Algoritmo genético para determinar la impedancia necesaria de un componente para que un circuito AC le transfiera la máxima potencia de energía.

- **Introducción**

El algoritmo genético se basa en el proceso de la teoría de Darwin de la evolución. A partir de un conjunto de posibles soluciones, el algoritmo las modifica durante varias iteraciones esperando que converger en la solución más óptima. El proceso comienza con un conjunto de posibles soluciones o cromosomas (generalmente en forma de cadenas de bits) que se generan al azar o son seleccionados. El conjunto de estos cromosomas se denomina población. Los cromosomas evolucionan durante varias iteraciones o generaciones. Las nuevas generaciones (hijos) se generan mediante la técnica de cruce (Crossover) y mutación (Mutation). Crossover consiste en dividir dos cromosomas y combinar la mitad de cada uno con el otro par. La mutación consiste en cambiar un solo bit de un cromosoma. Los cromosomas entonces son evaluados usando un criterio determinado y los mejores se mantienen, mientras que se descartan los demás. Este proceso se repite hasta que un cromosoma tiene el mejor ajuste al criterio planteado y por lo tanto se considera como la mejor solución del problema.

El algoritmo genético tiene varias ventajas. Funciona bien para optimización global de funciones especialmente cuando la función objetivo es discontinua o con varios mínimos locales. Sin embargo, esto conduce a posibles inconvenientes. Dado que no se utiliza información adicional, tales como evaluación de gradientes y pendientes el algoritmo genético tiene una tasa de convergencia lenta con funciones objetivo sin muchas variaciones.

El algoritmo genético puede ser utilizado tanto en la optimización con o sin restricciones. Puede ser aplicado a la programación lineal, programación estocástica (el caso donde los problemas de programación matemática tienen variables aleatorias, introduciendo así un grado de incertidumbre), y los problemas de optimización combinatoria, como el Problema del Viajero Vendedor.

- **Aplicaciones**

Dado que el algoritmo genético puede ser utilizado para resolver problemas tanto sin y con restricciones, es simplemente una manera de obtener una solución a un problema de optimización estándar. Por lo tanto, se puede utilizar para resolver problemas de optimización clásicos tales como la maximización del volumen o la reducción al mínimo de la cantidad de material necesario para producir un recipiente. Mediante la aplicación de algoritmos genéticos a los problemas de programación lineal y no lineal, es posible resolver los problemas típicos como el problema de la dieta (eligiendo la más barata de un conjunto de alimentos que deben cumplir con ciertos requerimientos nutricionales). También se pueden aplicar a los problemas de optimización combinatoria en la ciencia de la computación como es el manejo de horarios.

- **Problema de máxima transferencia de potencia**

Se considera el circuito en la **Figura 1**, donde un circuito de corriente alterna (ac) se conecta al componente Z_L y se representa mediante su equivalente de Thévenin. El componente generalmente se representa por su impedancia que puede modelar un motor eléctrico, una antena, una TV, entre otros. En forma rectangular, la impedancia de Z_{Th} y Z_L son:

$$Z_{Th} = R_{Th} + jX_{Th}$$

$$Z_L = R_L + jX_L$$

Donde, R es la resistencia y X la reactancia.

Entonces la corriente a través de componente es,

$$I = \frac{V_{Th}}{Z_{Th} + Z_{Th}} = \frac{V_{Th}}{\left(R_{Th} + jX_{Th}\right) + \left(R_L + jX_L\right)}$$

Y la potencia promedio

$$P = \frac{1}{2}\left|I^2\right|R_L = \frac{1}{2}R_L \frac{\left(V_{Th}\right)^2}{\left(R_{Th} + R_L\right)^2 + \left(X_{Th} + X_L\right)^2}$$

El objetivo es variar los parámetros R_L y X_L para obtener P máxima. Matemáticamente, esto se puede hacer derivando parcialmente con respecto a cada parámetro e igualando a cero. Con esto se obtiene:

$$\frac{\partial P}{\partial X_L} = -\frac{(V_{Th})^2 R_L (X_{Th} + X_L)}{\left[(R_{Th} + R_L)^2 + (X_{Th} + X_L)^2\right]^2}$$

$$\frac{\partial P}{\partial R_L} = -\frac{(V_{Th})^2 \left[(X_{Th} + X_L)^2 + (R_{Th} + R_L)^2 - 2R_L (R_{Th} + R_L)\right]}{\left[(R_{Th} + R_L)^2 + (X_{Th} + X_L)^2\right]^2}$$

