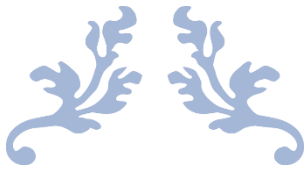


PRÁCTICA 1
SOFT COMPUTING

IMPLEMENTACIÓN DE ESTRATEGIAS EVOLUTIVAS BÁSICAS Y ALGORITMOS EVOLUTIVOS BINARIOS

28 DE DICIEMBRE DE 2020

CARMEN MARTÍNEZ ESTÉVEZ - 09070605A - ALBERTO MARTÍNEZ ORTEGA - 04638602P

EJERCICIO 1.1 – Estrategia evolutiva (1+1)-ES

Estrategia evolutiva que considera un único individuo, y que muta un hijo a partir de ruido gaussiano.

La función principal es *'unomasuno'*, a la que le pasamos por parámetro una sigma que es la varianza del ruido Gaussiano. Está función nos devuelve el individuo y su fitness.

Lo primero es generar el individuo, que será un vector de una fila y 30 columnas, de número aleatorios que van de -100 a 100. A continuación, calculamos su fitness llamando a la función *'funcion_optim'* a la que le pasamos el individuo generado.

Ahora entramos en un bucle for de 300000 iteraciones, en el que vamos a generar un hijo del individuo, con valores creados a partir del valor del individuo y aplicándole el valor sigma.

Una vez generado el hijo, comprobamos que ningún valor de los 30 que componen el hijo excede de los límites [-100,100]. Si algún valor es más pequeño que -100, lo actualizamos a -100, y si algún valor supera 100, se actualiza a 100.

Tras la comprobación calculamos el fitness del hijo, llamando nuevamente a *'funcion_optim',* pasándole el hijo.

Por último, comprobamos si el hijo generado tiene un fitness mejor que el del individuo, en cuyo caso actualizamos el valor de individuo sustituyéndole por el hijo, y también actualizamos el valor fitness.

Y así en cada iteración del bucle.

```
function [individuo, individuo_fit, grafica_fitness] = unomasuno(sigma)

individuo = -100+(100+100)*rand(1, 30)
individuo_fit = funcion_optim(individuo)
grafica_fitness=zeros(1, 300001);
grafica_fitness(1)=individuo_fit;
for i = 1:300000
    hijo=individuo+(randn(1,30)*sigma);

    for j=1:30
        if hijo(j)>100
            hijo(j)=100;
        elseif hijo(j)<(-100)
            hijo(j)=-100;
        end
    end

    hijo_fit=funcion_optim(hijo);


   if hijo_fit < individuo_fit
       individuo_fit = hijo_fit;
       individuo = hijo;
   end
   grafica_fitness(i+1)=individuo_fit;
end
```

A la funcion_optim le pasamos un individuo, y nos calcula su fitness. Con un bucle for nos recorre el vector del individuo, y va sumando cada valor del vector al cuadrado.

```
function individuo_fit = funcion_optim(individuo)

individuo_fit=0;

for i=1:length(individuo)
   individuo_fit=individuo_fit+(individuo(i)^2);
end
```

El main es el siguiente:

```
clc;
clear all;
close all;

[individuo, fitness1, grafical]=unomasuno(0.01);
disp("Individuo");
disp(individuo);
disp("Fitness");
disp(fitness1);
axis on;
hold on;
title("UNOMASUNO");
xlabel("N° de iteraciones");
ylabel("Fitness");
plot(grafical);
hold off;
```

El resultado obtenido de ejecutarlo es el siguiente:

Individuo padre y su fitness:

```
individuo =

  Columns 1 through 15

   28.5306   48.5773   29.9126   91.4320   52.6178  -78.9626  -94.0684  -36.4126  -60.0983   91.5944  -67.7954  -59.4303   -6.9713   -0.7896  -86.2944

  Columns 16 through 30

  -23.5014  -72.1797    4.3729   58.2768   49.6282   54.6515   36.2285    3.5907   72.8966   61.3576    7.5992  -71.8352  -59.3034   77.0531   47.9678


individuo_fit =

   1.0124e+05
```

El hijo con mejor fitness y su fitness:

```
Individuo
  Columns 1 through 15

   -0.0036    0.0034    0.0005   -0.0033    0.0014   -0.0018    0.0031    0.0038    0.0006   -0.0057    0.0120    0.0019   -0.0007    0.0001    0.0044

  Columns 16 through 30

   -0.0085   -0.0033   -0.0092    0.0015   -0.0108   -0.0005    0.0086    0.0001    0.0128    0.0033   -0.0018    0.0115    0.0146   -0.0025    0.0019

Fitness
    0.0012
```

EJERCICIO 1.2 – Estrategia evolutiva ($\lambda + \mu$)-ES

En este caso hay lambda padres y forman mu hijos cada uno. El procedimiento es el mismo que en el apartado anterior solo que considerando cada uno de los individuos con sus hijos.

