APÉNDICE I

Se muestra el código del programa que implementa el AGS explicado en el apartado 2.3 de la Introducción. (En el CD, carpeta CODIGO/AGS).

1. AGS.M

```
function ags(xinf,xsup,num_ind,num_gen,pcr,pmu)

% AGS implementa el Algoritmo Genético Simple (A.G.S., apartado 2.3 del
proyecto).
%
% AGS(XINF,XSUP,NUM_IND,NUM_GEN,PCR,PMU) busca el óptimo de una función en el
intervalo
% [XINF,XSUP]. Emplea poblaciones de tamaño NUM_IND que evolucionan a lo largo de
NUM_GEN
% generaciones. PCR y PMU son las probabilidades de cruce y mutación,
respectivamente.
% La función objetivo se encuentra en el archivo FUNC.M.
%
% Llama a los módulos INICIO, POB_INICIAL, EVOLUCION Y RESULTADOS.

inicio        % Inicia las variables del A.G.S
pob_inicial      % Genera la población inicial de forma aleatoria
evolucion     % Genera sucesivas poblaciones empleando operadores genéticos
resultados       % Muestra los resultados
```

2. INICIO.M

```
% INICIO script M-file inicia las variables del A.G.S:
%
% LCR, longitud del cromosoma.
% POB y NPOB, poblaciones de padres e hijos, respectivamente.
% MEJOR_IND Y PEOR_IND, registran los mejores y peores individuos de cada
generación.
% CG, contador de generaciones.
% SUMA, registra la suma de bondades de todos los individuos en cada generación.
% NMUT, contabiliza el número total de mutaciones efectuadas.
%
% Las variables de entrada al módulo son:
% XINF y XSUP, NUM_GEN y NUM_IND.
% Véase la función AGS.

% Determina la longitud del cromosoma(número de alelos)
lcr = ceil(log2(xsup-xinf)+1);

% Define la estructura "individuo" descrita en el apartado 2.3 del proyecto.
ind = struct('crom',repmat(0,1,lcr),'x',0,'bon',0);
```

```
% Construye sendos vectores de individuos:
pob = repmat(ind,1,num_ind);    % Población de padres
npob = pob;                         % Población de hijos

% Crea dos vectores que registren los mejores y los peores individuos de cada
generación
mejor_ind = repmat(ind,1,num_gen);
peor_ind= mejor_ind;
for k=1:num_gen
   peor_ind(k).bon=Inf;
end

% Agita la semilla de la serie pseudoaleatoria
rand('state',sum(100*clock));

% Inicia contador de generaciones
cg=2;

% Inicia vector suma de bondades extendida a toda la población
suma=zeros(1,num_gen);

% Inicia contador de mutaciones
global NMUT
NMUT=0;
```

3. POB_INICIAL.M

```
% POB_INICIAL script M-file genera una población inicial aleatoriamente.
%
% POB_INICIAL introduce en POB un conjunto de individuos candidatos a ser
solución.
% Guarda en MEJOR_IND(1) y PEOR_IND(1) los valores que correspondan.
% Las variables de entrada al módulo son:
% NUM_IND, POB, MEJOR_IND, PEOR_IND, XINF, XSUP, SUMA.
% Véase AGS e INICIO.
%
% Llama a las funciones UOC, DECOD y FUNC.


k = 1; % Inicia contador de individuos a 1

while k <= num_ind % Para todos los individuos de la población inicial:

   for i = 1 : lcr % Para todos los alelos del cromosoma de un individuo:
      pob(k).crom(i) = uoc(0.5); % Genera el alelo: 1 ó 0, al azar
   end

   pob(k).x = decod(pob(k).crom) + xinf; % A partir del cromosoma, genera el
fenotipo

   if pob(k).x <= xsup                  % Si pertenece al intervalo:
      pob(k).bon = func(pob(k).x);      % Evalúa la función para este valor de x
      if pob(k).bon > mejor_ind(1).bon  % Registra el mejor individuo hasta la
fecha
          mejor_ind(1) = pob(k);
      end
      if pob(k).bon < peor_ind(1).bon   % Y también el peor
          peor_ind(1) = pob(k);
      end
```

```
      k = k+1;                                % Aumenta en uno el contador de
individuos
   end
end

% Guarda el sumatorio de las bondades de todos los individuos de esta primera
población
suma(1) = sum(cat(2,pob.bon));

% Muestra en pantalla la población
disp(' ')
disp([' 1', '.... ',num2str( cat(2,pob.x))])
```

