PROGRAMACION EVOLUTIVA A PARTIR DE UN EJEMPLO

Horacio Caraballo, Cecilia González
Facultad de Ciencias Agrarias y Forestales. Universidad Nacional de La Plata. Argentina.
caraballohoracio@gmail.com cgonzalez@agro.unlp.edu.ar

Área temática: Aplicaciones de la Matemática
Palabras clave: programación evolutiva, algoritmo genético, ejemplo.

Resumen

Partiendo de las ideas de la teoría de la evolución se pueden desarrollar algoritmos que resuelvan problemas de búsqueda y optimización según el principio de "supervivencia del más apto". Estos algoritmos se denominan algoritmos evolutivos y conforman en general el tema de la computación evolutiva. A partir de ésta se han resuelto de manera exitosa una amplia variedad de problemas.

Nuestro propósito es motivar el estudio del tema mostrando en detalle un ejemplo de búsqueda de un extremo de una función real a partir de la técnica de los algoritmos genéticos. Si bien este ejemplo es elemental y la técnica podría considerarse no adecuada, cumple el objetivo de ilustrar el método y su estructura.

Introducción

En este artículo se muestra un ejemplo detallado sobre la construcción de un algoritmo genético aplicado a encontrar el extremo de una función real de una variable. La intención es facilitar la iniciación en el tema de la programación evolutiva. Desde un punto de vista técnico el problema elegido no es indicado para este método pero desde un punto de vista didáctico es ideal, la posibilidad de mostrar gráficamente a las distintas poblaciones evolucionando hacia la solución y los detalles relativamente simples de los módulos del programa, hacen que su análisis brinde al lector un esquema básico para comenzar el estudio del tema. El desarrollo está hecho en Matlab y estructurado mediante un conjunto de archivos .m que se incluyen en este documento de una manera literal para que puedan ser copiados y corridos con el software o sirvan de modelo para programarlo con otras aplicaciones (Scilab, R, Octave, etc.)

Fundamentos

En los últimos años, la computación evolutiva se ha consolidado como un conjunto de técnicas efectivas y robustas como resultado de su a aplicación a la resolución de problemas de diversa complejidad en múltiples áreas. Siguiendo a Santana Quintero y Coello Coello (2006) podríamos decir que el término "computación evolutiva" o "algoritmos evolutivos" engloba una serie de técnicas inspiradas en los principios de la teoría Neo-Darwiniana de la evolución natural. En términos generales, para simular el proceso evolutivo en una computadora se requiere:

• Codificar las estructuras que se replicarán (o sea, una estructura de datos que se utilice para almacenar a un "individuo").

• Operaciones que afecten a los "individuos" (típicamente, se usa cruce y mutación).

• Una función de aptitud que nos indique qué tan buena es una solución con respecto a las demás.

• Un mecanismo de selección que implemente el principio de "supervivencia del más apto".

En los años sesenta John H. Holland estudió los procesos lógicos involucrados en la adaptación y consideró el proceso de adaptación en términos de un formalismo en el que los individuos de una población interactúan y mejoran en base a un cierto ambiente que determina lo apropiado de su comportamiento. El combinar variaciones aleatorias con un proceso de selección (en función de que tan apropiado fuese el comportamiento), debía entonces conducir a un sistema adaptativo general. Este sistema fue publicado en su libro "Adaptation in Natural and Artificial Systems" en 1975, donde lo denominó "plan reproductivo genético", aunque después se popularizó bajo el nombre de "algoritmo genético". Coello Coello (2010).

Los algoritmos genéticos (AG) son métodos sistemáticos para la resolución de problemas de búsqueda y optimización que aplican los métodos de la evolución biológica: selección basada en la población, reproducción sexual y mutación. Se enfatiza la importancia de la cruza sexual (operador principal) sobre el de la mutación (operador secundario), y se usa selección probabilística.

Un AG básico sigue el siguiente esquema:

• Genera (aleatoriamente) una población inicial.

• Calcula la aptitud de cada individuo.

• Selecciona (probabilísticamente) en base a la aptitud.

• Aplica los operadores genéticos (cruza y mutación) para generar la siguiente población.

• Cicla hasta que cierta condición se satisfaga.

La representación tradicional es la binaria, cada uno de los pasos consiste en una actuación sobre las cadenas de bits, es decir, la aplicación de un operador a una cadena binaria.

Un algoritmo genético tiene también una serie de parámetros que se fijan para cada ejecución:

• Tamaño de la población: debe ser suficiente para garantizar la diversidad de las soluciones, y además tiene que crecer con el número de bits del cromosoma. Por supuesto, depende también del sistema en el que se esté ejecutando.