Igualando a cero $\frac{\partial P}{\partial X_L}$ obtenemos

$$X_L = -X_{Th}$$

E igualando a cero $\frac{\partial P}{\partial R_L}$,

$$R_L = \sqrt{(R_L)^2 + (X_{Th} + X_L)^2}$$

Y si $X_L = -X_{Th}$, $R_L = R_{Th}$. Por lo que para obtener la máxima potencia la impedancia del componente debe ser:

$$Z_L = R_L + jX_L = R_{Th} - jX_{Th}$$

Ahora bien, en este documento se pretende encontrar estos valores por medio de un algoritmo genético que maximice la función *P* de potencia variando los parámetros R_L y X_L, dados R_{th} y X_{Th} fijos.

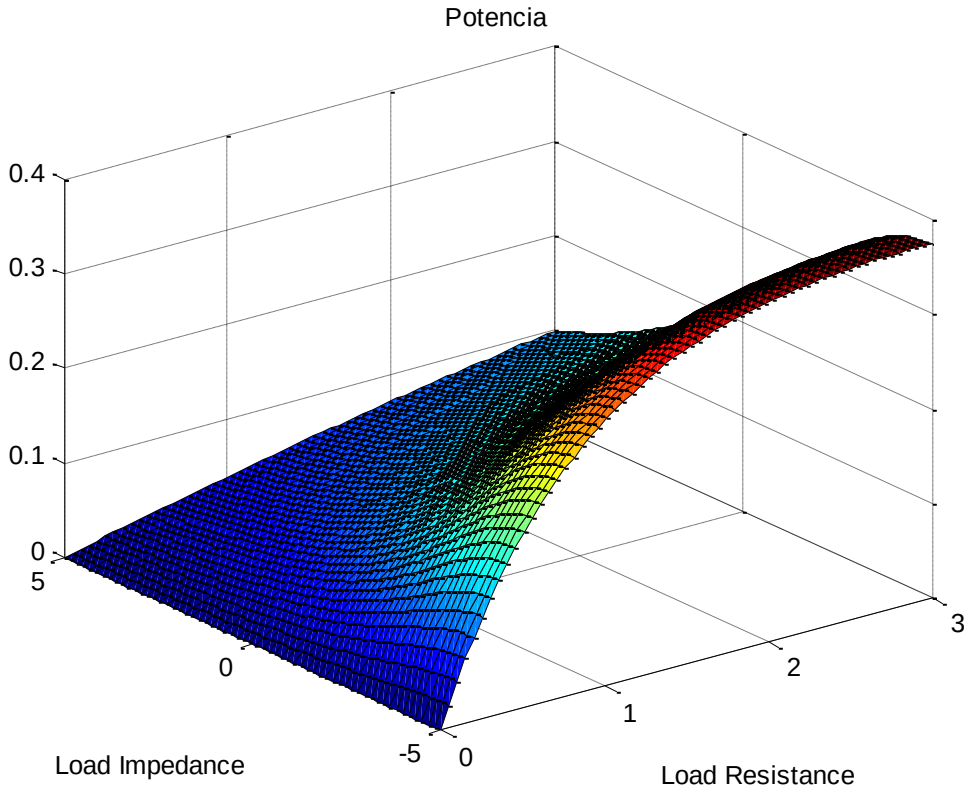
- **El algoritmo genético**

1. Problema de optimización

La función objetivo es la ecuación de la Potencia, la cual se muestra a continuación junto con una gráfica representativa.

$$P = \frac{1}{2} R_L \frac{(V_{Th})^2}{(R_{Th} + R_L)^2 + (X_{Th} + X_L)^2} \qquad \textit{Función obejtivo}$$

Gráfica1. Función de Potencia



2. Representación

Es necesario codificar las variables de decisión en una cadena binaria. Aquí se utilizan 16 bits para representar una variable. La asignación de una cadena binaria a un número real para las variables R_L y X_L se completa como sigue:

variable = valor decimal*((rango(sup)-rango(inf))/(2^length(bit)-1)) + range(inf)

Por ejemplo, la longitud total de un cromosoma es de 32 bits, 16 para la variable R_L y 16 para X_L:

1110011011001100 0110101110000010

Entonces:

