Codificación de la función de *fitness* y búsqueda de soluciones mediante Algoritmos Genéticos

(Global Optimization Toolbox)

Introducción

Estas páginas contienen un ejemplo de la creación de una función de fitness y optimizar esa función, minimizar el valo, mediante el uso de un Algoritmo Genético (GA). El GA esa función se implementó con la *toolbox* de optimización global de MATLAB.

Por ejemplo, si se desea encontrar la combinación de dos valores que minimice la siguiente función:

$$min\ f(x)\ =\ 100 * ({x_1}^2 - x_2)^2\ + (1 - x_1)^2$$

En primer lugar, tenemos que crear un archivo en MATLAB llamado "simple_fitness.m" con la representación de la función a minimizar en MATLAB:

```
function fitness = simple_fitness(individual)
fitness = 100 *(individual(1)^2 - individual(2))^2 + (1-individual(1))^2
```

EL toolbox de MATLAB's para Algoritmos Genéticos asumirá un único valor de entrada para la función de fitness que en este caso sería un vector, individual. El vector de entrada tendría tantos valores como la cantidad de variables en el problema, siendo una función en este caso particular. Debe devolver un valor de *fitness* que es el resultado de la evaluación de un individuo con la función de *fitness* definida.

El siguiente paso consiste en proporcionar al AG la función de *fitness* y el número de variables para minimizarla:

```
FitnessFunction = @simple_fitness

numberOfVariables = 2;

[x, fval] =ga(FitnessFunction, numberOfVariables)
```

```
x =          0.9652      0.9340

fval=        0.0017
```

La toolbox devolverá los mejores valores para x dentro de la población creada y evolucionada por el AG. Por sí solo, fval tiene el valor de aptitud para el mejor **x**.

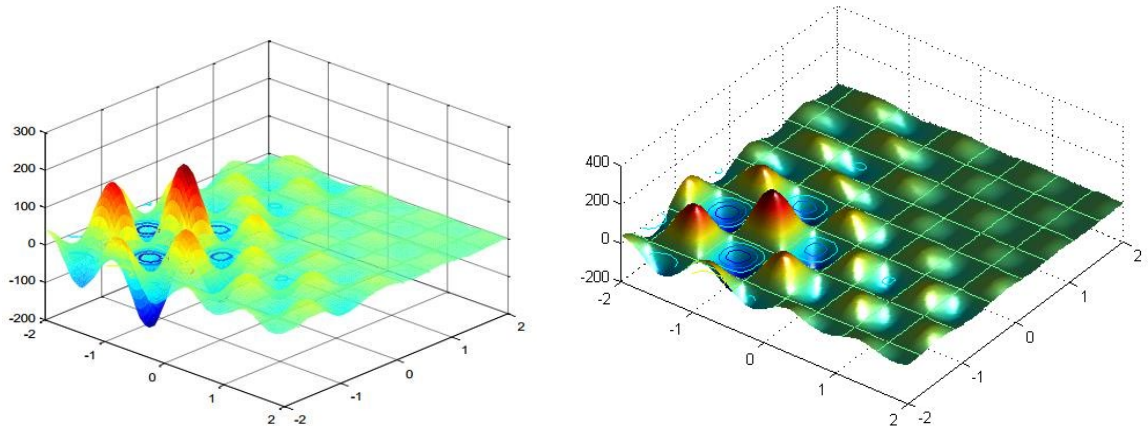
Opciones de Genetic Algorithm

La función de MATLAB (Toolbox) llamada 'ga' busca el valor mínimo de una función usando un Algoritmo Genético. Supongamos que se requiere el valor mínimo de una función llamada 'SHUFCN'. Esa función está en MATLAB y tiene dos variables de entrada. El código es como:

```
%SHUFCN Shu objective function

function f = shufcn(y)

for j=1:size(y)

        f(j) = 0.0;
        x = y(j, :);
        temp1 = 0;
        temp2 = 0;
        x1 = x(1);
        x2 = x(2);
        for i = 1:5
                temp1 = temp1 + i .* cos((i + 1).*x1 + i);
                temp2 = temp2 + i .* cos((i + 1).*x2 + i);
        end
        f(j) = temp1 .* temp2;
end
```

PLOTOBJECTIVE se puede utilizar para dibujar la función en el rango de [-2 2; -2 2] como:

```
plotobjective(@shufcn, [-2 2;-2 2]);
```



La función shu puede invocarse estableciendo el número de variables y poniendo FitnessFunction con el puntero adecuado

```
FitnessFunction = @shufcn

numberOfVariables = 2;
```

Las dos primeras salidas de 'ga' son los mejores puntos, x, alcanzados y su valor de fitness, fval. Además, se puede encontrar una tercera salida, exitFlag, que indica la razón para detener el AG. Por último, la cuarta salida contiene información adicional sobre el rendimiento del algoritmo, como el número de generaciones, el número de evaluaciones de aptitud, etc.

