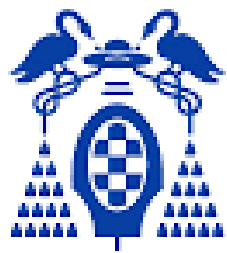


# Práctica Final – Diseño óptimo de un parque eólico.



Universidad  
de Alcalá

**Asignatura:** Soft Computing

**Titulación:** Ingeniería en Sistemas de Información

**Autores:** Iván Sánchez Ranz – Walter Huatay Rodríguez

# ÍNDICE

1. Introducción .....	1
2. Explicación código fuente.....	2
2.1. windSym.m .....	2
2.2. main.m .....	3
2.3. cruzar.m .....	8
3. Pruebas .....	9
3.1. Prueba con 20 individuos, 10 iteraciones y umbral 0.3: .....	9
3.2. Prueba con 20 individuos, 20 iteraciones y umbral 0.3: .....	10
3.3. Prueba con 20 individuos, 50 iteraciones y umbral 0.3:.....	11
3.4. Prueba con 20 individuos, 75 iteraciones y umbral 0.3: .....	12
3.5. Prueba con 20 individuos, 100 iteraciones y umbral 0.3:.....	13
3.5. Tabla comparativa .....	14
4. Conclusiones .....	15
4.1 Ventajas y desventajas .....	15
4.1.1 VENTAJAS .....	15
4.1.2 DESVENTAJAS .....	15
5. Bibliografía. ....	17
6. Anexo – Código fuente .....	18

## **1. Introducción**

A lo largo de la historia de la humanidad, los seres humanos hemos aprovechado al máximo todas las materias primas que tiene el planeta, sin tener en cuenta las consecuencias que podría tener la sobreexplotación del mismo (deforestación, aguas inundadas de petróleo, sobrepesca, extinción de especies vegetales y animales...). Una de las mayores consecuencias de esto es el aumento del calentamiento global.

Es por esto que en los últimos años, los diferentes gobiernos y empresas del mundo han optado por no sobreexplotar al máximo el planeta e incluso optar por el uso de **energías renovables** (hidráulica, fósiles, eólicas...).

En España la energía renovable más usada es la energía eólica con **28.139 MW** de potencia acumulada. En los últimos años es la energía renovable más importante en nuestro país y se encarga de suministrar **más del 20% de la demanda**.

Como prueba de la importancia que está tomando este tipo de energía, muchas empresas están realizando proyectos donde el producto principal es la creación de parques eólicos tanto en el territorio español como en muchos países del resto del mundo.

Como ejemplos de estos proyectos encontramos los siguientes: En primer lugar, **Endesa** ha conectado, hace menos de un mes, 2 plantas eólicas en Cuenca, que suman una potencia total instalada de 175 megavatios (MW). En segundo lugar, **Iberdrola** pujará por más de 18 Gigavatios (GW) de eólica marina en Europa, Asia y Oceanía. Finalmente, la Xunta de Galicia acaba de dar su visto bueno para que **Naturgy** pueda realizar su primer proyecto eólico en la Mariña lucense, el cual generará anualmente 103 gigavatios/hora.

El objetivo principal de la práctica es la mejora del diseño de un parque eólico a través de una simulación de viento. Esta mejora estará basada en encontrar el posicionamiento óptimo de 20 turbinas, teniendo en cuenta la dirección y la velocidad del viento.

Mediante la manipulación de todos los elementos, tanto creados como proporcionados para el desarrollo de la práctica, seremos capaces de crear un algoritmo lo suficientemente óptimo como para que se encargue de la disposición óptima de estos molinos/turbinas.

## 2. Explicación código fuente

Antes de explicar el código y funcionamiento en cada clase, tenemos que comentar en qué nos hemos basado para la realización de la práctica. Para la realización de la práctica hemos implementado un algoritmo meta-heurístico, el cual hemos decidido que sea un Algoritmo Genético.

Al principio empezamos con la idea de realizar un Temple Simulado, pero visto la complejidad de este pensamos en descartar esta idea y centrarnos en el Algoritmo Genético, ya que nos sentíamos más cómodos con éste.

### 2.1. windSym.m

```
4      clear all; close all; clc;
5      addpath('utils/');addpath('dt/')
6      load('dt/pwrCurve.mat');
7
8      %% Load Data
9      load('WindSym_1.mat');
10
11     %% Kgr Grid with Nturb Turbines
12     Kgr = 20; % Tamaño del Grid
13     Nturb = 20; % Numero de Turbinas
14     promptIndi = "Cuantos Individuos deseas: ";
15     individuos = input(promptIndi);
16
17     %Se genera población
18     poblacion = zeros(individuos,Kgr*Kgr);%generamos una matriz de zeros 20x20
19     for i=1:individuos
20         poblacion(i,randperm(Kgr*Kgr,Nturb))=1;
21     end
22     for i=1:individuos
23         [lista,indices] = sort(poblacion(i,:), 'descend');
24         indicesGenera(i,:) = indices(1:20);
25     end
26     indices = indicesGenera;
```

Este script es la base para el funcionamiento de nuestro programa. Aquí, primero se cargan los datos que serán necesarios para el funcionamiento de este.

Posteriormente, se encuentran los datos relativos al problema, como el tamaño del grid y el número de turbinas que deberán ser colocadas buscando su mayor rendimiento posible.