Número binario	**Valor decimal**
1110011011001100	59076
0110101110000010	27522

Suponiendo que para ambas variables el rango superior es 2 y el inferior -2,

$$VariableR_L = \left(59706 \times \frac{4}{2^{16}-1}\right) - 2 = 1.606$$

$$VariableX_L = \left(27522 \times \frac{4}{2^{16}-1}\right) - 2 = -0.320$$

3. Población inicial

En cada generación el tamaño de la población es 20. La población inicial se genera aleatoriamente como sigue:

TABLA1. Población inicial

#	Población inicial (Binario)	Pob. Inicial (Decimal)
1	0 1 1 1 0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 0 0 0 0 1 0 1 0 1 0 1 1 1	f(1.342870, -1.197604)=0.181367
2	1 0 0 1 0 0 0 1 1 1 1 1 1 1 0 1 1 0 1 1 0 1 0 1 1 1 0 0 1 0 0 0	f(1.710826, 2.100938)=0.106021
3	0 0 0 1 1 1 1 0 1 1 1 1 0 1 0 0 1 0 0 0 1 0 1 1 1 1 0 1 1 0 0 1	f(0.362737, 0.462882)=0.039666
4	1 1 0 1 0 1 1 0 1 0 0 1 1 1 1 0 1 0 1 0 0 0 1 1 1 0 1 1 1 0 0 1	f(2.515084, 1.395514)=0.158694
5	1 1 1 0 1 0 1 1 1 1 0 1 1 0 1 0 0 0 1 1 1 1 1 1 0 1 1 0 0 1 1 0 1	f(2.763928, -2.546807)=0.316956
6	1 1 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1 0 0 1 0 1 1 0 1 1 0 0 0 1 1	f(2.251087, -2.055161)=0.279476
7	0 1 0 0 0 0 1 1 0 1 1 1 1 0 1 0 0 1 0 1 1 1 0 0 0 0 0 0 1 1 0 1	f(0.790753, -1.404211)=0.130346
8	0 0 0 1 0 1 0 1 0 0 0 0 1 1 1 0 1 0 1 1 0 0 1 1 0 1 0 1 0 0 0 1	f(0.246738, 2.004654)=0.018628
9	1 0 1 0 0 1 1 1 0 1 1 0 0 0 1 0 0 0 1 1 1 1 0 1 0 1 1 1 0 1 1 0	f(1.961547, -2.599145)=0.290542
10	0 0 0 0 1 0 0 0 0 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 0 0 0 0 1 0	f(0.097276, 2.119554)=0.007262
11	1 1 0 1 0 1 0 0 0 0 1 0 0 0 1 1 1 0 1 1 1 1 0 0 0 0 1 0 0 0 1 0	f(2.486015, 2.349050)=0.133013
12	0 1 0 0 1 1 0 0 1 1 1 1 1 1 1 0 1 0 1 0 1 0 1 1 0 1 0 1 0 0 0 1 1	f(0.902220, -1.615702)=0.152169
13	1 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 0 1 1 1 0 0	f(2.449256, -4.546502)=0.368616
14	1 1 0 1 1 0 0 1 1 1 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0 0 1 1 0 1 0 1	f(2.551888, 3.133211)=0.118420
15	0 1 1 0 0 0 0 1 1 1 1 0 1 0 0 0 1 1 1 1 1 1 1 0 0 0 1 1 1 0 1 0 0	f(1.147356, 4.861601)=0.045112
16	1 0 1 1 1 0 1 1 0 0 0 0 0 0 0 1 0 1 1 1 0 0 1 1 1 1 1 1 0 0 1 1 0	f(2.191485, -0.472648)=0.207840
17	1 0 1 1 0 1 1 1 1 1 1 1 0 1 1 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0	f(2.155871, -4.804379)=0.364424
18	1 1 1 0 1 0 1 1 0 0 1 0 0 0 0 1 1 1 1 1 0 0 0 1 1 0 1 1 1 1 0 1	f(2.755459, 4.443046)=0.101389
19	0 0 1 0 1 0 0 1 1 0 1 0 0 1 0 1 1 1 0 0 1 1 1 0 1 0 0 1 0 1 1 1	f(0.488029, 3.070039)=0.028413
20	0 0 0 1 1 1 1 0 1 1 0 0 0 0 0 1 0 0 0 1 1 1 1 0 0 0 1 1 1 0 1 1	f(0.360403, -3.819104)=0.127834

4. Evaluación

El primer paso después de la creación de una generación es la de calcular el valor de aptitud de cada miembro. El proceso de evaluación de la aptitud de un cromosoma consiste en los tres pasos siguientes:

1) Convertir el genotipo del cromosoma a su fenotipo. Esto significa convertir a las cadenas binarias en los correspondientes valores reales.

