APÉNDICE II

Se muestra íntegramente el código de programación en lenguaje Matlab, versión 5.1, del algoritmo genético principal (AGP) y básico (AGB). (En el CD, carpeta CODIGO/COALGEN).

1. COALGEN.M

```
% COALGEN script M-file contiene todos los bloques constitutivos del programa.
%
% Tecleando COALGEN se arranca el algoritmo genético.

clc       % Limpia pantalla
clear        % Borra work-space(espacio de variables de Matlab)
tic       % Pon en marcha el reloj
datos        % Lee los datos
inicio    % Inicia y define variables
pob_inicial   % Genera una población inicial
evolucion % Crea sucesivas poblaciones
resultados    % Escribe resultados
toc       % Para el reloj
```

2. DATOS.M

```
% DATOS script M-file recoge los datos de entrada del programa
%
% Lee los ficheros interfaz.txt, condensadores.txt y ficheros .raw, y almacena
los datos en
% las siguientes variables:
%
% CASOS_BASE contiene los nombres de los casos base cuya descripción reside en
ficheros .raw.
% NUM_ESCENARIOS, número total de casos base.
% HA contiene el número de horas/año para cada escenario.
% EMA es el error máximo admitido para el residuo en el algoritmo de Newton-
Raphson.
% VLIMPQ y VLIMPV contiene las tensiones límites para nudos PQ y PV
respectivamente.
% VU, vida útil de la inversión para el cálculo del valor actual neto(VAN).
% TA, tasa de actualización para el cálculo del VAN.
% PKWH, precio del Kilovatio-hora.
% NC, número total de nudos candidatos(tamaño del vector NdC).
% NUM_PASOS, número máximo de pasos para los bancos de condensadores.
% KV_T, clase(tamaño) de condensador(1,2,3...) escogido para cada paso y nivel de
tensión.
% NUM_IND, tamaño de las poblaciones(constante).
```

```
% NUM_GEN, número máximo de generaciones.
% PCR y PMU, probabilidades de cruce y mutación.
% PMU, probabilidad de mutación.
% NM y NP, tamaños de las subpoblaciones de mejores y peores.
% ELITE, si vale 1 se realiza una práctica elitista.
% UMBRAL_DIV, umbral de diversidad en tanto por ciento.
%
% DC, matriz cuyas columnas representan:
% Nivel de Tensión(KV) Precio Fijo(euros)  Potencia_Nominal_1(MVAr)
Precio_1(euros)...
%
% BASE, vector que contiene las potencias base en MVA de los casos.
% N, vector con el número total de nudos de los casos.
% TIPO, matriz cuyas columnas contienen el tipo de los nudos para cada caso:
%     1,PQ; 2,PV; 3,slack.
% NIVELV, matriz cuyas columnas contienen la tensión nominal de los nudos para
cada caso.
% V, matriz cuyas columnas contienen el módulo de la tensión en pu para cada caso
base.
% D, matriz cuyas columnas contienen el argumento de la tensión en grados para
cada caso base.
% PD, matriz cuyas columnas contienen las potencias activas demandadas en los
nudos para cada
% caso base.
% QD, idem reactivas.
% PG, idem activas generadas.
% QG, idem reactivas.
% QMIN, idem límite inferior para la reactiva o cero si no hay generador.
% QMAX, idem superior.
% VCONSIG, idem tensiones de consigna para los generadores o cero si no hay.
% TAD, datos de ajuste de los trafos LTC. Matriz n-dimensional (con
n=num_escenarios).
% YBUS, matriz de admitancia de nudos. Matriz n-dimensional (con
n=num_escenarios).
% BRD, idem datos de las ramas.
% SHUNT, idem  admitancia de los dispositivos shunt conectados en los nudos.

% Véase COALGEN.


%Lee datos del fichero interfaz.txt
[casos_base,ha,ema,vlimPQ,vlimPV,vu,ta,pKWh,nc,num_pasos,KV_T,num_ind,num_gen,...
      pcr,pmu,nm,np,elite,umbral_div]=leeinterfaz('interfaz.txt');

%Lee datos del fichero codensadores.txt
[DC]=leecondensadores('condensadores.txt');

%Lee datos del fichero raw de cada caso base
num_escenarios=size(casos_base,1); % Número total de casos base

for k=1:num_escenarios
   [basek,nk,tipok,nivelvk,vk,dk,Pdk,Qdk,Pgk,Qgk,Qmink,Qmaxk,vconsigk,TADk,...
         Ybusk,BrDk,SHUNTk]=leecaso(deblank(casos_base(k,:)));
   base(k)=basek;   % Potencia base en MVA para el caso k
   n(k)=nk;            % Número total de nudos del caso k
   tipo(:,k)=tipok; % Tipo de los nudos: 1,PQ; 2,PV; 3,slack
   nivelv(:,k)=nivelvk; % Tensión nominal de los nudos
   v(:,k)=vk;          % Módulo de la tensión en pu. Valor inicial
   d(:,k)=dk;       % Argumento de la tensión en grados. Valor inicial
   Pd(:,k)=Pdk;     % Potencias activas demandadas
   Qd(:,k)=Qdk;     % Idem reactivas
   Pg(:,k)=Pgk;     % Potencias activas generadas
   Qg(:,k)=Qgk;     % Idem reactivas
   Qmin(:,k)=Qmink; % Límite inferior para la reactiva o cero si no hay generador
```

```
   Qmax(:,k)=Qmaxk; % Idem superior
   vconsig(:,k)=vconsigk;  % Tensiones de consigna para los generadores o cero si
no hay
   TAD(:,:,k)=TADk; % Datos de ajuste de los trafos LTC
   Ybus(:,:,k)=Ybusk;% Matriz de admitancia de nudos
   BrD(:,:,k)=BrDk; % Datos de las ramas
   SHUNT(:,:,k)=SHUNTk;% Datos de los dispositivos shunt conectados en los nudos
end
```

2.1. LEEINTERFAZ.M

```
[casos_base,ha,ema,vlimPQ,vlimPV,vu,ta,pKWh,nc,num_pasos,KV_T,num_ind,num_gen,...
      pcr,pmu,nm,np,elite,umbral_div]=leeinterfaz('interfaz.txt');

%Lee datos del fichero codensadores.txt
[DC]=leecondensadores('condensadores.txt');

%Lee datos del fichero raw de cada caso base
num_escenarios=size(casos_base,1); % Número total de casos base

for k=1:num_escenarios
   [basek,nk,tipok,nivelvk,vk,dk,Pdk,Qdk,Pgk,Qgk,Qmink,Qmaxk,vconsigk,TADk,...
         Ybusk,BrDk,SHUNTk]=leecaso(deblank(casos_base(k,:)));
   base(k)=basek;   % Potencia base en MVA para el caso k
   n(k)=nk;             % Número total de nudos del caso k
   tipo(:,k)=tipok; % Tipo de los nudos: 1,PQ; 2,PV; 3,slack
   nivelv(:,k)=nivelvk; % Tensión nominal de los nudos
   v(:,k)=vk;           % Módulo de la tensión en pu. Valor inicial
   d(:,k)=dk;       % Argumento de la tensión en grados. Valor inicial
   Pd(:,k)=Pdk;     % Potencias activas demandadas
   Qd(:,k)=Qdk;     % Idem reactivas
   Pg(:,k)=Pgk;     % Potencias activas generadas
   Qg(:,k)=Qgk;     % Idem reactivas
   Qmin(:,k)=Qmink; % Límite inferior para la reactiva o cero si no hay generador
   Qmax(:,k)=Qmaxk; % Idem superior
   vconsig(:,k)=vconsigk;  % Tensiones de consigna para los generadores o cero si
no hay
   TAD(:,:,k)=TADk; % Datos de ajuste de los trafos LTC
   Ybus(:,:,k)=Ybusk;% Matriz de admitancia de nudos
   BrD(:,:,k)=BrDk; % Datos de las ramas
   SHUNT(:,:,k)=SHUNTk;% Datos de los dispositivos shunt conectados en los nudos
end
```

2.2. LEECONDENSADORES.M

```
function DC=leecondensadores(nombre_fichero)

% LEECONDENSADORES lee las potencias y precios fijo y por paso para cada nivel de
tensión
%
% DC= LEECONDENSADORES(NOMBRE_FICHERO) lee el fichero NOMBRE_FICHERO con
estructura
% del tipo del fichero condensadores.txt y guarda los datos en la matriz DC,
cuyas
% columnas representan:
%
```

```
% Nivel de Tensión(KV) Precio Fijo(euros)  Potencia_Nominal_1(MVAr)
Precio_1(euros)...

fid=fopen(nombre_fichero); % Apertura del fichero
fgets(fid);
fgets(fid);
fgets(fid);
%
% Lectura de las potencias nominales y precios por paso para cada nivel de
tensión
%
DC=[];%Matriz de datos de condensadores
FDC=[];%DC en fila
fila=[]; %vector fila auxiliar
while 1
   c=fscanf(fid,'%g',1); % Lectura del nivel de tensión,potencia y precio del
paso
   if isempty(c)~=1
      FDC=[FDC,c];
   else
      break
   end
end
for i=FDC
   if i~=0
      fila=[fila,i];
   else
      DC=[DC;fila];
      fila=[];
   end
end

% Cierre del fichero
fclose(fid);
```

2.3. LEECASO.M

```
function [base,n,tipo,nivelv,v,d,Pd,Qd,Pg,Qg,Qmin,Qmax,...
      vconsig,TAD,Y,BrD,SHUNT]=leecaso(caso)

% LEECASO realiza la lectura de datos contenidos en ficheros PTI RAW DATA
FORMAT(versión 24)
%
% [....]= LEECASO(NOMBRE_FICHERO) lee el fichero NOMBRE_FICHERO con extensión
.raw. Las
% variables de salida son:
%
% BASE, potencia base en MVA para el caso k
% N, número total de nudos del caso k
% TIPO, tipo de los nudos: 1,PQ; 2,PV; 3,slack
% NIVELV, tensión nominal de los nudos
% V, módulo de la tensión en pu. Valor inicial
% D, argumento de la tensión en grados. Valor inicial
% PD, potencias activas demandadas en MW
% QD, idem reactivas en MVA
% PG, potencias activas generadas en MW
% QG, idem reactivas en MVA
% QMIN, límite inferior para la reactiva en MVA o cero si no hay generador
% QMAX, idem superior
% VCONSIG, tensiones de consigna en pu para los generadores o cero si no hay
```

```
% TAD, datos de ajuste de los trafos LTC
% Y, matriz de admitancia de nudos
% BRD, datos de las ramas
% SHUNT, datos de los dispositivos shunt conectados en los nudos

fid=fopen(caso); % Apertura del fichero
%
%Reserva de i y j para la unidad compleja
%
i=sqrt(-1);j=i;
%
% Bloque de datos de identificación del caso
%
a=fscanf(fid,'%i',2);
base=a(2);   % Lectura de SBASE
fgets(fid);
fgets(fid);
fgets(fid);
%
% Bloque de datos de los buses
%
n=0;
SHUNT=[];
while 1
   nudo=fscanf(fid,'%i',1);   % Lectura de I
   if nudo~=0
      n=n+1;
      a=[];                   % Lectura de NAME
      while 1
         b=fscanf(fid,'%s',1);
         a=[a b];
         if a(end)==''''
            break
         end
        nombre(nudo).n=a;
      end
       a=fscanf(fid,'%g',1);   % Lectura de BASKV
       nivelv(nudo)=a;
      a=fscanf(fid,'%g',7); % Lectura de IDE, GL, BL, IA, ZONE, VM y VA
      fgets(fid);
      tipo(nudo)=a(1);
      if (a(2)~=0)|(a(3)~=0)   % Hay una admitancia shunt
         SHUNT=[SHUNT;[nudo (a(2)+i*a(3))/base]];
      end
      v(nudo)=a(6);
      d(nudo)=a(7);
   else
      break
   end
end
v=v';
d=d';
nivelv=nivelv';
tipo=tipo';
%
% Bloque de datos de las cargas
%
Pd=zeros(n,1);
Qd=Pd;
while 1
   nudo=fscanf(fid,'%i',1);   % Lectura de I
   if nudo~=0
      a=fscanf(fid,'%s',1);   % Lectura de ID
      a=fscanf(fid,'%g',5);   % Lectura de STAT, IA, ZONE, PL, QL
```

```
      Pd(nudo)=a(4)*a(1);
      Qd(nudo)=a(5)*a(1);

   else
      break
   end
end

%
% Bloque de datos de los generadores
%
Pg=zeros(n,1);
Qg=Pg;
Qmin=Pg;
Qmax=Pg;
vconsig=Pg;
while 1
   nudo=fscanf(fid,'%i',1);   % Lectura de I
   if nudo~=0
      a=fscanf(fid,'%s',1);   % Lectura de ID
      a=fscanf(fid,'%g',5);   % Lectura de PG, QG, QT, QB y VS
      Pg(nudo)=Pg(nudo)+a(1);
      Qg(nudo)=Qg(nudo)+a(2);
      Qmax(nudo)=Qmax(nudo)+a(3);
      Qmin(nudo)=Qmin(nudo)+a(4);
      vconsig(nudo)=a(5);
      fgets(fid);
   else
      break
   end
end
%
% Bloque de datos de las ramas
%
BrD=[];
while 1
   desp=0;
   p=fscanf(fid,'%i',1);   % Lectura de I
   if p~=0
      q=abs(fscanf(fid,'%i',1));% Lectura de J
      a=fscanf(fid,'%g',1);   % Lectura de CKT
      ckt=a;
      a=fscanf(fid,'%g',8);   % Lectura de R, X, B, RATEA, RATEB, RATEC, RATIO y
ANGLE
      if size(a,1)==6         % Línea sin trafo
         a=[a; 0; 0];desp=4;
      end
      admit=a(1:3); %Define la admitancia
      potmax=a(4);  %Flujo de potencia máximo(RATEA)
      c=a(7:8);     %Define la relación de transformación compleja
      fseek(fid,40+desp,0);
      status=fscanf(fid,'%i',1);   % Lectura de ST
      fgets(fid);
      if status==1
         BrD=[BrD;[p q admit' c' ckt potmax]];
      end
   else
      break
   end

end
%
% Bloque de datos de ajuste de los trafos
%
```

```
TAD=[];
while 1
   p=fscanf(fid,'%i',1);   % Lectura de I

   if p~=0
      q=fscanf(fid,'%i',1);% Lectura de J
      a=fscanf(fid,'%g',9);% Lectura de CKT, ICONT, RMA, RMI, VMA, VMI, STEP,
TABLE, CNTRL
      TAD=[TAD; p q a(1:7)' a(9)];
      fgets(fid);
   else
      break
   end
end
%
% Salto de datos de área
%
while 1
   a=fscanf(fid,'%i',1);
   if a~=0
      fgets(fid);
   else
      break
   end
end
%
% Salto de datos de líneas de continua
%
while 1
   a=fscanf(fid,'%i',1);
   if a~=0
      fgets(fid);
   else
      break
   end
end
%
% Bloque de datos de condensadores variables
%
Yc=zeros(n,1);
while 1
   nudo=fscanf(fid,'%i',1);   % Lectura de I
   if nudo~=0
      a=fscanf(fid,'%g',7);   % Lectura MODSW, VSWHI, VSWLO, SWREM, BINIT, N, B
      Yc(nudo)=(i*a(5))/base;
      SHUNT=[SHUNT;[nudo Yc(nudo)]];
      fgets(fid);
   else
      break
   end
end

fclose(fid); % Cierre del fichero
%
% CÁLCULO DE LA MATRIZ DE ADMITANCIAS NODALES
%
Y=zeros(n); % Matriz de admitancia de nudos
%
% Consideración de las líneas y de los trafos en el cálculo de Y
%
for a=1:size(BrD,1)
   p=BrD(a,1);
   q=BrD(a,2);
```

```
   if (BrD(a,6)==0)&(BrD(a,7)==0)   % Se trata de una línea
      Yl=1/(BrD(a,3)+i*BrD(a,4));BrD(a,12)=Yl;BrD(a,13)=Yl;
      Yshunt=i*BrD(a,5)/2;BrD(a,10)=Yshunt;BrD(a,11)=Yshunt;
      Y(p,q)=Y(p,q)-Yl;
      Y(q,p)=Y(q,p)-Yl;
      Y(p,p)=Y(p,p)+Yl+Yshunt;
      Y(q,q)=Y(q,q)+Yl+Yshunt;
   else   % Se trata de un trafo
      c=BrD(a,6)*(cos(BrD(a,7)*(pi/180))+i*sin(BrD(a,7)*(pi/180)));%Relación de
transformación
      Yt=1/(BrD(a,3)+i*BrD(a,4));                                    % compleja
      Yshunt=i*BrD(a,5)/2;
      Y(p,q)=Y(p,q)-Yt/c';BrD(a,12)=Yt/c';
      Y(q,p)=Y(q,p)-Yt/c;BrD(a,13)=Yt/c;
      Y(p,p)=Y(p,p)+Yshunt+Yt/abs(c)^2;BrD(a,10)=Yshunt+Yt*(1-c)/abs(c)^2;
      Y(q,q)=Y(q,q)+Yshunt+Yt;BrD(a,11)=Yshunt+Yt*(1-1/c);


      ckt=BrD(a,8); % Completa TAD
      K=1;
      for trafo=TAD.'
         if trafo(1)==p & trafo(2)==q & trafo(3)==ckt
            TAD(K,11)=BrD(a,6);TAD(K,12)=BrD(a,7);TAD(K,13)=Yt;
         end
         K=K+1;
      end

   end
end
if isempty(TAD)
   TAD=0;
end

%
% Consideración de las admitancias conectadas a los nudos en el cálculo de Y
%
if isempty(SHUNT)==0
   for a=1:size(SHUNT,1)
      p=SHUNT(a,1);
      Y(p,p)=Y(p,p)+SHUNT(a,2);
   end
else
   SHUNT=0;
end
```

3. INICIO.M

```
% INICIO script M-file. Inicia  variables del programa, recalcula los valores
iniciales
% de las tensiones en los nudos de los casos base, determina las pérdidas de los
casos base,
% el caso principal y los nudos candidatos.
%
% Define las variables:
% LGEN, longitud de un gen.
% LCR, idem cromosoma.
% IND, estructura individuo.
% POBP y NPOBP, poblaciones padre e hija de estructuras individuo.
% MEJORES y PEORES, subpoblaciones de nm y np individuos, resp.
```

```
% MEJOR_IND y PEOR_IND, registran el mejor y el peor individuo de cada
generación.
% CG, contador de generaciones.
% SUMA, suma de bondades poblacional en cada generación.
% NMUT, registra el número total de mutaciones en cada generación.
% NOCONV, número total de no convergencias del Newton-Raphson.
% PLOSS0, vector de pérdidas de los casos base.
% NDC, vector de nudos candidatos.
% CP, número de orden del caso principal.
% QIP, vector que contiene la potencia por condensador para cada nudo candidato.
% PP, vector que contiene el precio por condensador(de tipo fijo) para cada nudo
candidato.
% PF, vector que contiene el precio de instalación(indep. del número de
condensadores)para
% cada nudo candidato.
% TIPO_COND, vector que contiene el tipo de condensador: 1,fijo; 2,variable.

% Véase COALGEN.

% Longitud de la subcadena(gen) que codifica el nº pasos del condensador en un
nudo
lgen = ceil(log2(num_pasos+1));

% Determina la longitud del cromosoma(número total de alelos)
lcr= lgen*nc; %Tantos bancos posibles como nudos candidatos

% Estructura individuo
ind =
struct('crom',repmat(0,num_escenarios,lcr),'x',repmat(0,num_escenarios,nc),...

'estado',repmat(0,n(1),2*num_escenarios),'van',0,'loss',repmat(0,num_escenarios,1
));

% Define dos vectores de individuos
global pobp npobp
pobp = repmat(ind,1,num_ind);%el de la población "padre"
npobp = pobp;%el de la población "hija"

% Define dos vectores que registren los nm mejores y los np peores individuos
% en cada generación
mejores = repmat(ind,1,nm);
peores= repmat(ind,1,np);

% Define dos vectores que registren el mejor y el peor individuo de cada
generación
mejor_ind = repmat(ind,1,num_gen);
peor_ind= mejor_ind;
for k=1:num_gen
   peor_ind(k).van=Inf;
end

% Agitar semilla de la serie aleatoria
rand('state',sum(1000*clock));

% Contador de generaciones
global cg
cg=1;

% Define vector suma de bondades
suma=zeros(1,num_gen);

% Número de mutaciones y no convergencias del N-R
global NMUT NOCONV
NMUT=zeros(1,num_gen);NOCONV=0;
```

```
% Determina el caso principal y el vector de nudos candidatos
for k=1:num_escenarios
   [v(:,k),d(:,k),Ploss0(k),err]= reparto_carga(base(k),n(k),tipo(:,k),...
      nivelv(:,k),v(:,k),d(:,k),Pd(:,k),...
      Qd(:,k),Pg(:,k),Qg(:,k),Qmin(:,k),Qmax(:,k),vconsig(:,k),TAD(:,:,k),...
      ,Ybus(:,:,k),ema,BrD(:,:,k),SHUNT(:,:,k));
end

% Ordena vector de pérdidas Ploss0 de los casos base
[Ploss0_ord,caso_ord]=sort(Ploss0); %en orden ascendente
caso_ord=caso_ord([num_escenarios:-1:1]);    %en orden descendente
cp=caso_ord(1); %caso principal

fprintf('\n\n    Si desea introducir manualmente los nudos introduzca "S" y pulse
return.\n');
fprintf('    En caso contrario, pulse return ');
du=input('para continuar.\n \n>> ', 's');

if du=='S' | du=='s'
   for i=1:nc
      fprintf('\n     ( %u )Introduzca el índice del nudo ',i);
      NdC(i)=input(' >> ');
      fprintf('\n');
   end
else
   % Ordena vector de tensiones del caso principal
    [v_ord,n_ord]=sort(v(:,cp)); %en orden ascendente
    NdC=[];
    for k=1:n
    if tipo(n_ord(k),cp)==1  %(sólo nudos PQ)
        NdC=[NdC n_ord(k)];   % vector de nudos candidatos
    end
    if size(NdC,2)==nc
       break
    end
   end
end

% Compara casos base: Deben ser iguales el número de nudos en todos ellos
if prod(n/n(1))~=1
   error('Todos los casos base deben tener el mismo número de nudos');
end

% Potencia y precio por paso para cada nivel de tensión
for k=1:nc % Para todos los nudos candidatos
   t=KV_T(find(KV_T(:,1)==nivelv(NdC(k),cp)),2); % Tamaño(clase 1,2,...)
   % Potencia y precio por paso y fijo para cada nudo candidato
   Qip(k)=DC(find(DC(:,1)==nivelv(NdC(k),cp)),2*t+1);
   pp(k)=DC(find(DC(:,1)==nivelv(NdC(k),cp)),2*t+2);
   pf(k)=DC(find(DC(:,1)==nivelv(NdC(k),cp)),2);
end

% Inicia tipo de condensadores a condensadores tipo 1(fijos)
tipo_cond=ones(1,nc);
```

4. POB_INICIAL.M

```
% POB_INICIAL script M-file crea una población inicial aleatoriamente.
%
```

```
% La población se obtiene individuo a individuo, a partir de la generación
aleatoria de
% su cromosoma CROM. En caso de producirse un individuo fuera de rango(con más
condensadores
% en un nudo que los permitidos), se descarta pero aprovechando los genes válidos
% y generando sólo los 'defectuosos'. Para ello se forma un patrón cromosómico,
% PATRONCROM, donde las posiciones de los alelos defectuosos toman valor
uno.Inicialmente
% todos están puestos a uno.
% A partir del cromosoma, se completa el individuo. Si no cumple las
restricciones
% de tensiones y flujos de potencia, se descarta totalmente.
% Guarda en SUMA la suma de bondades(valores actuales netos) poblacional,
registra el
% MEJOR_IND y el PEOR_IND y vuelca en pantalla el informe de la primera
generación.
% Finalmente, guarda la población obtenida en POBP(población padre de la
siguiente
% generación).
%
% Véase COALGEN.

k = 1;                          %Inicia contador de individuos a 1
patroncrom=ones(lgen,nc);  % Inicia patrón cromosómico a 1
esc=(num_gen+cg)/(1.5*num_gen); % factor de escala

% Mientras no se alcance el límite poblacional, haz:
while k <= num_ind
   % Genera el cromosoma de un individuo
   for i = 1 : lcr
      if patroncrom(i)==1
         crom(i) = uoc(0.5); % Uno o cero con probabilidad 0.5
      end
   end
   % Completa individuo
   completa_ind
end

% Guarda en 'suma' el sumatorio de las bondades de todos
% los individuos de esta primera población
suma(cg) = sum(cat(2,npobp.van));
% Suma escalada
suma_esc= sum(cat(2,npobp.van).^esc);

% Crea las subpoblaciones  'mejores' y 'peores'
subpoblaciones

% Registra el mejor y el peor individuo de la presente generación
mejor_ind(cg) = mejores(1);
peor_ind(cg)= peores(1);

% Calcula indicador de diversidad
diversidad

% Informe en pantalla
infor_generacion

% La población hija es la población padre para la siguiente generación
pobp=npobp;

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

5. EVOLUCION.M

```
% EVOLUCION script M-file que genera sucesivas poblaciones utilizando operadores
genéticos.
%
% Los operadores genéticos son selección, cruce y mutación, implementados en las
funciones
% SELEC, CRUCE y MUTA, repectivamente.
%
% Se llega a una generación final confiando que ésta contenga un individuo
% que proporcione un valor casi-óptimo.
%
% Véase COALGEN.

num_ind_a_generar=num_ind; % En principio, se genera un número de individuos
igual al tamaño
                                % poblacional
for cg=2:num_gen % Para todas las generaciones que siguen a la primera, haz:

    k=1; % Inicia contador de individuos
   while 1 % Hasta alcanzar el tamaño cte de la población, haz:

      % Selección de una pareja reproductora: pad1 y pad2
      pad1=selec(sign(cat(2, pobp.van)).*abs(cat(2, pobp.van)).^esc,suma_esc);%
Padre 1
      pad2=selec(sign(cat(2, pobp.van)).*abs(cat(2, pobp.van)).^esc,suma_esc);%
Padre 2

      % Cruce y mutación: Se generan los cromosomas de dos candidatos para
      % formar parte de la población hija
      cr=repmat(0,2,lcr);
      [cr(1,:),cr(2,:)]=
cruce(pobp(pad1).crom(cp,:),pobp(pad2).crom(cp,:),pcr,pmu);

      % Completa sus respectivas estructuras (en caso de ser válidos)
      for i=1:2
         crom=cr(i,:);
         completa_ind
       if k> num_ind_a_generar
           break % Pasa a otra generación si se alcanza el límite poblacional
         end
      end % fin de for i

      if k> num_ind_a_generar
         break % Pasa a otra generación si se alcanza el límite poblacional
      end

   end % fin de while 1

   % Crea las subpoblaciones  'mejores' y 'peores'
   subpoblaciones

   % Registra el mejor y el peor individuo de la presente generación
   mejor_ind(cg) = mejores(1);
   peor_ind(cg)= peores(1);

  % Conservación del mejor individuo generación tras generación
  if mejor_ind(cg-1).van > mejor_ind(cg).van
     % Reemplaza el peor de la hija por el mejor de la generación padre
     npobp(i_peores(1))=mejor_ind(cg-1);
     % Actualiza subpoblaciones
```

```
      subpoblaciones
      % Actualiza el mejor y el peor individuo de la presente generación
       mejor_ind(cg) = mejores(1);
      peor_ind(cg)= peores(1);
   end

   % Guarda en 'suma' el sumatorio de las bondades de todos
    % los individuos de esta primera población
    suma(cg) = sum(cat(2,npobp.van));
   % Escalado: bondad^esc
   if cg<(num_gen/2)
      esc=(num_gen+cg)/(1.5*num_gen); % Factor de escala
   else
      esc=(num_gen+0.5*cg)/num_gen;  % Factor de escala
   end
   suma_esc= sum(cat(2,npobp.van).^esc);

   % Calcula indicador de diversidad
    diversidad

   % Informe en pantalla
    infor_generacion

   % Si cae la diversidad, finaliza
   if div<umbral_div
      break
   end

   % La población hija es la población padre para la siguiente generación
   pobp=npobp;

   % Elitismo: Los mejores individuos obtenidos pasan directamente a la
   % siguiente generación
   if elite == 1
      num_ind_a_generar=num_ind-nm; % En la siguiente generación se engendran
sólo num_ind-nm
      npobp((num_ind_a_generar+1):end)= mejores;
   end

end % fin de for cg
```

5.1. 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
proporcional a
% B(K)/SB. B es el vector de bondades poblacional y SB es el sumatorio de las
mismas.
%
% Se emplean dos métodos: Rueda Ruleta y Torneo.
%
% El primero es sencillamente una 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. Se usa cuando todos los
elementos
% de B  son positivos o nulos.
%
% El segundo consiste en seleccionar el mejor de un grupo de individuos extraídos
% aleatoriamente (sin reposición) de la población.
% Se emplea sólo cuando algún elemento de B es negativo.
%
% Véase EVOLUCION.

if  isempty(find(sign(b)==-1)) % Si todos los valores son mayores o iguales que
cero

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

else   % Si hay valores negativos o nulos

   % Torneo
    % Extracción aleatoria del grupo reproductor
    grupo=[];
    while isempty(grupo)==1
        for i=1:size(b,2)
        if uoc(0.10)
            grupo=[grupo i];
        end
    end
    end
    % Se toma el mejor del grupo
    [b_ord,i_ord]=sort(b(grupo));   % Ordena de menor a mayor bondad
    y=grupo(i_ord(end));

end
```

5.2. CRUCE.M

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

% CRUCE cruza dos individuos y llama a la función MUTACION.

% [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.

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
```

5.3. 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(alelo),
cuando
% ésta se produce con probabilidad PM.
% Véase EVOLUCION y CRUCE.

global NMUT NMUTs cg

if uoc(pm)
   y=not(bit); % Muta el bit
   NMUT(cg)=NMUT(cg)+1;% Suma uno al contador de mutaciones del algoritmo
principal
   NMUTs=NMUTs+1;      % Suma uno al contador de mutaciones del AGB
else
   y=bit; % Devuelve el bit sin mutar
end
```

6. RESULTADOS.M

```
% RESULTADOS script M-file escribe el fichero solucion.txt y guarda el gráfico
% de la evolución en graf_evo.mat.
%
% Véase COALGEN.

% Escribe en el fichero solucion.txt

fid = fopen('solucion.txt','w');
fprintf(fid,'**********************************************************************
**********');
fprintf(fid,'*****************\n');
fprintf(fid,'                  FICHERO DE RESULTADOS DEL PROGRAMA COALGEN:
solucion.txt\n');
fprintf(fid,'**********************************************************************
**********');
fprintf(fid,'*****************\n');
fprintf(fid,'\n');
fprintf(fid,'**********************************************************************
**********');
```

```
fprintf(fid,'*****************\n');
fprintf(fid,'                                DATOS DE ENTRADA: interfaz.txt\n');
fprintf(fid,'*************************************************************************
**********');
fprintf(fid,'*****************\n');
fprintf(fid,'\n');

fprintf(fid,'*** Nombres de los ficheros *.raw y horas/año de vigencia: \n');
for i=1:num_escenarios
   fprintf(fid,'%s  %u\n',casos_base(i,:),ha(i));
end
fprintf(fid,'\n');
fprintf(fid,'*** Error máximo admisible para el reparto de carga');
fprintf(fid,'(potencia en p.u.): ema= %g\n',ema);
fprintf(fid,'\n');
fprintf(fid,'*** Tensiones límites(en p.u.) \n');
fprintf(fid,'\n');
fprintf(fid,'Nudos PQ:   Límite Superior  Límite inferior\n',vlimPQ');
fprintf(fid,'                  %g           %g\n',vlimPQ(1),vlimPQ(2));
fprintf(fid,'Nudos PV:\n                 %g
%g\n',vlimPV(1),vlimPV(2));
fprintf(fid,'\n');
fprintf(fid,'*** Vida útil de la inversión(años), tasa de actualización y ');
fprintf(fid,'precio del Kilovatio-hora(euros): \n');
fprintf(fid,'vu= %g años, ta= %g, pKWh= %g euros\n',vu,ta,pKWh);
fprintf(fid,'\n');
fprintf(fid,'*** Número de nudos candidatos y número máximo de pasos para los
bancos \n');
fprintf(fid,'nc= %g nudos, num_pasos= %g\n',nc,num_pasos);
fprintf(fid,'\n');
fprintf(fid,'*** Tamaños de paso para cada nivel de tensión\n');
fprintf(fid,'KV    Tamaño\n');
for i=KV_T'
   fprintf(fid,'%3i     %g\n',i);
end
fprintf(fid,'\n');
fprintf(fid,'*** Parámetros de funcionamiento del AG:\n');
fprintf(fid,'num_ind= %g individuos, num_gen= %g generaciones, pcr= %g, pmu=
%g\n',...
   num_ind,num_gen,pcr,pmu);
fprintf(fid,'nm= %g individuos, np= %g individuos, elite= %g, umbral_div= %g
%\n',...
   nm,np,elite,umbral_div);
fprintf(fid,'\n');
fprintf(fid,'\n');
fprintf(fid,'*************************************************************************
**********');
fprintf(fid,'*****************\n');
fprintf(fid,'                                      RESULTADOS\n');
fprintf(fid,'*************************************************************************
**********');
fprintf(fid,'*****************\n');
fprintf(fid,'\n');
fprintf(fid,'*** MEJORES DISTRIBUCIONES OBTENIDAS:\n');
fprintf(fid,'\n');
for s=1:nm
   fprintf(fid,'********************\n');
   fprintf(fid,'*    Solución %i *\n',s);
   fprintf(fid,'********************\n');
   fprintf(fid,'\n');
   fprintf(fid,'*** Distribución de condensadores\n');
    fprintf(fid,'Escenarios:');
   for c=caso_ord
      fprintf(fid,'   %15s  ',casos_base(c,:));
```

```
   end
    fprintf(fid,'\n');
   fprintf(fid,'   (Nudo)  _____________________(MVAr
nominales)_____________________\n');
   for p=1:nc
      fprintf(fid,'%6i',NdC(p));
      for c=caso_ord
         fprintf(fid,'         %2.3f       ',Qip(p)*mejores(s).x(c,p));
      end
      fprintf(fid,'\n');
   end
   fprintf(fid,'
_____________________________________________________\n');
    fprintf(fid,'\n');
   fprintf(fid,'Escenarios:');
   for c=caso_ord
      fprintf(fid,'       %15s',casos_base(c,:));
   end
   fprintf(fid,'\n');
    fprintf(fid,'*** Pérdidas (MW):');

   for c=caso_ord
      fprintf(fid,'          %4.3f      ',mejores(s).loss(c));
   end
   fprintf(fid,'\n');
   fprintf(fid,'*** Valor Actual Neto de la Solución (euros) %i
:%10.0f\n',s,mejores(s).van);
   fprintf(fid,'\n');
   fprintf(fid,'Estado:\n');
   fprintf(fid,'   NUDO   ');
   for c=caso_ord
      fprintf(fid,'       v(p.u.)        d(º)');
   end
    fprintf(fid,'\n');
   for p=1:n
      fprintf(fid,'%6i     ',p);
      for c=caso_ord
         fprintf(fid,'       %5.3f       %6.2f',...
            mejores(s).estado(p,2*c-1),mejores(s).estado(p,2*c));
      end
      fprintf(fid,'\n');
   end
   fprintf(fid,'\n');
   fprintf(fid,'\n');
end

fprintf(fid,'\n');
fprintf(fid,'*** ESTADÍSTICAS:\n');
fprintf(fid,'\n');
fprintf(fid,'Número total de mutaciones: %g\n',sum(NMUT));
fprintf(fid,'Número total de ''no convergencias'': %g\n',NOCONV);
fprintf(fid,'Número de generaciones: %g, de un máximo de %g\n',cg,num_gen);
%
%Cierre del fichero
%
fclose(fid);
%
% Visualización en pantalla:
%
fprintf('\n');
fprintf('  MEJOR DISTRIBUCIÓN OBTENIDA:\n');
fprintf('\n');
fprintf('  Nudo            Número de escalones ( ');
for h=caso_ord
```

```
   fprintf('%9s ',casos_base(h,:));
end
fprintf(')\n');
fprintf('  ____________________________________________________\n\n');
for p=1:nc
   fprintf('%6i       %6i\n',NdC(p));
   for h=caso_ord
      fprintf('  %6i',mejores(1).x(h,p));
   end
   fprintf('\n');
end
fprintf('  ____________________________________________________\n\n');
disp(['     VAN... ', num2str(mejores(1).van),'euros.'])
fprintf('\n\n    Escenario:     Pérdidas:\n\n');
for h=caso_ord
   fprintf('    %9s      %4.3f MW\n',casos_base(h,:),mejores(1).loss(h));
end
fprintf('\n');


fprintf('\n');
fprintf('  Tras %u generaciones:\n\n',cg);
fprintf('   * El número de mutaciones ha sido %u\n',sum(NMUT));
fprintf('   * El número de ''no convergencias'' ha sido %u\n',NOCONV);
fprintf('   * Diversidad Final: %2.2f %%\n', div);
ejex=1:cg;
plot(ejex,cat(2,mejor_ind(1:cg).van),'rx-',ejex,cat(2,peor_ind(1:cg).van),...
   'bx-',ejex,suma(1:cg)/num_ind,'k:');
title('Evolución del valor del VAN a lo largo de sucesivas generaciones')
xlabel('Generaciones')
ylabel('VAN')

% Guarda gráfico de la evolución en graf_evo.mat y crea el módulo graf_evo.m
% Tecleando 'graf_evo', se muestra el gráfico en pantalla

print graf_evo -dmfile
```

7. REPARTO_CARGA.M

```
function [v,d,Ploss,err]= reparto_carga(base,n,tipo,nivelv,v,d,Pd,Qd,Pg,Qg,...
   Qmin,Qmax,vconsig,TAD,Y,ema,BrD,SHUNT);

% REPARTO_CARGA calcula iterativamente el estado de un sistema de potencia
% (Algoritmo de Newton-Raphson).
%
% [V,D,PLOSS,ERR]=REPARTO_CARGA(...) devuelve los valores de la tensión en
módulo(V,en pu) y
% en argumento(D, en grados) de los nudos, las pérdidas PLOSS(en MW) y el error
final ERR
% en pu(valor máximo del residuo). Las variables de entrada son:
%
% BASE, potencia base en MVA para el caso k
% N, número total de nudos del caso k
% TIPO, tipo de los nudos: 1,PQ; 2,PV; 3,slack
% NIVELV, tensión nominal de los nudos
% v, módulo de la tensión en pu.
% d, argumento de la tensión en grados.
% PD, potencias activas demandadas en MW
% QD, idem reactivas en MVA
% PG, potencias activas generadas en MW
```

```
% QG, idem reactivas en MVA
% QMIN, límite inferior para la reactiva en MVA o cero si no hay generador
% QMAX, idem superior
% VCONSIG, tensiones de consigna en pu para los generadores o cero si no hay
% TAD, datos de ajuste de los trafos LTC
% Y, matriz de admitancia de nudos
% EMA, error máximo admisible en pu
% BRD, datos de las ramas
% SHUNT, datos de los dispositivos shunt conectados en los nudos
%
% REPARTO_CARGA llama a los siguientes módulos: INI_NEWRAPH,
ITERACION_ESTADO_RED,
% VIOLACION_NUDOS_PV, CAMBIAR_NUDO, AJUST_TOMAS, ESTADO_FINAL

global NOCONV                       % Acumulador de 'no convergencias'
k=0;                                   % Contador de iteraciones
ini_newraph                            % Inicia variables del algoritmo
CPV=PV;                                % Copia de PV (nudos PV)
Qlim=[Qmax,Qmin];                      % Límites de reactiva de los generadores
viol=[PV,zeros(size(PV,1),3)];      % Inicia VIOL(matriz que registra la violación
                                       % de reactiva en los nudos PV. Véase
VIOLACION_NUDOS_PV)
if TAD==0
   TAD=[];
end

if isempty(TAD) == 0
   control=zeros(size(TAD,1),1); % Variable de control para los trafos LTC
else
   control=0;                      % Vale cero si no hay trafos LTC
end

while 1
   while err>ema
      k=k+1;
      if k>200                         % Número máximo de iteraciones permitido
         break
      end

       Iteracion_Estado_Red       % Realiza una iteración
      Violacion_Nudos_PV          % Comprueba violación de reactiva
      if k>=10
         Cambiar_Nudo             % Permite cambiar a nudo PQ después de 9
iteraciones
      end
      dP=Pesp-Pcal; dQ=Qesp-Qcal;% Calcula los residuos
    dRes=[dP([PV;PQ]);dQ([PQ])];
      err=max(abs(dRes));         % Calcula el error
   end
   if k>200
      NOCONV=NOCONV+1; % El algoritmo no converge tras 200 iteraciones
      break
   end

   if isempty(TAD) == 0 % Ajusta tomas de los trafos LTC
      ajust_tomas
   end

   % Control de salida
   if sum(viol(:,2))==0 & sum(control)==0 % Termina si no hay violaciones de
reactiva
      break                                      % ni se han cambiado tomas en la
última iteración
   else
```

```
     err=ema+1;                          % En caso contrario, aumenta el error
artificialmente
    end
end

Estado_Final                        % Prepara variables de salida
```

7.1. INI_NEWRAPH.M

```
% INI_NEWRAPH script M-file inicia las variables para el reparto de carga
% (Algoritmo de Newton_Raphson)
%
% Véase REPARTO_CARGA. Las variables con las que trabaja el algoritmo son:
% SL, PV, PQ: Vectores de tipos de nudos(SL,slack; PV,nudos PV; PQ,nudos PQ).
% v,d: vectores módulo(pu) y desfase(radianes) de las tensiones nodales(variables
de estado).
% V: Vector fasor tensión de los nudos.
% I: Vector fasor de intensidad en cada nudo hacia la red.
% PD,QD,PG,QG: Vectores de potencias en pu.
% QMAX,QMIN: Límites de reactiva de los nudos de generación.
% PESP, QESP: Potencias especificadas(generadas-demandadas).
% PCAL, QCAL: Potencias calculadas.
% DP,DQ,DRES: Diferencias entre las potencias especificadas y calculadas;
residuos.
% ERR: Mayor componente del residuo.
% dv,dd: Incremento de las variables de estado(pu y radianes, respectivamente).

% Nudo slack, vector de nudos PV y vector de nudos PQ
SL=find(tipo==3);PV=find(tipo==2); PQ=find(tipo==1);

% Vector de tensiones de nudos(a partir de VM y VA de la lectura del .raw)
d=d*pi/180;  % Pasa desfases a radianes

% Se asegura que los generadores están a sus tensiones de consigna
for nudo=PV'
   v(nudo)=vconsig(nudo);
end
v(SL)=vconsig(SL);

% Fasores tensión inicial de los nudos
V=v.*exp(d*j);

% Intensidad que sale de cada nudo
I=Y*V;

% Vectores de potencias especificadas en pu en cada nudo
Pd=Pd/base; Qd=Qd/base; Pg=Pg/base; Qg=Qg/base; Qmax=Qmax/base; Qmin=Qmin/base;

% Potencias especificadas netas en cada nudo
Pesp=sparse(Pg-Pd); Qesp=sparse(Qg-Qd);

% Potencias calculadas en cada nudo
Pcal=sparse(real(V.*conj(I))); Qcal=sparse(imag(V.*conj(I)));

% Residuos iniciales
dP=Pesp-Pcal; dQ=Qesp-Qcal;
dRes=[dP([PV;PQ]);dQ([PQ])];

% Error inicial
```

```
err=max(abs(dRes));

% Incrementos del módulo y del ángulo de V
dv=zeros(n,1);
dd=zeros(n,1);
```

7.2. ITERACIÓN_ESTADO_RED.M

```
% ITERACION_ESTADO_RED  script M-file construye el Jacobiano y resuelve el sistema
de ecuaciones.
%
% Llama a la función JAC y devuelve el estado resultante.
% Véase REPARTO_CARGA,INI_NEWRAPH y JAC.

J=jac(Y,v,d,Pcal,Qcal,PV,PQ);              % Jacobiano
perm=colmmd(J);                            % Vector de permutaciones de columnas
J=J(:,perm);                               % Reordena columnas
dx= J\dRes;                                   % Resuelve el sistema
dx(perm)=dx;                                  % Guarda resultado en el puesto
que corresponda
dd([PV;PQ])= dx(1:(size(PV,1)+size(PQ,1)));
dv([PQ])=dx((size(PV,1)+size(PQ,1)+1):(size(PV,1)+2*size(PQ,1)));

d=d+dd;                                                  % Actualiza estado
v=v.*(1+dv); dv=zeros(n,1);
V= v.*exp(d*j);
I=Y*V;
Pcal=sparse(real(V.*conj(I))); Qcal=sparse(imag(V.*conj(I)));    % Calcula
potencias
```

7.2.1. JAC

```
function JB=jac(Y,v,d,pest,qest,PV,PQ)

% JAC calcula la matriz jacobiana para el flujo de cargas normal.
%
% JB=JAC(Y,V,D,PEST,QEST,PV,PQ) devuelve en JB el Jacobiano de la red que tiene
una matriz
% de admitancias nodales Y, particularizado en el estado determinado por V y D,
módulo(en pu)
% y argumento(en radianes)de las tensiones en los nudos, a partir de las
potencias calculadas
% en una iteración anterior del algoritmo de Newton-Raphson, PEST y QEST. PV y PQ
son sendos
% vectores columna que contienen respectivamente los índices de los nudos PV y
PQ.

% Inicia submatrices del Jacobiano
H=sparse(zeros(size(v,1)));
N=sparse(zeros(size(v,1)));
J=sparse(zeros(size(v,1)));
L=sparse(zeros(size(v,1)));
```

```
% Calcula H
for k=[PV;PQ]'
   for m=[PV;PQ]'
      if k~=m
         H(k,m)=v(k)*v(m)*(real(Y(k,m))*sin(d(k)-d(m))-imag(Y(k,m))*cos(d(k)-
d(m)));
      else
     H(k,k)=-qest(k)-imag(Y(k,k))*v(k)^2;
      end
   end
end

% Calcula N
for k=[PV;PQ]'
   for m=PQ'
      if k~=m
         N(k,m)=v(k)*v(m)*(real(Y(k,m))*cos(d(k)-d(m))+imag(Y(k,m))*sin(d(k)-
d(m)));
      else
     N(k,k)=pest(k)+real(Y(k,k))*v(k)^2;
      end
   end
end

% Calcula J
for k=PQ'
   for m=[PV;PQ]'
      if k~=m
         J(k,m)=-v(k)*v(m)*(real(Y(k,m))*cos(d(k)-d(m))+imag(Y(k,m))*sin(d(k)-
d(m)));
      else
     J(k,k)=pest(k)-real(Y(k,k))*v(k)^2;
      end
   end
end

% Calcula L
for k=PQ'
   for m=PQ'
      if k~=m
         L(k,m)=v(k)*v(m)*(real(Y(k,m))*sin(d(k)-d(m))-imag(Y(k,m))*cos(d(k)-
d(m)));
      else
     L(k,k)=qest(k)-imag(Y(k,k))*v(k)^2;
      end
   end
end

% Devuelve el Jacobiano, JB
H=H([PV;PQ],[PV;PQ]);
N=N([PV;PQ],PQ);
J=J(PQ,[PV;PQ]);
L=L(PQ,PQ);
JB=[H,N;J,L];
```

7.3. VIOLACIÓN_NUDOS_PV.M

```
% VIOLACION_NUDOS_PV script M-file averigua si se produce la violación de los
% límites de reactiva de los nudos PV y registra si es por defecto o por exceso.
%
```

```
% Actualiza la 2ªcolumna de VIOL.
% VIOL es una matriz con tantas filas como nudos PV y cuyas columnas son:
%
%  INDICE NUDO         TIPO VIOLACION     NºVIOLACIONES    1 SI V~=VCONSIG
%
% Véase REPARTO_CARGA y CAMBIAR_NUDO.

for i=1:size(viol,1)
   if Qcal(viol(i,1))> Qlim(viol(i,1),1)-Qd(viol(i,1))
      viol(i,2)=1;%  violación por exceso
   elseif Qcal(viol(i,1)) < Qlim(viol(i,1),2)-Qd(viol(i,1))
      viol(i,2)=2;%  violación por defecto
   else
      viol(i,2)=0;%  no hay violación
   end
end
```

7.4. CAMBIAR_NUDO.M

```
% CAMBIAR_NUDO script M-file pasa nudos PV de estado PV a estado PQ y viceversa.
%
% Actualiza las columnas 3ª y 4ª de VIOL.
% VIOL es una matriz con tantas filas como nudos PV y cuyas columnas son:
%
%  INDICE NUDO         TIPO VIOLACION     NºVIOLACIONES    1 SI V~=VCONSIG
%
% Para evitar reiterados cambios de estado PV a PQ, se implementa el módulo
APROXIMA_TENSION.
% Véase también REPARTO_CARGA y VIOLACION_NUDOS_PV.

for i=1:size(viol,1) % Para todos los nudos PV, haz:

   if tipo(viol(i,1))==2 & viol(i,2)~=0 % Se trata de un nudo en estado PV que
viola límites
       viol(i,3)=viol(i,3)+1;           % Contabiliza número de violaciones
alternativas
          Qesp(viol(i,1))=Qlim(viol(i,1),viol(i,2))-Qd(viol(i,1)); % Fija Q
         tipo(viol(i,1))=1;                % Pasa a ser PQ (se libera v)

   elseif tipo(viol(i,1))==1 & viol(i,2)==0 % Nudo en estado PQ que no viola
límites
      tipo(viol(i,1))=2;                   % Pasa a ser PV
      % Asigno al nudo su tensión de consigna(si nºviolaciones<4) o próxima a
ella
      if viol(i,3)<4

v(viol(i,1))=vconsig(viol(i,1));V(viol(i,1))=v(viol(i,1))*exp(j*d(viol(i,1)));
      else
         Aproxima_Tension
      end

   elseif tipo(viol(i,1))==2 & viol(i,4)==1 %nudo en estado PV con v distinta a
la de consigna
      Aproxima_Tension
   end

   % NOTA: Puede existir violación en un nudo en estado PQ, pero se corrige con
la condición
   % err<ema
```

```
   viol(i,2)=0;% Se pone a cero el marcador de violaciones(una condición
                                                                %de salida de
REPARTO_CARGA)
end
clear PV PQ
PV=find(tipo==2);PQ=find(tipo==1); % Actualiza estados(PV o PQ)de los nudos
```

7.5. APROXIMA_TENSIÓN.M

```
% APROXIMA_TENSION  acerca la tensión del nudo a su tensión de consigna.
%
% En lugar de devolver el valor de set-point a un nudo PV que vuelva a estar
dentro
% de límites en la generación de reactiva, se le da un valor cada vez más
cercano.
% Modifica el valor de la 4ªcolumna de VIOL.
% VIOL es una matriz con tantas filas como nudos PV y cuyas columnas son:
%
%   INDICE NUDO          TIPO VIOLACION     NºVIOLACIONES    1 SI V~=VCONSIG
%
%Véase REPARTO_CARGA, CAMBIAR_NUDO, VIOLACION_NUDOS_PV.

viol(i,4)=1;  % Registra que el nudo en estado PV tiene una tensión distinta a la
de consigna
if v(viol(i,1))>1.00001*vconsig(viol(i,1))
   v(viol(i,1))=v(viol(i,1))-abs(v(viol(i,1))-vconsig(viol(i,1)))*0.00001;
elseif v(viol(i,1))<0.99999*vconsig(viol(i,1))
   v(viol(i,1))=v(viol(i,1))+abs(v(viol(i,1))-vconsig(viol(i,1)))*0.00001;
else
   viol(i,4)=0; %vuelve a tener su tensión de consigna(muy aproximadamente)
end
V(viol(i,1))=v(viol(i,1))*exp(j*d(viol(i,1))); % Fasor tensión del nudo
```

7.6. AJUST_TOMAS.M

```
% AJUST_TOMAS    cambia las tomas de los trafos LTC que se encuentren en control
automático.
%
% Ajusta automáticamente las tomas para llevar al nudo de tensión controlada
dentro de su
% rango de variabilidad. Utiliza los datos contenidos en la matriz TAD.
% Véase REPARTO_CARGA, DATOS, LEECASO.

kt=1; % Índice de trafo
for trafo=TAD.' % Para todos los trafos LTC

   ntc=trafo(4); % Nudo de tensión controlada
   estado=trafo(10); % =1, habilita el control automático, =0, lo deshabilita.
    if isempty(find(PV==ntc))& estado==1 % Para nudos(salvo PV) que estén en
control aut.,haz:
    p=trafo(1); % Lado de las tomas
    q=trafo(2); % Lado sin tomas
    vmax=trafo(7);vmin=trafo(8); % Límites de tensión para el nudo ntc
    paso=trafo(9);rmax=trafo(5);rmin=trafo(6); % paso y relaciones de
transformación límites
    r=trafo(11); ang=trafo(12);r=(round(10^5*r)/10^5);
```

```
    c=r*exp(j*ang); % Relación de transformación actual
    Yt=trafo(13); % Admitancia del trafo

    if ntc==q
        if v(ntc)<vmin & r~=rmin
            rn= r-paso;control(kt)=1;
        elseif v(ntc)>vmax & r~=rmax
            rn= r+paso;control(kt)=1;
        else
            rn=r;control(kt)=0;
        end
    elseif ntc==p
        if v(ntc)<vmin & r~=rmax
            rn= r+paso;control(kt)=1;
           elseif v(ntc)>vmax & r~=rmin
            rn= r-paso;control(kt)=1;
        else
            rn=r;control(kt)=0;
        end
    else
        error('Error: El nudo de control del trafo debe ser uno de sus bornes')
       end

        if rn~=r                % Actualiza relación de transformación y matriz de
admitancias
          TAD(kt,11)=rn;
          cn=rn*exp(j*ang); % Nueva relación de transformación
           Y(p,q)=Y(p,q)+Yt/c'-Yt/cn';
        Y(q,p)=Y(q,p)+Yt/c-Yt/cn;
          Y(p,p)=Y(p,p)-Yt/abs(c)^2+Yt/abs(cn)^2;
        end
    end

    kt=kt+1;      % Incrementa índice del trafo
end
```

7.7. ESTADO_FINAL

```
% ESTADO_FINAL scrip M-file prepara las variables de salida de la función
REPARTO_CARGA.


% Potencias
Pg(SL)=Pcal(SL)+Pd(SL);
Qg([SL;CPV])=Qcal([SL;CPV])+Qd([SL;CPV]);
% Pérdidas
Ploss=sum(Pg)-sum(Pd);
% Intensidades hacia la red en los nudos
Ibase=(base/sqrt(3))./nivelv;
Ikamp=I.*Ibase;
% Número de iteraciones
disp(['k= ' num2str(k)])
% Desfases en grados
d=d*180/pi;

% Calcula potencia consumida por resistencias shunt
j=sqrt(-1);
ls=0;
if SHUNT~=0
 for i=SHUNT'
```

```
    p=i(1);y=real(i(2));ir=y*v(p);
    if ir~=0
      ls=ls+ir^2/y;
    end
 end
end

% Pérdidas descontando la potencia de las resistencias shunt
Ploss=(Ploss-ls)*base;
disp(['Ploss(MW) = ' num2str(Ploss.')])
```

8. COMPLETA_IND.M

```
% COMPLETA_IND script M-file completa cada estructura tipo individuo de la
población en
% ciernes.
%
% Empieza por decodificar el cromosoma CROM, en X. Comprueba que la distribución
X esté
% dentro de rango. En caso afirmativo, obtiene la matriz de admitancias de nudos
Y a partir de
% la correspondiente al caso base principal, YBUS. Resuelve el reparto de carga
para el caso
% principal y comprueba las restricciones. Si se verifican, calcula el flujo de
caja para
% el caso principal, N(CP),y arranca el Algoritmo Genético Básico(AGB) para cada
caso
% restante.
% Con los resultados, completa la estructura del individuo NPOBP(K).
% Véase POB_INICIAL y EVOLUCION.

% A partir del cromosoma(genotipo), genera el fenotipo:
x = decod(crom,lgen);

% Si pertenece al espacio de soluciones admisibles, haz:
if prod(x <= num_pasos)

   % Modifica la matriz de admitancias de nudos
    Y= modY(x,Ybus(:,:,cp),NdC,Qip/base(cp));
   Y=sparse(Y);

   % Resuelve el reparto de carga para este valor de x:
   [vk,dk,Ploss,err]= reparto_carga(base(cp),...
   n(cp),tipo(:,cp),nivelv(:,cp),v(:,cp),d(:,cp),Pd(:,cp),...

Qd(:,cp),Pg(:,cp),Qg(:,cp),Qmin(:,cp),Qmax(:,cp),vconsig(:,cp),TAD(:,:,cp),Y,ema,
BrD(:,:,cp),SHUNT(:,:,cp));

   % Comprueba cumplimiento de las restricciones(tensiones y potencias)
   w=cp;
   restricciones % Devuelve restric=1 si se verifican
   if restric
      npobp(k).crom(cp,:)=crom;
      npobp(k).x(cp,:)=x;
    npobp(k).estado(:,2*cp-1)=vk;    % Tensiones
      npobp(k).estado(:,2*cp)=dk;    % Desfases
      npobp(k).loss(cp)=Ploss;          % Pérdidas

      % Flujo neto de caja anual caso principal
      N(cp)=(Ploss0(cp)-Ploss)*ha(cp)*1000*pKWh;
```

```
      if num_escenarios >1

         if  sum(npobp(k).crom(cp,:))~=0

          % Algoritmo Genético Básico sobre el resto de escenarios
       AGB

       else  % Los casos secundarios son los casos base
          for w=caso_ord(2:end)                    % Para los casos
secundarios:
                N(w)=0;                                  % Flujo de Caja
                npobp(k).x(w,:)=npobp(k).x(cp,:);       % Fenotipo
                npobp(k).crom(w,:)=npobp(k).crom(cp,:); % Cromosoma
                npobp(k).estado(:,2*w-1)=v(:,w);        % Tensiones
                npobp(k).estado(:,2*w)=d(:,w);             % Desfases
                npobp(k).loss(w)=Ploss0(w);                % Pérdidas
             end
          end

         % Compara los fenotipos de los distintos escenarios para determinar
         % qué condensadores son fijos y cuáles son variables
         for i=1:nc
            tipo_cond(i)=1; % fijo, en principio
            for w=1:(num_escenarios-1)
               if npobp(k).x(w,i)~=npobp(k).x(w+1,i)
                  tipo_cond(i)=2; % variable
                  break
               end
            end
         end

      end % fin de if num_escenarios>1

     % Desembolso(inversión)
     D=sum(npobp(k).x(cp,:).*pp.*tipo_cond)+sum(sign(npobp(k).x(cp,:)).*pf);

     % Cálculo del Valor Actual Neto
      npobp(k).van=VAN(D,sum(N),ta,vu);

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

    end % fin de if restric

    if cg==1 %Únicamente para la primera generación
      patroncrom=ones(lgen,nc);
    end

elseif cg==1 % Se evita generar un cromosoma entero de nuevo en la población
inicial
    patronx=zeros(1,nc);
    patronx(find(x>num_pasos))=1;
    patroncrom=repmat(patronx,lgen,1); %genes que deben modificarse
end % fin de if prod(x<=num_pasos)
```

8.1. DECOD.M

```
function y = decod(crom,lgen)
```

```
% DECOD decodifica un cromosoma.
%
% Y=DECOD(CROM,LGEN) devuelve el fenotipo codificado por CROM.
% CROM está constituido a su vez por subcadenas de longitud LGEN que codifican
% los valores(nº de pasos)de los NC bancos de condensadores,
% siendo NC el número de nudos candidatos.
% Llama a la función BINADEC.
% Véase COMPLETA_IND.

k=1;                %Inicia el contador de nudos
lcr=size(crom,2); %Longitud de la cadena cromosómica
nc=lcr/lgen;     %Número de nudos candidatos

%Para todos los bancos
for i=1:nc
   y(i)=binadec(crom((i-1)*lgen+1 : lgen*i)); %Agrupa los alelos de lgen en lgen
end
```

8.1.1. BINADEC.M

```
function y=binadec(bin)

% BINADEC convierte un vector fila binario en un escalar(número en base 10).
%
% Y=BINADEC(BIN) devuelve en y el equivalente en base 10 del vector BIN, que
representa a
% un número en base 2.
% Véase DECOD.

l=size(bin,2);          % Longitud del vector fila
y=0; pot2=1;         % Acumulador y potencia de dos.
for k=l:-1:1         % Recorre el vector de atrás a delante.
   if bin(k)
      y=y+pot2;
   end
   pot2=pot2*2;
end
```

8.2. MODY.M

```
function Y = modY(x,Ybus,NdC,Qip)

% MODY modifica la matriz de de admitancia de nudos incluyendo la   de los
condensadores.
%
% Y = MODY(X,YBUS,NDC,QIP) retorna la YBUS alterada por la distibución de
condensadores X
% en los nudos NDC. QIP contiene la potencia nominal(en pu) por paso para cada
nudo.
% Véase COMPLETA_IND.

j=sqrt(-1);
Y=Ybus;       % Inicia a la matriz del caso base
i=1;          % Contador de bancos

% Para todos los bancos de condensadores
for k=NdC
   Y(k,k) = Y(k,k) + j*Qip(i)*x(i);
```

```
   i=i+1;
end
```

8.3. RESTRICCIONES.M

```
% RESTRICCIONES script M-file devuelve un cero si no se verifican y un uno en
caso contrario.
%
% Se encarga de comprobar que la distribución generada cumple las
especificaciones relativas
% a las tensiones de los nudos y a los flujos de potencia por las líneas.
% Sus variables de entrada son los módulos y desfases de las tensiones obtenidos
del reparto
% de carga: VK y DK.
% W es el índice del escenario.
% Véase COMPLETA_IND, DATOS.

if err>ema       % No comprueba restricciones si el error del reparto de carga es
superior al
   restric=0;  % máximo admisible

else
j=sqrt(-1);

% Límites de tensión en los nudos
SL=find(tipo(:,w)==3);PV=find(tipo(:,w)==2); PQ=find(tipo(:,w)==1); % Tipos de
nudo
restric=prod(vk(PV)<=vlimPV(1)) & prod(vk(PV)>=vlimPV(2))...
   & prod(vk(PQ)<=vlimPQ(1)) & prod(vk(PQ)>=vlimPQ(2));

if restric
   % Límites de flujo de potencia por las líneas
   for i=1:size(BrD,1)
      p=BrD(i,1,w); q=BrD(i,2,w);% Nudos de salida y llegada
      if p==0
         break
      end
      Smax=BrD(i,9,w);                  % Flujo máximo
      ycp=BrD(i,10,w);ypq=BrD(i,12,w);  % Admitancias en el lado p(charging y
serie)
      ycq=BrD(i,11,w);yqp=BrD(i,13,w);  % Admitancias en el lado q(charging y
serie)
      V_p= vk(p)*exp(j*dk(p)*pi/180); V_q= vk(q)*exp(j*dk(q)*pi/180); % Tensiones
complejas
      S_p= abs(V_p*conj(V_p*ycp+(V_p - V_q)*ypq))*base(w);         % Potencias
aparentes(MVA)
      S_q= abs(V_q*conj(V_q*ycq+(V_q - V_p)*yqp))*base(w);

      if max([S_p S_q]) > Smax
         restric=0;
         break
      end

   end % for
end % if restric

clear SL PV PQ i p q Smax ycp ycq ypq yqp V_p V_q S_p S_q

end %else
```

```
restric
```

8.4. AGB.M

```
% AGB script M-file algoritmo genético básico sobre todos los escenarios
secundarios.
%
% Para cada distribución principal obtenida en el algoritmo externo, encuentra
% las distribuciones derivadas casi-óptimas  en el resto de los escenarios.
% Con pequeñas diferencias, reproduce la estructura del algoritmo principal, con
el que
% comparte alguno de sus módulos y funciones.
% Véase COMPLETA_IND.

for w=caso_ord(2:end)   % Para todos los casos secundarios
AGB_inicio                 % Inicia variables
AGB_pob_inicial         % Genera una población inicial aleatoria
AGB_evolucion           % Crea sucesivas poblaciones empleando operadores
genéticos
AGB_resultados             % Muestra resultados
end
```

8.5. VAN.M

```
function y=VAN(D,N,i,t)

% VAN calcula el valor actual neto.
%
% Y=VAN(D,N,i,t) devuelve en Y el V.A.N. calculado a partir de:
%      D desembolso inicial en €, N flujo de caja neto anual en €, i rendimiento
del capital,
%      t vida útil de la inversión en años.
%
% N es el producto del decremento de las pérdidas eléctricas, en KWh/año, por
% el precio medio, en €, del Kilovatio-hora en el mercado eléctrico.
% Se supone que no hay costes de mantenimiento y que N es contante a lo largo de
los años.
% Para sumar unidades homogéneas se debe actualizar N del año k dividiendo por
% (1+i)^k.
%
% VAN = -D + N/(1+i) + N/(1+i)^2 + ... + N/(1+i)^t
%
% Véase COMPLETA_IND.

r=1/(1+i);
y= -D + N*(r-r^(t+1))/(1-r);
```

9. SUBPOBLACIONES.M

```
% SUBPOBLACIONES script M-file encuentra los NP peores y los NM mejores de NPOBP.
%
% Crea dos subpoblaciones: MEJORES y PEORES.
% Véase POB_INICIAL, EVOLUCION.
```

```
[van_ord,i_ord]=sort(cat(2,npobp.van));    % Ordena de menor a mayor bondad
i_peores=i_ord(1:np);                      % Lugar que ocupan los peores en la
población
peores=npobp(i_peores);                    % Subpoblación de peores

i_ord=i_ord([num_ind:-1:1]);              % Ordena de mayor a menor bondad
i_mejores=i_ord(1:nm);                     % Lugar que ocupan los mejores en la
población
mejores=npobp(i_mejores);                  % Subpoblación de mejores

clear van_ord i_ord
```

10.DIVERSIDAD.M

```
% DIVERSIDAD script M-file calcula el indicador DIV de diversidad.
%
% Calcula el porcentaje de alelos distintos a los del cromosoma correspondiente
al
% mejor individuo hasta la fecha.
% Véase POB_INICIAL, EVOLUCION.

A=[];
for i=1:num_ind
   A=[A; npobp(i).crom(cp,:)];
end
div=0;
for i=1:lcr
   for z=1:num_ind
      div=div+xor(mejores(1).crom(cp,i),A(z,i));
   end
end
div=100*div/(num_ind*lcr);
```

11.INFOR_GENERACION.M

```
% INFOR_GENERACION script M-file vuelca en pantalla información sobre la
generación obtenida.
%
% La información se presenta tabulada y se complementa con un diagrama de barras
que muestra
% la bondad de cada individuo de la población.

disp('_____________________________________________________________________________'
)
disp(' ')
disp('              ALGORITMO GENÉTICO PRINCIPAL                  ')
disp('_____________________________________________________________________________'
)
disp(' ')
disp(['   ESCENARIO PRINCIPAL... ', casos_base(cp,:)])
disp('_____________________________________________________________________________'
)
disp(' ')
fprintf('  Individuo nº:      Fenotipo:       VAN:     Pérdidas (en %s:\n');
for h=caso_ord
   fprintf('%s ',casos_base(h,:));
end
```

```
fprintf(' ):\n\n');
for i=1:num_ind
   fprintf('    (%4g)',i);
   fprintf('%20s' ,num2str( npobp(i).x(cp,:)));
   fprintf('%10.0f euros',npobp(i).van);
   for h=caso_ord
      fprintf('  %4.3f MW',npobp(i).loss(h));
   end
   fprintf('\n');
end
disp('_____________________________________________________________________________'
)
fprintf('\n\n');
disp(['    Mejor distribución encontrada en la ', num2str(cg),'ª generación:'])
fprintf('\n');
disp(['    Individuo nº... ',num2str(i_mejores(1))])
fprintf('\n');
fprintf('    Escenario:          Fenotipo:              Pérdidas:\n\n');
for h=caso_ord
   fprintf('    %9s  %20s              %4.3f MW\n',casos_base(h,:),...
      num2str(mejores(1).x(h,:)),mejores(1).loss(h));
end
fprintf('\n');

disp(['    VAN... ', num2str(mejores(1).van),'euros.'])
disp('_____________________________________________________________________________'
)
disp(' ')
disp(['    Diversidad... ', num2str(div),' %'])
disp('_____________________________________________________________________________'
)
bar(cat(2,npobp.van))
title({[num2str(cg),'ª generación(población principal)'],['Escenario:
',casos_base(cp,:)]})
xlabel('Individuo nº')
ylabel('VAN')

% Convertir en instrucción la línea siguiente para visualizar el informe
% keyboard
% Para continuar la ejecución del programa teclear en el prompt 'return' y
después RETURN

fprintf('Break point')
```

12.UOC.M

```
function y=uoc(p)

% UOC genera una variable aleatoria de Bernouilli.
%
% Y=UOC(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
```

13.AGB_INICIO

```
%AGB_INICIO script M-file inicia las variables del AGB.
%
% Es básicamente igual que el módulo INICIO, salvo que trabaja con estructuras
más
% simples y un fenotipo reducido: se eliminan aquellas posiciones a las que
corresponda
% un valor nulo de potencia reactiva instalada en la distribución principal.
%
% Véase AGB.

Xmax=npobp(k).x(cp,:); % Fenotipo de la distribución principal, Xmax
pos_nsc=find(Xmax==0); % Posiciones asociadas a nudos SIN condensadores en Xmax
pos_ncc=find(Xmax~=0); % Posiciones asociadas a nudos CON condensadores en Xmax
NdCs=NdC(pos_ncc);     % Nudos considerados candidatos para los casos secundarios
Xmaxr=Xmax(pos_ncc);   % Fenotipo reducido
ncs=size(Xmaxr,2);    % Número de nudos considerados candidatos para los casos
secundarios

% Determina la longitud del cromosoma(número de alelos)
lcrs= lgen*ncs; %tantos bancos posibles como nudos candidatos

% Define la estructura "individuo"
ind = struct('crom',repmat(0,1,lcrs),'x',repmat(0,1,ncs),'estado',...
   repmat(0,n(w),2),'bon',0,'loss',0);

% Define dos vectores de individuos
global pobs npobs
pobs = repmat(ind,1,num_ind);%el de la población "padre"
npobs = pobs;%el de la población "hija"

% Define dos vectores que registren los nm mejores y los np peores
% individuos en cada generación
mejoress = repmat(ind,1,nm);
peoress= repmat(ind,1,np);

% Define dos vectores que registren el mejor y el peor individuo de cada
generación
mejor_inds = repmat(ind,1,num_gen);
peor_inds= mejor_inds;
for ks=1:num_gen
   peor_inds(ks).bon=Inf;
end

% Contador de generaciones
cgs=1;

% Define vector suma de bondades
sumas=zeros(1,num_gen);

% Número de mutaciones y no convergencias del N-R
global NMUTs NOCONVs
NMUTs=0;NOCONVs=0;

% Potencia y precio por paso para cada nivel de tensión
Qips=Qip(pos_ncc);
```

14.AGB_POB_INICIAL

```
% AGB_POB_INICIAL scrip M-file genera una población inicial aleatoriamente para
el AGB.
%
% Véase AGB.

% Inicia contador de individuos a 1
ks = 1;
clear croms xs
patroncrom=ones(lgen,ncs);
escs=(num_gen+cgs)/(1.5*num_gen); % factor de escala
marca=0; % Se pone a uno cuando el bucle while a hecho una iteración

% Mientras no se alcance el límite poblacional, haz:
while ks <= num_ind
   % Genera el cromosoma de un individuo
   if prod(patroncrom)==1
      contador=0;prob=0.5;
   else
      cotador=contador+1;
   end
   %Introduce en la población inicial parte(o todo, si es posible)
   if marca==0    % de la distrib. secundaria asociada
    croms= cod(mejor_ind(cg-(cg~=1)).x(w,pos_ncc),lgen); % a la mejor distrib.
principal
    else % Genera aleatoriamente el cromosoma
    for i = 1 : lcrs
        if patroncrom(i)==1
            croms(i) = uoc(prob);
        end
       end
   end
   % Completa individuo
   AGB_completa_ind
   marca=1;
end

% Guarda en 'sumas' el sumatorio de las bondades de todos
% los individuos de esta primera población
sumas(cgs) = sum(cat(2,npobs.bon));
% Suma escalada
sumas_esc= sum(cat(2,npobs.bon).^escs);

% Crea las subpoblaciones  'mejores' y 'peores'
AGB_subpoblaciones

% Registra el mejor y el peor individuo de la presente generación
mejor_inds(cgs) = mejoress(1);
peor_inds(cgs)= peoress(1);

% Calcula indicador de diversidad
AGB_diversidad

% Informe en pantalla
AGB_infor_generacion

% La población hija es la población padre para la siguiente generación
pobs=npobs;

%Incrementa contador de generaciones
cgs=cgs+1;
```

15.AGB_EVOLUCION

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

% Los operadores genéticos son SELECCION, CRUCE Y MUTACIÓN.
% Se llega a una generación final dada por num_gen y se espera que ésta contenga
un individuo
% que proporcione un valor casi-óptimo.
%
% Véase AGB.

num_ind_a_generar=num_ind;
for cgs=2:num_gen % Para todas las generaciones que siguen a la primera, haz:

    ks=1; % Inicia contador de individuos
   while 1 % Hasta alcanzar el tamaño cte de la población, haz:

      % Selección de una pareja reproductora: pad1 y pad2

      pad1=selec(sign(cat(2, pobs.bon)).*abs(cat(2, pobs.bon)).^escs,sumas_esc) %
Padre 1
      pad2=selec(sign(cat(2, pobs.bon)).*abs(cat(2, pobs.bon)).^escs,sumas_esc) %
Padre 2

      %Cruce y mutación: Se generan los cromosomas de dos candidatos para
      % formar parte de la población hija
      c=repmat(0,2,lcrs);
      [c(1,:),c(2,:)]= cruce(pobs(pad1).crom,pobs(pad2).crom,pcr,pmu);

      for i=1:2
         croms=c(i,:)
         AGB_completa_ind
       if ks> num_ind_a_generar
           break % Pasa a otra generación si se alcanza el límite poblacional
         end
      end %for

      if ks> num_ind_a_generar
         break % Pasa a otra generación si se alcanza el límite poblacional
      end

   end % while 1

   % Crea las subpoblaciones  'mejoress' y 'peoress'
   AGB_subpoblaciones

   % Registra el mejor y el peor individuo de la presente generación
   mejor_inds(cgs) = mejoress(1);
   peor_inds(cgs)= peoress(1);

  % Conservación del mejor individuo generación tras generación
  if mejor_inds(cgs-1).bon > mejor_inds(cgs).bon
     % Reemplaza el peor de la hija por el mejor de la generación padre
     npobs(i_peoress(1))=mejor_inds(cgs-1);
     % Actualiza subpoblaciones
     AGB_subpoblaciones
     % Actualiza el mejor y el peor individuo de la presente generación
      mejor_inds(cgs) = mejoress(1);
     peor_inds(cgs)= peoress(1);
  end

  % Guarda en 'sumas' el sumatorio de las bondades de todos
   % los individuos de esta primera población
```

```
   sumas(cgs) = sum(cat(2,npobs.bon));

   % Escalado: bondad^esc
   if cgs<(num_gen/2)
      escs=(num_gen+cgs)/(1.5*num_gen); % Factor de escala
   else
      escs=(num_gen+0.5*cgs)/num_gen;   % Factor de escala
   end
   sumas_esc= sum(cat(2,npobs.bon).^escs);

   % Calcula indicador de diversidad
    AGB_diversidad

   % Informe en pantalla
    AGB_infor_generacion

   % Si cae la diversidad, finaliza
   if div<umbral_div
      break
   end

   % La población hija es la población padre para la siguiente generación
   pobs=npobs;

   % Elitismo: Los mejores individuos obtenidos pasan directamente a la
   % siguiente generación
   if elite == 1
      num_ind_a_generar=num_ind-nm;
      npobs((num_ind_a_generar+1):end)= mejoress;
   end
end
```

16.AGB_RESULTADOS

```
% AGB_RESULTADOS script M-file que muestra la evolución de la población
% a lo largo de sucesivas generaciones.
%
% Véase AGB.

% Expande e incluye al mejor derivado encontrado en la estructura individuo de la
% población principal
npobp(k).x(w,pos_ncc)=mejoress(1).x;              % Fenotipo
npobp(k).x(w,pos_nsc)=0;                             %      expandido
npobp(k).crom(w,:)=cod(npobp(k).x(w,:),lgen);     % Cromosoma expandido
npobp(k).estado(:,2*w-1)=mejoress(1).estado(:,1);  % Tensiones
npobp(k).estado(:,2*w)=mejoress(1).estado(:,2);      % Desfases
npobp(k).loss(w)=mejoress(1).loss;                   % Pérdidas

% Flujo neto de caja anual caso secundario
N(w)=mejoress(1).bon;


% Visualización en pantalla:
fprintf('\n_____________________________________________________________________
___\n\n')
fprintf('        RESUMEN DE LA EVOLUCION DEL ALGORITMO GENÉTICO BÁSICO (AGB)\n');
fprintf('\n_____________________________________________________________________
___\n\n')
fprintf('\n');
disp(['   MEJOR DISTRIBUCIÓN DERIVADA DE Xmax=',num2str(Xmax)]);
```

```
disp(['   (Individuo', num2str(k),'º DE LA ',num2str(cg),'ªgeneración del AGP)'])
fprintf('\n');
disp(['   OBTENIDA EN EL ESCENARIO SECUNDARIO: ', num2str(casos_base(w,:))]);
fprintf('\n\n  Nudo         Número de escalones\n');
fprintf('  ____________________________\n\n');
for p=1:nc
   fprintf('%6i          %6i\n',NdC(p),npobp(k).x(w,p));
end
fprintf('  ____________________________\n\n');

disp(['     Flujo de caja... ',num2str(N(w)),'euros.     Pérdidas... ',...
      num2str(npobp(k).loss(w)),'MW'])
fprintf('\n_____________________________________________________________________
___\n\n')
fprintf('  Tras %u generaciones:\n',cgs);
fprintf('   * El número de mutaciones ha sido %u\n',NMUTs);
fprintf('   * El número de ''no convergencias'' ha sido %u\n',NOCONVs);
fprintf('   * Diversidad Final: %2.2f %%\n', div);
ejex=1:cgs;
plot(ejex,cat(2,mejor_inds(1:cgs).bon),'rx-
',ejex,cat(2,peor_inds(1:cgs).bon),'bx-',ejex,...
   sumas(1:cgs)/num_ind,'k:');
title('Evolución del valor del Flujo de Caja a lo largo de sucesivas
generaciones')
xlabel('Generaciones')
ylabel('Flujo de Caja(euros)')

% Convertir en instrucción la línea siguiente para visualizar el informe
% keyboard
% Para continuar la ejecución del programa teclear en el prompt 'return' y
después RETURN

fprintf('Break point')
```

16.1. COD.M

```
function y = cod(x,lgen)

% COD codifica el fenotipo X.

% Y=COD(X,LGEN) devuelve el cromosoma que codifica X.
% CROM está constituido a su vez por subcadenas de longitud LGEN que codifican
% los valores(nº de pasos)de los NC bancos de condensadores,
% siendo NC el número de nudos candidatos.
% Llama a la función DECABIN.
% Véase AGB_RESULTADOS.

k=1;                              % Inicia el contador de nudos
nc=size(x,2);                   % Número de bancos
lcr=nc*lgen;                    % Longitud de la cadena cromosómica
y=[];
for i=1:nc                        % Para todos los bancos
   y=[y decabin(x(i),lgen)];  % Construye el cromosoma gen a gen
end
```

16.1.1. DECABIN.M

```
function y=decabin(dec,lcr)
```

```
% DECABIN script M-file convierte un número en base 10 en un vector binario.
%
% Y=DECABIN(DEC,LCR) devuelve en Y la representación binaria de DEC en un vector
de
% longitud LCR.

bin=[];

while 1
   bin=[bin mod(dec,2)];
   dec=floor(dec/2);
   if dec==0
      break
   end
end
y=[bin(end:-1:1)];
nbits=size(y,2);
if nbits<lcr
   y=[repmat(0,1,lcr-nbits) y];
end

if nbits>lcr
   error('La longitud de la cadena binaria es insuficiente para codificar el
número decimal');
end
```

17.AGB_COMPLETA_IND

```
% AGB_COMPLETA_IND script M-file completa la estructura tipo individuo para el
AGB.
%
% Véase AGB_POB_INICIAL, AGB_EVOLUCION.

% A partir del cromosoma(genotipo), genera el fenotipo:
xs = decod(croms,lgen)

% Si pertenece al espacio de soluciones admisibles, haz:
if prod(xs <= Xmaxr)
   % Modifica la matriz de admitancias de nudos
   Y= modY(xs,Ybus(:,:,w),NdCs,Qips/base(w));
   Y=sparse(Y);

   % Resuelve el reparto de carga para este valor de x:
   [vk,dk,Ploss,err]= reparto_carga(base(w),...
      n(w),tipo(:,w),nivelv(:,w),v(:,w),d(:,w),Pd(:,w),...
      Qd(:,w),Pg(:,w),Qg(:,w),Qmin(:,w),Qmax(:,w),vconsig(:,w),TAD(:,:,w),...
      Y,ema,BrD(:,:,w),SHUNT(:,:,w));

   % Comprueba cumplimiento de las restricciones(tensiones y potencias)

   restricciones % Devuelve restric=1 si se verifican

   if restric
      npobs(ks).crom=croms;       % Cromosoma
      npobs(ks).x=xs;             % Fenotipo
    npobs(ks).estado(:,1)=vk;  % Tensiones
      npobs(ks).estado(:,2)=dk;   % Desfases
```

```
      npobs(ks).loss=Ploss;      % Pérdidas

      % Flujo neto de caja anual caso secundario
      npobs(ks).bon=(Ploss0(w)-Ploss)*ha(w)*1000*pKWh;

       % Aumenta en uno el contador de individuos:
      ks = ks+1;
   end

   if cgs==1 %Únicamente para la primera generación
      patroncrom=ones(lgen,ncs);
   end


elseif cgs==1 % Se evita generar un cromosoma entero de nuevo en la población
inicial

    patronx=zeros(1,ncs);
    patronx(find(xs>Xmaxr))=1;
    patroncrom=repmat(patronx,lgen,1); %genes que deben modificarse
     if contador>2*lcrs
        prob=min(patronx.*Xmaxr)/num_pasos;
     end
end
```

18.AGB_SUBPOBLACIONES

```
% AGB_SUBPOBLACIONES script M-file las subpoblaciones  'mejoress' y 'peoress'
%
% Véase  AGB_POB_INICIAL,AGB_EVOLUCION.

[van_ord,i_ord]=sort(cat(2,npobs.bon));   % Ordena de menor a mayor bondad
i_peoress=i_ord(1:np);                    % Lugar que ocupan los peores en la
población
peoress=npobs(i_peoress);                 % Subpoblación de peores

i_ord=i_ord([num_ind:-1:1]);             % Ordena de mayor a menor bondad
i_mejoress=i_ord(1:nm);                  % Lugar que ocupan los mejores en la
población
mejoress=npobs(i_mejoress);              % Subpoblación de mejores

clear van_ord i_ord
```

19.AGB_DIVERSIDAD

```
% AGB_DIVERSIDAD script M-file análogo a DIVERSIDAD.
%
% Véase AGB_POB_INICIAL, AGB_EVOLUCION.


A=[];
for i=1:num_ind
   A=[A; npobs(i).crom];
```

```
end
div=0;
for i=1:lcrs
   for z=1:num_ind
      div=div+xor(mejoress(1).crom(i),A(z,i));
   end
end
div=100*div/(num_ind*lcrs);
```

20.AGB_INFOR_GENERACION

```
% AGB_INFOR_GENERACION script M-file análogo a INFOR_GENERACION
%
% Vuelca en pantalla información sobre la generación obtenida.

fprintf('_____________________________________________________________________________
_\n\n');
disp('                        ALGORITMO GENÉTICO BÁSICO (AGB)')
disp('_____________________________________________________________________________')
fprintf('\n');
disp(['   ESCENARIO SECUNDARIO... ', casos_base(w,:)])
disp('_____________________________________________________________________________')
fprintf('\n   Individuo nº:  Fenotipo reducido:   Pérdidas:   Flujo de caja:\n');
for i=1:num_ind
   fprintf('   (%4g)   %20s       %4.3f MW %10.0f
euros\n',i,num2str(npobs(i).x),...
      npobs(i).loss,npobs(i).bon);
end
disp('_____________________________________________________________________________')
fprintf('\n');
disp(['    ESCENARIO SECUNDARIO... ', casos_base(w,:)])
disp(['    AGB: Mejor Distribución Derivada  de Xmax=
',num2str(npobp(k).x(cp,:))])
disp(['    en la ', num2str(cgs),'ª generación:'])
fprintf('\n');
disp(['      Individuo nº... ',num2str(i_mejoress(1))])
fprintf('\n');
disp(['      Fenotipo reducido... ',num2str(mejoress(1).x)])
fprintf('\n');
disp(['      Flujo de caja... ', num2str(mejoress(1).bon),'euros'])
fprintf('\n');
disp(['      Pérdidas... ',num2str(mejoress(1).loss),'MW'])
disp('_____________________________________________________________________________')
fprintf('\n')
disp(['      Diversidad... ', num2str(div),' %'])
disp('_____________________________________________________________________________')
bar(cat(2,npobs.bon))
title({[num2str(cgs),'ª generación de distribuciones derivadas de Xmax= ',...
         num2str(npobp(k).x(cp,:))],['Escenario: ',casos_base(w,:)]})
xlabel('Individuo nº')
ylabel('Flujo de Caja')

% Convertir en instrucción la línea siguiente para visualizar el informe
% keyboard
% Para continuar la ejecución del programa teclear en el prompt 'return' y
después RETURN
fprintf('Break point')
```