A continuación, se va a crear la “**población**”, la cual contará con 20 generadores representados con 1’s. De primeras se va a convertir en vector de tamaño 400 con valor ceros para posteriormente ser modificada a una matriz

20x20 para poder trabajar mejor con ello, ya que es mucho más fácil la manipulación y recorrido de datos con una matriz de 20x20 que un vector de 400.

Una vez creada la población, se procede a la creación de la matriz “índices”, esta matriz de 20x20 representa la posición de cada generador en el grid.

```
28 %Llamamos al main
29
30 final=main(vVec,poblacion,indices,individuos);
31 disp("                                MATRIZ RESULTANTE")
32 disp(final)
33 % Outputs: pwr_t: Potencia Total
34 %           pwrGen: Potencia Individual por Generador
35 %           Ux: Viento en cada Generador
36 %           gan: Precio (KW) * Potencia Total
37 %           cost: Coste de las Turbinas
38 %           obj: cost / gan
39 [pwr_T2,gan_T2,cost_T2,obj_T2] = f_powerPlantsT_fast(vVec,final);
40
41 disp("POTENCIA GENERADA")
42 disp(pwr_T2)
```

Por último, ya tenemos los Vectores del Viento, los individuos deseados (los molinos), la población (el terreno) y los índices. Todos estos valores se los pasaremos por parámetro a la función “**main.m**” donde se realizará todo lo posteriormente explicado. Con los valores ya finales y precisos, se mostrará por pantalla tanto la Matriz Resultante Final, como la Potencia a Maximizar.

## 2.2. main.m

Esta clase main será donde realicemos todo nuestro algoritmo, el cual será llamado por la clase “**windSym.m**” pasándole por parámetros, el Vector del Viento, la matriz de población, el número de individuos y los índices.

### Creación de la población y petición de datos al usuario:

```
Editor - C:\Users\isr10\Desktop\UNIVERSIDAD\Soft Computing\PLFinal\van-Walter\main.m
main.m  windSym.m  cruzar.m  f_powerPlantsT_fast.m  +
1  function matrizFinal = main(vVec,poblacion,indices,individuos)
2  %AQUI SE REALIZARÁN EL Nº DE ITERACIONES EN LA EJECUCIÓN
3  prompt = "Cuántas Iteraciones deseas: ";
4  iteracionesBucleInput = input(prompt);%Variable encargada de pedir al usuario cuantas ite
5  pedirUmbral = "Cuanto umbral de mutación ponemos (valor entre 0 y 1): (EJEMPLO = 0.5) ";
6  umbralInput=input(pedirUmbral);
7  filas = length(indices(:,1));
8  columnas = length(indices(1,:));
9  mitadFilas= filas/2;
10 figure;
11 xlabel("individuos");
12 ylabel("fitness");
13 hold on;
14
15 % NOS ENCARGAMOS DE RECORRER TODAS LAS COLUMNAS Y MODIFICAR LA MATRIZ%
```

En esta primera parte de la función main, declaramos las filas, columnas y las variables necesarias para la posterior creación de la gráfica (plot). Esta gráfica mostrará los individuos en su eje vertical y el **valor del fitness** en su eje horizontal. También cabe destacar que mediante la variable “prompt” le pediremos al usuario el número de iteraciones que se recorran y el umbral de mutación que tendrá el programa. Este umbral de mutación tiene que ser obligatoriamente un **valor entre 0 y 1**, el cual, si no es así, indicará al usuario que el umbral no está entre los valores posibles, provocando el cierre del programa mediante un “break”. El número de iteraciones lo pedimos para las posteriores pruebas, ya que en algunos casos desearemos que se realicen 10 iteraciones, pero en otros casos desearemos que se realicen 1000 iteraciones.

### Inicio del bucle y realizamos la “Selección”:

```
17 for iteraciones=1:iteracionesBucleInput %se hace un bucle dependiendo de las iteraciones que queramos
18     if(umbralInput<0 || umbralInput>1)
19         disp("EL UMBRAL DEBE TENER UN VALOR ENTRE 0 Y 1, SE VA A CERRAR EL PROGRAMA")
20         break
21     end
22     disp(iteraciones) %imprime en que iteración estamos
23     columnasPoblacion=length(poblacion(:,1)); %cogemos la longitud de las columnas de población
24     columnasPobZeros=zeros(columnasPoblacion,1);
25     for recorrerColumnas=1:length(poblacion(:,1))
26         conversion = reshape(poblacion(recorrerColumnas,:),[20,20]); %Remoleda la matriz, pasa de vector de 400 a matriz
27
28         [pwr_T2,gan_T2,cost_T2,obj_T2] = f_powerPlantsT_fast(vVec,conversion);
29         columnasPobZeros(recorrerColumnas) = pwr_T2;
30     end
31     vectorFit=columnasPobZeros;
32
33     %SELECCIONAMOS LOS MEJORES INDIVIDUOS DE LA ANTERIOR POBLACIÓN
34     longitud=length(vectorFit);
35     [lista,indicesNuevos]=sort(vectorFit,'descend');
36     indicesSelec=indicesNuevos(1:longitud/2);
37     poblacionSelec=indices(indicesSelec,:);
38     indices(1:mitadFilas,1:columnas) = poblacionSelec;
39
```

Empezamos realizando un bucle donde se va a ir realizando todo el código, este bucle tendrá el valor de las iteraciones que desee el usuario. Dependiendo del valor del bucle, cuanto mayor sea, más precisos serán los datos. A posteriori de declarar el bucle, reconvertiremos el vector de 400 ceros pasados por parámetros desde la clase “**windSym.m**” a una matriz de 20x20. Esta reconversión será posible gracias a la función “**reshape**”.