Cogemos cada uno de los 10 padres y, mutándolos con ruido Gaussiano, obtenemos 5 hijos para cada uno. Después comparamos el fitness de cada uno de los hijos y escogemos el mejor.

Posteriormente comparamos el fitness del mejor hijo con el de su padre y, en caso de ser mejor, actualizamos la población sustituyendo al padre por dicho hijo, así como el conjunto de fitness de la población, donde introducimos el nuevo fitness del individuo.

Las funciones desarrolladas para esto son las siguientes:

La función lambdamasmu realiza el grueso de las operaciones, con un total de 100000 iteraciones. Le pasamos los parámetros sigma (definimos el ruido), tam_pob, lambda y mu. Estos 3 últimos definen el tamaño de cada individuo, el número de padres y de hijos respectivamente. También nos encontramos dentro del lambdamasmu.m otra función llamada 'hijos_dev' que actualiza los valores de los hijos en caso de que estos estén fuera de los límites de [-100, 100].

```
function [poblacion, poblacion_fit, grafica_fitness] = lambdamasmu(sigma, tam_pob, lambda, mu)

poblacion = -100+(100+100)*randn(lambda, tam_pob)

poblacion_fit=zeros(1,lambda);
grafica_fitness=zeros(100001, lambda);


for i=1:lambda
    poblacion_fit(i) = funcion_optim(poblacion(i,:))
end

grafica_fitness(1,:)=poblacion_fit;

for j = 1:100000

   for k=1:lambda
       hijos=zeros(mu, tam_pob);
       hijos_fitness=zeros(1,mu);
       for l=1:mu
           hijos(l,:)=(poblacion(k,:)+(randn(1, tam_pob)*sigma));
       end

       hijos=comprobar_hijos(hijos, tam_pob, mu);

       for m=1:mu
          hijos_fitness(m)=funcion_optim(hijos(m,:));
          if hijos_fitness(m)<poblacion_fit(k)
              poblacion(k,:)=hijos(m,:);
              poblacion_fit(k)=hijos_fitness(m);
          end
       end

   end
   grafica_fitness(j+1,:)=poblacion_fit;
end
end
```

```
function hijos_dev = comprobar_hijos(hijos, tam_pob, mu)
hijos_dev=zeros(mu, tam_pob);
    for i=1:mu
        for j=1:tam_pob
            if hijos(i,j)<-100
                hijos_dev(i,j)=-100;
            elseif hijos(i,j)>100
                hijos_dev(i,j)=100;
            else
                hijos_dev(i,j)=hijos(i,j);
            end
        end
    end
end
```

La función main, limpia la consola, cierra posibles ventanas abiertas y borra todas las variables previas del Workspace. Después llama a la función 'lambdamasmu' con los parámetros sigma=0.01, tam_pob=30, lambda=10, mu=5. Una vez terminada la ejecución recoge en las variables 'poblacion', 'fitness2' y 'grafica2' el resultado final obtenido que es mostrado por consola en el caso de las dos primeras y con un plot gráfico para observar la evolución del algoritmo.

```
clc;
clear all;
close all;
[poblacion, fitness2, grafica2]=lambdamasmu(0.01,30,10,5);
disp("Poblacion");
disp(poblacion);
disp("Fitness");
disp(fitness2);
axis on;
hold on;
title("LAMBDAMASMU");
xlabel("N° de iteraciones");
ylabel("Fitness");
plot(grafica2);
hold off;
```

Por último, comentar que la función optimizadora es la misma que en el caso anterior. Se trata de minimizar a 0 los datos de cada individuo. La única diferencia es que, en este caso, la función ha de recorrer un array bidimensional, para lo cual se han usado 2 bucles anidados a diferencia del bucle único del caso anterior.

```
function [hijo_fit]=funcion_optim2(hijos)

hijo_fit=0;
for i = 1:10
    for j=1:30
        for k=1:5
            hijo_fit=(sum(hijos(i,j,k))^2);
        end
    end
end
```

