ANEXO

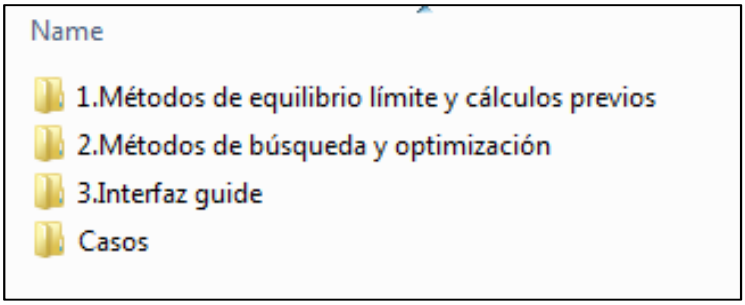
Este anexo contiene los pasos para comenzar a usar el programa desarrrollado y por otra parte contiene el código fuente de todas las funciones esenciales de dicho programa.

Tabla de contenido del anexo

1 INSTALACIÓN DEL PROGRAMA E INTERFAZ

Todas las funciones que se han desarrollado en el programa vienen incluidas en la carpeta *TaludMet Ulpgc,* incluidas en el CD entregado. Dentro de esta carpeta se pueden observar las siguientes subcarpetas:

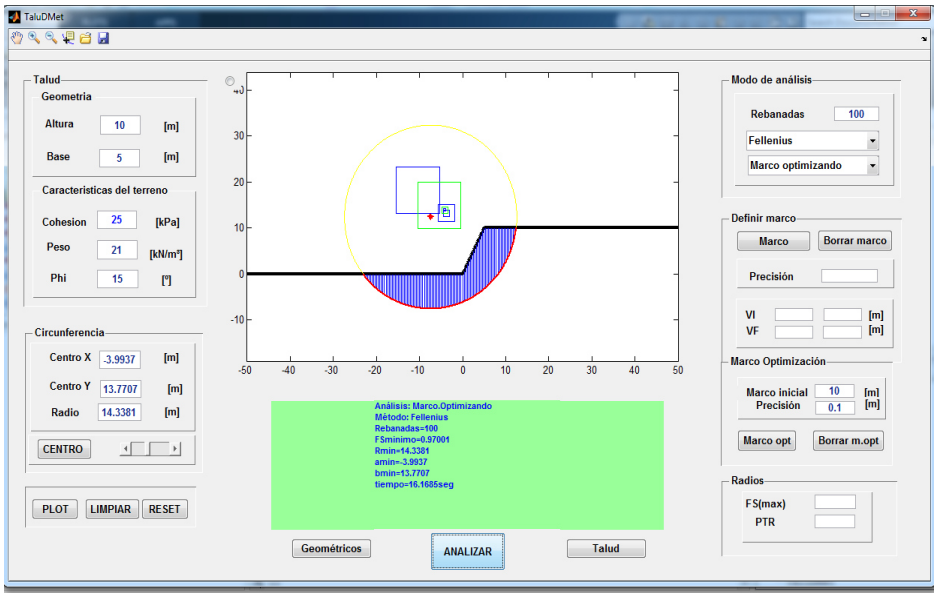
ILUSTRACIÓN I



En la imagen se observan las carpetas en las que se ha subdivido el programa. El código fuente de todas las funciones que se encuentran en las carpetas uno y dos se muestra en este anexo. La carpeta número tres contiene las funciones que se desarrollaron para implementar la interfaz de usuario. La carpeta titulada *Casos* contiene siete taludes de referencia, que son los que han sido usados en las simulaciones en la memoria técnica.

Por otro lado, para poder usar el programa, será necesario tener instalado Matlab. Se tendrán que introducir dentro del *path* de Matlab todos los archivos mostrados en la ilustración I. Una vez están todos los archivos incluidos en el path de Matlab, se deberá teclear en el *Command Window* de Matlab la palabra *TaludMet.* Después de esto se abrirá la interfaz de usuario, mostrado en la ilustración II.

ILUSTRACIÓN II



En la Ilustración III se puede observar el aspecto de esta interfaz. Esta interfaz contiene una serie de paneles en la que se introducen los datos necesarios para realizar los métodos explicados en la memoria técnica del proyecto. Además se pueden cargar los taludes que están en la carpeta *Casos.*

Por último cabe decir, que todas las simulaciones realizadas en la memoria se pueden llevar a cabo con facilidad con esta aplicación, ya que esta misma fue desarrollada para servir en la redacción del proyecto.

2 CÓDIGO FUENTE

A continuación se muestra el código fuente de las funciones y procedimientos que se encuentran en la carpeta número uno y dos de la ilustración I.

2.1 MÉTODOS DE EQUILIBRIO LÍMITE Y CÁLCULOS PREVIOS

2.1.1 CÁLCULO DE LA SUPERFICIE

2.1.1.1 FUNCIÓN TALUDGEOMETRIA

```
        function    y_talud = taludgeometria( B,x,H )
%  Devuelve la superficie del talud
% Inputs
% B (m):Base del talud
% H (m):Altura del talud
% x:Puntos del eje x sobre los que se quiere saber su ordenada en
% en el eje y
% Outputs
% y_talud: Ordenada de los puntos x sobre la superficie del talud
m=H/B; % Se calcula pendiente de la recta inclinada
y1=m.*x;
y2=0; % El talud siempre arranca en y=0;x=0.
y3=H;
y_talud=y2.*(x<=0)+y1.*((x>0)&(x<B))+y3.*(x>=B);
end
```

2.1.1.2 FUNCIÓN RAICIRCUNFERECTA

```
        function [ X ] = raicircunferecta( a,b,m,n,R )
% Calcula los puntos de corte entre una circunferencia y una y una recta
% Devuelve las soluciones en la variable X
% ordenados de menor a mayor.
% Inputs
% a=Desplazamiento del centro sobre el eje x
% b=Desplazamiento del centro sobre el eje y
% m=Pendiente de la recta
% n=Ordenada en el origen
% R=Radio de la circunferencia
% Outputs
% X=puntos de corte de la circunferencia ordenados de menor a mayor.
A=m^2+1;
B=2*m*(n-b)-2*a;
C=(a^2)-(R^2)+(n-b)^2;
coeficientes=[A B C];
```

```
[X] = segundogrado( coeficientes);
end
```