• Condición de terminación: lo más habitual es que sea la convergencia del algoritmo genético o un número prefijado de generaciones.

Podemos resumir un glosario como sigue.

Un *cromosoma* es la estructura de datos que contiene una cadena de parámetros de diseño o genes. Esta estructura de datos puede almacenarse, por ejemplo, como una cadena de bits o un arreglo de enteros.

Se llama *gen* a una subsección de un cromosoma que (usualmente) codifica el valor de un solo parámetro.

Se denomina *genotipo* a la codificación (por ejemplo, binaria) de los parámetros que representan una solución del problema a resolverse.

Se denomina *fenotipo* a la decodificación del cromosoma. Es decir, a los valores obtenidos al pasar de la representación (binaria) a la usada por la función objetivo.

Se denomina *individuo* a un solo miembro de la población de soluciones potenciales a un problema. Cada individuo contiene un cromosoma (o de manera más general, un genoma) que representa una solución posible al problema a resolverse.

Se denomina *aptitud* al valor que se asigna a cada individuo y que indica que tan bueno es este con respecto a los demás para la solución de un problema.

Se denomina *alelo* a cada valor posible que puede adquirir una cierta posición genética. Si se usa representación binaria, un alelo puede valer 0 ó 1.

Llamamos *generación* a una iteración de la medida de aptitud y a la creación de una nueva población por medio de operadores de reproducción.

La *cruza* es un operador que forma un nuevo cromosoma combinando partes de cada uno de sus cromosomas padres.

Se denomina *mutación* a un operador que forma un nuevo cromosoma a través de alteraciones (usualmente pequeñas) de los valores de los genes de un solo cromosoma padre.

Se denomina *elitismo* al mecanismo utilizado en algunos AG para asegurar que los cromosomas de los miembros más aptos de una población se pasen a la siguiente generación sin ser alterados por ningún operador genético.

En este marco nos proponemos desarrollar un ejemplo simple, de manera detallada, que sirva para comprender las distintas aristas de este tipo de programación.

Desarrollo

Cálculo del mínimo de una función mediante un AG.

El AG esta desarrollado en Matlab, consiste en un conjunto de archivos .m. Se ha buscado simplificar la programación, perdiendo refinamiento en el algoritmo, debido a que la intención es mostrar la idea y la estructura general. El AG define una población inicial que es una matriz de unos y ceros donde cada columna representa a un cromosoma (o también individuo) y cada uno o cero es un gen del cromosoma. El AG evalúa la aptitud de cada individuo, selecciona a los más aptos, los cruza y les introduce una mutación con cierta probabilidad, genera una nueva población y repite el proceso hasta que se cumple una condición de salida, esto está implementado mediante un conjunto de siete archivos .m que realizan la tarea. Estos archivos son:

• AG_basico.m: Define los parámetros del problema. Grafica la función. Construye la población inicial. Itera hasta la condición de finalización. Calcula el resultado.

• funcion.m: Define la función.

• cromnum.m: Convierte la matriz de población cuyas columnas son cromosomas en un vector fila de números reales que son los fenotipos.

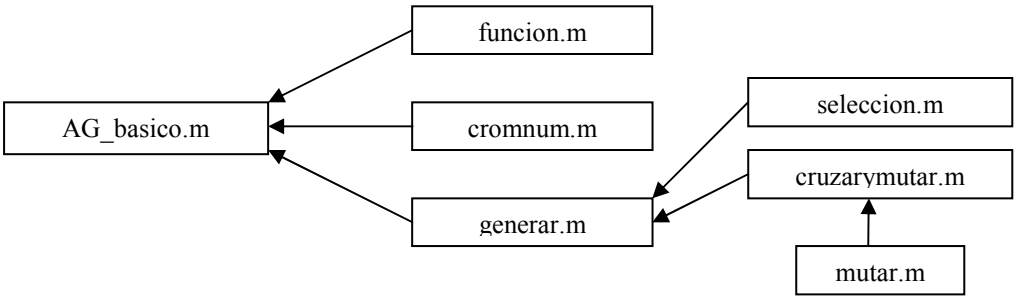
• generar.m: Genera una nueva población utilizando selección, cruzamiento y mutación.

• seleccion.m: Selecciona los padres.

• cruzarymutar.m: Con los padres seleccionados obtiene hijos.

• mutar.m: Aplica mutación a un alelo según corresponda.