Tras reconvertir el vector de 400 en una matriz 20x20, la pasamos como parámetro a las funciones proporcionadas para la realización de la práctica. En la clase “**f\_powerPlantsT\_fast.m**” se generará el valor de la potencia en la variable “**pwr\_T2**”, la cual será el valor que le asignaremos a nuestro “**vectorFit**”.

Tras tener los valores del “**vectorFit**” seleccionaremos cuáles serán los mejores valores de los individuos que nos han salido.

#### **Cruzamiento en el main:**

```
36      %CRUZAMOS LOS INDIVIDUOS PARA OBTENER HIJOS
37
38      poblacionCruzada= cruzar(indices,poblacionSelec);
39      indices(mitadFilas+1:filas,1:columnas) = poblacionCruzada;
```

Respecto al cruzamiento, éste consiste en realizar el cruce entre 2 individuos de la población que ha sido seleccionada anteriormente.

Para realizar este cruzamiento recurriremos a una función que ha sido creada específicamente para esto, dicha función se llama “cruzar”, de la cual almacenamos el resultado que devuelve. Esta función será explicada más en detalle posteriormente como una clase independiente.

### Mutar:

```
44 %MUTAR
45 umbralMutacion=umbralInput; %podemos poner un valor cualquiera entre 0 y 1
46 for j=1:iteracionesBucleInput
47     probabilidadMutar = rand(1); %solo queremos generar un unico nºrandom
48     if(umbralMutacion<probabilidadMutar)
49         posicionColumnasRandi = randi([1 columnas]); %Numero aleatorio de columna
50         posicionFilasRandi = randi([1 filas]); %Numero aleatorio de fila
51         if(indices(posicionFilasRandi,posicionColumnasRandi)==1)
52             indices(posicionFilasRandi,posicionColumnasRandi)=...
53                 indices(posicionFilasRandi,posicionColumnasRandi) + 1;
54         else
55             indices(posicionFilasRandi,posicionColumnasRandi)=...
56                 indices(posicionFilasRandi,posicionColumnasRandi) - 1;
57         end
58     end
59 end
60
```

En esta parte del código procederemos a mutar el algoritmo. En resumen y definitiva es la parte del código encargada de **asignar una posición** a los individuos, basándonos en los hijos anteriormente obtenidos, podremos modificar la posición de estos de izquierda a derecha dependiendo del **umbral de mutación**.

Para empezar, el usuario tendrá que indicar el umbral de mutación que desea para realizar la mutación. Este valor debe ser un valor entre 0 y 1 y nunca puede ser diferente a este en ningún caso. Para que todas las pruebas tengan la máxima similitud, nosotros hemos hecho que, en todas las pruebas, el umbral de mutación tenga un valor de “0.3”. Tras esto se recorrerá un bucle de la cantidad de veces que deseamos mutar. Nosotros por simplificar hemos hecho que este bucle sea el mismo que la cantidad de las iteraciones para simplificar, pero un valor en torno a 10 por defecto será bastante adecuado para la cantidad de mutaciones que debe haber.

Tras esto cogeremos un valor random **entre 0 y 1.**, este valor será la probabilidad de mutación. Si este valor es superior al umbral se producirá la mutación.

En caso de mutación, cogeremos posiciones aleatorias en nuestra matriz, cogiendo las posiciones de las filas y columnas aleatoriamente. Usando las funciones “if” y “else”. Dependiendo si el valor de las posiciones nos da 1 o no sumaremos o restaremos 1, eso hará que la posición del molino se mueva a la izquierda o a la derecha en nuestra matriz final.



### Generación de la matriz final:

```
60
61 %Creamos población con los nuevos elementos
62 %SE GENERA LA MATRIZ CON LOS MOLINOS, DONDE HAYA 1 SERÁ
63 %DONDE HAYA UN MOLINO
64 matrizPob = zeros(individuos,400);
65 for bucleIndividuos=1:individuos
66     matrizPob(bucleIndividuos,indices(bucleIndividuos,:)) = 1;
67 end
68 %MOSTRAR TODOS LOS VALORES FITNESS, LA GRÁFICA Y
69 %LA MATRIZ RESULTANTE FINAL
70 vectorFit
71 poblacion=matrizPob;
72 plot (vectorFit,individuos, "o-");
73 hold off;
74 end
75 [lista,indices] = sort(vectorFit,'descend');
76 matrizFinal = reshape(poblacion(indices(1),:),[20,20]);
77
```

Esta parte del código será la encargada de adjudicar 1's a la matriz final. Es decir, adjudicará la posición de los molinos/turbinas. Para ello se realizará un bucle de 1 a los individuos, es decir, habrá 20 molinos/turbinas. La posición adjudicada de los molinos se basará en los valores de los índices anteriormente modificados e iterados, con la intención de que esa posición sea la más precisa y adecuada teniendo en cuenta todos los valores anteriormente mencionados (potencia, vectores de dirección...)