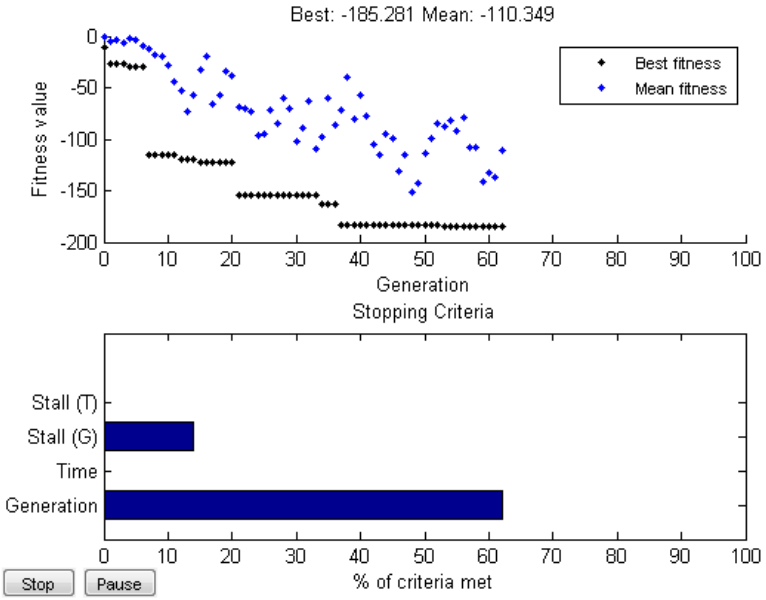
```
[x, Fval, exitFlag, Output] = ga(FitnessFunction, numberOfVaribles);

fprintf('Number of Generations: %d\n', Output.generations);
fprintf('Number of Fitness Evaluations: %d\n', Output.funccount);
fprintf('Best value:  %d\n', Fval);

Number of Generations: 54
Number of Fitness Evaluations: 1100
Best value: -186.381
```

El AG parte de una población aleatoria y utiliza un conjunto de operadores genéticos sobre esa población para encontrar la solución de un problema. A partir de la población inicial se obtienen las siguientes aplicando los operadores y se calcula su bondad gracias a la función fitness.
Se pueden invocar otras opciones para analizar el comportamiento del algoritmo sobre un problema. La función 'ga' acepta una o más opciones de trazado con el argumento OPTIONS. Se pueden elegir diferentes funciones de trazado utilizando 'gaoptimset'. Por ejemplo, se puede definir una estructura de opciones con 2 funciones de trazado, primero, 'GAPLOTBESTF' que traza el mejor y el promedio para cada iteración y, segundo, 'GAPLOTSTOPPING' que traza el porcentaje de la condición de parada establecida en el algoritmo.

```
opts = gaoptimset('PlotFcns',{ @gaplotbestf, @gaplotstopping});
```

Esa llamada define la estructura que posteriormente se puede utilizar en la llamada a la función 'ga':

```
[x, Fval, exitFlag, Output] = ga(FitnessFunction, numberOfVariables, opts);
```



También es posible modificar los criterios de parada del algoritmo. Hay 4 criterios de parada posibles:

1. El algoritmo se detiene cuando se alcanza un número máximo de generaciones, cuyo valor por defecto es 100.
2. El algoritmo termina cuando no se mejora el mejor valor de fitness después de N segundos
3. Algoritmo terminará, si después de N generaciones el mejor valor de fitness no se modifica.
4. El algoritmo se detiene después de un tiempo máximo establecido en segundos.

A continuación, se establecen dos criterios de parada. En primer lugar, el número máximo de generaciones se ha elevado a 150 y, en segundo lugar, el número de generaciones sin ninguna mejora se ha fijado en 100.

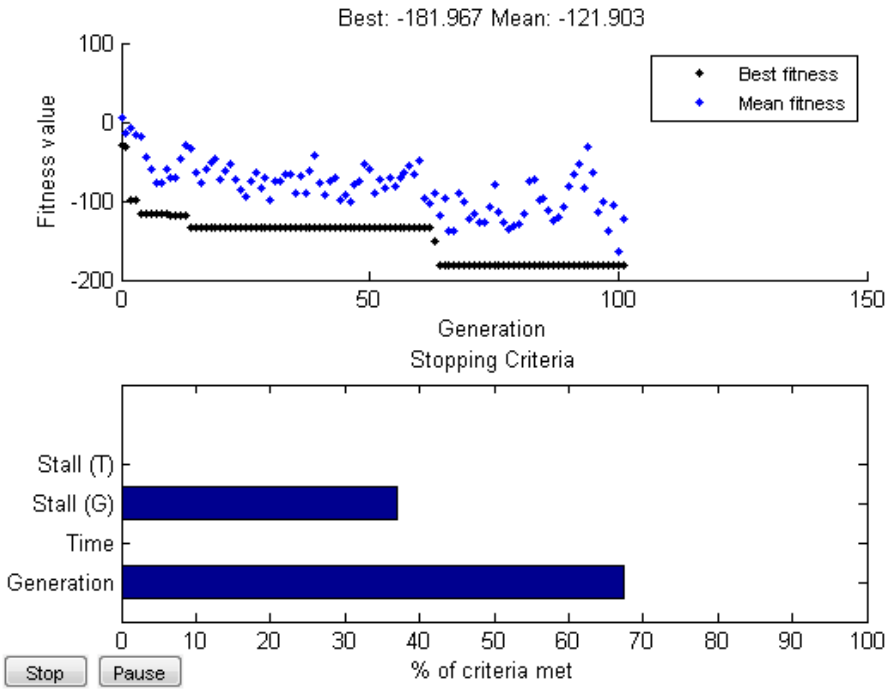
```
opts = gaoptimset('Generations',150,'SmallGenLimit', 100);
```