Tras ejecutar el main obtenemos lo siguiente:

Una muestra de los 10 padres y el fitness de cada uno

```
 Columns 1 through 14

   7.5334 -369.9774   34.2994   77.6791 -120.4485 -272.7306 -317.8129 -223.1204  183.8620 -329.5906   68.0751 -527.6711  481.6016 -170.7700
 266.7770  506.9847 -341.4974 -329.4140 -148.2894  -84.5282  -93.4885   49.6154  -41.6831  -79.0251 -277.6064 -267.9177   65.0438 -264.7173
-551.7694   45.0808   43.4477 -313.7741  -36.1587 -342.8234   10.5054 -138.4837  -60.4378   44.4508  -79.9814  170.9189  175.7944 -415.4114
  72.4347 -112.6110  226.0471 -261.8997  -37.4283 -322.7001  120.1220   77.7221  217.5398  417.0983 -208.9058 -314.4311 -311.6361    1.5949
 -36.2470   42.9486   -2.2212 -688.8568 -272.9760 -101.3699  208.8424 -252.9698 -260.8932 -233.3781  -39.2958   92.1908 -193.7231  -43.6032
-361.5377 -140.9932  106.9386  187.6761 -106.0103  206.5261  -82.8138 -380.4538   39.3249  -62.5338 -220.0653  -75.1900 -154.4939  -93.3040
-186.7184 -124.8289   45.3770  -34.9619 -132.9758 -253.9332 -398.3181 -384.4752   67.0176 -116.4989   -2.0069  187.3393  119.6849 -366.7356
 -31.4751  197.9395 -160.6882 -250.9857   25.5415  -25.7242 -248.4604   -2.3612 -148.7430 -486.6046   47.8726 -492.1800 -155.5744  125.4985
 615.6794  181.8069  -41.2257  174.0597  118.6531 -145.1169 -312.3163 -135.4750  -56.8660 -187.7932  242.3776 -139.5396   40.3083  -29.9641
 453.8874  183.4385 -257.4566 -442.3033  121.8547  123.4712  370.0914 -139.2107 -333.1688 -458.9358 -138.8247 -341.5691 -510.3633 -159.8132

 Columns 15 through 28

 -95.4220    4.0120 -158.7507 -366.4009 -372.3389 -139.0442  -63.3545 -206.4023 -312.8427  -21.6212 -313.3403 -246.8338  100.0122  -34.5880
-152.3991 -104.0056 -269.5852 -565.9734   -8.9941 -143.5213 -305.9535  236.4207  220.6915 -350.1358   86.7456 -106.1627 -432.8329  116.5267
-450.0425 -106.9542 -324.0257 -389.8195 -269.7419 -160.6215   89.8444 -275.1459  146.9358 -289.5922  -29.9358  -53.5306 -218.0069  101.2154
-157.1302 -259.6327  405.1999  -33.2978 -166.9774  -95.3909  -38.5876 -196.7630 -145.9253 -248.2212 -105.8012  -14.7225 -155.6128 -230.1815
-266.2733  103.7371  231.0995  -21.7293   10.5567  -89.7419  -72.9650 -242.4009 -401.2319 -201.5635  -63.5096 -174.5617  -15.4569  -48.5888
-295.8413 -126.6435  -38.4930   -9.6641  107.8181   65.2126    3.0493 -334.8425 -188.9256 -164.1151 -413.0112 -147.2909 -434.0401 -288.8756
-331.2803 -242.9060 -351.4237 -126.0569 -323.5277  205.3953  -47.7187 -138.4479 -131.1882  -97.5062 -116.9079  304.7382   -5.6731 -364.3577
-206.7114  170.2772 -273.0936  -63.2622  152.1317   -6.6171 -288.2972 -154.8140  -44.7863 -705.8355  220.7893 -551.6708 -342.5694   84.9652
-500.5271 -144.9542 -135.3068 -195.2306   32.0286 -141.9427 -132.4675  206.0145 -152.2327 -191.4029  -80.3304  345.8891  -86.7620  -99.9900
  92.8459 -217.8058   58.2832   72.4043 -113.5731   25.0381 -129.2109 -149.8049  -11.3156  148.4897  -91.7253  -32.4873   30.4712 -110.9838

poblacion_fit =

   1.0e+06 *

    1.7300    2.0533    1.6216    1.2584    1.3059    1.2705    1.8002    2.0472    1.6361    2.0902
```