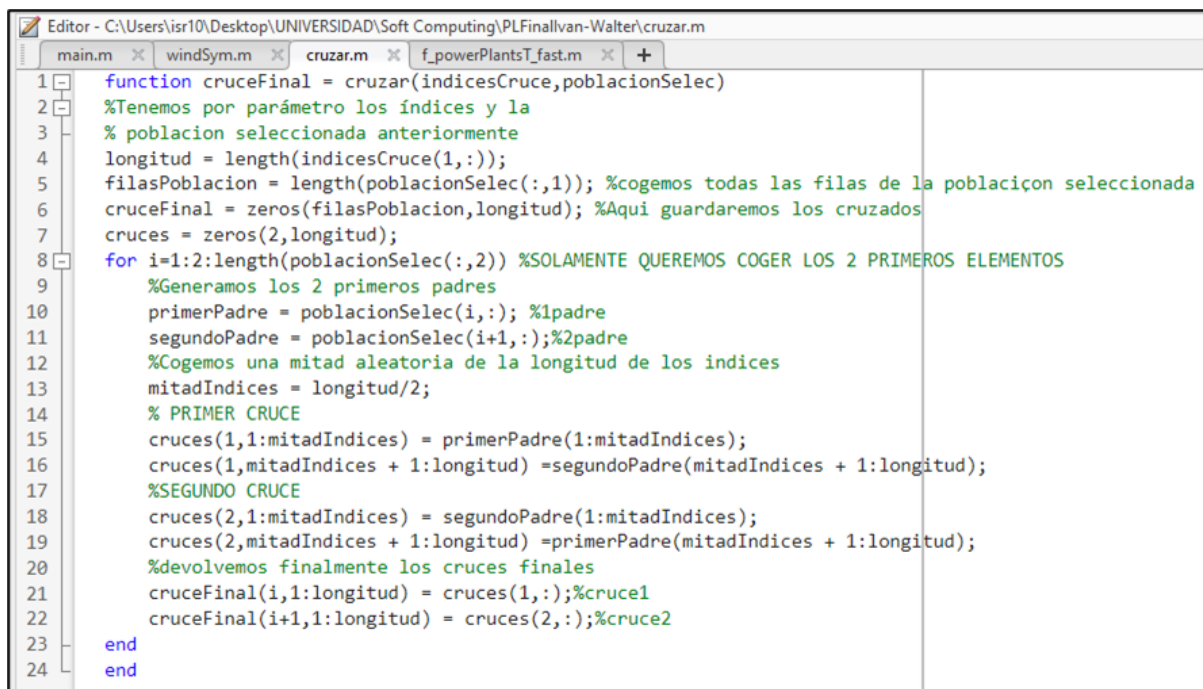
### Visualizar Fitness, Mostrar Gráfica y Matriz Final

```
70 %MOSTRAR TODOS LOS VALORES FITNESS, LA GRÁFICA Y
71 %LA MATRIZ RESULTANTE FINAL
72 vectorFit
73 poblacion=matrizPob;
74 plot (vectorFit,individuos, "o-");
75 hold off;
76 end
77 [lista,indices] = sort(vectorFit,'descend');
78 matrizFinal = reshape(poblacion(indices(1),:),[20,20]);
79
```

Esta es la parte final de la clase, donde mostraremos todos los valores que deseamos visualizar. Estos valores han sido modificados a lo largo del algoritmo y cuentan con un valor preciso. Los valores a visualizar serán:

- ❖ **El vectorFitness:** Serán todos los valoresFitness que se irán generando a lo largo de las iteraciones y los individuos. Es decir, lo normal es que hagamos pruebas **con 20 individuos**, por tanto, tendremos 20 valores Fitness. Por cada iteración que se realice se irán mostrando todos los valoresFitness de cada individuo en la consola. Estos valores posteriormente serán representados en una gráfica final para que sea más visible para el usuario. Los mejores valores y más precisos serán los de la última iteración, los cuales serán los que se muestren en la gráfica final y los que generen la posición de los molinos final en la Matriz Resultante. Estos valores últimos serán más precisos ya que por cada iteración que se haga se irá mejorando y precisando dichos valores.
- ❖ **La gráfica final:** una gráfica final que haremos mediante un “plot” de los valoresFitness de la última iteración resultante, los cuales serán los más precisos a nuestro algoritmo.
- ❖ **La matriz final:** esta matriz será la posición de los molinos/turbinas final, basándose en los valores de los índices que han sido modificados y precisados a lo largo del algoritmo.

### 2.3. cruzar.m



```

1 function cruceFinal = cruzar(indicesCruce,poblacionSelec)
2 %Tenemos por parámetro los índices y la
3 % poblacion seleccionada anteriormente
4 longitud = length(indicesCruce(1,:));
5 filasPoblacion = length(poblacionSelec(:,1)); %cogemos todas las filas de la poblacion seleccionada
6 cruceFinal = zeros(filasPoblacion,longitud); %Aqui guardaremos los cruzados
7 cruces = zeros(2,longitud);
8 for i=1:2:length(poblacionSelec(:,2)) %SOLAMENTE QUEREMOS COGER LOS 2 PRIMEROS ELEMENTOS
9     %Generamos los 2 primeros padres
10    primerPadre = poblacionSelec(i,:); %1padre
11    segundoPadre = poblacionSelec(i+1,:); %2padre
12    %Cogemos una mitad aleatoria de la longitud de los indices
13    mitadIndices = longitud/2;
14    % PRIMER CRUCE
15    cruces(1,1:mitadIndices) = primerPadre(1:mitadIndices);
16    cruces(1,mitadIndices + 1:longitud) =segundoPadre(mitadIndices + 1:longitud);
17    %SEGUNDO CRUCE
18    cruces(2,1:mitadIndices) = segundoPadre(1:mitadIndices);
19    cruces(2,mitadIndices + 1:longitud) =primerPadre(mitadIndices + 1:longitud);
20    %devolvemos finalmente los cruces finales
21    cruceFinal(i,1:longitud) = cruces(1,:);%cruce1
22    cruceFinal(i+1,1:longitud) = cruces(2,:);%cruce2
23 end
24 end

