conscientes y que mejor se adapten a las condiciones reales del problema.

Referencias

[1] A. P. Engelbrecht, *Computational Intelligence. An Introduction*. West Sussex, England: John Wiley & Sons, Ltd, 2007.

[2] J. L. Crespo Mariño y R. Loaiza Baldares, "Tema 3: Computación evolutiva (I): Introducción al paradigma evolutivo", *tecdigital.tec.ac.cr*. [En línea]. Disponible: https://tecdigital.tec.ac.cr/dotlrn/classes/IMT/MT8008/S-2-2022.CA.MT8008.1/file-storage/view/presentaciones%2FTEMA3.1-V2.1.pdf. [Accedido nov. 12, 2022].

[3] J. L. Crespo Mariño y R. Loaiza Baldares, "Tema 3: Computación evolutiva (II): Componentes de un genético fundamental", *tecdigital.tec.ac.cr*. [En línea]. Disponible: https://tecdigital.tec.ac.cr/dotlrn/classes/IMT/MT8008/S-2-2022.CA.MT8008.1/file-storage/view/presentaciones%2FTEMA3.2-V2.3.pdf. [Accedido nov. 12, 2022].

[4] S. Franco, *Design with Operational Amplifiers and Analog Integrated Circuits*. 2 Penn Plaza, New York: McGraw-Hill Education, 2015.

[5] DEAP Project, "One Max Problem", *deap.readthedocs.io*. [Online]. Available: https://deap.readthedocs.io/en/master/examples/ga_onemax.html. [Accessed Nov. 12, 2022].