Esquemáticamente:



Código

A continuación se muestra el código fuente de cada módulo del algoritmo. Bastará con copiar en el editor de Matlab cada uno salvándolo con el nombre correspondiente para luego ejecutar el archivo AG_basico en Command Window. Se puede cambiar la función editando funcion.m y los parámetros iniciales, longcrom, tampoblacion, etc. en AG_basico.

AG_basico.m

```
% Búsqueda del mínimo de una función continua en un intervalo [a,b]
longcrom = 20            % longitud del cromosoma
tampoblacion = 100       % tamaño de la población
probcruza = 0.95         % probabilidad de cruzamiento
probmutacion = 0.01      % probabilidad de mutación
maxgen = 80              % máxima cantidad de generaciones
a = -15                  % mínimo del intervalo
b = 20                   % máximo del intervalo
% Función definida en funcion.m:
intervalo = a:0.001:b;   % define los puntos para la grafica de la función
y = funcion(intervalo);  % valores de la función en el intervalo
% Construcción de la población inicial
pob = round(rand(longcrom,tampoblacion)); %población matriz de ceros y unos
fenotipos = cromnum(pob, a, b);   %vector de reales calculado en cromnum.m
valores = funcion(fenotipos);   %valores de la función
fitness = abs(max(valores)) * ones(1,tampoblacion) - valores ; %ajuste
% iteración para las sucesivas generaciones:
gen = 0;
while (gen <= maxgen),
   gen = gen + 1;
   nuevapob = generar(pob, fitness, probcruza, probmutacion);
   fenotipos = cromnum(nuevapob, a, b);
   valores = funcion(fenotipos);
   fitness = abs(max(valores)) * ones(1,tampoblacion) - valores ;
   hold off  % limpia la ventana de gráficos
   plot(intervalo,y,'r-'); % visualiza la función en el intervalo
   hold on   % mantiene la ventana de gráficos
   plot(fenotipos,valores,'k*');
   pause(0.5)
   pob = nuevapob;
end
% Elegimos un valor representante del conjunto de individuos de la última
% generación. Por ejemplo la mediana.
mediana=median(fenotipos)
% El valor de la función
minimo=funcion(mediana)
```

funcion.m

```
function y = funcion(x)
% Define la función de estudio:
y =x.* sin (x);
```

cromnum.m

```
function y = cromnum(x, a, b)
% Convierte la matriz de población, cuyas columnas son cromosomas,
% en un vector fila de números reales con los fenotipos:
[fila , col] = size(x);  % define fila y columna según x
y = [];                  % inicializa un vector vacío
for i=1:1:col,           % cada columna i es un cromosoma
    suma =0;
    potencia =1;
    for j= fila:-1:1,
       if x(j,i)==1,
            suma = suma + potencia; % convierte cada cromosoma  de binario
a decimal
       end
       potencia = potencia * 2;
    end
    aux = 2^fila -1;   % aux es el número decimal máximo para un cromosoma
    y(i) = a + suma * ( (b - a) / aux ); % transforma el número suma en un
número del intervalo [a,b]
end
```

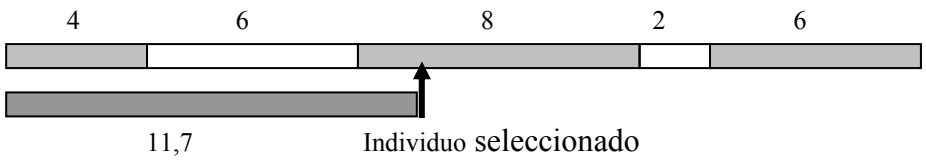
generar.m

```
function y = generar(pob, fitness, probcruza, probmutacion)
% Genera una nueva población de individuos a partir de pob
% utilizando seleccion, cruzarmutar y mutar:
[fila, col] = size(pob);
col = round(col / 2);   % por cada iteración se generan dos hijos
y = [];
for i=1:1:col,
    % selecciona dos padres:
    padre1 = seleccion(fitness); % numero de columna del individuo
seleccionado
    padre2 = seleccion(fitness); % numero de columna del individuo
seleccionado
    %genera los dos hijos:
    [hijo1, hijo2] = cruzarmutar(pob,padre1,padre2,probcruza,probmutacion);
    y = [y, hijo1, hijo2];
end
```

seleccion.m:

```
function y = seleccion(fitness)
% Dado un vector fila con los fitness de los individuos de la población
% selecciona utilizando selección proporcional
cuantos = length(fitness);
sumfitness = sum(fitness);
aleatorio = rand * sumfitness;  %posición dentro de la ruleta
%encuentra el número del individuo señalado por la posición
suma = fitness(1);
j = 1;
while (suma < aleatorio) & (j<cuantos),
    j = j + 1;
    suma = suma + fitness(j);
end
y = j;     %retorna el número del individuo elegido
```