```

El objetivo de esta función es producir 2 hijos a partir de 2 padres. En primer lugar, creamos las variables que almacenarán la longitud de la población

inicial, el número de filas de esta población y la matrices finales resultantes de los cruces.

Posteriormente, inicializamos el bucle en búsqueda de 2 padres, mediante estos padres procederemos a la creación de los hijos. Para ello, el primer hijo será generado a partir de la unión de la primera mitad de un padre y de la segunda mitad del otro padre. Mientras que el segundo hijo será generado a partir de las mitades restantes que no fueron utilizadas en el cruce previo.

Finalmente, esta función devolverá las matrices correspondientes a cada hijo.

### 3. Pruebas

En este apartado vamos a realizar las pruebas del código:

#### 3.1. Prueba con 20 individuos, 10 iteraciones y umbral 0.3:

En primer lugar, el usuario introduce por teclado el número de individuos, el número de iteraciones a realizar y el valor del umbral de mutación a utilizar:

```
Command Window
Cuantos Individuos deseas: 20
Cuantas Iteraciones deseas: 10
Cuanto umbral de mutación ponemos (valor entre 0 y 1): (EJEMPLO = 0.5) 0.3
```

\*Esta imagen solo se mostrará una vez a lo largo de las pruebas para reducir el volumen de imágenes en este apartado.

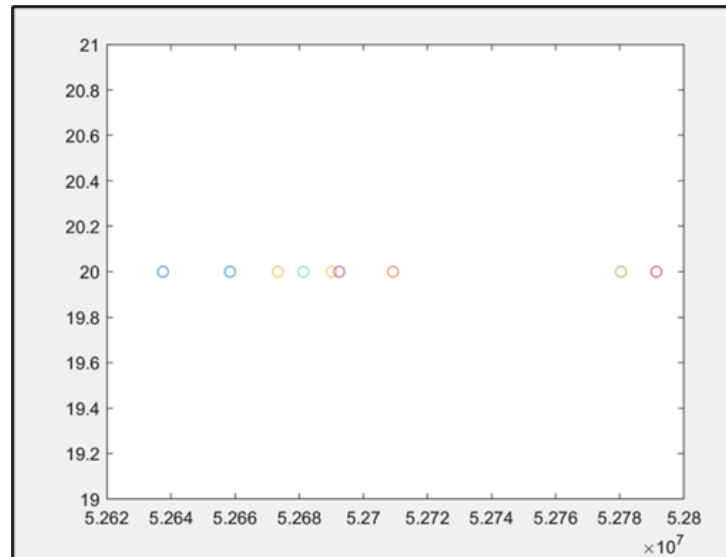
Como resultado de la ejecución del programa, obtendremos la matriz resultante y la potencia generada.