Si ejecutamos estas opciones debería verse así:

```
[x, Fval, exitFlag, Output] = ga(FitnessFunction, numberOfVaribles, [], [], [], [], [], [], opts);
```

```
fprintf('Number of Generations: %d\n', Output.generations);
fprintf('Number of Fitness Evaluations: %d\n', Output.funccount);
fprintf('Best value:  %d\n', Fval);
```

```
Number of Generations: 101
Number of Fitness Evaluations: 1020
Best value: -181.967
```



El tamaño inicial de la población, que tiene como valor por defecto 20 ,también podría ser modificado. El valor por defecto es seguramente un poco bajo para cualquier problema con una, variable de dimensionalidad media/alta. 'PopulationSize' puede pasarse a una estructura 'opts' para modificar ese parámetro.

```
opts = gaoptimset(opts,'PopulationSize', 10);
```

El rango inicial podría establecerse mediante la opción 'PopInitRange', que debe tener como valor una matriz con dos filas. Esas filas establecen el umbral máximo y mínimo para cada valor, por lo tanto debe haber tantas columnas como variables tenga el problema. En caso de que la matriz tenga una sola columna se supone que es el rango para cualquier variable. Por ejemplo, una matriz [-1 0;1 2] establece un rango [-1 1] para la primera variable y [0 2] para la segunda.

```
opts = gaoptimset(opts,'PopInitRange', [-1 0; 1 2]);
```

El 'ga' utiliza por defecto un generador de números aleatorios para generar la población inicial. Mientras que, las siguientes generaciones se generan utilizando el operador genético, que utiliza el mismo generador de números aleatorios, en la generación anterior. Por lo tanto, el estado del generador de números aleatorios cambia cada vez que se genera un número aleatorio. Como consecuencia, cuando el algoritmo se detiene y se invoca una nueva instancia de "ga", los resultados de ambas ejecuciones serían diferentes. Por ejemplo:

```
[x, Fval, exitFlag, Output] = ga(FitnessFunction, numberOfVaribles);

fprintf('Best value: ', Fval);
```

```
Best Value: -186.209
```

Una vez más:

```
[x, Fval, exitFlag, Output] = ga(FitnessFunction, numberOfVaribles);

fprintf('Best value: ', Fval);
```

```
Best Value: -186.362
```

Las anteriores ejecuciones de 'ga' tuvieron diferentes resultados debido a la aleatoriedad y a los diferentes estados del generador aleatorio dentro de la ejecución del algoritmo GA. Ese estado influye en los diferentes operadores genéticos que son:

- Selección
- Cruce
- Mutación

La Toolbox de MATLAB dispone de varias funciones para cada uno de los operadores. Por ejemplo, en el siguiente ejemplo, 'seleciontournament' se establece como método de selección 'SelectionFcn'; el cruce se establece para un único punto con una proporción del 90%; y la mutación es una mutación uniforme con un 10% de probabilidad.

```
opts = gaoptimset(opts,'SelectionFcn', @selectiontournament);
opts = gaoptimset(opts,'CrossoverFcn', @crossoversinglepoint);
opts = gaoptimset(opts,'CrossoverFraction', 0.9);
opts = gaoptimset(opts,'MutationFcn', {@mutationuniform, 0.1});
```

Once the algorithm is executed:

```
[x, Fval, exitFlag, Output] = ga(FitnessFunction, numberOfVaribles,opts);

fprintf('Number of Generations: %d\n', Output.generations);
fprintf('Number of Fitness Evaluations: %d\n', Output.funccount);
fprintf('Best value:  %d\n', Fval);
```

```
Number of Generations: 51
Number of Fitness Evaluations: 1040
Best value: -180.399
```

Una elección diferente para cada operador podría mejorar los resultados o empeorarlos. La selección dependería del problema y es frecuente que se ejecuten varias pruebas para comprobar los enfoques más adecuados.

Compruebe las posibles opciones en:
https://es.mathworks.com/help/gads/ga.html#mw_4a8bfdb9-7c4c-4302-8f47-d260b7a43e26