2.1.1.3 FUNCIÓN PUNTOSPOSIBLES

```
       function [puntosposibles] = intersectaludcirc( a,b,H,R,B )
% Esta función calcula la matriz puntos posibles.
% Inputs
%   a,b,R : Centro y radio de la circunferencia
%   H,B: Altura y base del talud
% Outputs
% puntosposibles: Matriz que contiene las raíces de los posibles
% extremos del talud

% Recta k1. y=m*x , n=0
m1=H/B;
n1=0;
    %raices
    [ X1 ] = raicircunferecta( a,b,m1,n1,R );
    k1x1=X1(1);k1x2=X1(2);
%     Eliminamos casos de tangencia y complejos
    if  k1x1==k1x2 || imag(k1x1)~=0
    k1x1=NaN;k1y1=NaN;
    k1x2=NaN;k1y2=NaN;
    else
    %imagen
    k1y1=m1*k1x1;k1y2=m1*k1x2;
    end
    %puntos
    k1p1=[k1x1 k1y1];k1p2=[k1x2 k1y2];
%Recta k2. y=0 para todo x
m2=0;
n2=0;
    %raices
    [ X2 ] = raicircunferecta( a,b,m2,n2,R);
    k2x1=X2(1);k2x2=X2(2);
    %     Eliminamos casos de tangencia y complejos
    if k2x1==k2x2 || imag(k2x1)~=0
        k2x1=NaN;k2y1=NaN;
    else
    %Imagen
    k2y1=0;k2y2=0;
    end
    %Puntos
    k2p1=[k2x1 k2y1];

%Recta k3. y=H
m3=0;
n3=H;
    %raices
    [ X3 ] = raicircunferecta( a,b,m3,n3,R );
    k3x1=X3(1);k3x2=X3(2);
%     Eliminamos casos de tangencia y complejos
    if k3x1==k3x2 || imag(k2x1)~=0
    k3x2=NaN;k3y2=NaN;
```

```
    else
    %imagen
    k3y1=H;k3y2=H;
    end
     %Puntos
    k3p2=[k3x2 k3y2];
% Se agrupan en la matriz puntosposibles los puntos que pertenecen
% al conjunto de posibles soluciones
puntosposibles=[k1p1; k1p2;k2p1; k3p2];

% Eliminamos la posibilidad de que dos puntos sean iguales
if  round(puntosposibles(1,:)*100000)/100000==round(puntosposibles(3,:)*100000)/100000
%R==Dv1 % % puntosposibles(3,:)==puntosposibles(1,:)
    puntosposibles(3,:)=nan;
    puntosposibles(1,:)=[0 0];
end

if  round(puntosposibles(2,:)*100000)/100000==round(puntosposibles(4,:)*100000)/100000
%R==Dv2 % puntosposibles(2,:)==puntosposibles(4,:)
    puntosposibles(4,:)=nan;
    puntosposibles(2,:)=[B H];
end

end
```

2.1.1.4 FUNCIÓN PUNTOSPOSIBLES

```
       function [ xti,xtd,k4,r] = calculoextremos(a,b,B,H,R)
% Esta función finaliza con el procedimiento de búsqueda de extremos
% del talud
% Inputs
% Mismos inputs que funcion intersectalud
% Outputs
% xti,xtd : Raíces de los extremos del talud
% k4 : Informa si la superficie de deslizamiento es válida
% r: Informa de las rectas con las que corta la superficie de deslizamiento
% B=H/tand(beta); % Base del talud
[puntosposibles] = intersectaludcirc( a,b,H,R,B );            % Puntos de corte
circunferencia-talud
% Corte con la recta k1. Si no corta con k1 la solucion no es valida.k4=0
k1x1=puntosposibles(1,1);
Q1=~isnan(k1x1);
if Q1==1
 %   Eliminamos las ordenadas mayores a b (centro de la circunferencia)
        ordmayqueb= puntosposibles(:,2)>b;
        puntosposibles(ordmayqueb,:)=nan;
% Comprobamos que los valores de las ordenadas de las rectas coinciden
% con los valores del talud
        y_t =taludgeometria( B,puntosposibles(:,1),H );
        d_t=find(y_t==puntosposibles(:,2));
        % Minimo dos puntos en comun
        if length(d_t)<2
```

```
        k4=0;r=[0 0 0];xti=NaN;xtd=NaN;
        else
%       Extraemos de la matriz puntosposibles la solucion
        puntosextremos=puntosposibles(d_t,:); %pares extremos
        raicesextremas=puntosextremos(:,1); %extraemos las raices extremas y ordenamos
        raicesextremas=sort(raicesextremas);
%       Eliminamos la situacion "doble entrada" escogiendo el ultimo y
%       el penultimo valor despues de ordenar
        xti=raicesextremas(end-1);
        xtd=raicesextremas(end);
%       Buscamos en la matriz puntosextremos la imagen de xti,xtd
        reunimos1=find(xti==puntosextremos(:,1));
        reunimos2=find(xtd==puntosextremos(:,1));
        yti=puntosextremos(reunimos1(1),2);
        ytd=puntosextremos(reunimos2(1),2);
%       Eliminamos los casos limite en los que k1x1=k2x1 etc. Un talud
%       formado en la misma altura.
            if yti==ytd;
                k4=0;r=[0 0 0];xti=NaN;xtd=NaN;
            else
            k4=1;
            r=[ ((xti>=0 & xti<=B )|(xtd>=0 & xtd<=B))  xti<0   xtd>B];
            end
        end
else
k4=0;xti=NaN;xtd=NaN;r=[0 0 0];
end
end
```