Respecto a la matriz resultante, esta muestra la posición de cada turbina en el grid, cada turbina está representada con un 1.

```
Command Window
MATRIZ RESULTANTE
0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Finalmente, obtenemos la potencia generada y la representación gráfica que representa las valoraciones fitness de los individuos de la última iteración, ya que ésta debería ser la más precisa.

POTENCIA GENERADA	5.2753e+07
-------------------	------------



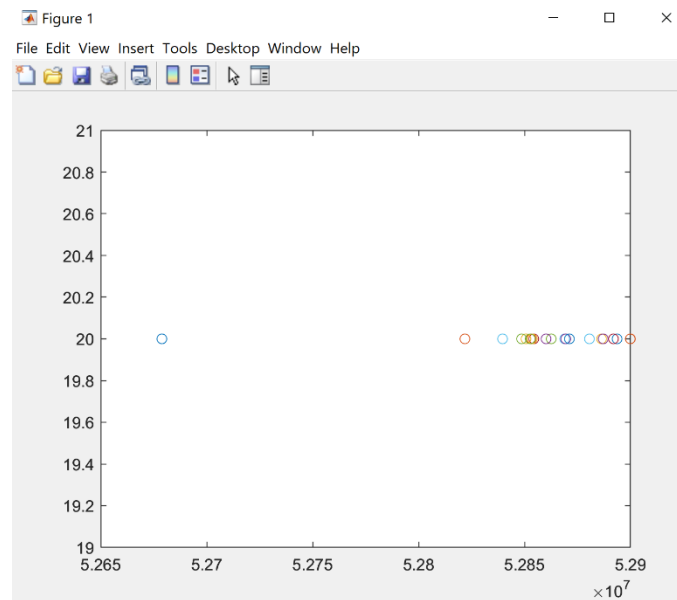
### 3.2. Prueba con 20 individuos, 20 iteraciones y umbral 0.3:

Matriz resultante junto a la Potencia generada:

[illegible]

POTENCIA GENERADA  
5.2894e+07

Representación gráfica:



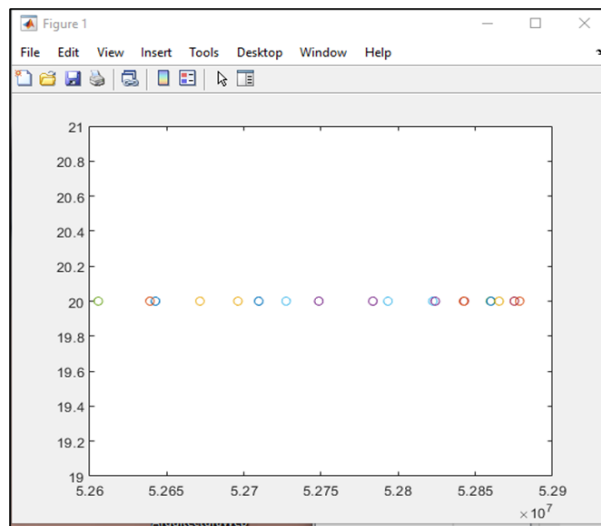
### 3.3. Prueba con 20 individuos, 50 iteraciones y umbral 0.3:

Matriz resultante junto a la Potencia generada:

MATRIZ RESULTANTE																			
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

POTENCIA GENERADA  
5.2875e+07

Representación gráfica:

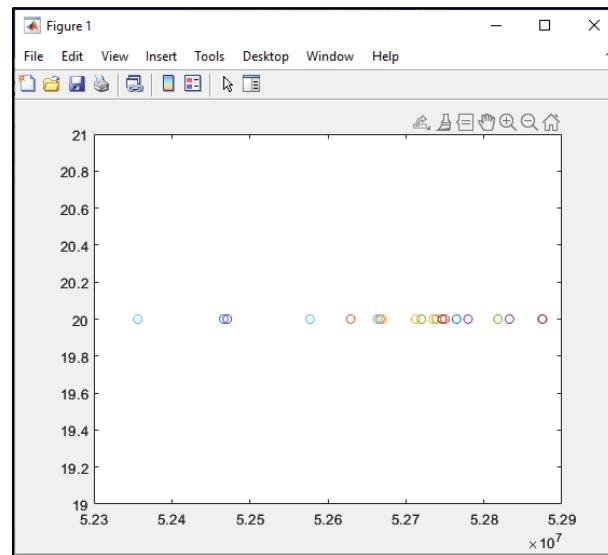


### 3.4. Prueba con 20 individuos, 75 iteraciones y umbral 0.3:

MATRIZ RESULTANTE																			
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

POTENCIA GENERADA  
5.2874e+07

Representación gráfica:



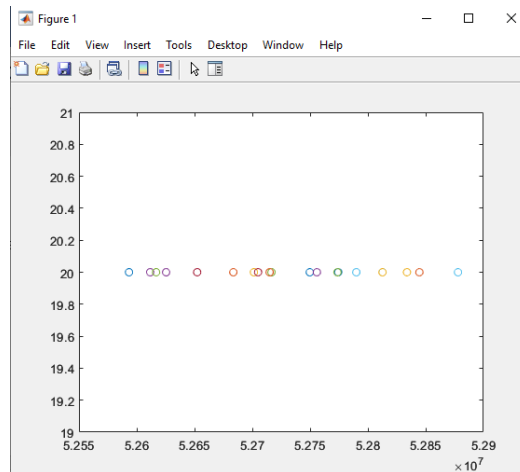
### 3.5. Prueba con 20 individuos, 100 iteraciones y umbral 0.3:

Matriz resultante junto a la Potencia generada:

MATRIZ RESULTANTE																			
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

POTENCIA GENERADA  
5.2907e+07

## Representación gráfica:



### 3.5. Tabla comparativa

A continuación vamos a mostrar las pruebas que hemos realizado con nuestro algoritmo. En las pruebas con menores iteraciones, 2 y 5 respectivamente, se han obtenido potencias generadas muy inferiores respecto al resto de pruebas. Esto se debe a la necesidad de un mayor número de iteraciones para que la potencia llegue a niveles más óptimos. Respecto al resto de pruebas, obtenemos en todas ellas, una potencia superior a 5.28e+07. Corroborando así que el algoritmo creado optimiza de manera satisfactoria la potencia.

Tabla Pruebas	20 Individuos
2 iteraciones	5.2417e+07
5 iteraciones	5.2527e+07
10 iteraciones	5.2753e+07
20 iteraciones	5.2802e+07
50 iteraciones	5.2875e+07
75 iteraciones	5.2874e+07
100 iteraciones	5.2907e+07



## 4. Conclusiones

- Hemos realizado la práctica usando un Algoritmo Genético.
- Partimos de una simulación de viento con valores ya predefinidos.
- Como algoritmo genético, hemos seguido el modelo de selección, cruce y mutación, por ese orden.
- Hemos realizado pruebas con diferentes rangos de iteraciones para poder sacar una conclusión de la misma.
- Las pruebas han sido, en su mayoría, de buena calidad, exceptuando algunas de ellas.
- Las primeras iteraciones como es normal nos dan valores de mala calidad al ser los primeros.
- Después, al optimizar la potencia con el algoritmo genético, obtenemos valores de buena calidad.

### 4.1 Ventajas y desventajas

En este apartado, se comentan las ventajas y desventajas existentes al utilizar un algoritmo genético:

#### 4.1.1 VENTAJAS

- No es necesario tener conocimientos específicos del problema para resolverlo mediante algoritmo evolutivo.
- Es realmente cómodo su ejecución, salvo al realizar pruebas con muchas iteraciones, los tiempos de ejecución son realmente aceptables.
- Nos basamos en probabilidad, en vez de operadores determinísticos
- Se opera de forma simultánea con diferentes soluciones en vez de secuencialmente.

#### 4.1.2 DESVENTAJAS

- Es una técnica robusta.
- Hay técnicas especializadas que pueden resolver los problemas más rápido y con más eficacia que el algoritmo genético.
- No es capaz de tolerar cambios aleatorios

- Lo que anteriormente era una ventaja, también puede ser una desventaja, y es que nos basamos en probabilidad y no en operadores deterministas.

#### **4.2. Problemas encontrados y Soluciones aportadas**

- A la hora de elegir los individuos para simplificar el problema, algunas elecciones de la cantidad de individuos que deseamos nos daban errores. Por tanto, hemos realizado las pruebas en base los individuos por defecto (20) y en las que hay muchas iteraciones, con una cantidad de individuos que no nos diese errores.
- Hemos tenido que volver a ordenar mediante un sort los valores fitness porque nos daba errores a la hora de hacer algunas iteraciones.
- Ha sido tedioso el controlar los rangos de las matrices, sobre todo a la hora de intercalar vectores-matrices y no se produjera ningún error de ejecución.
- El problema en sí es que esto es algo a lo que nunca nos hemos enfrentado ni nunca hemos tratado, pero hay información disponible para poder entender la mayoría de los problemas encontrados.

## 5. Bibliografía.

- Información, L. (2022, 9 diciembre). *Endesa conecta a la red dos plantas eólicas en Cuenca y ya suman 175 MW*. La Información. <https://www.lainformacion.com/empresas/endesa-conecta-dos-plantas-eolicas-cuenca-175-mw/2878087/>
- Pérez, A. (2022, 15 diciembre). *Iberdrola pujará por más de 18 GW de eólica marina en Europa, Asia y Oceanía*. elEconomista.es. <https://www.eleconomista.es/energia/noticias/12073416/12/22/Iberdrola-pujara-por-mas-de-18-GW-de-eolica-marina-en-Europa-Asia-y-Oceania.html>
- *La Xunta da su visto bueno ambiental al parque que Naturgy quiere erigir en la Mariña lucense*. (2022, 28 diciembre). Energías Renovables, el periodismo de las energías limpias. Recuperado 29 de diciembre de 29d.C.,de <https://www.energias-renovables.com/eolica/naturgy-obtiene-luz-verde-ambiental-para-el-20221228>
- *Wind Turbine - MATLAB & Simulink - MathWorks España*. (s. f.). <https://es.mathworks.com/help/sps/ug/wind-turbine.html>
- *What Is the Genetic Algorithm? - MATLAB & Simulink - MathWorks España*. (s. f.). <https://es.mathworks.com/help/gads/what-is-the-genetic-algorithm.html>

## 6. Anexo – Código fuente

### CLASE MAIN