4. EVOLUCION.M

```
% EVOLUCION scrip M-file que genera sucesivas poblaciones utilizando operadores
genéticos.

% Los operadores genéticos son SELECCION(SELEC), CRUCE Y MUTACIÓN(MUTA).
% Se llega a una generación final esperando alcanzar un entero que proporcione
% un valor casi-óptimo: MEJOR_IND(CG).X.
% Las variables de entrada al módulo son:
% NUM_IND, NUM_GEN, PCR, PMU, CG, POB, NPOB, MEJOR_IND, PEOR_IND, XINF, XSUP,
SUMA.
% Véase AGS e INICIO.
%
% Llama a las funciones SELEC, CRUCE, y DECOD.

while cg<= num_gen  % Para todas las generaciones que siguen a la primera:

   k=1;                %Inicia contador de individuos

   while k< num_ind     %Hasta alcanzar el tamaño cte de la población:
                                              % Selecciona una pareja
reproductora:
      pad1=selec(cat(2, pob.bon),suma(cg-1));% Padre 1
      pad2=selec(cat(2, pob.bon),suma(cg-1));% Padre 2

      % Cruce y mutación: Se obtienen los cromosomas de dos candidatos para
      % formar parte de la población hija
      [c1,c2]= cruce(pob(pad1).crom,pob(pad2).crom,pcr,pmu);

      x1=decod(c1)+xinf; x2=decod(c2)+xinf;%Decodificación: Obtención del
fenotipo

      if x1<=xsup & x2<=xsup  %Si son admisibles, incluirlos como individuos en
la
                              % población hija
         npob(k).crom=c1; npob(k+1).crom=c2;             %Genotipo
         npob(k).x=x1; npob(k+1).x=x2;                   %Fenotipo
         npob(k).bon=func(x1); npob(k+1).bon=func(x2);   %Bondad

         for i=k:(k+1) %Registrar el mejor y peor hasta el momento de la nueva
población

            if npob(i).bon > mejor_ind(cg).bon
               mejor_ind(cg)=npob(i);
             elseif npob(i).bon < peor_ind(cg).bon
               peor_ind(cg)=npob(i);
```

```
            end
         end

         k=k+2;         %Incrementa en dos el contador de individuos
      end
   end

   %Conservación del mejor individuo generación tras generación
    if mejor_ind(cg-1).bon > mejor_ind(cg).bon
        npob(ceil(rand*num_ind))=mejor_ind(cg-1); % Reemplaza uno cualquiera de la
población
         mejor_ind(cg)=mejor_ind(cg-1);           % hija por el mejor hasta la
fecha
   end

   % Muestra en pantalla la población
   if cg<10
    disp([' ',num2str(cg) ,'.... ' , num2str(cat(2,pob.x))])
    else
      disp([num2str(cg) ,'.... ' , num2str(cat(2,pob.x))])
   end

   %Actualización de la suma de las bondades
   suma(cg)=sum(cat(2,npob.bon));

   %La población hija es la población padre para la siguiente generación
    pob=npob;

   cg=cg+1; % Incrementa el contador de generaciones
end
```

5. RESULTADOS.M

```
% RESULTADOS script M-file escribe en pantalla el resultado de la evolución.
%
% RESULTADOS muestra el valor casi-óptimo, MEJOR_IND(NUM_GEN).X, y una gráfica
con la
% evolución de la bondad poblacional.
% Las variables de entrada al módulo son:
% XINF, XMAX, MEJOR_IND, PEOR_IND, SUMA, NUM_GEN, NMUT
% Véase AGS e INICIO.

fprintf('\n');
fprintf('El entero perteneciente al intervalo (%u,%u) que ha sido hallado
',xinf,xsup);
fprintf('como el que da el mayor valor\nde la función implementada en func.m, ');
fprintf('ha resultado ser %u, tras %u
generaciones\n',mejor_ind(num_gen).x,num_gen);
fprintf('\n');
fprintf('Número de mutaciones producidas: %u\n',NMUT);

% Gráfica
ejex=[1:num_gen];
plot(ejex,cat(2,mejor_ind.bon),'r:',ejex,cat(2,peor_ind.bon),'k:',ejex,suma/num_i
nd,'b:');
title('Evolución del valor de la función a lo largo de sucesivas generaciones')
xlabel('Generaciones')
ylabel('Valor de la función')

 % Representación de la función
```

```
fprintf('\nTeclee "return" y pulse INTRO para ver la representación de la función
');
fprintf('que se ha ensayado \nen el AGS\n');
keyboard
rfunc
```