2.1.2 REBANADAS

2.1.2.1 FUNCIÓN DIVISORDEREBANADAS

```
       function [ x,x_medio,pasovector ] = divisorderebanadas( xti,xtd,rebanadas,r,B)
% Esta función divide la superficie de deslizamiento en rebanadas
% Inputs
% xti,xtd: Abscisas de los puntos de corte de la superficie deslizante
% con el talud
% rebanadas: Número de rebanadas en que se quiere dividir la superficie
% r: Vector que informa de las rectas sobre las que se encuentran los
% puntos de corte pti,ptd
% B: Base del talud
% Outputs:
% x: Vector que contiene la abscisa de los puntos sobre los que se dibujan
% las rebanadas.
% x_medio: Vector que contiene la abscisa del punto medio de cada rebanada
% pasovector: Vector que contiene el ancho de cada rebanada.
%
%

% % Subproceso repartoderebanadas
% Definimos el caso en el que estamos
```

```
    if isequal(r,[0 1 1])
        l1=xti;l2=0;l3=B;l4=xtd;
    elseif isequal(r,[1 1 0])
        l1=xti;l2=0;l3=xtd;l4=[];
    elseif  isequal(r,[1 0 1])
        l1=xti;l2=B; l3=xtd;l4=[];
    elseif isequal(r,[ 1 0 0])
        l1=xti;l2=xtd; l3=[];l4=[];
    end
%
%     Distancias y proporciones por tramo
    distanciatotal=xtd-xti;
    dist1=l2-l1;
    dist2=l3-l2;
    dist3=l4-l3;
%     Proporciones por tramo
    prop1=dist1/distanciatotal;
    prop2=dist2/distanciatotal;
    prop3=dist3/distanciatotal;
%     Rebanadas enteras por tramo
    reb1=round(prop1*rebanadas);
    reb2=round(prop2*rebanadas);
    reb3=round(prop3*rebanadas);

%    Tramos no representados
   if reb1==0;
       reb1=reb1+1;
   end;
   if reb2==0;
       reb2=reb2+1;
   end;
   if reb3==0;
       reb3=reb3+1;
   end

%   Generamos el vector reb
    reb=[reb1 reb2 reb3];
    dimensionreb=length(reb);

%   Reajuste de rebanadas
    while rebanadas < sum(reb)
        [rebmayor,pos]=max(reb); %#ok<*ASGLU>
        reb(pos)=reb(pos)-1;
    end

    while rebanadas > sum(reb)
        [rebmayor,pos]=max(reb);
        reb(pos)=reb(pos)+1;
    end


    %Se vuelven a construir las variables reb1, reb2 reb3
    if dimensionreb==1

            reb1=reb(1);
            reb2=[];
```

```
            reb3=[];

    elseif dimensionreb==2
            reb1=reb(1);
            reb2=reb(2);
            reb3=[];

    elseif dimensionreb==3
            reb1=reb(1);
            reb2=reb(2);
            reb3=reb(3);
    end
% %
% % Subproceso vectoresrebanada.
% Calculo del vector x, x_medio, pasovector
%   Numero de total de puntos del vector x:x1,x2,x3
    Mtotal=rebanadas+1; %#ok<NASGU>
    M1=reb1+1;
    M2=reb2+1;
    M3=reb3+1;
    % Tramo 1
    x1=linspace(l1,l2,M1); % vector x1
    paso1=x1(2)-x1(1);     % Paso1
    x1_medio=x1+paso1/2;x1_medio(end)=[]; % vector x1 medios
    paso1vector=paso1*ones(1,reb1); % Vector de pasos por rebanada
    % Tramo 2
    if length(dist2)==1
        x2=linspace(l2,l3,M2); % vector x2
        paso2=x2(2)-x2(1);     % Paso2
        x2_medio=x2+paso2/2;x2_medio(end)=[];  % vector x2 medios
        paso2vector=paso2*ones(1,reb2); % Vector de pasos por rebanada
    else
        x2=[];paso2vector=[];x2_medio=[];
    end
    % Tramo 3
    if length(dist3)==1
        x3=linspace(l3,l4,M3); % Puntos x3
        paso3=x3(2)-x3(1);     % Paso3
        x3_medio=x3+paso3/2;x3_medio(end)=[];% Puntos x3 medios
        paso3vector=paso3*ones(1,reb3);% Vector de pasos por rebanada
    else
        x3=[];paso3vector=[];x3_medio=[];
    end
    %     Juntamos todos los resultados
    pasovector=[paso1vector paso2vector paso3vector];%Pasos vector
    x_medio=[x1_medio x2_medio x3_medio];% Vector x medio
    x= [x1 x2 x3 ];%Vector x. Este vector tiene valores repetidos

          % Eliminamos los valores repetidos de x.
           countrep=0;
           repetidos=0;
           for s=2:1:length(x)
               if x(s)== x(s-1)
                   countrep=countrep+1;
                   repetidos(countrep)=s; %#ok<*AGROW>
               end
```

```
            end

            if repetidos~=0
            x(repetidos)=[];
            end
end
```

2.1.2.2 FUNCIÓN PARAMETROS

```
       function [ x,x_medio,y_talud_med ,y_circ_med,alfa,pasovector,y_talud,y_circ ] =
parametros( rebanadas,xti,xtd,B,H,a,b,R,r)
% Esta función calcula valores asociados a las rebanadas
% Inputs
% rebanadas: Número de rebanadas
% xti,xtd: Puntos de corte entre la superficie de rotura y el talud
% B,H: Base y altura del talud
% a,b,R: Centro y radio de la circunferencia
% r: Vector que informa de las rectas sobre las que se encuentran los
% puntos de corte pti,ptd

% Outputs
% x,x_medio,pasovector: Outputs de la funcion divisorderebanadas
% y_talud_med: Ordenada del talud en los puntos del vector x_med
% y_circ_med: Ordenada de la circunferencia en los puntos del vector x_med
% alfa: Angulo respecto de la horizontal de la linea inferior de cada
% rebanada.
% y_talud: Ordenada del talud en los puntos del vector x
% y_circ: Ordenada de la circunferencia en los puntos del vector x
%
%
%

% Llamamos al divisor de rebanadas
[ x,x_medio,pasovector ] = divisorderebanadas( xti,xtd,rebanadas,r,B);

% Talud
y_talud = taludgeometria( B,x,H);
y_talud_med = taludgeometria( B,x_medio,H);


% Calculo de ordenada de los puntos M
y_circ=real(-sqrt(R^2-(x-a).^2)+b);
% y_circ=(round(y_circ*precisiondecimal)/precisiondecimal);


y_circ_sinpri=y_circ;y_circ_sinpri(1)=[];
y_circ_sinfin=y_circ;y_circ_sinfin(end)=[];
x_sinpri=x;x_sinpri(1)=[];x_sinfin=x;x_sinfin(end)=[];

% Altura rebanadas punto medio
y_circ_med=(y_circ_sinpri+y_circ_sinfin)./2;
% y_circ_med=(round(y_circ_med*masprecision)/masprecision);
```

```
% Angulo de cada rebanada
pendiente=(y_circ_sinpri-y_circ_sinfin)./(x_sinpri-x_sinfin);
alfa=real(atand(pendiente));
% alfa=(round(alfa*precisiondecimal)/precisiondecimal);
 end
```