Para aclarar el módulo de selección proporcional consideremos el siguiente ejemplo:
La matriz de fitness se modifica de modo que a los individuos de mejor ajuste le correspondan números mayores, se suman todos estos números y se los multiplica por un número entre cero y uno elegido aleatoriamente. Este último número se utiliza para seleccionar la posición de un individuo de manera aleatoria con probabilidad proporcional al "tamaño" del mismo. Consideremos el siguiente ejemplo:
Números de ajuste: 4 6 8 2 6
Suma: 26
Numero aleatorio: 0,450
Suma por número aleatorio: 11,7
El individuo seleccionado es el tercero que tiene un número de ajuste ocho. Podemos representarlo del modo siguiente:



cruzarymutar.m

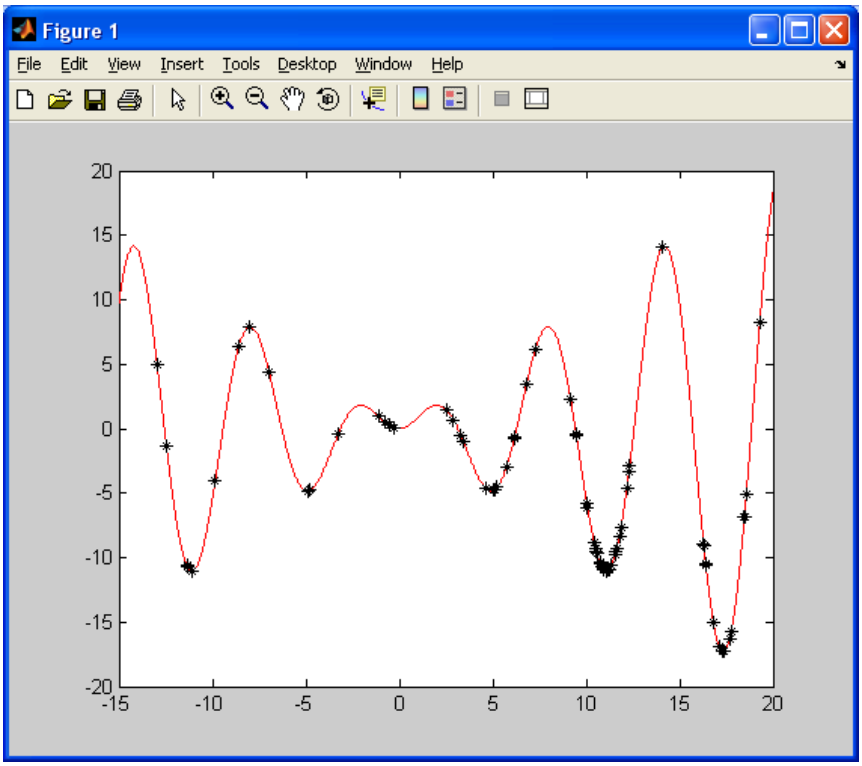
```
function [hijo1, hijo2] = cruzarmutar(pob,padre1,padre2,probcruza,
probmutacion)
%Aplica cruzamiento y mutación a las columnas padre1 y padre2 de pob y
%calcula dos vectores columna con ambos hijos
[fila, col] = size(pob);
if padre1>col, padre1=col; end
if padre2>col, padre2=col; end
% ver si corresponde aplicar cruzamiento:
haycruza =(rand<=probcruza)
```

```
if haycruza,
          posicion = round(rand * (fila-2)) + 1;
    else  posicion = fila;
end
% Si no hay cruzamiento el hijo1 será igual al padre1 con alguna mutación
% y el hijo2 será igual al padre2 con alguna mutación (posición = fila):
for i=1:1:posicion,
  hijo1(i,1) = mutar(pob(i,padre1),probmutacion); % genes padre1 hijo1
  hijo2(i,1) = mutar(pob(i,padre2),probmutacion); % genes padre2 hijo2
end
% Si hay cruzamiento posición es menor que fila y cada hijo tiene genes de
% ambos padres con alguna mutación:
if haycruza,
  for i=posicion+1:1:fila,
     hijo1(i,1) = mutar(pob(i,padre2),probmutacion); % genes padre2 hijo1
     hijo2(i,1) = mutar(pob(i,padre1),probmutacion); % genes padre1 hijo2
  end
end
```