6. FUNC.M

```
function y=func(x)

% FUNC es la función que se desea maximizar mediante el A.G.S.

y=0.001+(((15*x^2*log(x+1)-x^3)+x^2)/10000)*round(abs(sin(x*4-23)))*((round(x/2)-
x/2)==0);
```

7. DECOD.M

```
function y=decod(bin)

% DECOD decodifica un cromosoma.
%
% Y=DECOD(BIN) devuelve en Y el entero equivalente en base 10 del vector fila
binario BIN.
% Véase POB_INICIAL y EVOLUCION.

l=size(bin,2); % Longitud del cromosoma
y=0; pot2=1;

for j=l:-1:1
   if bin(j)
      y=y+pot2;
   end
   pot2=pot2*2;
end
```

8. SELEC.M

```
function y=selec(b,sb)

%SELEC selecciona un individuo de la población para ser padre de la siguiente
generación.

%Y=SELEC(B,SB) devuelve en Y el índice de un individuo con probabilidad igual a
% B(K)/SB. B es el vector de bondades poblacional y SB es el sumatorio de las
mismas.

%El procedimiento de selección se basa en el método de la ruleta ponderada: Se
divide
%la rueda en tantos sectores como individuos hay en la población, siendo el
ángulo del
%sector proporcional a la bondad del individuo. Cada uno de estos sectores se
marca
%con el índice del individuo al que representa. Se hace girar la rueda. Al
pararse, una
```

```
%aguja fija apuntando hacia el sector afortunado, indica qué individuo es
finalmente
%seleccionado como padre para la siguiente generación.

acumula=0;
cont=0;
parada=rand*sb;
while acumula < parada
   cont=cont+1;
   acumula=acumula+b(cont);
end

y=cont;
```

9. CRUCE.M

```
function [a,b]=cruce(p,q,pc,pm)

% CRUCE cruza y muta dos cromosomas.
%
% [A,B]=CRUCE(P,Q,PC,PM) cruza los vectores cromosómicos P y Q en un punto
% de la cadena elegido al azar, generando los cromosomas hijos A y B.
% Esto lo hace con probabilidad PC. El proceso incluye también una eventual
% mutación de alelos de los cromosomas hijos con probabilidad PM.
% Llama a la función MUTA.
% Véase EVOLUCION.

lcrom=size(p,2); % Longitud del cromosoma

if uoc(pc) %si se produce el cruce, haz:
   corte=ceil((lcrom-1)*rand); %determina aleatoriamente el punto de cruce
   for k=1:corte
      a(k)=muta(p(k),pm); b(k)=muta(q(k),pm);
   end

   for k=corte+1:lcrom
      a(k)=muta(q(k),pm); b(k)=muta(p(k),pm);
   end

else
   for k=1:lcrom
      a(k)=muta(p(k),pm); b(k)=muta(q(k),pm);
   end

end
```

10.MUTA.M

```
function y=muta(bit,pm)

% MUTA invierte un alelo uno en cero y viceversa.
%
% Y=MUTA(BIT,PM) devuelve en Y el resultado de la negación lógica de BIT con
probabilidad PM,
% o el mismo valor BIT con probabilidad 1-PM.
% Véase CRUCE.
```

```
global NMUT
if uoc(pm)
   y=not(bit);NMUT=NMUT+1;
else
   y=bit;
end
```

11.UOC.M

```
function y=uoc(p)

%UOC(P) genera una variable aleatoria de Bernouilli.
%
%Y=UCO(P) asigna a Y un uno con probabilidad P o un cero con probabilidad 1-P.

if rand < p
   y=1;
else
   y=0;
end
```

12.RFUNC.M

```
% RFUNC script M-file representa la función implementada en FUNC.M.

x=xinf:xsup; %Intervalo
y=zeros(1,xsup-xinf+1);

for k= 1: (xsup-xinf+1)
   y(k)=(((15*x(k)^2*log(x(k)+1)-x(k)^3)+x(k)^2)/10000)*round(abs(sin(x(k)*4-
23)))...
      *((round(x(k)/2)-x(k)/2)==0);
end

plot(x,y)
title ('Función de prueba para el AGS')
```