```
function matrizFinal = main(vVec,poblacion,indices,individuos)
%AQUI SE REALIZARÁN EL N° DE ITERACIONES EN LA EJECUCIÓN
prompt = "Cuántas Iteraciones deseas: ";
iteracionesBucleImput = input(prompt);%Variable encargada de pedir al
usuario cuántas iteraciones quiere
pedirUmbral = "Cuanto umbral de mutación ponemos (valor entre 0 y 1):
(EJEMPLO = 0.5) ";
umbralInput=input(pedirUmbral);
filas = length(indices(:,1));
columnas = length(indices(1,:));
mitadFilas= filas/2;
figure;
xlabel("individuos");
ylabel("fitness");
hold on;

% NOS ENCARGAMOS DE RECORRER TODAS LAS COLUMNAS Y MODIFICAR LA MATRIZ%
for iteraciones=1:iteracionesBucleImput %se hace un bucle dependiendo de
las iteraciones que queramos
    if(umbralInput<0 || umbralInput>1)
        disp("EL UMBRAL DEBE TENER UN VALOR ENTRE 0 Y 1, SE VA A CERRAR EL
PROGRAMA")
        break
    end
    disp(iteraciones) %imprime en que iteración estamos
    columnasPoblacion=length(poblacion(:,1)); %cogemos la longitud de las
columnas de población
    columnasPobZeros=zeros(columnasPoblacion,1);
    for recorrerColumnas=1:length(poblacion(:,1))
        conversion = reshape(poblacion(recorrerColumnas,:),[20,20]);
%Remoleta la matriz, pasa de vector de 400 a matriz 20x20

        [pwr_T2,gan_T2,cost_T2,obj_T2] =
f_powerPlantsT_fast(vVec,conversion);
        columnasPobZeros(recorrerColumnas) = pwr_T2;
    end
    vectorFit=columnasPobZeros;