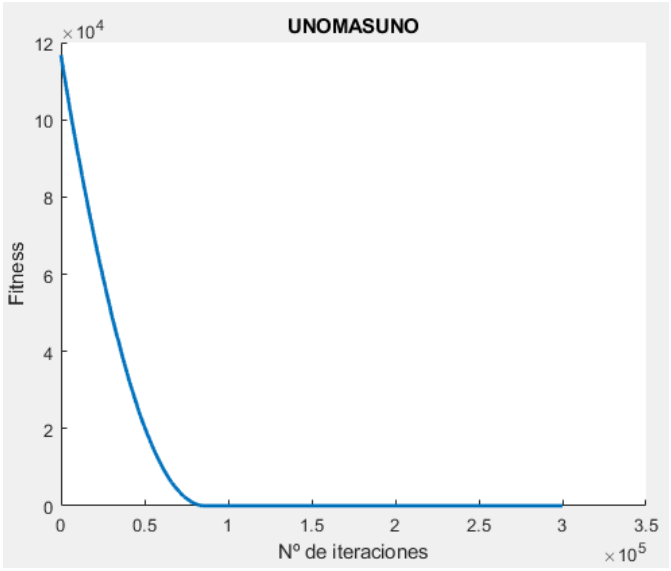
Tras superar las 100.000 iteraciones el resultado es el siguiente:

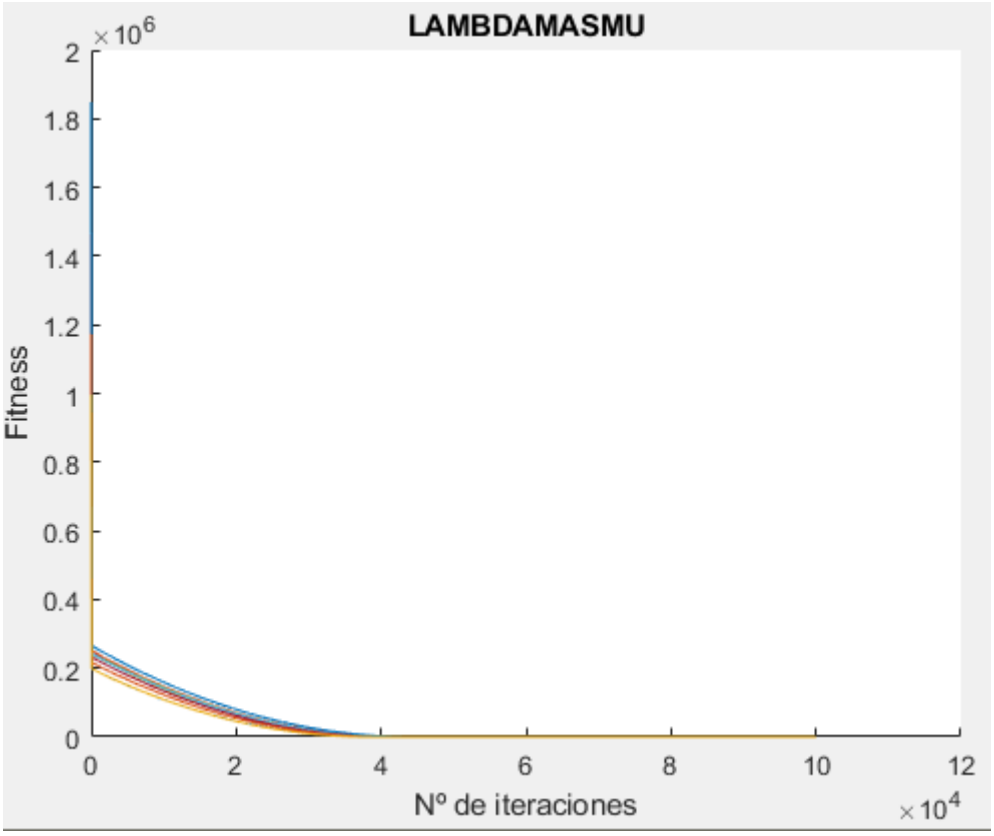
```
 Columns 1 through 14

   0.0058    0.0020   -0.0001   -0.0054   -0.0020   -0.0039   -0.0067    0.0042    0.0054   -0.0100    0.0004   -0.0068    0.0025    0.0010
  -0.0030   -0.0054    0.0051   -0.0014    0.0070    0.0080    0.0113    0.0031   -0.0081    0.0037   -0.0033    0.0104    0.0001   -0.0026
   0.0081   -0.0029    0.0045    0.0019    0.0047    0.0016    0.0096   -0.0010   -0.0025   -0.0052   -0.0010    0.0028   -0.0054    0.0037
  -0.0113    0.0067    0.0092   -0.0040    0.0021    0.0060   -0.0078   -0.0096   -0.0080   -0.0038    0.0015    0.0016   -0.0048   -0.0008
  -0.0040   -0.0006    0.0013    0.0050    0.0071    0.0061    0.0114    0.0149    0.0057    0.0057    0.0022    0.0003   -0.0047    0.0002
   0.0146   -0.0068   -0.0067    0.0022    0.0096    0.0050   -0.0030   -0.0058    0.0041   -0.0020    0.0045    0.0019   -0.0021   -0.0009
   0.0028    0.0019    0.0062   -0.0016    0.0094   -0.0079    0.0001   -0.0015   -0.0017   -0.0060   -0.0003    0.0067   -0.0086    0.0001
   0.0016    0.0011    0.0027   -0.0068   -0.0083   -0.0003   -0.0047    0.0021    0.0035    0.0032   -0.0039    0.0035    0.0009   -0.0019
  -0.0052   -0.0079    0.0051   -0.0060   -0.0070    0.0007   -0.0051    0.0098    0.0050    0.0073   -0.0045   -0.0013    0.0072   -0.0005
   0.0050    0.0089    0.0002   -0.0095   -0.0022   -0.0038   -0.0056    0.0013   -0.0025   -0.0065    0.0029    0.0057    0.0025    0.0036

 Columns 15 through 28

  -0.0096    0.0001   -0.0051    0.0190   -0.0033   -0.0058    0.0008    0.0088   -0.0068   -0.0037   -0.0039   -0.0007   -0.0001   -0.0018
  -0.0016   -0.0010   -0.0016   -0.0020   -0.0036   -0.0017    0.0015   -0.0024   -0.0008   -0.0057    0.0044    0.0022    0.0052    0.0047
   0.0008    0.0176    0.0101   -0.0033   -0.0022    0.0029    0.0011    0.0024   -0.0070    0.0060   -0.0056   -0.0052    0.0147    0.0038
   0.0040   -0.0046    0.0121    0.0019   -0.0009   -0.0047    0.0001   -0.0086   -0.0021   -0.0028    0.0125    0.0033    0.0041   -0.0017
   0.0094   -0.0113    0.0056    0.0007   -0.0062    0.0072   -0.0053   -0.0060   -0.0028    0.0005   -0.0045   -0.0007   -0.0006   -0.0015
   0.0021    0.0029   -0.0048    0.0101    0.0011    0.0034    0.0007    0.0040    0.0103   -0.0029    0.0082   -0.0025   -0.0018    0.0026
   0.0150    0.0045   -0.0062    0.0053   -0.0089    0.0041   -0.0007    0.0010   -0.0023    0.0029   -0.0019   -0.0079    0.0001   -0.0096
  -0.0013   -0.0100    0.0039   -0.0007    0.0079    0.0079   -0.0015   -0.0144   -0.0055   -0.0079   -0.0007   -0.0023    0.0003    0.0005
   0.0096    0.0027    0.0023   -0.0033   -0.0016    0.0015   -0.0012   -0.0020    0.0141   -0.0078    0.0031   -0.0048    0.0090   -0.0067
  -0.0091    0.0031    0.0037    0.0000   -0.0093   -0.0011   -0.0073   -0.0052   -0.0024   -0.0030   -0.0000    0.0046   -0.0034    0.0036

Fitness
    0.0011    0.0009    0.0011    0.0011    0.0010    0.0009    0.0009    0.0009    0.0011    0.0007
```

Por último, nos gustaría comentar los resultados, así como los parámetros elegidos para la simulación de estos algoritmos.