Anexos

A. Código de Python implementado

```
#MT-8008 Inteligencia Artificial
#Tarea Evaluable: Computación Evolutiva parte I

#Elaborado por:
#César Argüello Salas
#Jose Fabio Navarro Naranjo


#instalación de la librería DEAP
!pip install deap

#importación de librerías adicionales
import random
import numpy
import math
import matplotlib.pyplot as plt

#importación de módulos de DEAP
from deap import base
from deap import creator
from deap import tools


#se define que el problema es de minimización
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))

#creación de la clase para los individuos
creator.create("Individual", list, fitness=creator.FitnessMin)


alelosRC = [10, 12, 22, 33, 47, 56, 68, 82] #espacio de alelos de la mantisa

def gen_tuplaRC(): #función para generar las tuplas de genes de cada componente
  return (random.choice(alelosRC), random.randint(0, 9))


toolbox = base.Toolbox() #se inicializa el toolbox

toolbox.register("tuplaRC", gen_tuplaRC) #se registra la función para crear genes

toolbox.register("individual", tools.initRepeat, creator.Individual,
                 toolbox.tuplaRC, n=7) #se registra la estructura del cromosoma
```

```
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
                                                        #se registra la población


def fcalidad(ind): #se define la función de calidad

    #cálculo de los valores de los componentes a partir de sus genes
    R1 = ind[0][0] * pow(10, ind[0][1]-2)
    R2 = ind[1][0] * pow(10, ind[1][1]-2)
    Rf = ind[2][0] * pow(10, ind[2][1]-2)
    Ra = ind[3][0] * pow(10, ind[3][1]-2)
    Rb = ind[4][0] * pow(10, ind[4][1]-2)
    C1 = ind[5][0] * pow(10, ind[5][1]-12)
    C2 = ind[6][0] * pow(10, ind[6][1]-12)

    #porcentaje de error de la frecuencia central
    error_f0 = abs(((1/(20000*math.pi))*math.sqrt((R1+Rf)/(R1*R2*Rf*C1*C2)))-1)

    #porcentaje de error de la calidad Q del filtro
    error_Q = abs((0.25*(math.sqrt((R1+Rf)*R1*R2*Rf*C1*C2))/((R1*Rf*(C1+C2))+(R2*C2*
(Rf-((Rb/Ra)*R1)))))-1)

    calidad = 10*error_f0 + error_Q #suma ponderada de los errores

    return calidad,


def mutar_ind(ind, prob): #función para mutar individuos
                          #la probabilidad de mutación le ingresa como parámetro
  n = random.random()         #número aleatorio entre 0 y 1
  if n < prob:                #condición para que ocurra la mutación
    idx = random.randrange(7) #se selecciona una tupla de genes al azar
    ind[idx] = gen_tuplaRC()  #se modifica la tupla seleccionada
  return ind,


#se registran los operadores en el toolbox
toolbox.register("select", tools.selTournament, tournsize=2) #operador de selección
                                                             #para recombinación

toolbox.register("mate", tools.cxOnePoint)        #operador de recombinación

toolbox.register("mutate", mutar_ind, prob=0.1)   #operador de mutación con
                                                  #probabilidad del 10%
```

```
toolbox.register("evaluate", fcalidad)            #función de calidad

toolbox.register("delete", tools.selWorst) #operador de selección para reemplazo


def main(): #función principal, donde se programa el algoritmo

  #definición de hiperparámetros
  SIZE_POP = 400          #Tamaño de la población
  N_HIJOS = 100           #Número de hijos por generación
  N_GENERACIONES = 200    #Número máximo de generaciones


  pop = toolbox.population(n=SIZE_POP) #se inicializa la población inicial


  #se inicializa la herramienta para generar las estadísticas
  stats = tools.Statistics(key=lambda ind: ind.fitness.values)
  stats.register("min", numpy.min)  #calidad del mejor individuo
  stats.register("std", numpy.std)  #desviación estándar de calidades

  hof = tools.HallOfFame(1) #salón de la fama para guardar al mejor individuo
  logbook = tools.Logbook() #variable que lleva registro de las estadísticas


  #Se evalúan las calidades de toda la población
  fitnesses = list(map(toolbox.evaluate, pop))
  for ind, fit in zip(pop, fitnesses):
    ind.fitness.values = fit

  #Se crea una lista para que contiene las calidades
  fits = [ind.fitness.values[0] for ind in pop]


  g = 0 #variable para llevar la cuenta del número de generaciones


  while g < N_GENERACIONES: #ciclo para llevar a cabo la evolución

    g = g + 1 #se empieza una nueva generación


    #Selección de los individuos para recombinación (padres)
    offspring = toolbox.select(pop, N_HIJOS)

    #Se crea una copia de los padres
```

```
    offspring = list(map(toolbox.clone, offspring))


    #Se aplica el operador de recombinación para generar los hijos
    for child1, child2 in zip(offspring[0::2], offspring[1::2]):
      toolbox.mate(child1, child2)


    for child in offspring:
      toolbox.mutate(child)      #se aplica el operador de mutación
                                 #sobre los hijos recientemente creados

      del child.fitness.values   #se borran sus valores de calidad
                                 #porque necesitan ser reevaluados


    #Se evalúan las calidades de los nuevos individuos
    fitnesses = list(map(toolbox.evaluate, offspring))
    for ind, fit in zip(offspring, fitnesses):
      ind.fitness.values = fit


    #Se eliminan N_HIJOS de la población inicial
    #usando el operador de selección para reemplazo
    for i in range(N_HIJOS):
      selected = toolbox.delete(pop, 1)[0]
      pop.remove(selected)

    #Se agregan los nuevos individuos a la población
    pop.extend(offspring)


    #Se calculan y guardan las estadísticas de la nueva generación
    record = stats.compile(pop)
    logbook.record(gen=g, **record)
    hof.update(pop)              #se actualiza el salón de la fama



#se leen las estadísticas guardadas durante la evolución
gen, min_fit, std_fit = logbook.select("gen", "min", "std")


#creación del gráfico para mostrar las curvas de calidad y desv. estándar
fig, ax1 = plt.subplots()
line1 = ax1.plot(gen, min_fit, "b-", label="Calidad del mejor individuo")
```

```
  ax1.set_xlabel("Generación")
  ax1.set_ylabel("Calidad", color="b")
  for tl in ax1.get_yticklabels():
    tl.set_color("b")

  ax2 = ax1.twinx()
  line2 = ax2.plot(gen, std_fit, "r-", label="Desviación estándar de calidad")
  ax2.set_ylabel("Desviación estándar", color="r")
  for tl in ax2.get_yticklabels():
    tl.set_color("r")

  lns = line1 + line2
  labs = [l.get_label() for l in lns]
  ax1.legend(lns, labs, loc="upper right")

  plt.show() #se muestra el gráfico


  #se extrae el genoma del mejor individuo y su valor de calidad
  mejor_ind = hof[0]
  calidad_ind = mejor_ind.fitness.values[0]

  #se calcula el valor de los componentes a partir de sus genes
  R1 = mejor_ind[0][0] * pow(10, mejor_ind[0][1]-2)
  R2 = mejor_ind[1][0] * pow(10, mejor_ind[1][1]-2)
  Rf = mejor_ind[2][0] * pow(10, mejor_ind[2][1]-2)
  Ra = mejor_ind[3][0] * pow(10, mejor_ind[3][1]-2)
  Rb = mejor_ind[4][0] * pow(10, mejor_ind[4][1]-2)
  C1 = mejor_ind[5][0] * pow(10, mejor_ind[5][1]-12)
  C2 = mejor_ind[6][0] * pow(10, mejor_ind[6][1]-12)

  #se calcula la frecuencia central y la calidad del filtro resultante
  f0 = (1/(2000*math.pi))*math.sqrt((R1+Rf)/(R1*R2*Rf*C1*C2))  #en KHz
  Q = (math.sqrt((R1+Rf)*R1*R2*Rf*C1*C2))/((R1*Rf*(C1+C2))+(R2*C2*(Rf-
((Rb/Ra)*R1))))

  #se imprimen los resultados obtenidos
  print("----------------------------------------------------------------")
  print("Genoma del mejor individuo:")
  print("%s" % mejor_ind)
  print("\nCalidad del mejor individuo:   %f" % calidad_ind)
  print("----------------------------------------------------------------")
  print("Valores de los componentes del filtro:")
  print("R1:   %E [Ω]" % R1)
  print("R2:   %E [Ω]" % R2)
```

```
  print("Rf:   %E [Ω]" % Rf)
  print("Ra:   %E [Ω]" % Ra)
  print("Rb:   %E [Ω]" % Rb)
  print("C1:   %E [F]" % C1)
  print("C2:   %E [F]" % C2)
  print("\nFrecuencia central del filtro:   %f [kHz]" % f0)
  print("\nCalidad Q del filtro:            %f" % Q)
  print("-----------------------------------------------------------------")


main() #llamada a la función main para ejecutar el algoritmo
```