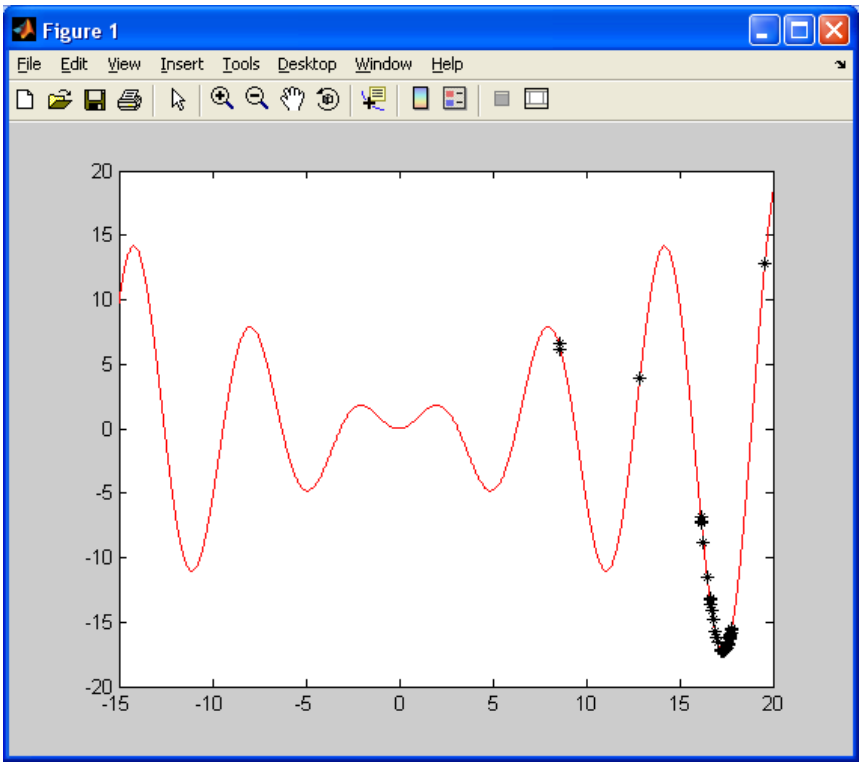
mutar.m

```
function NuevoAlelo = mutar( Alelo, probabilidad)
% Muta el Alelo con la probabilidad indicada:
if (rand<=probabilidad),
    NuevoAlelo = ~Alelo;     % hay mutación
else
    NuevoAlelo = Alelo;      % no hay mutación
end
```

Luego de la ejecución del código se obtuvieron los siguientes resultados:
Como representante de los fenotipos de la última generación tomamos, arbitrariamente, la mediana que resultó igual a 17.3187. Para este valor la función resulta igual a -17.3049. El valor exacto es de 11/2 π (aproximadamente 17,278) y -11/2 π para el valor de la función. La población inicial (100 individuos) puede verse en la siguiente figura:



La población luego de 80 iteraciones:



Conclusiones

Los argumentos de la programación evolutiva brindan herramientas muy potentes para enfrentar un gran número de problemas matemáticos derivados de la aplicación y el modelado de situaciones complejas, el hecho de que una población de "individuos" evolucione hacia la solución hace que el problema sea considerado en paralelo, esto es, escudriñado en su totalidad. La representación del problema en términos genéticos y su programación no son cuestiones simples en la mayoría de los casos y aunque existen aplicaciones de software específicamente diseñadas (inclusive Matlab tiene una) es necesario entender los mecanismos internos y los detalles para llegar a buenos resultados. En nuestra experiencia intentando transmitir el tema hemos notado que es conveniente que el primer paso sea a partir de un ejemplo simple como el presentado en este artículo. El familiarizarse con los argumentos puestos en juego en la ejecución del algoritmo permite una comprensión mas acabada.

Bibliografía

Coello Coello, Carlos A. *Introducción a la Computación Evolutiva*. CINVESTAV-IPN Departamento de Computación. México; 2010

Coello Coello, Carlos A. Santana Quintero, Luis Vicente. *Revista de métodos cuantitativos para la economía y la empresa*. Páginas 3–26. ISSN: 1886-516X. D.L: SE-2927-06. Diciembre de 2006.

John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Michigan; 1975.

Nakamura, S. *Análisis Numérico y visualización gráfica con Matlab*. Prentice Hall, México; 1997