%SELECCIONAMOS LOS MEJORES INDIVIDUOS DE LA ANTERIOR POBLACIÓN
longitud=length(vectorFit);
[lista,indicesNuevos]=sort(vectorFit,'descend');
indicesSelec=indicesNuevos(1:longitud/2);
poblacionSelec=indices(indicesSelec,:);
indices(1:mitadFilas,1:columnas) = poblacionSelec;
```

```

%CRUZAMOS LOS INDIVIDUOS PARA OBTENER HIJOS

poblacionCruzada= cruzar(indices,poblacionSelec);
indices(mitadFilas+1:filas,1:columnas) = poblacionCruzada;

%MUTAR
umbralMutacion=umbralInput; %podemos poner un valor cualquiera entre 0
y 1
for j=1:iteracionesBucleInput
    probabilidadMutar = rand(1); %solo queremos generar un unico
nºrandom
    if(umbralMutacion<probabilidadMutar)
        posicionColumnasRandi = randi([1 columnas]); %Numero aleatorio
de columna
        posicionFilasRandi = randi([1 filas]); %Numero aleatorio de
fila
        if(indices(posicionFilasRandi,posicionColumnasRandi)==1)
            indices(posicionFilasRandi,posicionColumnasRandi)=...
                indices(posicionFilasRandi,posicionColumnasRandi) + 1;
        else
            indices(posicionFilasRandi,posicionColumnasRandi)=...
                indices(posicionFilasRandi,posicionColumnasRandi) - 1;
        end
    end
end

%Creamos población con los nuevos elementos
%SE GENERA LA MATRIZ CON LOS MOLINOS, DONDE HAYA 1 SERÁ
%DONDE HAYA UN MOLINO
matrizPob = zeros(individuos,400);
for bucleIndividuos=1:individuos
    matrizPob(bucleIndividuos,indices(bucleIndividuos,:)) = 1;
end
%MOSTRAR TODOS LOS VALORES FITNESS, LA GRÁFICA Y
%LA MATRIZ RESULTANTE FINAL
vectorFit
poblacion=matrizPob;
plot (vectorFit,individuos, "o-");
hold off;
end
[lista,indices] = sort(vectorFit,'descend');
matrizFinal = reshape(poblacion(indices(1),:),[20,20]);

```

## CLASE WINDSYM

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%  
%% WinSym Code  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%  
clear all; close all; clc;  
addpath('utils/');addpath('dt/')  
load('dt/pwrCurve.mat');  
  
%% Load Data  
load('WindSym_1.mat');  
  
%% Kgr Grid with Nturb Turbines  
Kgr = 20; % Tamaño del Grid  
Nturb = 20; % Numero de Turbinas  
promptIndi = "Cuantos Individuos deseas: ";  
individuos = input(promptIndi);  
  
%Se genera población  
poblacion = zeros(individuos,Kgr*Kgr);%generamos un vector de 400  
for i=1:individuos  
    poblacion(i,randperm(Kgr*Kgr,Nturb))=1;  
end  
for i=1:individuos  
    [lista,indices] = sort(poblacion(i,:), 'descend');  
    indicesGenera(i,:) = indices(1:20);  
end  
indices = indicesGenera;  
%Llamamos al main  
  
final=main(vVec,poblacion,indices,individuos);  
disp("                                MATRIZ RESULTANTE")  
disp(final)  
% Ouputs: pwr_t: Potencia Total  
%         pwrGen: Potencia Individual por Generador  
%         Ux: Viento en cada Generador  
%         gan: Precio (KW) * Potencia Total  
%         cost: Coste de las Turbinas  
%         obj: cost / gan  
[pwr_T2,gan_T2,cost_T2,obj_T2] = f_powerPlantsT_fast(vVec,final);  
  
disp("POTENCIA GENERADA")  
disp(pwr_T2)
```

### CLASE CRUZAR

```
function cruceFinal = cruzar(indicesCruce,poblacionSelec)
%Tenemos por parámetro los índices y la
% poblacion seleccionada anteriormente
longitud = length(indicesCruce(1,:));
filasPoblacion = length(poblacionSelec(:,1)); %cogemos todas las filas de
la poblacion seleccionada
cruceFinal = zeros(filasPoblacion,longitud); %Aqui guardaremos los
cruzados
cruces = zeros(2,longitud);
for i=1:2:length(poblacionSelec(:,2)) %SOLAMENTE QUEREMOS COGER LOS 2
PRIMEROS ELEMENTOS
    %Generamos los 2 primeros padres
    primerPadre = poblacionSelec(i,:); %1padre
    segundoPadre = poblacionSelec(i+1,:);%2padre
    %Cogemos una mitad aleatoria de la longitud de los indices
    mitadIndices = longitud/2;
    % PRIMER CRUCE
    cruces(1,1:mitadIndices) = primerPadre(1:mitadIndices);
    cruces(1,mitadIndices + 1:longitud) =segundoPadre(mitadIndices +
1:longitud);
    %SEGUNDO CRUCE
    cruces(2,1:mitadIndices) = segundoPadre(1:mitadIndices);
    cruces(2,mitadIndices + 1:longitud) =primerPadre(mitadIndices +
1:longitud);
    %devolvemos finalmente los cruces finales
    cruceFinal(i,1:longitud) = cruces(1,:);%cruce1
    cruceFinal(i+1,1:longitud) = cruces(2,:);%cruce2
end
end
```