2.1.3 MÉTODOS DE EQUILIBRIO LÍMITE

2.1.3.1 FUNCIÓN MFELLE

```
       function [ FS,num,den ] = mFelle( y_talud_med,y_circ_med,alfa
,pasovector,gd,C,fi )
% Esta función calcula el FS por el método de Fellenius
% Inputs
% y_talud_med,y_circ_med,alfa,pasovector: Outputs función parametros
% Outputs
% gd: Peso especifico del terreno
% C: Cohesión efectica del terreno
% fi: Angulo de rozamiento del terreno
%
%
%
% 1) Calculamos la superficie inferior de la rebanada
l=pasovector./cosd(alfa);
% 2) Peso de cada rebanada
altura=y_talud_med-y_circ_med;
area=pasovector.*altura;
W=gd*area;
% 3) Calculamos Denominador (fuerzas desesestabilizadoras (Stress))
T=W.*sind(alfa);    % descomposicion del peso(W) en la tangencial
den=sum(T);
% 4) Calculamos Numerador (fuerzas estabilizadoras (Strength))
cohesion=C*l;
N=W.*cosd(alfa);    % descomposicion del peso(W) en la normal
rozamiento=N*tand(fi);
num=sum(cohesion+rozamiento);
% 5) Calculamos el factor de seguridad (FS)
FS=abs(num/den);
end
```

2.1.3.2 FUNCIÓN MBISHOP

```
        function [FS,num,den] = mBishop( y_talud_med,y_circ_med,alfa,pasovector,gd,C,fi
)
% Esta función calcula el FS por el método de Bishop simplificado
% Inputs
% y_talud_med,y_circ_med,alfa,pasovector: Outputs función parametros
% Outputs
% gd: Peso especifico del terreno
% C: Cohesión efectica del terreno
% fi: Angulo de rozamiento del terreno
% Outputs
% FS: Factor de seguridad por el método de Bishop simplificado
% num: Numerador del cociente FS=num/den
% den: Denominador del cociente FS=num/den
%
%
%

% 1) Peso de cada rebanada
altura=y_talud_med-y_circ_med;
area=pasovector.*altura;
W=gd*area;

% 2) Calculamos Denominador (fuerzas desestabilizadoras (Stress))
T=W.*sind(alfa);
den=sum(T);

% 3) Calculamos Numerador (fuerzas estabilizadoras (Strength))y FSB
B1=C*pasovector;B2=W*tand(fi);B3=cosd(alfa);B4=tand(fi)*sind(alfa);
% Contador i=0 % Primer FS=1 % definimos toleranciaB1=1>0.001 para entrar
i=0;FS(1)=1;toleranciaB=1;
    while toleranciaB>0.0001
        i=i+1 ;
        B5=B3+B4./FS(i) ;B6=1./B5;
       num=sum((B1+B2).*B6);
        FSB=abs(num/den);
% 4) Guardamos el FSB en un vector y calculamos el error
        FS(i+1)=FSB; %#ok<AGROW> % Guardamos cada valor de FSB en el vector FS(i)
        toleranciaB=abs(FS(i+1)-FS(i)); % Calculamos el error
    end
% 5) Escogemos el último puesto como representante
        FS=FS(end); % Nos quedamos con el último FS
end
```

2.1.3.3 FUNCIÓN MMORGPRI

```
       function [FS,lambda] = mMorgPri( x,y_talud_med,y_circ_med,alfa
,pasovector,gd,C,fi )
% Esta función calcula el FS por el método de Morgenstern-Price
% Inputs
% y_talud_med,y_circ_med,alfa,pasovector: Outputs de la función parametros
% Outputs
% gd: Peso especifico del terreno
% C: Cohesión efectica del terreno
% fi: Angulo de rozamiento del terreno
% Outputs
% FS: Factor de seguridad por el método de Morgenstern-Price
% lambda: Factor de corrección lambda
%
%
%
%
% Calculo de R,T
altura=y_talud_med-y_circ_med;
area=pasovector.*altura;
W=gd*area;
% R
N=W.*cosd(alfa);
rozamiento=N*tand(fi);
cohesion=C*pasovector.*secd(alfa);
R=rozamiento+cohesion;
% T
T=W.*sind(alfa);
% Elección de f(x)
for mu=1:1:2  %0-5
    for uve=0.5:0.5:2 %0.5-2
aa=x(1);
bb=x(end);
argumento=pi*(((x-aa)/(bb-aa)).^uve);
medioseno=(sin(argumento)).^mu;
f=medioseno;  %f(1)=f0,f(2)=f1,f(i)=fi-1,f(n)=f(n-1)
% Si se quiere plotear la funcuion activar este modulo
%plot(x,f);
% axis([aa bb 0 1.1])

% Metodo para Fi,Psi, Factor de seguridad
[FS,lambda] = ZhuLeeChen(alfa ,pasovector,fi,R,T,f);
% 5) Escogemos el último puesto como representante
        FS=FS(end); % Nos quedamos con el último FS
if isnan(FS)==0
    break
end
    end
end
end
```