2) Evaluar la función objetivo.

3) Convertir el valor de la función objetivo en un valor de ajuste. Aquí, a fin de sólo tener valores positivos, la aptitud de cada cromosoma es igual a la función objetivo evaluada para cada cromosoma menos el mínimo valor obtenido para la función objetivo.

Los valores de la función objetivo F y la aptitud de los cromosomas de los valores Eval de la población de arriba mostrada son los siguientes:

Tabla2. Ajuste de la población inicial

#	Función objetivo	Eval (Ajuste)= F(#)-Fmin
1	f(1.342870, -1.197604)=0.181367	0.174
2	f(1.710826, 2.100938)=0.106021	0.099
3	f(0.362737, 0.462882)=0.039666	0.032
4	f(2.515084, 1.395514)=0.158694	0.151
5	f(2.763928, -2.546807)=0.316956	0.310
6	f(2.251087, -2.055161)=0.279476	0.272
7	f(0.790753, -1.404211)=0.130346	0.123
8	f(0.246738, 2.004654)=0.018628	0.011
9	f(1.961547, -2.599145)=0.290542	0.283
10	f(0.097276, 2.119554)=0.007262	0.000
11	f(2.486015, 2.349050)=0.133013	0.126
12	f(0.902220, -1.615702)=0.152169	0.145
13	f(2.449256, -4.546502)=0.368616	0.361
14	f(2.551888, 3.133211)=0.118420	0.111
15	f(1.147356, 4.861601)=0.045112	0.038
16	f(2.191485, -0.472648)=0.207840	0.201
17	f(2.155871, -4.804379)=0.364424	0.357
18	f(2.755459, 4.443046)=0.101389	0.094
19	f(0.488029, 3.070039)=0.028413	0.021
20	f(0.360403, -3.819104)=0.127834	0.121

En esta primera generación, el cromosoma con mejor ajuste es el 13 y el 10 el peor, ya que se quiere un máximo.

5. Crear una nueva población

Después de la evaluación, tenemos que crear una nueva población de la generación actual. Los tres operadores (reproducción, cruce, y mutación) se utilizan.

- Reproducción

Los dos cromosomas (cadenas) con la aptitud pueden "vivir" y producir descendencia en la próxima generación. Por ejemplo, en la primera población, los cromosomas 13 y 16 se les permite vivir en la segunda población.

- Selección y Cruce

La probabilidad acumulada se utiliza para decidir que cromosomas se seleccionarán para cruce. La probabilidad acumulada se calcula en los siguientes pasos:

a) Calcular el ajuste total de la población.

$$AjsuteTotal = \sum_{i=1}^{TamañoPob} Ajsute(i)$$

b) Calcular la probabilidad *Pi* para cada cromosoma.

$$P_i = \frac{Ajsute(i)}{AjsuteTotal}$$

c) Calcular la probabilidad acumulada *Qi* para cada cromosoma.

$$Q_k = \sum_{k=0}^{i} P_k$$

Para la población mostrada anteriormente el ajuste total es de 3.03096, y la probabilidad de selección y acumulada para cada individuo es:

TABLA3. Probabilidad de selección y probabilidad acumulada para cada miembro.

i	P_i	Q_i
1	0.057	0.057
2	0.033	0.090
3	0.011	0.101
4	0.050	0.151
5	0.102	0.253
6	0.090	0.343
7	0.041	0.383
8	0.004	0.387
9	0.093	0.480
10	0.000	0.480
11	0.041	0.522
12	0.048	0.570

13	0.119	0.689
14	0.037	0.726
15	0.012	0.738
16	0.066	0.804
17	0.118	0.922
18	0.031	0.953
19	0.007	0.960
20	0.040	1.000

El cruce se utiliza aquí es el método de punto de corte (Xover point), que selecciona al azar un punto de corte e intercambia la parte derecha de los padres para generar descendencia.

a) Generar un número aleatorio *r* en el intervalo [0,1];
b) Si Qi-1 < *r* < = Qi, se selecciona el i-ésimo cromosoma para ser el padre 1.
c) Repita el paso a y b para reproducir los padres dos.
d) Generar un número aleatorio r en el intervalo [0,1]. Si *r* es menor que la probabilidad de cruce (elegimos la probabilidad de cruce igual a 1,0), el cruce se realiza, el punto de corte se selecciona detrás del gen cuyo lugar es el entero más cercano (mayor o igual) a *r*x (longitud(bit)-1). En este caso, la longitud es de 32 bits.
e) Se repita del paso *a* al paso *d* en total nueve veces para terminar el cruce. La creación de 18 hijos, más 2 cromosomas (lo más aptos de la anterior) mantiene la población de 20 en cada generación.