B. *Resultados impresos por el código*

Para la solución #1

```
------------------------------------------------------------
Genoma del mejor individuo:
[(47, 2), (12, 3), (68, 0), (12, 9), (22, 7), (56, 5), (56, 4)]

Calidad del mejor individuo:   0.046078
------------------------------------------------------------
Valores de los componentes del filtro:
R1:   4.700000E+01 [Ω]
R2:   1.200000E+02 [Ω]
Rf:   6.800000E-01 [Ω]
Ra:   1.200000E+08 [Ω]
Rb:   2.200000E+06 [Ω]
C1:   5.600000E-06 [F]
C2:   5.600000E-07 [F]

Frecuencia central del filtro:   10.020888 [kHz]

Calidad Q del filtro:            4.100758
------------------------------------------------------------
```

Para la solución #2

```
------------------------------------------------------------
Genoma del mejor individuo:
[(47, 2), (68, 7), (56, 0), (47, 8), (47, 5), (56, 5), (12, 0)]

Calidad del mejor individuo:   0.009191
------------------------------------------------------------
Valores de los componentes del filtro:
R1:   4.700000E+01 [Ω]
R2:   6.800000E+06 [Ω]
Rf:   5.600000E-01 [Ω]
Ra:   4.700000E+07 [Ω]
Rb:   4.700000E+04 [Ω]
C1:   5.600000E-06 [F]
C2:   1.200000E-11 [F]

Frecuencia central del filtro:   10.008270 [kHz]

Calidad Q del filtro:            3.996317
------------------------------------------------------------
```