2.1.3.4 FUNCIÓN ZHULEECHEN

```
       function [FS,lambda,num,den] = ZhuLeeChen(alfa,pasovector,fi,R,T,f)
% Esta funcion forma parte de la funcion mMorgPri. Esta calcula el FS de
% Mogenstern-Price por el método de Zhu, Lee y Chen.
% Inputs
% alfa,pasovector,fi: Imputs de la duncion mMorgPri
% R,T,f: Variable calculada en la funcion mMorgPri
% Outputs
% FS: FS en la iteracion i por el método de Morgenster-Price
% lambda: Factor de corrección en la iteración i.
% num,den: numerador y denominador del cociente FS=num/den.
%
%
%
%
% Definicion de variables necesarias
toleranciaMP=0.0001;
FS(1)=1;lambda(1)=0;
contadorFS=1;contadorlambda=1;
errorFS=1;errorlambda=1;
        while (errorFS>=toleranciaMP || errorlambda>=toleranciaMP )
         if contadorFS>=20
             FS=nan;lambda=nan;
         break
         end
                contadorFS=contadorFS+1;

            for j=1:2
                 % FI (FI1....FIn)
                 fsinfo=f;fsinfo(1)=[];
                 FI=(sind(alfa)-lambda(end)*fsinfo.*cosd(alfa))...
                      *tand(fi)+(cosd(alfa)+lambda(end)*fsinfo...
                      .*sind(alfa))*FS(end);
                % PSI  (PSI1....PSI(n-1))
                fsinfofn=fsinfo; fsinfofn(end)=[];
                asint1=alfa;asint1(1)=[];
                FIsinFIn=FI;FIsinFIn(end)=[];

                PSI=((sind(asint1)-lambda(end)*fsinfofn.*cosd(asint1))...
                    *tand(fi)+(cosd(asint1)+lambda(end)...
                    *fsinfofn.*sind(asint1))*FS(end))./FIsinFIn;
                % FS
                N=length(alfa); %N rebanadas;
                numi=zeros(1,N-1);deni=zeros(1,N-1);
                for i=1:N-1
                    prodPSI=prod(PSI(i:end));
                 % numerador
                    numi(i)=R(i)*prodPSI;
                 % denominador
                    deni(i)=T(i)*prodPSI;
                end
                num=sum(numi)+R(end);den=sum(deni)+T(end);
                FS(contadorFS)=abs(num/den); %#ok<AGROW>
            end
```

```
    % E (E1...E(n-1))
         E=zeros(1,N-1);
         E(1)=FS(end)*T(1)-R(1)/FI(1);

        for i=2:N-1
            E(i)=(PSI(i-1)*E(i-1)*FIsinFIn(i-1)+FS(end)*T(i)-R(i))/FI(i);
        end
    % Lambda
    % Adaptamos E a la ecuación de lambda. Eo y En =0.
        fsinfn=f;
        fsinfn(end)=[];

        Eimenos1=[0 E];Ei=[E 0];
        Enum=Ei+Eimenos1;
        fsinfnEimenos1=Eimenos1.*fsinfn;fsinfoEi=Ei.*fsinfo;
        fEden=fsinfoEi+fsinfnEimenos1;

        contadorlambda=contadorlambda+1;

        lambdanum=sum(pasovector.*(Enum).*tand(alfa));
        lambdaden=sum(pasovector.*fEden);
        lambda(contadorlambda)=lambdanum/lambdaden;

    % errores
        errorFS=abs(FS(end)-FS(end-1));
        errorlambda=abs(lambda(end)-lambda(end-1));

        end
    end
```

2.2 ALGORITMOS DE BÚSQUEDA Y OPTIMIZACIÓN

2.2.1 FACTOR DE SEGURIDAD EN UN PUNTO

2.2.1.1 PROCEDIMIENTO ELEGIRMETODO

```
       % Procedimiento: elegirmetodo
% Este procedimiento adapta una cadena para poder llamar a cada método
if strcmp(Metodo,'mMorgPri')==1
    Met=['[FS,lambda] =' Metodo '( x,y_talud_med,y_circ_med,alfa,pasovector,gd,C,fi);'];
 else
    Met=['[ FS ] =' Metodo '( y_talud_med,y_circ_med,alfa,pasovector,gd,C,fi );'];
 end
```

2.2.1.2 FUNCIÓN FSP

```
       function [ FS,xti,xtd,k4,lambda] = FSP( a,b,R,C,gd,fi,B,H,rebanadas,Met,Metodo)
%#ok<INUSL>
% Esta función permite calcular el FS en un punto.
% Inputs
% a,b,R: Centro y radio de la circunferencia
% C,gd,fi: Cohesión, Peso específico y angulo de rozamiento del terreno
% B,H: Base y altura del terreno
% rebanadas: Numero de rebanadas
% Met,Metodo: Cadenas de caracteres que informa del método elegido
% Outputs
% FS: Factor de seguridad por el metodo que aparezca en la variable Met
% xti,xtd: Abscisas de los puntos de corte de la circunferencia con el
% talud.
% k4: Variable que informa si la superficie de deslizamiento es valida o
% no.
% lambda: Factor de corrección calculado en la funcion ZhuLeeChen.
[ xti,xtd,k4,r] = calculoextremos(a,b,B,H,R);
    if  k4==0 % No hay superficie de deslizamiento válida
        FS=nan;
        lambda=nan;
    else
      [ x,x_medio,y_talud_med ,y_circ_med,alfa,...
      pasovector,y_talud,y_circ ] = parametros( rebanadas,...
      xti,xtd,B,H,a,b,R,r); %#ok<*ASGLU,*NASGU>
          switch Metodo
              case 'mMorgPri'
                   eval(Met);
              case {'mFelle','mBishop'}
                   eval(Met);lambda=nan;
          end
    end
 end
```

2.2.2 FACTOR DE SEGURIDAD VARIANDO EL RADIO EN UN PUNTO

2.2.2.1 FUNCIÓN FSRMINSINOPT

```
       function [FSRmin,FSR]=FSRMinSinOpt(a,b,C,gd,fi,B,H,rebanadas,Met,Metodo)
% Esta función calcula el FS mínimo asociado a un punto sin aplicar técnicas
% de optmización.
pr=0.01;
% Calculamos distancias
[Dmin,R2max,Dv1,~,Dv2,~]=distminR1(B,H,a,b);
Dfin=0;
% Distancia final
if b>H
cFSRDf=0;
R=max([Dv1 Dv2]);
while 2<3
        cFSRDf=cFSRDf+1;
         R=R+0.5;
        [ FS] = FSP( a,b,R,C,gd,fi,B,H,rebanadas,Met,Metodo);
```

```
        FSRDf(cFSRDf,:)=[FS R];  % contiene [FS R]
        try
            [ derivadaprimera] = FSderivada( a,b,R,pr,C,gd,fi,B,H,rebanadas,Met,Metodo
);
              if derivadaprimera>0
                Dfin=FSRDf(cFSRDf,2);
              break
              end
        end
end
end
%
l1=Dmin+pr;
l2=R2max*(b<=H)+Dfin*(b>H);
cFSR=0;
for R=l1:pr:l2
    cFSR=cFSR+1;
    [ FS] = FSP( a,b,R,C,gd,fi,B,H,rebanadas,Met,Metodo);
    FSR(cFSR,:)=[FS(end) R]; % contiene [FS; R]
    R=R+pr;
end
% Buscamos la pareja FSmin,R
 [FSmin,pFSmin] = min(FSR(:,1)); % Buscamos FSmin
 FSRmin=[FSmin FSR(pFSmin,2)]; % contiene [FSmin R]
end
```

2.2.2.2 FUNCIÓN FSRMINCONOPT

```
       function [ FSRmintotal,FSR,minFSR,FSRminFSR,minFSRminFSR ] = FSRMinConOpt(
a,b,PT,C,gd,fi,B,H,rebanadas,Met,Metodo )
% Esta función calcula el FS mínimo asociado a un punto aplicando técnicas
% de optmización.
% Inputs
% Mismos inputs que FSP
% Outputs
% FSRmintotal: El FS minimo asociado al punto
% Solo interesa FSRmintotal.
% El resto de outputs se utilizan para dibujar las gráficas.
%
%
pr=0.01; % Paso con el que avanza el radio
frenarderivada=0.05; % Derivada que se acepta como nula
% Calculamos distancia mínima y tramos
[Dmin,R2max,Dv1,~,Dv2,zona]=distminR1(B,H,a,b);
zona=zona';
if b>H
% % Subproceso R2max
cFSRDf=0;
R=max([Dv1 Dv2]);
while 2<3
        cFSRDf=cFSRDf+1;
         R=R+0.5;
        [ FS] = FSP( a,b,R,C,gd,fi,B,H,rebanadas,Met,Metodo);
        FSRDf(cFSRDf,:)=[FS R]; % contiene [FS; R]
```

```
        try %#ok<TRYNC>
            [ derivadaprimera] = FSderivada( a,b,R,pr,C,gd,fi,B,H,rebanadas,Met,Metodo
);
              if derivadaprimera>0
                Dfin=FSRDf(cFSRDf,2);
              break
              end
        end
end
end
% % fin R2max
%
% %Subproceso: asignacionli
% Las distintas posibilidades son
if b<=H
    if isequal(zona,[0 1 0])  %Dmin==Dmc % Se encuentra en la zona central
            if min([Dv1 R2max])==Dv1 % Primero llega a Dv1
                l1=Dmin;l2=Dv1;l3=R2max;l4=0;
            elseif min([Dv1 R2max])==R2max % No llega a Dv1
                l1=Dmin; l2=R2max; l3=0;l4=0;
            end
    elseif isequal(zona,[1 0 0])%Dmin~=Dmc Se encuentra en la zona exterior
        l1=Dv1;l2=R2max; l3=0;l4=0;
    end
elseif b>H
    if isequal(zona,[0 1 0]) && Dv1~=Dv2
        l1=Dmin;l2=min([Dv1 Dv2]);l3=max([Dv1 Dv2]);l4=Dfin;
    elseif  isequal(zona,[1 0 0]) || isequal(zona,[0 0 1])
        l1=Dmin;l2=max([Dv1 Dv2]);l3=Dfin;l4=0;
    elseif isequal(zona,[0 1 0]) && Dv1==Dv2
        l1=Dmin;l2=Dv1;l3=Dfin;l4=0;
    end
end
% % fin asignacionli
%
% Subproceso reparticionN
tramo1=l2-l1;
tramo2=l3-l2;
if tramo2==-l2
    tramo2=0;
end
tramo3=l4-l3;
if tramo3==-l3
    tramo3=0;
end
% Todo el recorrido
Tramototal=tramo1+tramo2+tramo3;
% Proporcines
prop1=tramo1/Tramototal;
prop2=tramo2/Tramototal;
prop3=tramo3/Tramototal;
% fin reparticionN
%
%  Empezamos con tramo 1
% Numero de puntos
puntostramo1=ceil(prop1*PT)+2;
```

```
Rtramo1=linspace(l1+pr,l2,puntostramo1);
paso1=Rtramo1(2)-Rtramo1(1);
if paso1>2*pr
        %Primera búsqueda.Matriz FSR
        cFSR=0;
        for R=Rtramo1
            cFSR=cFSR+1;
            [ FS] = FSP( a,b,R,C,gd,fi,B,H,rebanadas,Met,Metodo);
            FSR1(cFSR,:)=[FS R]; %#ok<AGROW>
        end
         % Primer mínimo.minFSR
         [FSmin1,pFSmin1] = min(FSR1(:,1)); %Buscamos FSmin
          minFSR1=[FSmin1 FSR1(pFSmin1,2)]; % contiene [FSmin R]
      if minFSR1(1,2)~=l1+pr && minFSR1(1,2)~=l2
         % Definimos la derivada para saber en que dirección hacemos la búsqueda
         [derivadaprimera1] = FSderivada(
a,b,minFSR1(1,2),pr,C,gd,fi,B,H,rebanadas,Met,Metodo );
         % Si la derivada es cercana a cero, ya se encontró el mínimo.
         if abs(derivadaprimera1)<frenarderivada
             FSR1minFSR1=[nan nan];
             minFSR1minFSR1=minFSR1;
         else
             FSR1minFSR1(1,:)=minFSR1;
             cFSR=1;
             R=minFSR1(1,2);
             sentido=-derivadaprimera1/abs(derivadaprimera1);
             nuevosentido=sentido;
             while sentido==nuevosentido
                 cFSR=cFSR+1;
                 R=R+pr*sentido;
                  if R>=l2 || R<=l1
                      break
                  end
                 [FS] = FSP( a,b,R,C,gd,fi,B,H,rebanadas,Met,Metodo);
                 FSR1minFSR1(cFSR,:)=[FS R];
                 [derivadaprimera1] = FSderivada(
a,b,R,pr,C,gd,fi,B,H,rebanadas,Met,Metodo );
                 nuevosentido=-derivadaprimera1/abs(derivadaprimera1);
             end
             % Buscamos el minimo del tramo minFSRminFSR
             [FSmin1,pFSmin1] = min(FSR1minFSR1(:,1));
             minFSR1minFSR1=[FSmin1 FSR1minFSR1(pFSmin1,2)];

         end
      else
          FSR1minFSR1=[nan nan];
          minFSR1minFSR1=minFSR1;
      end
else
 % Si el tramo es muy pequeño analizamos tres puntos nada más
     Rtramo1=linspace(l1,l2,3);
         cFSR=0;
     for R=Rtramo1
        cFSR=cFSR+1;
        [FS] = FSP( a,b,R,C,gd,fi,B,H,rebanadas,Met,Metodo);
        FSR1(cFSR,:)=[FS R];
```

```
     end
     % Buscamos el minimo total del tramo FSR1
       [FSmin1,pFSmin1] = min(FSR1(:,1));
       minFSR1minFSR1=[FSmin1 FSR1(pFSmin1,2)];
       % Asignamos valores a todas las variables
       FSR1minFSR1=[nan nan];
        minFSR1=[nan nan];
end

%  pause

% Empezamos con tramo 2

if tramo2~=0
    puntostramo2=ceil(prop2*PT)+2;
    Rtramo2=linspace(l2+pr,l3,puntostramo2);
    paso2=Rtramo2(2)-Rtramo2(1);
if paso2>2*pr
        %Primera búsqueda.Matriz FSR
        cFSR=0;
        for R=Rtramo2
        cFSR=cFSR+1;
            [FS] = FSP( a,b,R,C,gd,fi,B,H,rebanadas,Met,Metodo);
            FSR2(cFSR,:)=[FS R] ; %#ok<AGROW>
        end
        %Primer mínimo.minFSR
       [FSmin2,pFSmin2] = min(FSR2(:,1));
        minFSR2=[FSmin2 FSR2(pFSmin2,2)] ;
   if minFSR2(1,2)~=l2+pr && minFSR2(1,2)~=l3
        %Definimos la derivada para saber en que dirección hacemos la búsqueda
         [derivadaprimera2] = FSderivada(
a,b,minFSR2(1,2),pr,C,gd,fi,B,H,rebanadas,Met,Metodo );
         if abs(derivadaprimera2)<frenarderivada
             FSR2minFSR2=[nan nan];
             minFSR2minFSR2=minFSR2;
         else
             FSR2minFSR2(1,:)=minFSR2;
             cFSR=1;
             R=minFSR2(1,2);
             sentido=-derivadaprimera2/abs(derivadaprimera2);
             nuevosentido=sentido;
             while sentido==nuevosentido
                 cFSR=cFSR+1;
                 R=R+pr*sentido;
                 if R>=l3 || R<=l2
                      break
                  end
                 [ FS] = FSP( a,b,R,C,gd,fi,B,H,rebanadas,Met,Metodo);
                 FSR2minFSR2(cFSR,:)=[FS R];
                 [derivadaprimera2] = FSderivada(
a,b,R,pr,C,gd,fi,B,H,rebanadas,Met,Metodo );
                 nuevosentido=-derivadaprimera2/abs(derivadaprimera2);
             end
             %Buscamos el mínimo del tramo
        [FSmin2,pFSmin2] = min(FSR2minFSR2(:,1));
        minFSR2minFSR2=[FSmin2 FSR2minFSR2(pFSmin2,2)];
```

```
        end
   else
            FSR2minFSR2=[nan nan];
             minFSR2minFSR2=minFSR2;
    end
else
%Si el tramo es muy pequeño se analizan tres puntos
        cFSR=0;
    for Rtramo2=linspace(l2,l3,3)
        cFSR=cFSR+1;
        [ FS] = FSP( a,b,R,C,gd,fi,B,H,rebanadas,Met,Metodo);
        FSR2(cFSR,:)=[FS R];  %#ok<AGROW>
    end
    % Buscamos el minimo total del tramo FSR1
       [FSmin2,pFSmin2] = min(FSR2(:,1));
       minFSR2minFSR2=[FSmin2 FSR2(pFSmin2,2)];
       % Asignamos valor a todas las variables
       FSR2minFSR2=FSR2;
       minFSR2=[nan nan];
end
end
% pause
%    Tramo 3
if tramo3~=0
puntostramo3=ceil(prop3*PT)+2;
Rtramo3=linspace(l3+pr,l4,puntostramo3);
paso3=Rtramo3(2)-Rtramo3(1);
if paso3>2*pr
    %Primera búsqueda.Matriz FSR
        cFSR=0;
    for R=Rtramo3
        cFSR=cFSR+1;
        [FS] = FSP( a,b,R,C,gd,fi,B,H,rebanadas,Met,Metodo);
        FSR3(cFSR,:)=[FS R];  %#ok<AGROW>
    end
    %Buscamos el primer mínimo minFSR
    [FSmin3,pFSmin3] = min(FSR3(:,1));
    minFSR3=[FSmin3 FSR3(pFSmin3,2)];
  if minFSR3(1,2)~=l3+pr && minFSR3(1,2)~=l4
    %Definimos la derivada para saber en que dirección hacemos la búsqueda
    [derivadaprimera3] = FSderivada(
a,b,minFSR3(1,2),pr,C,gd,fi,B,H,rebanadas,Met,Metodo);
    if abs(derivadaprimera3)<frenarderivada
        FSR3minFSR3=[nan nan];
        minFSR3minFSR3= minFSR3;
    else
    FSR3minFSR3(1,:)=minFSR3;
    cFSR=1;
    R=minFSR3(1,2);
    sentido=-derivadaprimera3/abs(derivadaprimera3);
    nuevosentido=sentido;
         while sentido==nuevosentido
             cFSR=cFSR+1;
             R=R+pr*sentido;
              if R>=l4 || R<=l3
```

```
                    break
                end
            [ FS] = FSP( a,b,R,C,gd,fi,B,H,rebanadas,Met,Metodo);
            FSR3minFSR3(cFSR,:)=[FS R];
           [derivadaprimera3] = FSderivada( a,b,R,pr,C,gd,fi,B,H,rebanadas,Met,Metodo );
            nuevosentido=-derivadaprimera3/abs(derivadaprimera3);
        end
        %Buscamos el minimo del tramo  minFSR3minFSR3
    [FSmin3,pFSmin3] = min(FSR3minFSR3(:,1)); %Buscamos FSmin
     minFSR3minFSR3=[FSmin3 FSR3minFSR3(pFSmin3,2)]; % contiene [FSmin R]
    end
  else
        FSR3minFSR3=[nan nan];
        minFSR3minFSR3= minFSR3;
  end
else
         cFSR=0;
     for Rtramo3=linspace(l3,l4,3)
        cFSR=cFSR+1;
        [ FS] = FSP( a,b,R,C,gd,fi,B,H,rebanadas,Met,Metodo);
        FSR3(cFSR,:)=[FS R];    %#ok<AGROW>
     end
     % Buscamos el minimo total del tramo FSR1
       [FSmin3,pFSmin3] = min(FSR3(:,1));
       minFSR3minFSR3=[FSmin3 FSR3(pFSmin3,2)];
       % Asignamos valor a todas las variables
       FSR3minFSR3=FSR3;
       minFSR3=[nan nan];
end
end
%
% Se unen aquí todos los vectores
FSR=FSR1;
FSRminFSR=FSR1minFSR1;
minFSR=minFSR1;
minFSRminFSR=minFSR1minFSR1;

if tramo2~=0
    FSR=[FSR1;nan nan; FSR2];
    FSRminFSR=[FSR1minFSR1; nan nan; FSR2minFSR2];
    minFSR=[minFSR1; minFSR2];
    minFSRminFSR=[minFSR1minFSR1; minFSR2minFSR2];
end
if tramo3~=0
    FSR=[FSR1;nan nan; FSR2;nan nan; FSR3];
    FSRminFSR=[FSR1minFSR1;nan nan; FSR2minFSR2; nan nan;FSR3minFSR3];
    minFSR=[minFSR1; minFSR2; minFSR3];
    minFSRminFSR=[minFSR1minFSR1; minFSR2minFSR2; minFSR3minFSR3];
end

%Elegimos el mínimo total
[FSmin,pFSmin] = min(minFSRminFSR(:,1)); %Buscamos FSmin
FSRmintotal=[FSmin minFSRminFSR(pFSmin,2)]; % contiene [FSmin R]

end
```

2.2.3 FACTOR DE SEGURIDAD GLOBAL

2.2.3.1 FUNCIÓN RASTREO

```
       function [FSRabmin,FSRmab,FS3D,i,j] =
Rastreo(a1,a2,b1,b2,pa,pb,C,PT,gd,fi,B,H,rebanadas,Met,Metodo )
% Esta función permite calcular el mínimo FS asociado a un talud. Necesita
% ser usada por otra función que la utilice, como por ejemplo la funcion
% marcosoptimos
% Inputs
% a1,a2,b1,b2: Son las esquinas del marco de búsqueda
% pa,pb: Pasos con los que se quiere rastrear dicho marco
% resto de variables: ver funcion FSP
% Outputs
% FSRabmin: Es el factor de seguridad mínimo global.
% Resto de variable: Ver memoria descriptiva


    cFSRmab=0;
        j=0;            % Este contador permite crear la matriz FS3D
    for a=linspace(a1,a2,pa+1)
        j=j+1;
        i=0;    % Este contador permite crear la matriz FS3D
        for b=linspace(b1,b2,pb+1)
            i=i+1;
            cFSRmab=cFSRmab+1;
                [D,R2max,D1,D2,D3,zona]=distminR1(B,H,a,b);

                % Parche de proteccion
                    if b<=0 || D2<=0 || (a>=B && b<=H)
                        FSRmab(cFSRmab,:)=[nan nan a b]; %FSRmab=[FSmin R(FSmin) a  b]
                        FS3D(i,j)=nan;
                        continue
                    end
            [FSminR]=FSRMinConOpt( a,b,PT,C,gd,fi,B,H,rebanadas,Met,Metodo );
            FSRmab(cFSRmab,:)=[FSminR a b]; %FSRmab=[FSmin R(FSmin) a  b]
            FS3D(i,j)=FSminR(1,1);          % Matriz FSmin en 3D
        end
    end
    %        Buscamos la pareja FSmin,R
            [FSmin,pFSmin] = min(FSRmab(:,1)); %Buscamos FSmin
            FSRabmin=[FSmin FSRmab(pFSmin,2:end)]; % FSRabmin=[FSmin R]

end
```

2.2.3.2 FUNCIÓN MARCOSOPTIMOS

```
       function [FSRabmin,FRab,npmt,et,p,npt] =
marcosoptimos(ca,cb,l1,pn,fe,C,PT,gd,fi,B,H,rebanadas,Met,Metodo )
% Esta función calcula el mínimo FS asociado a un talud aplicando técnicas
% de optimización.
% Inputs
% ca,cb: punto alrededor del cual se dibujará un marco cuadrado
% l1: mitad del largo de cada lado de cuadrado
```

```
% pn : precisión final con la que se quiere dar el resultado
% fe: factor de ensanchamiento a aplicar a los marcos sucesivos
% Resto de variables: ver funcion FSP
% Outputs
% FSRabmin: Es el factor de seguridad mínimo global.
% Resto de variable: Ver memoria descriptiva
%
%
%
%
pregunta=questdlg('¿Desea  dibujar los marcos?','SALIR','Si','No','No');
if strcmp(pregunta,'Si')
dibujarmarco=1;
else
dibujarmarco=0;
end
% Calculamos los parámetros de optimización
po=l1/2;
et=round(log(po/pn)); % número de etapas o marcos optimo
% Definimos el vector p=(p1,p2...pet)
p=0*(1:et-1); % Restamos 1 porque el último el primero son dato
    for q=1:et-1
        p(q)=nthroot((pn^q)*po^(et-q),et);
    end
p=[po p pn]; % Conjunto de pasos optimo
npt=round(2*fe*nthroot((po/pn),et)+1);% El número de puntos a repartir en cada tramo
será
pa=npt;pb=npt;
%
repeticiones=0; % Esta variable informará las veces que se tuvo que repetir una
operación porque el centro quedó en el borde
hold on
for q=1:et
a1=ca-fe*p(q);
a2=ca+fe*p(q);
b1=cb-fe*p(q);
b2=cb+fe*p(q);
        if dibujarmarco==1
            hold on
            vp=[a1 b1];vf=[a2 b2];
            rectangulo
            rectang(q)=plot([a1,a2,a2,a1,a1],[b1,b1,b2,b2,b1]);
        end
   [FSRabmin] = Rastreo(a1,a2,b1,b2,pa,pb,C,PT,gd,fi,B,H,rebanadas,Met,Metodo );
        if isnan(FSRabmin(1,1))
            return
        end
   % El centro encontrado no puede estar en el borde
   w=0;
  while FSRabmin(3)==a1 || FSRabmin(3)==a2 || FSRabmin(4)==b1 || FSRabmin(4)==b2
      w=w+1;
      if w>10
          break
      end
        a1=FSRabmin(3)-fe*p(q);
        a2=FSRabmin(3)+fe*p(q);
```

```
        b1=FSRabmin(4)-fe*p(q);
        b2=FSRabmin(4)+fe*p(q);
        repeticiones=repeticiones+1;
             if dibujarmarco==1
                 hold on
                vp=[a1 b1];vf=[a2 b2];
                rectangulo
                 rectang(q+repeticiones)=plot([a1,a2,a2,a1,a1],[b1,b1,b2,b2,b1],'g');
             end
        [FSRabmin] = Rastreo(a1,a2,b1,b2,pa,pb,C,PT,gd,fi,B,H,rebanadas,Met,Metodo );
        if isnan(FSRabmin(1,1))
            continue
        end
   end
   % Asignamos los nuevos centros sobre los que dibujar el marco
   ca=FSRabmin(3);
   cb=FSRabmin(4);
 % Guardamos todos los valores en un vector
   FRab(q,:)=FSRabmin;
end
npmt=(et+repeticiones)*(npt+1)^2; % Calculamos el número total de simulaciones
end
```