Por ejemplo el primer cruce de la población mostrada es:

Punto de cruce = 22

Padre1

1 1 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1 0 0 1 0 1 1 0 1 1 0 0 0 1 1

Padre2

1 1 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1 0 0 1 0 1 1 0 1 1 0 0 0 1 1

Nueva población1

1 1 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1 0 0 1 0 1 1 0 1 1 0 0 0 1 1

Nueva población2

1 1 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1 0 0 1 0 1 1 0 1 1 0 0 0 1 1

- Mutación

La mutación se realiza después del cruce. La mutación altera uno o más genes con una probabilidad igual a la tasa de mutación. (En este caso, la probabilidad de mutación es 0.01).

a) Generar una secuencia de números aleatorios de r_k (k = 1 ,....., 640) (El número de bits total en una población 20 × 32 = 640).
b) Si r_i es 1, se cambia el bit i-ésimo de 1 a 0 o de 0 a 1.
c) Los cromosomas elite de la población anterior no están sujetos a mutación, así que después de la mutación, deben ser restaurados.

Una nueva población es creada después de una iteración del algoritmo genético. Este procedimiento se puede repetir las veces deseadas pero en este ejemplo se cierra el ciclo cuando no se muestra una mejoría en el mejor miembro de cada generación después de 20 iteraciones. Para este ejemplo, se tomo el voltaje de Thevenin como 3.0 volts, la resistencia de Thevenin como 3 ohms y la reactancia como 5j. Se muestra al individuo con el mejor ajuste en cada iteración.

$$V_{Th} = 3.0[V] \quad R_{Th} = 3.0[\Omega] \quad X_L = 5.0\,j[\Omega]$$

Generación 2: f(2.449256, -4.546502)=0.368616
Generación 3: f(2.449256, -4.649958)=0.369644
Generación 4: f(2.824262, -4.533989)=0.372275
Generación 5: f(2.824262, -4.533989)=0.372275
Generación 6: f(2.825727, -4.533837)=0.372281
Generación 7: f(2.825727, -4.649958)=0.373317
Generación 8: f(2.825727, -4.649958)=0.373317
Generación 9: f(2.825727, -4.649958)=0.373317
Generación 10: f(2.918013, -4.649958)=0.373621
Generación 20: f(2.966354, -4.962463)=0.374973
Generación 30: f(2.966354, -4.985657)=0.374986
Generación 40: f(2.967269, -4.998474)=0.374989
Generación 50: f(2.973953, -4.996033)=0.374993