Comenzamos por los parámetros haciendo especial mención al parámetro sigma. Comenzamos probando con una sigma de valor 10 y los resultados obtenidos eran totalmente aleatorios. Conforme fuimos bajando el parámetro, íbamos obteniendo valores mucho más ajustados a lo que se nos pedía. Bajamos tanto el valor de sigma, a 0.0001, que el algoritmo no se ejecutaba correctamente. Entonces comprendimos que bajando tanto la sigma, los padres apenas mutaban para 'dar a luz' a sus hijos y, por ello, por muchas iteraciones que realizásemos, el resultado nunca iba a converger a 0.

Hay que destacar también que, tal y como podemos observar en el código, hemos ido guardando el fitness de cada una de las iteraciones en una variable para luego poder visualizarlos en forma de gráfica. Estos son los resultados obtenidos:





Tal y como cabía esperar, al principio el fitness cae en picado por la rápida mutación de los individuos. Como en el lambdamasmu tenemos 5 hijos para elegir, es lógico que este descienda aún más rápido que en el unomasuno. Después, conforme nos acercamos a 0, se ralentiza la bajada hasta llegar a valores de $x*10^{-3}$ aproximadamente en ambos casos. La diferencia al principio entre individuos en el segundo caso es ínfima y depende de la generación aleatoria inicial.

EJERCICIO 2 – Algoritmo genético básico con codificación binaria.

Vamos a proceder a la programación de un algoritmo genético siguiendo el esquema ($\lambda + \mu$), es decir, lambda padres que generan un hijo cada uno. Esto lo hacemos utilizando 3 operadores.

Primero cruzamos a los padres para generar a los hijos. El esquema seguido ha sido el siguiente: en cada iteración se escoge un punto de cruce distinto, del 1 al 1000. Con esta división hecha, el hijo 1 estará formado por los elementos del padre 1 hasta dicho punto de corte más los elementos del padre 2 que estén a partir del punto de corte. El hijo 2 será formado por los padres 2 y 3 y así sucesivamente. Cuando lleguemos al hijo 1000, cogeremos los x primeros dígitos del padre 1000 y los x segundos del padre 1 para completar el círculo.

Una vez obtenidos los hijos mediante el cruce procedemos a mutarlos para otorgarles algo más de diversidad genética. Cada hijo tiene una probabilidad de mutar del 3%, y en caso de que mute, lo hace un 20% del total de sus dígitos (200 de 1000). Esto se ha decidido así tras escuchar las consideraciones hechas en clase al respecto por el profesor. Nos gustaría aclarar que generar números aleatorios es un mundo complejo y se ha estudiado la mejor forma de que estos sean realmente aleatorios, llegando a una solución que consideramos la mejor de las posibles.

La función 'binario' se encarga de generar la población inicial, calcular su fitness y entrar en las 5000 iteraciones con el procedimiento arriba descrito:

```
function [poblacion, fitness, grafica]=binario(n_iteraciones, n_individuos, longitud)

poblacion=zeros(n_individuos, longitud);
fitness=zeros(1, n_individuos);
grafica=zeros(n_iteraciones, n_individuos);
for i=1:n_individuos
    for j=1:longitud
        poblacion(i,j) = randi([0,1]);
    end
    fitness(i)=funcion_fit(poblacion(i,:));
end
grafica(1,:)=fitness;
poblacion
fitness

fitness_hijos=zeros(1, n_individuos);

for k=1:n_iteraciones
    hijos_cruce=crucelpunto(poblacion);
    hijos_mutados=mutacion(hijos_cruce);
    for m=1:n_individuos
        fitness_hijos(m)=funcion_fit(hijos_mutados(m,:));
        if fitness_hijos(m)>fitness(m)
            poblacion(m,:)=hijos_mutados(m,:);
            fitness(m)=fitness_hijos(m);
        end
    end
    grafica(k+1,:)=fitness;
end
end
```

Tal y como sucedía en el apartado 1, se compara el fitness de cada hijo con su padre y, en caso de ser mejor, se actualiza la población y el valor fitness correspondiente.