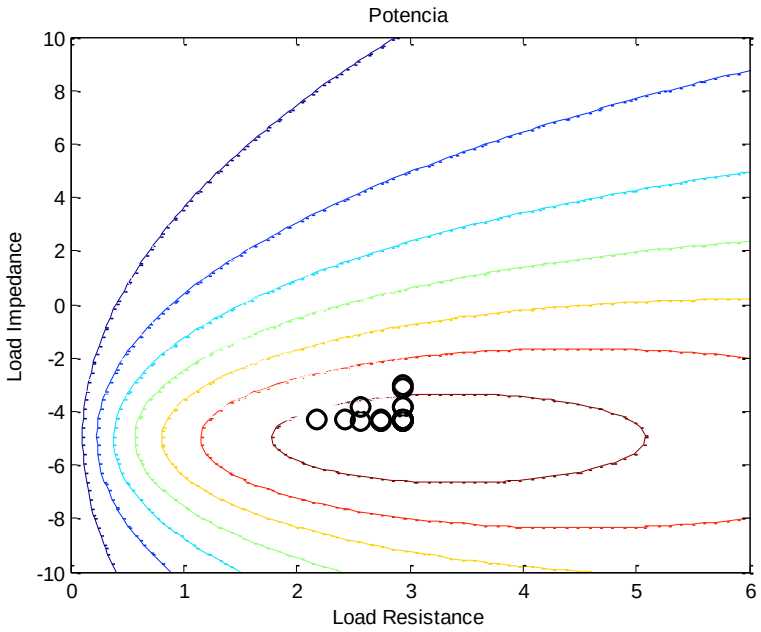
Generación 60: f(2.976608, -4.995422)=0.374994
Generación 70: f(2.997391, -4.998779)=0.375000
Generación 80: f(2.998535, -4.998779)=0.375000
Generación 90: f(2.998718, -5.000000)=0.375000
Generación 100: f(2.999268, -5.000000)=0.375000
Generación 110: f(3.000000, -5.000000)=0.375000
Generación 111: f(3.000000, -5.000000)=0.375000
Generación 112: f(3.000000, -5.000000)=0.375000
Generación 113: f(3.000000, -5.000000)=0.375000
Generación 114: f(3.000000, -5.000000)=0.375000
Generación 115: f(3.000000, -5.000000)=0.375000
Generación 116: f(3.000000, -5.000000)=0.375000
Generación 117: f(3.000000, -5.000000)=0.375000
Generación 118: f(3.000000, -5.000000)=0.375000
Generación 119: f(3.000000, -5.000000)=0.375000
Generación 120: f(3.000000, -5.000000)=0.375000
Generación 121: f(3.000000, -5.000000)=0.375000
Generación 122: f(3.000000, -5.000000)=0.375000
Generación 123: f(3.000000, -5.000000)=0.375000
Generación 124: f(3.000000, -5.000000)=0.375000
Generación 125: f(3.000000, -5.000000)=0.375000
Generación 126: f(3.000000, -5.000000)=0.375000
Generación 127: f(3.000000, -5.000000)=0.375000

Maxima potencia: 3.750000e-001 Watts
Cuando: Rth = 3.000000 ohms, Xth = -5.000000j ohms

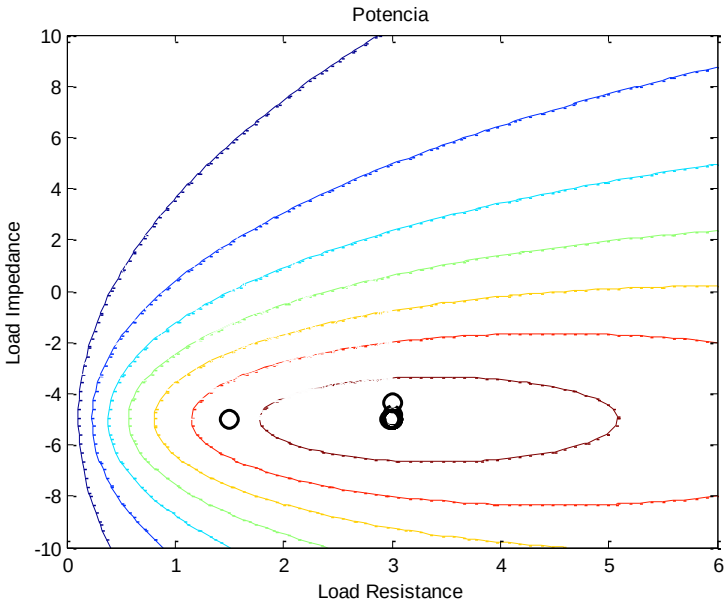
Así comprobamos lo que matemáticamente sabíamos, la máxima potencia se alcanza cuando la impedancia del componente conectado a un circuito ac es el conjugado de la impedancia de dicho circuito.

Se muestran unas gráficas con el mapa de contornos de la función potencia, y la ubicación de los miembros creados por el algoritmo genético para la décima generación y la última para ver su convergencia.

Gráfica2. Mapa de contornos de la función objetivo con la población inicial.



Gráfica3. Mapa de contornos de la función objetivo con la última generación.



- **Apéndice – Archivos M de Matlab para el algoritmo.**

MAIN

```
popuSize = 20; % Población inicial
xover_rate = 1.0; % Tasa de Cruzamiento
mutate_rate = 0.01; % Tasa de mutación
bit_n = 16; % Número de bits para cada variable
limit = 0;
global Vth;
global Rth;
global Xth;
%Se pide por los valores de entrada
Vth=input('Valor del voltaje Vth: ');
if isempty(Vth)
    Vth=0;
end
Rth=input('Valor de la resistencia Rth: ');
if isempty(Rth)
    Rth=0;
end
Xth=input('Valor de la impedancia Xth: ');
if isempty(Xth)
    Xth=0;
end

obj_fcn = 'G_Pfunction2';% Función objetivo
var_n = 2; % Numero de variables de entrada
range = [0, Rth; -Xth, Xth]; % Rango de la entradas
% Grafica de la función de potencia
syms RL;
syms XL;
z = (0.5*(Vth^2)*RL)/((Rth+RL)^2+(Xth+XL)^2);
figure;
ezsurf(z,[0,Rth,-Xth,Xth]);
xlabel('Load Resistance'); ylabel('Load Impedance'); title('Potencia');
% Mapa de contornos de la función potencia
figure;
ezcontour(z,[0,2*Rth,-2*Xth,2*Xth]);
hold off;
xlabel('Load Resistance'); ylabel('Load Impedance'); title('Potencia');

% Creación de la población inicial
popu = rand(popuSize, bit_n*var_n) > 0.5;

fprintf('Población inicial.\n');
for i=1:popuSize
for j=1:bit_n*var_n
fprintf('%1.0f ',popu(i,j));
end
fprintf('\n');
end

upper = zeros(50, 1); %Creación de la matriz de mejor individuo
```

```
% Loop Principal
i=0;
%Cuando el ind. Más apto no mejora después de 20 generaciones el GA
%termina
while(limit <=20)
i=i+1;
k=i;
% Reseteo de variables
delete(findobj(0, 'tag', 'member'));
delete(findobj(0, 'tag', 'count'));
% Evaluación de la función para cada miembro de la población
fcn_value = evalpopu(popu, bit_n, range, obj_fcn);
if (i==1),
fprintf('Población inicial\n ');
for j=1:popuSize
fprintf('f(%f, %f)=%f\n', ...
bit2num(popu(j, 1:bit_n), range(1,:)), ...
bit2num(popu(j, bit_n+1:2*bit_n), range(2,:)), ...
fcn_value(j));
end
end

% Llenar la matriz con los mejores individuos
upper(i) = max(fcn_value);

%Registro de si el ind. Más apto mejora o no
if(i>=2)
    if(upper(i)==upper(i-1))
        limit = limit +1;
    else
        limit=0;
    end
end

% Grafica de los mejores individuos
[best, index] = max(fcn_value);
fprintf('Generación %i: ', i);
fprintf('f(%f, %f)=%f\n', ...
bit2num(popu(index, 1:bit_n), range(1,:)), ...
bit2num(popu(index, bit_n+1:2*bit_n), range(2,:)), ...
best);
% Creación de la siguiente población con selección, cruce y mutación
popu = nextpopu(popu, fcn_value, xover_rate, mutate_rate,k);
if(mod(i,10)==0)
fprintf('Población después de la %d ° generación.\n',i);
fprintf('Presione cualquier tecla...\n');
pause;
end
%end
end
[best, index] = max(fcn_value);
fprintf('\nMaxima potencia: %i Watts\n ', best);
fprintf('Cuando: Rth = %f ohms, Xth = %fj ohms\n', ...
bit2num(popu(index, 1:bit_n), range(1,:)), ...
bit2num(popu(index, bit_n+1:2*bit_n), range(2,:)));
```

Función objetivo

```
function z = G_Pfunction2(input)
%Regresa el valor de la función de acuerdo al input.

global PREV_PT % Salida de la función anterior
global Vth;
global Rth;
global Xth;
RL= input(1); XL = input(2);
z = (0.5*(Vth^2)*RL)/((Rth+RL)^2+(Xth+XL)^2);
% Grafica de la función potencia
property='Marker';
line(RL, XL, property, 'o', 'markersize', 10, ...
'clipping', 'off', 'erase', 'xor', 'color', 'k', ...
'tag', 'member', 'linewidth', 2);

if ~isempty(PREV_PT),% Actualiza la grafica
line([PREV_PT(1) RL], [PREV_PT(2) XL], 'linewidth', 1, ...
'clipping', 'off', 'erase', 'none', ...
'color', 'w', 'tag', 'traj');
end

PREV_PT = [RL XL];
drawnow;
```

Eval. Población

```
function fitness = evalpopu(popu, bit_n, range, obj_fcn)
%EVALPOPU Evaluación de ajuste de la población.

global count %Contador global de individuos
pop_n = size(popu, 1);
fitness = zeros(pop_n, 1);
%Evaluación miembro por miembro
for count = 1:pop_n,
fitness(count) = evaleach(popu(count, :), bit_n, range, obj_fcn);
end
```

Eval. Each

```
function out = evaleach(string, bit_n, range, obj_fcn)
% EVALEACH Evaluación de cada individuo.
% string: cadena de bits que representan un individuo.

var_n = length(string)/bit_n;
input = zeros(1, var_n);
%Ciclo para convertir cada indiviudo de bits a decimal
for i = 1:var_n,
input(i) = bit2num(string((i-1)*bit_n+1:i*bit_n), range(i, :));
end
out = feval(obj_fcn, input);%Regresa el valor de salida
```

Convertir BIT2NUM

```
function num = bit2num(bit, range)
% BIT2NUM Conversión de cadena de bits a número decimal.
integer = polyval(bit, 2);
num = integer*((range(2)-range(1))/(2^length(bit)-1)) + range(1);
```

NEXT Population

```
function new_popu = nextpopu(popu, fitness, xover_rate, mut_rate,k)
new_popu = popu;
popu_s = size(popu, 1);
string_leng = size(popu, 2);
% ====== ELITISMO: Se conservan los mejores 2 individuos
tmp_fitness = fitness;
[a, index1] = max(tmp_fitness); % Encuentra al mejor
tmp_fitness(index1) = min(tmp_fitness);
[a, index2] = max(tmp_fitness); % Encuentra al segundo mejor
new_popu([1 2], :) = popu([index1 index2], :);
% Se reescala la function de ajuste
fitness = fitness - min(fitness);% Positiva
total = sum(fitness);
if(k==1)
fprintf('Función de ajsute despues de nueva escala\n');
for i=1:popu_s
fprintf('%10.3f \n',fitness(i));
end
fprintf('La suma de cada ajuste %10.5f\n',total);
end
if total == 0,
fprintf('=== Error ===\n');
fitness = ones(popu_s, 1)/popu_s;% sum is 1
else
fitness = fitness/sum(fitness); % sum is 1
end
cum_prob = cumsum(fitness);
if(k==1)
fprintf('La probabilidad de cada cromosoma, y su prob. acumulada \n');
for i=1:popu_s
fprintf('%10.3f %10.3f\n',fitness(i),cum_prob(i));
end
end
% ====== SELECCIÓN Y CRUZAMIENTO
for i = 2:popu_s/2,
% === Se seleccionan dos padres de acuerdo a su nivel de ajuste
tmp = find(cum_prob - rand > 0);
parent1 = popu(tmp(1), :);
tmp = find(cum_prob - rand > 0);
parent2 = popu(tmp(1), :);
% === Se determina si se cruza o no
if rand < xover_rate,
% Operación de cruzamiento
xover_point = ceil(rand*(string_leng-1));
```

```
new_popu(i*2-1, :) = ...
[parent1(1:xover_point) parent2(xover_point+1:string_leng)];
new_popu(i*2, :) = ...
[parent2(1:xover_point) parent1(xover_point+1:string_leng)];
end
if(k==1)
fprintf('Punto de cruce = %d \n', xover_point);
fprintf('Padre1\n');
for j=1:string_leng
fprintf('%d ',parent1(j));
end
fprintf('\n');
fprintf('Padre2\n');
for j=1:string_leng
fprintf('%d ',parent2(j));
end
fprintf('\n');
fprintf('Nueva población1\n');
for j=1:string_leng
fprintf('%d ',new_popu(i*2-1,j))
end
fprintf('\n');
fprintf(' Nueva población2\n');
for j=1:string_leng
fprintf('%d ',new_popu(i*2,j))
end
fprintf('\n');
end
end

if(k==1)
fprintf('El resultado después del cruce de la 1a población\n');
for i=1:popu_s
for j=1:string_leng
fprintf('%d ',new_popu(i,j))
end
fprintf('\n');
fprintf('\n');
end
end
% ====== MUTACION (los elites no se mutan)
mask = rand(popu_s, string_leng) < mut_rate;
new_popu = xor(new_popu, mask);
if(k==1)
fprintf('El resultado de la mutación de la 1a población\n');
for i=1:popu_s
for j=1:string_leng
fprintf('%d ',new_popu(i,j))
end
fprintf('\n');
fprintf('\n');
end
end
% Se restaurant los miembros elite
new_popu([1 2], :) = popu([index1 index2], :);
```