Conforme se comentaba antes, aquí podemos observar el código de la función de cruce, que utiliza una mitad de un individuo y otra mitad del siguiente para generar cada hijo.

```
function hijos = crucelpunto(padres)

hijos=zeros(100,1000);
puntodecruce=int16(rand()*1000);

for i=1:100
    for j=1:puntodecruce
        hijos(i,j)=padres(i,j);
    end
    indice=puntodecruce+1;
    for k=indice:1000
        if i~=100
            indice2=i+1;
            hijos(i,k)=padres(indice2,k);
        else
            hijos(100,k)=padres(1,k);
        end
    end
end

end
```

En la función mutación se ha utilizado un generador de números aleatorios en función de la fecha actual para que cada ejecución genere una tupla de números aleatorios distintos. Así conseguimos que la elección de si el individuo muta, y en su caso, qué valores mutan, sea lo más aleatoria posible.

```
function hijos=mutacion(padres)

hijos=padres;
rng('shuffle');
for i=1:100

    random=rand();
    if random<0.02 %probabilidad de 2%. si muta, muta un 20% del individuo.
        valor_mutante= randi([1 1000],1,200);
        for j=1:200
            random2=rand();
            if random2<=0.5
                hijos(i, valor_mutante(j))=1;
            else
                hijos(i, valor_mutante(j))=0;
            end
        end
    end
end

end
```

Para el procedimiento de mutación se ha decidido hacer un 50/50 sobre si el valor muta a uno o a 0, independientemente del valor que tuviera. Puede sonar extraño, pero tras unas cuantas pruebas nos dimos cuenta de que, continuamente estábamos teniendo en cuenta el objetivo (maximizar el número de unos) y esta fue la solución más justa y alejada de partidismos que encontramos.

Nos queda comentar dos funciones muy sencillas como son la función optimizadora:

```
function unos = funcion_fit(vector)
unos=0;
for i=1:length(vector)
    if vector(i)==1
        unos=unos+1;
    end
end
end
```

Esta función tiene la sencilla misión de contar el número de unos de un vector y devolvernos dicho resultado. Después en la función binario, se mira a ver qué individuo tiene mayor número de unos para conseguir el objetivo de maximizar el número de unos de la población.

Por último, el main que ejecuta la función binario y que muestra los resultados finales:

```
clc;
clear all;
[poblacion, fitness, grafica]=binario(5000, 100, 1000);
disp("Poblacion");
disp(poblacion);
disp("Fitness");
disp(fitness);
hold on;
axis on;
title("Maximizar unos");
ylabel("Fitness");
xlabel("Iteraciones");
plot(grafica);
```

Si ejecutamos dicha función obtenemos los siguientes resultados:

```
fitness =

  Columns 1 through 23

   488   518   524   514   501   489   514   489   489   516   505   475   516   514   481   510   495   507   514   512   467   509   482

  Columns 24 through 46

   491   498   513   491   504   483   508   537   518   506   478   518   490   509   516   492   468   502   491   506   488   498   483

  Columns 47 through 69

   518   490   519   474   500   494   493   504   510   502   481   474   491   473   511   478   498   513   494   498   493   501   521

  Columns 70 through 92

   515   482   500   512   514   489   519   507   505   523   513   494   494   484   489   521   514   502   501   505   505   494   504

  Columns 93 through 100

   517   469   488   524   506   493   503   503
```

Estos son los valores fitness de la población inicial. Como podemos observar están alrededor de 500 (número de unos) pues la generación inicial se hace de manera aleatoria.

```
Fitness
  Columns 1 through 23

   762   762   762   761   761   761   761   761   761   761   761   761   761   760   761   761   762   762   762   761   761   760

  Columns 24 through 46

   761   761   761   760   759   759   759   759   760   761   760   760   760   761   762   762   761   761   763   762   762   762   762

  Columns 47 through 69

   762   762   762   762   762   762   762   762   761   760   760   759   762   762   762   762   762   762   762   763   762   762   762

  Columns 70 through 92

   763   763   763   763   763   762   763   763   762   762   761   761   760   761   761   761   762   762   761   760   761   761   761

  Columns 93 through 100

   762   762   762   762   762   762   761   762
```

Tras terminar la ejecución obtenemos un fitness comprendido entre 759 y 763.

Cabe destacar, como curiosidad, que en las primeras columnas de datos encontramos resultados de unos y ceros más intercalados. Conforme avanzamos en las columnas nos encontramos con filas enteras de unos o ceros. Se han omitido estos resultados en la memoria porque son 100*1000 unos o ceros que coparían las páginas de este documento sin ningún sentido.

Para finalizar podemos observar la evolución del fitness a lo largo de las ejecuciones gracias a la variable gráfica que va almacenando dicho valores iteración tras iteración. Se observa un gran lío debido a que estamos tratando con 100 individuos diferentes. No obstante, se puede observar correctamente la tendencia del fitness: