

## ANEJOS

---

## **ANEJOS**

ANEJO 1. Código ejemplo 1.....	iii
ANEJO 2. Código ejemplo 2.....	xi
ANEJO 3. Código ejemplo 3.....	xvi
ANEJO 4. Código Ejemplo 4.....	xxiv

## ANEJO 1. Código ejemplo 1

```
function CPbyTM_v12
% Cálculo Plástico (CP) / Análisis Límite mediante el Método
Directo y
% Teoría de Mecanismos (TM). Pórtico biempotrado

% Datos
L=4.0;
F=1.0*10^3;
vE=2.1*10^11;
Iz=8360.0*10^-8;
S=628.0*10^-6;
sigma_F=275*10^6;
Mp=S*sigma_F;

% Geometría
nodos = {[0, 0], {'Fx', 'Fy', 'Mz'}},
        {[0, L], {0, 0}},
        {[L, L], {0, -F}},
        {[2*L, L], {F, 0}},
        {[2*L, 0], {'Fx', 'Fy', 'Mz'}}
        ]; % coordenadas / fuerzas
material = {[vE]}; % una lista por material
perfil = {[Iz, Mp]}; % una lista por tipo de perfil
barras = {[1, 2], [1, 1]},
        {[2, 3], [1, 1]},
        {[3, 4], [1, 1]},
        {[4, 5], [1, 1]}
        ]; % definicion, propiedades(material,perfil)

nd=length(nodos);
nb=length(barras);

% numero de ecuaciones de equilibrio
NPR=nd;
GH=0;
for i=1:nd
    kk=nodos{i}{2}{1};
    if kk=='Fx'
        GH=GH+1;
    end
end

GH=3*(GH-1);
EQ=NPR-GH;

fprintf('Ecuaciones de equilibrio = %d \n',EQ);
fprintf('Ecuaciones de compatibilidad = %d \n',GH);

% mecanismos propuestos / ecuaciones de equilibrio

dv=0.0001;
mt=zeros(nd,3,3); % matrices Ti
mmT=zeros(3,3); % matriz T
```

```

qi=zeros(EQ,NPR); % giros virtuales
di=zeros(EQ,2,NPR); % desplazamientos virtuales
Mi=ones(1,NPR); % momentos reales

% mecanismo (MC1)
q0=[0 1 1 1 0];
options = optimoptions('fsolve','Display','iter','TolFun',1e-30,'TolX',1e-50);
[q,fun]=fsolve(@posicion,dv*q0,options);
qi(1,:)=q;

% desplazamiento de los nodos
di(1,:,:)=delta(q);

% mecanismo (MC2)
q0=[1 1 0 1 1];
[q,fun]=fsolve(@posicion,dv*q0,options);
qi(2,:)=q;

di(2,:,:)=delta(q);

% plantear las ecuaciones de compatibilidad (PFV)

mi=zeros(GH,NPR); % momentos virtuales
m0=[1 0 0 0 1]; % EC1
[m,fun]=fsolve(@momentosVirtuales,m0,options);
mi(1,:)=m;

m0=[0 1 0 0 0]; % EC2
[m,fun]=fsolve(@momentosVirtuales,m0,options);
mi(2,:)=m;

m0=[0 0 1 0 0]; % EC3
[m,fun]=fsolve(@momentosVirtuales,m0,options);
mi(3,:)=m;

% ecuacion de la energia (funcion a minimizar) + restricciones
s0=zeros(1,1+2*NPR);
s0(1)=100;
s0(2:2+NPR-1)=zeros(1,NPR);
s0(2+NPR:end)=zeros(1,NPR);

lb=zeros(1,1+2*NPR);
lb(2:2+NPR-1)=-1.0;
lb(2+NPR:end)=-50.0;

ub=zeros(1,1+2*NPR);
ub(1)=1000;
ub(2:2+NPR-1)=1.0;
ub(2+NPR:end)=50.0;

options = optimoptions('fmincon','Display','iter','TolFun',1e-30,'TolX',1e-50);
[s,fval]=fmincon(@energia,s0,[],[],[],[],lb,ub,@restricciones,options);

```

```

lambda=s(1);
M=Mp*s(2:1+NPR);

s0=ones(1,NPR-1);

[s,fval]=fmincon(@energiaDisipada,s0,[],[],[],[],[],[],@restriccio
nes2,options);

q=zeros(1,NPR);
q([1 3 4 5])=s(:);

fprintf('Carga de colapso: %4.2f \n',F*lambda);
fprintf('Momento flector en "a": %4.2f \n',M(1));
fprintf('Momento flector en "b": %4.2f \n',M(2));
fprintf('Momento flector en "c": %4.2f \n',M(3));
fprintf('Momento flector en "d": %4.2f \n',M(4));
fprintf('Momento flector en "e": %4.2f \n',M(5));
fprintf('\n');

fprintf('Giro en "a": %2.5f \n',q(1));
fprintf('Giro en "b": %2.5f \n',q(2));
fprintf('Giro en "c": %2.5f \n',q(3));
fprintf('Giro en "d": %2.5f \n',q(4));
fprintf('Giro en "e": %2.5f \n',q(5));

kk=1;

function v=posicion(q)

    q=q.*q0;
    kk=matrizT(q);
    vP=cat(1, kk(1:2,3), kk(2,1));
    v0=cat(2, nodos{nd}{1}, 0.0);
    v=v0'-vP;

end

function v=matrizT(q)

    lx=0;
    ly=0;
    i=1;
    Li=norm(nodos{i+1}{1}-nodos{i}{1});
    v=mT(q(1),Li,lx,ly);
    mt(1,:,:) =v;

    for i=1:nb
        Li=norm(nodos{i+1}{1}-nodos{i}{1});
        kk=(nodos{i+1}{1}-nodos{i}{1})/Li;
        lx=kk(1);

```

```

        ly=kk(2);
        kk=mT(q(i+1),Li,lx,ly);
        mt(i+1,:,:) =kk;
        v=v*kk;

    end

    mmT=v;

    function v=mT(x,Li,lx,ly)
        v=[ [1,-x,Li*lx]
            [x,1,Li*ly]
            [0,0,1]
            ];
    end

end

function v=delta(q)

    v=zeros(2,nd);

    for i=1:nd
        kk1=diag(ones(1,3));
        for j=1:i
            kk1=kk1*reshape(mt(j,:,:),[3,3]);
        end
        kk=kk1(1:2,3);
        kk2=nodos{i}{1}';
        v(:,i)=kk-kk2;
    end
end

function v=momentosVirtuales(m)
    % calcula los momentos virtuales (mi)

    v=zeros(1,EQ);
    q0=qi;
    for j=1:EQ
        q0(j,:)=q0(j,:)/max(q0(j,:));
        for i=1:nd
            v(j)=v(j)-m*(q0(j,:))';
        end
    end
end

end

function v=energia(s)
    % energia de deformacion y energia disipada

    M=Mp*s(2:2+NPR-1); % momentos reales
    q=Mp*L/(vE*Iz)*s(2+NPR:2+2*NPR-1); % giros acumulados en
    la RPs
    %         for i=1:NPR

```

```

%             if(abs(M(i))<Mp)
%                 q(i)=0.0;
%             end
%         end

v=0;
for i=1:nb
    ij=barras{i}{1};
    Li=norm(nodos{i+1}{1}-nodos{i}{1});
    vEi=material{barras{i}{2}(1)}{1};
    Izi=perfil{barras{i}{2}(2)}{1};
    Ma=M(ij(1)); Mb=M(ij(2));
    ma=Ma; mb=Mb;
    v=v+Li/(6*vEi*Izi)*(ma*(2*Ma+Mb)+mb*(Ma+2*Mb));
end
for i=1:NPR
    v=v+M(i)*q(i);
end

v=-v;

end

function [c,ceq]=restricciones(s)

    lambda=s(1); % factor de carga
    M=Mp*s(2:2+NPR-1); % momentos reales
    q=Mp*L/(vE*Iz)*s(2+NPR:2+2*NPR-1); % giros acumulados en
la RPs
%         for i=1:NPR
%             if(abs(M(i))<Mp)
%                 q(i)=0.0;
%             end
%         end
%
ceq=zeros(1,NPR);

% aplicar el Principio de los Desplazamientos Virtuales
(PDV)
ceq(1:EQ)=PDV(di); % ecuaciones de equilibrio

% aplicar PFV sistematico
ceq(EQ+1:NPR)=PFV(mi); % GH ecuaciones de compatibilidad

c=zeros(1,NPR);

for i=1:NPR
    c(i)=-M(i)*q(i);
end

function v=PDV(di)
% aplica el PDV -> obtener EQ ecuaciones de equilibrio

v=zeros(1,EQ);

```

```

        for j=1:EQ
            desp=zeros(2,NPR);
            desp=reshape(di(j, :, :), [2, NPR]);
            for i=1:nd
                v(j)=v(j)-M(i)*qi(j,i);
                kk=nodos{i}{2};
                if kk{1}~='Fx'

v(j)=v(j)+lambda*(desp(1,i)*kk{1}+desp(2,i)*kk{2});
                    end
                end
            end

        end

        function v=PFV(mi)
            % aplica el PFV -> obtener GH ecuaciones de
            compatibilidad

            v=zeros(1,GH);
            for j=1:GH
                for i=1:nb
                    ij=barras{i}{1};
                    Li=norm(nodos{i+1}{1}-nodos{i}{1});
                    vEi=material{barras{i}{2}(1)}{1};
                    Izi=perfil{barras{i}{2}(2)}{1};
                    Ma=M(ij(1)); Mb=M(ij(2));
                    ma=mi(j,ij(1)); mb=mi(j,ij(2));

v(j)=v(j)+Li/(6*vEi*Izi)*(ma*(2*Ma+Mb)+mb*(Ma+2*Mb));
                    end
                    for i=1:NPR
                        v(j)=v(j)+mi(j,i)*q(i);
                    end
                end
            end

        end

        function v=energiaDisipada(s)
            % energia de deformacion y energia disipada

            q=zeros(1,NPR); % giros acumulados en la RPs
            q([1 3 4 5])=s(:);

            v=0;
            for i=1:NPR
                v=v+M(i)*q(i);
            end

        end

        function [c,ceq]=restricciones2(s)

            q=zeros(1,NPR); % giros acumulados en la RPs
            q([1 3 4 5])=s(:);

```



```

ceq=zeros(1,GH);

% aplicar PFV sistematico
ceq=PFV(mi); % GH ecuaciones de compatibilidad

c=zeros(1,NPR);

for i=1:NPR
    c(i)=-M(i)*q(i);
end

function v=PDV(di)
% aplica el PDV -> obtener EQ ecuaciones de equilibrio

v=zeros(1,EQ);
for j=1:EQ
    desp=zeros(2,NPR);
    desp=reshape(di(j,:,:),[2,NPR]);
    for i=1:nd
        v(j)=v(j)-M(i)*qi(j,i);
        kk=nodos{i}{2};
        if kk{1}~='Fx'
            v(j)=v(j)+lambda*(desp(1,i)*kk{1}+desp(2,i)*kk{2});
        end
    end
end

end

function v=PFV(mi)
% aplica el PFV -> obtener GH ecuaciones de
compatibilidad

v=zeros(1,GH);
for j=1:GH
    for i=1:nb
        ij=barras{i}{1};
        Li=norm(nodos{i+1}{1}-nodos{i}{1});
        vEi=material{barras{i}{2}(1)}{1};
        Izi=perfil{barras{i}{2}(2)}{1};
        Ma=M(ij(1)); Mb=M(ij(2));
        ma=mi(j,ij(1)); mb=mi(j,ij(2));

v(j)=v(j)+Li/(6*vEi*Izi)*(ma*(2*Ma+Mb)+mb*(Ma+2*Mb));
    end
    for i=1:NPR
        v(j)=v(j)+mi(j,i)*q(i);
    end
end

end

```

end

end

## ANEJO 2. Código ejemplo 2.

```
function Ejemplo_1_v1
% Cálculo Plástico (CP) / Análisis Límite mediante el Método
Directo y
% Teoría de Mecanismos (TM) empotrado apoyado

% Datos
L=4.0;
F=1.0*10^3;
vE=2.1*10^11;
Iz=8360.0*10^-8;
S=628.0*10^-6;
sigma_F=275*10^6;
Mp=S*sigma_F;

% Geometría - Ejemplo 1 (sistematizar)
nodos = {[0, 0], {'Fx', 'Fy', 'Mz'}}, % empotramiento
        {[0, L], {0, 0, 0}},
        {[L, L], {0, -F, 0}},
        {[2*L, L], {F/6, 0, 0}},
        {[2*L, 0], {'Fx', 'Fy', 0}} % apoyo fijo
        ]; % coordenadas / fuerzas
material = {[vE]}; % una lista por material
perfil = {[Iz, Mp]}; % una lista por tipo de perfil
barras = {[1, 2], [1, 1]},
        {[2, 3], [1, 1]},
        {[3, 4], [1, 1]},
        {[4, 5], [1, 1]}
        ]; % definicion, propiedades(material,perfil)

nd=length(nodos);
nb=length(barras);

% numero de ecuaciones de equilibrio
NPR=nd;
GH=0;
for i=1:nd
    kk1=nodos{i}{2}{1};
    kk3=nodos{i}{2}{3};
    if (kk1=='Fx') & (kk3=='Mz')
        GH=GH+3; % empotramiento
    elseif (kk1=='Fx') & (kk3==0)
        GH=GH+2; % apoyo fijo
        NPR=NPR-1;
    end
end

GH=GH-3;
EQ=NPR-GH;

fprintf('Ecuaciones de equilibrio = %d \n',EQ);
fprintf('Ecuaciones de compatibilidad = %d \n',GH);
```

```

% mecanismos propuestos / ecuaciones de equilibrio

dv=0.0001;
mt=zeros(nd,3,3); % matrices Ti
mmT=zeros(3,3); % matriz T

qi=zeros(EQ,nd); % giros virtuales
di=zeros(EQ,2,nd); % desplazamientos virtuales
Mi=zeros(1,nd); % momentos reales
Mi(1:end-1)=ones(1,NPR); % (sistematizar)

% mecanismo (MC1)
q0=[0 1 1 1 0];
options = optimoptions('fsolve','Display','iter','TolFun',1e-30,'TolX',1e-50);
[q,fun]=fsolve(@posicion,dv*q0,options);
qi(1,:)=q;

% desplazamiento de los nodos
di(1,:,:)=delta(q);

% mecanismo (MC2)
q0=[1 1 0 1 1];
[q,fun]=fsolve(@posicion,dv*q0,options);
qi(2,:)=q;

di(2,:,:)=delta(q);

qi(:,5)=0.0; % apoyo fijo

% plantear las ecuaciones de compatibilidad (PFV) - (sistematizar)

mi=zeros(GH,nd); % momentos virtuales
m0=[0 1 0 0 0]; % EC1
[m,fun]=fsolve(@momentosVirtuales,m0,options);
mi(1,:)=m;

m0=[1 0 0 0 0]; % EC2
[m,fun]=fsolve(@momentosVirtuales,m0,options);
mi(2,:)=m;

% ecuacion de la energia (funcion a minimizar) + restricciones
s0=zeros(1,EQ+GH);

options = optimoptions('fsolve','Display','iter','TolFun',1e-30,'TolX',1e-50);
[s,fval]=fsolve(@ecuaciones,s0,options);

M(1:4)=s;

fprintf('Momento flector en "a": %5.2f \n',M(1));
fprintf('Momento flector en "b": %5.2f \n',M(2));
fprintf('Momento flector en "c": %5.2f \n',M(3));
fprintf('Momento flector en "d": %5.2f \n',M(4));
fprintf('Momento flector en "e": %5.2f \n',M(5));

```

```

fprintf('\n');

fprintf('Giro en "a": %2.5f \n',q(1));
fprintf('Giro en "b": %2.5f \n',q(2));
fprintf('Giro en "c": %2.5f \n',q(3));
fprintf('Giro en "d": %2.5f \n',q(4));
fprintf('Giro en "e": %2.5f \n',q(5));

function v=posicion(q)

    q=q.*q0;
    kk=matrizT(q);
    vP=cat(1, kk(1:2,3), kk(2,1));
    v0=cat(2, nodos{nd}{1}, 0.0);
    v=v0'-vP;

end

function v=matrizT(q)

    lx=0;
    ly=0;
    i=1;
    Li=norm(nodos{i+1}{1}-nodos{i}{1});
    v=mT(q(1), Li, lx, ly);
    mt(1, :, :) = v;

    for i=1:nb
        Li=norm(nodos{i+1}{1}-nodos{i}{1});
        kk=(nodos{i+1}{1}-nodos{i}{1})/Li;
        lx=kk(1);
        ly=kk(2);
        kk=mT(q(i+1), Li, lx, ly);
        mt(i+1, :, :) = kk;
        v=v*kk;

    end

    mmT=v;

function v=mT(x, Li, lx, ly)
    v=[ [1, -x, Li*lx]
        [x, 1, Li*ly]
        [0, 0, 1]
        ];

end

end

function v=delta(q)

    v=zeros(2, nd);

    for i=1:nd
        kk1=diag(ones(1,3));

```

```

        for j=1:i
            kk1=kk1*reshape(mt(j, :, :), [3, 3]);
        end
        kk=kk1(1:2, 3);
        kk2=nodos{i}{1}';
        v(:, i)=kk-kk2;
    end
end

function v=momentosVirtuales(m)
    % calcula los momentos virtuales (mi)

    v=zeros(1, EQ);
    q0=qi;
    for j=1:EQ
        q0(j, :)=q0(j, :)/max(q0(j, :));
        for i=1:nd
            v(j)=v(j)-m*(q0(j, :)' );
        end
    end
end

end

function v=ecuaciones(s)

    M=zeros(1, nd);
    M(1:4)=s(:); % momentos reales

    v=zeros(1, EQ+GH);

    % aplicar el Principio de los Desplazamientos Virtuales
(PDV)
    v(1:EQ)=PDV(di); % ecuaciones de equilibrio

    % aplicar PFV sistematico
    v(EQ+1:end)=PFV(mi); % GH ecuaciones de compatibilidad

    function v=PDV(di)
        % aplica el PDV -> obtener EQ ecuaciones de equilibrio

        v=zeros(1, EQ);
        for j=1:EQ
            desp=zeros(2, nd);
            desp=reshape(di(j, :, :), [2, nd]);
            for i=1:nd
                v(j)=v(j)-M(i)*qi(j, i);
                kk=nodos{i}{2};
                if kk{1}~='Fx'
                    v(j)=v(j)+(desp(1, i)*kk{1}+desp(2, i)*kk{2});
                end
            end
        end
    end

    function v=PFV(mi)

```

```

% aplica el PFV -> obtener GH ecuaciones de
compatibilidad

v=zeros(1,GH);
for j=1:GH
    for i=1:nb
        ij=barras{i}{1};
        Li=norm(nodos{i+1}{1}-nodos{i}{1});
        vEi=material{barras{i}{2}(1)}{1};
        Izi=perfil{barras{i}{2}(2)}{1};
        Ma=M(ij(1)); Mb=M(ij(2));
        ma=mi(j,ij(1)); mb=mi(j,ij(2));

v(j)=v(j)+Li/(6*vEi*Izi)*(ma*(2*Ma+Mb)+mb*(Ma+2*Mb));
    end
    for i=1:NPR
        v(j)=v(j)+mi(j,i)*q(i);
    end
end

end

end

end

```

## ANEJO 3. Código ejemplo 3.

```
function CPbyTM_v17
% Cálculo Plástico (CP) / Análisis Límite mediante el Método
Directo y
% Teoría de Mecanismos (TM)
% Carga distribuida (p) y puntual (F)

% Datos
L=1.0;
F=0*1.0*10^5;
vE=2.1*10^11;
Iz=8360.0*10^-8;
S=628.0*10^-6;
sigma_F=275*10^6;
Mp=S*sigma_F;
p=1000.0;

x0=0.5*3*L*ones(1,1);

lb1=zeros(1,1);
ub1=3*L*ones(1,1);

NPR=0; GH=0; EQ=0; mi=0; nb=0; barras=0; nodos=0; material=0;
perfil=0;

s=ones(1,1+2*5);

options = optimoptions('fmincon','Display','iter','TolFun',1e-
12,'TolX',1e-6);
[x,fval]=fmincon(@cargaDistriuida,x0,[],[],[],[],lb1,ub1,[],option
s);

lambda=s(1);
M=Mp*s(2:1+NPR);

% calcular los giros
s0=ones(1,NPR-1);

[s,fval]=fmincon(@energiaDisipada,s0,[],[],[],[],[],[],@restriccio
nes2,options);

q=zeros(1,NPR);
q([1 2 4 5])=s(:);

fprintf('x: %4.2f \n',x);

fprintf('Carga de colapso: %4.2f \n',p*lambda);
fprintf('\n');
fprintf('Momento flector en "a": %4.2f \n',M(1));
fprintf('Momento flector en "b": %4.2f \n',M(2));
fprintf('Momento flector en "c": %4.2f \n',M(3));
fprintf('Momento flector en "d": %4.2f \n',M(4));
fprintf('Momento flector en "e": %4.2f \n',M(5));
```



```

fprintf('\n');
fprintf('Giro acumulado en "a": %2.5f \n',q(1));
fprintf('Giro acumulado en "b": %2.5f \n',q(2));
fprintf('Giro acumulado en "c": %2.5f \n',q(3));
fprintf('Giro acumulado en "d": %2.5f \n',q(4));
fprintf('Giro acumulado en "e": %2.5f \n',q(5));

kk=1;

function vl=cargaDistriuida(x)

% Geometría - Ejemplo 1 (sistematizar)
nodos = {[0, 0], {'Fx', 'Fy', 'Mz'}}, % empotramiento
        {[0, x(1)], {F, 0, 0}},
        {[0, 3*L], {0, 0, 0}},
        {[5*L, 3*L], {0, 0, 0}},
        {[5*L,0], {'Fx', 'Fy', 'Mz'}} % apoyo fijo
        ]; % coordenadas / fuerzas
material = {{vE}}; % una lista por material
perfil = {{Iz, Mp}}; % una lista por tipo de perfil
barras = {[1, 2], [1, 1], p},
         [2, 3], [1, 1], p},
         [3, 4], [1, 1], 0},
         [4, 5], [1, 1], 0}
        ]; % definicion, propiedades(material,perfil)

nd=length(nodos);
nb=length(barras);

% numero de ecuaciones de equilibrio
NPR=nd;
GH=0;
for i=1:nd
    kk1=nodos{i}{2}{1};
    kk3=nodos{i}{2}{3};
    if (kk1=='Fx') & (kk3=='Mz')
        GH=GH+3; % empotramiento
    elseif (kk1=='Fx') & (kk3==0)
        GH=GH+2; % apoyo fijo
        NPR=NPR-1;
    end
end

GH=GH-3;
EQ=NPR-GH;

fprintf('Ecuaciones de equilibrio = %d \n',EQ);
fprintf('Ecuaciones de compatibilidad = %d \n',GH);

% mecanismos propuestos / ecuaciones de equilibrio

dv=0.0001;
mt=zeros(nd,3,3); % matrices Ti
mmT=zeros(3,3); % matriz T

qi=zeros(EQ,nd); % giros virtuales

```

```

di=zeros(EQ,2,nd); % desplazamientos virtuales
Mi=ones(1,nd); % momentos reales

% mecanismo (MC1)
% q0=[1 1 1 0 0];
q0=[1 1 1 0 0];
options = optimoptions('fsolve','Display','iter','TolFun',1e-
30,'TolX',1e-50);
[q,fun]=fsolve(@posicion,dv*q0,options);
qi(1,:)=q;

% desplazamiento de los nodos
di(1,:,:)=delta(q);

% mecanismo (MC2)
% q0=[1 0 1 1 1];
q0=[1 0 1 1 1];
[q,fun]=fsolve(@posicion,dv*q0,options);
qi(2,:)=q;

di(2,:,:)=delta(q);

% plantear las ecuaciones de compatibilidad (PFV) -
(sistematizar)

mi=zeros(GH,nd); % momentos virtuales
m0=[1 0 0 0 1]; % EC1
[m,fun]=fsolve(@momentosVirtuales,m0,options);
mi(1,:)=m;

m0=[1 0 0 0 0]; % EC2
[m,fun]=fsolve(@momentosVirtuales,m0,options);
mi(2,:)=m;

m0=[0 0 1 0 0]; % EC3
[m,fun]=fsolve(@momentosVirtuales,m0,options);
mi(3,:)=m;

% (sistematizar)

% ecuacion de la energia (funcion a maximizar) + restricciones
s0=ones(1,1+2*NPR);
s0(1)=100;
s0(2+NPR:end)=ones(1,NPR);

lb=zeros(1,1+2*NPR);
lb(2:2+NPR-1)=-1.0;
lb(2+NPR:end)=-75.0;

ub=zeros(1,1+2*NPR);
ub(1)=1000;
ub(2:2+NPR-1)=1.0;
ub(2+NPR:end)=75.0;

```

```
options = optimoptions('fmincon','Display','iter','TolFun',1e-12,'TolX',1e-6);
```

```
[s,fval]=fmincon(@energia,s0,[],[],[],[],lb,ub,@restricciones,options);
```

```
v1=s(1);
```

```
function v=posicion(q)
```

```
q=q.*q0;
kk=matrizT(q);
vP=cat(1, kk(1:2,3), kk(2,1));
v0=cat(2, nodos{nd}{1}, 0.0);
v=v0'-vP;
```

```
end
```

```
function v=matrizT(q)
```

```
lx=0;
ly=0;
i=1;
Li=norm(nodos{i+1}{1}-nodos{i}{1});
v=mT(q(1), Li, lx, ly);
mt(1, :, :) = v;

for i=1:nb
    Li=norm(nodos{i+1}{1}-nodos{i}{1});
    kk=(nodos{i+1}{1}-nodos{i}{1})/Li;
    lx=kk(1);
    ly=kk(2);
    kk=mT(q(i+1), Li, lx, ly);
    mt(i+1, :, :) = kk;
    v=v*kk;
end
```

```
end
```

```
mmT=v;
```

```
function v=mT(x, Li, lx, ly)
```

```
v=[1, -x, Li*lx]
    [x, 1, Li*ly]
    [0, 0, 1]
    ];
```

```
end
```

```
end
```

```
function v=delta(q)
```

```
v=zeros(2, nd);
```

```
for i=1:nd
    kk1=diag(ones(1,3));
```

```

        for j=1:i
            kk1=kk1*reshape(mt(j, :, :), [3, 3]);
        end
        kk=kk1(1:2, 3);
        kk2=nodos{i}{1}';
        v(:, i)=kk-kk2;
    end
end

function v=momentosVirtuales(m)
    % calcula los momentos virtuales (mi)

    v=zeros(1, EQ);
    q0=qi;
    for j=1:EQ
        q0(j, :)=q0(j, :)/max(q0(j, :));
        for i=1:nd
            v(j)=v(j)-m*(q0(j, :)' );
        end
    end
end

end

function v=energia(s)
    % energia de deformacion y energia disipada

    lambda=s(1); % factor de carga
    M=Mp*s(2:2+NPR-1); % momentos reales - (sistematizar)
    q=Mp*L/(vE*Iz)*s(2+NPR:end); % giros acumulados en la

RPs

    v=0;
    for i=1:nb
        ij=barras{i}{1};
        Li=norm(nodos{i+1}{1}-nodos{i}{1});
        vEi=material{barras{i}{2}(1)}{1};
        Izi=perfil{barras{i}{2}(2)}{1};
        Ma=M(ij(1)); Mb=M(ij(2));
        ma=Ma; mb=Mb;
        p0=barras{i}{3};
        v=v+Li/(6*vEi*Izi)*(ma*(2*Ma+Mb)+mb*(Ma+2*Mb)+...

lambda*p0*Li^2*(ma+mb+Ma+Mb)/4+(lambda*p0)^2*Li^4/20);
    end
    for i=1:nd
        v=v+M(i)*q(i);
    end

    v=-v;

end

function [c, ceq]=restricciones(s)

```

```

        lambda=s(1); % factor de carga
        M=Mp*s(2:2+NPR-1); % momentos reales - (sistematizar)
        q=Mp*L/(vE*Iz)*s(2+NPR:2+2*NPR-1); % giros acumulados
en la RPs
        ceq=zeros(1,NPR);

        % aplicar el Principio de los Desplazamientos
Virtuales (PDV)
        ceq(1:EQ)=PDV(di); % ecuaciones de equilibrio

        % aplicar PFV sistematico
        ceq(EQ+1:NPR)=PFV(mi); % GH ecuaciones de
compatibilidad

        c=zeros(1,NPR);

        for i=1:NPR
            c(i)=-M(i)*q(i);
        end

        function v=PDV(di)
        % aplica el PDV -> obtener EQ ecuaciones de equilibrio

        v=zeros(1,EQ);
        for j=1:EQ
            desp=zeros(2,nd);
            desp=reshape(di(j,:,:),[2,nd]);
            for i=1:nd % nudos
                v(j)=v(j)-M(i)*qi(j,i);
                kk=nodos{i}{2};
                if kk{1}~='Fx'
v(j)=v(j)+lambda*(desp(1,i)*kk{1}+desp(2,i)*kk{2});
                end
            end
            for i=1:nb % barras
                ij=barras{i}{1};
                Li=norm(nodos{i+1}{1}-nodos{i}{1});
                p0=barras{i}{3};
                a=desp(1,ij(1));
                b=(desp(1,ij(2))-desp(1,ij(1)))/Li;
                d=Li*(a+b*Li/2); % integral del
desplazamiento
                v(j)=v(j)+lambda*p0*d;
            end
        end

        end

        function v=PFV(mi)
        % aplica el PFV -> obtener GH ecuaciones de
compatibilidad

        v=zeros(1,GH);
        for j=1:GH
            for i=1:nb
                ij=barras{i}{1};

```

```

        Li=norm(nodos{i+1}{1}-nodos{i}{1});
        vEi=material{barras{i}{2}{1}}{1};
        Izi=perfil{barras{i}{2}{2}}{1};
        Ma=M(ij(1)); Mb=M(ij(2));
        ma=mi(j,ij(1)); mb=mi(j,ij(2));
        p0=barras{i}{3};

v(j)=v(j)+Li/(6*vEi*Izi)*(ma*(2*Ma+Mb)+mb*(Ma+2*Mb)+lambda*p0*Li^2
*(ma+mb)/4);

        end
        for i=1:NPR
            v(j)=v(j)+mi(j,i)*q(i);
        end
    end

end

end

end

function v=energiaDisipada(s)
    % energia de deformacion y energia disipada

    q=zeros(1,NPR); % giros acumulados en la RPs
    q([1 2 4 5])=s(:);

    v=0;
    for i=1:NPR
        v=v+M(i)*q(i);
    end

end

function [c,ceq]=restricciones2(s)

    q=zeros(1,NPR); % giros acumulados en la RPs
    q([1 2 4 5])=s(:);

    ceq=zeros(1,GH);

    % aplicar PFV sistematico
    ceq=PFV(mi); % GH ecuaciones de compatibilidad

    c=zeros(1,NPR);

    for i=1:NPR
        c(i)=-M(i)*q(i);
    end

    function v=PFV(mi)
        % aplica el PFV -> obtener GH ecuaciones de
        compatibilidad

        v=zeros(1,GH);

```

```

        for j=1:GH
            for i=1:nb
                ij=barras{i}{1};
                Li=norm(nodos{i+1}{1}-nodos{i}{1});
                vEi=material{barras{i}{2}(1)}{1};
                Izi=perfil{barras{i}{2}(2)}{1};
                Ma=M(ij(1)); Mb=M(ij(2));
                ma=mi(j,ij(1)); mb=mi(j,ij(2));
                p0=barras{i}{3};

v(j)=v(j)+Li/(6*vEi*Izi)*(ma*(2*Ma+Mb)+mb*(Ma+2*Mb)+lambda*p0*Li^2
*(ma+mb)/4);
            end
            for i=1:NPR
                v(j)=v(j)+mi(j,i)*q(i);
            end
        end

    end

end

end

```

## ANEJO 4. Código Ejemplo 4.

```
function CPbyTM_v20
% Cálculo Plástico (CP) / Análisis Límite mediante el Método
Directo y
% Teoría de Mecanismos (TM)
% Carga distribuida (p) y puntual (F)
% Pórtico a 2 aguas

warning('off','all');

% Datos
Lp=4.0;
Ld=6.0;
F=1.0*10^3;
vE=2.1*10^11;
Iz=8360.0*10^-8;
S=628.0*10^-6;
sigma_F=275.0*10^6;
Mp=S*sigma_F;
p=1000.0;
alpha=10.0*pi/180; % grados

kk=cos(alpha);

% Geometría
nodos = {[0, 0], {'Fx', 'Fy', 'Mz'}}, % empotramiento
        {[0, Lp/2], {0, 0, 0}},
        {[0, Lp], {0, 0, 0}},
        {[Ld/2, (Lp+Ld/2*tan(alpha))], {0, 0, 0}},
        {[Ld, (Lp+Ld*tan(alpha))], {0, -F, 0}},
        {[2*Ld, Lp], {F, 0, 0}},
        {[2*Ld, 0], {'Fx', 'Fy', 'Mz'}}
        ]; % coordenadas / fuerzas
material = {[vE]}; % una lista por material
perfil = {[Iz, Mp]}; % una lista por tipo de perfil
barras = {[1, 2], [1, 1], p},
         {[2, 3], [1, 1], p},
         {[3, 4], [1, 1], p},
         {[4, 5], [1, 1], p},
         {[5, 6], [1, 1], 0},
         {[6, 7], [1, 1], 0},
        ]; % definicion, propiedades(material,perfil)

nd=length(nodos);
nb=length(barras);

lb1=zeros(1,2);
ub1=[Lp Ld/cos(alpha)];

x0=ub1/2;

NPR=0; GH=0; EQ=0; mi=0; s=0;
```



```

%                                options                                =
optimoptions('fmincon','Display','iter','MaxFunctionEvaluations',5
000,'TolFun',1e-12,'TolX',1e-10);
options = optimoptions('fmincon','Display','off','TolFun',1e-
12,'TolX',1e-6);
[x,fval]=fmincon(@cargaDistriuida,x0,[],[],[],[],lb1,ub1,[],option
s);

lambda=s(1);
M=Mp*s(2:1+NPR);

RPs=[]; % rotulas plasticas que se forman
for i=1:length(M)
    if abs(abs(M(i))-Mp)<0.01
        RPs=cat(2,RPs,i);
    end
end

% calcular los giros
s0=ones(1,GH+1);

[s,fval]=fmincon(@energiaDisipada,s0,[],[],[],[],[],[],@restriccio
nes2,options);

q=zeros(1,NPR);
q(RPs)=s(:);

fprintf('Factor de carga de colapso: %4.2f \n',lambda);
fprintf('\n');
fprintf('Momento flector \n');
fprintf('    a    /    b    /    c    /    d    /    e    /    f    /    g
\n');
fprintf('%4.2f / %4.2f / %4.2f / %4.2f / %4.2f / %4.2f / %4.2f
\n',M);
fprintf('\n');
fprintf('Giro acumulado \n');
fprintf('    a    /    b    /    c    /    d    /    e    /    f    /    g
\n');
fprintf('%2.5f / %2.5f / %2.5f / %2.5f / %2.5f / %2.5f / %2.5f
\n',q);

kk=1;

function vl=cargaDistriuida(x)

% Geometría
nodos = {[0, 0], {'Fx', 'Fy','Mz'}}, % empotramiento
        {[0, x(1)], {0, 0, 0}},
        {[0, Lp], {0, 0, 0}},
        {[x(2)*cos(alpha), Lp+x(2)*sin(alpha)], {0, 0, 0}},
        {[Ld, (Lp+Ld*tan(alpha))], {0, -F, 0}},
        {[2*Ld, Lp], {F, 0, 0}},
        {[2*Ld,0], {'Fx', 'Fy', 'Mz'}} % apoyo fijo
    }; % coordenadas / fuerzas
material = {{vE}}; % una lista por material
perfil = {{Iz, Mp}}; % una lista por tipo de perfil

```

```

barras = {[1, 2], [1, 1], p},
          {[2, 3], [1, 1], p},
          {[3, 4], [1, 1], p},
          {[4, 5], [1, 1], p},
          {[5, 6], [1, 1], 0},
          {[6, 7], [1, 1], 0}
]; % definicion, propiedades(material,perfil)

nd=length(nodos);
nb=length(barras);

% numero de ecuaciones de equilibrio
NPR=nd;
GH=0;
for i=1:nd
    kk1=nodos{i}{2}{1};
    kk3=nodos{i}{2}{3};
    if (kk1=='Fx') & (kk3=='Mz')
        GH=GH+3; % empotramiento
    elseif (kk1=='Fx') & (kk3==0)
        GH=GH+2; % apoyo fijo
        NPR=NPR-1;
    end
end

GH=GH-3;
EQ=NPR-GH;

% fprintf('Ecuaciones de equilibrio = %d \n',EQ);
% fprintf('Ecuaciones de compatibilidad = %d \n',GH);

% mecanismos propuestos / ecuaciones de equilibrio

dv=0.0001;
mt=zeros(nd,3,3); % matrices Ti
mmT=zeros(3,3); % matriz T

qi=zeros(EQ,nd); % giros virtuales
di=zeros(EQ,2,nd); % desplazamientos virtuales
Mi=ones(1,nd); % momentos reales

% mecanismo (MC1)
q0=[1 1 1 0 0 0 0];
% options
optimoptions('fsolve','Display','iter','TolFun',1e-30,'TolX',1e-50);
options = optimoptions('fsolve','Display','off','TolFun',1e-30,'TolX',1e-50);
[q,fun]=fsolve(@posicion,dv*q0,options);
qi(1,:)=q;

% desplazamiento de los nodos
di(1,:,:)=delta(q);

% mecanismo (MC2)
q0=[0 0 1 1 1 0 0];

```

```

[q,fun]=fsolve(@posicion,dv*q0,options);
qi(2,:)=q;

di(2,,:)=delta(q);

% mecanismo (MC3)
q0=[1 0 1 0 1 0 1];
[q,fun]=fsolve(@posicion,dv*q0,options);
qi(3,:)=q;

di(3,,:)=delta(q);

% mecanismo (MC4)
q0=[0 0 0 1 1 1 1];
[q,fun]=fsolve(@posicion,dv*q0,options);
qi(4,:)=q;

di(4,,:)=delta(q);

% plantear las ecuaciones de compatibilidad (PFV) -
(sistematizar)

mi=zeros(GH,nd); % momentos virtuales
m0=[1 0 0 0 1 0 0]; % EC1
[m,fun]=fsolve(@momentosVirtuales,m0,options);
mi(1,:)=m;

m0=[1 0 0 0 0 0 0]; % EC2
[m,fun]=fsolve(@momentosVirtuales,m0,options);
mi(2,:)=m;

m0=[0 0 1 0 0 0 0]; % EC3
[m,fun]=fsolve(@momentosVirtuales,m0,options);
mi(3,:)=m;

% (sistematizar)

% ecuacion de la energia (funcion a maximizar) + restricciones
s0=ones(1,1+2*NPR);
s0(1)=100;
s0(2+NPR:end)=ones(1,NPR);

lb=zeros(1,1+2*NPR);
lb(2:2+NPR-1)=-1.0;
lb(2+NPR:end)=-125.0;

ub=zeros(1,1+2*NPR);
ub(1)=1000;
ub(2:2+NPR-1)=1.0;
ub(2+NPR:end)=125.0;

options = optimoptions('fmincon','Display','off','TolFun',1e-
16,'TolX',1e-10);

[s,fval]=fmincon(@energia,s0,[],[],[],[],lb,ub,@restricciones,opti
ons);

```

```

v1=s(1);

function v=posicion(q)

    q=q.*q0;
    kk=matrizT(q);
    vP=cat(1, kk(1:2,3), kk(2,1));
    v0=cat(2, nodos{nd}{1}, 0.0);
    v=v0'-vP;

end

function v=matrizT(q)

    lx=0;
    ly=0;
    i=1;
    Li=norm(nodos{i+1}{1}-nodos{i}{1});
    v=mT(q(1), Li, lx, ly);
    mt(1, :, :)=v;

    for i=1:nb
        Li=norm(nodos{i+1}{1}-nodos{i}{1});
        kk=(nodos{i+1}{1}-nodos{i}{1})/Li;
        lx=kk(1);
        ly=kk(2);
        kk=mT(q(i+1), Li, lx, ly);
        mt(i+1, :, :)=kk;
        v=v*kk;
    end

    mmT=v;

function v=mT(x, Li, lx, ly)
    v=[ [1, -x, Li*lx]
        [x, 1, Li*ly]
        [0, 0, 1]
        ];

end

end

function v=delta(q)

    v=zeros(2, nd);

    for i=1:nd
        kk1=diag(ones(1,3));
        for j=1:i
            kk1=kk1*reshape(mt(j, :, :), [3,3]);
        end
        kk=kk1(1:2,3);
        kk2=nodos{i}{1}';

```

```

        v(:,i)=kk-kk2;
    end
end

function v=momentosVirtuales(m)
    % calcula los momentos virtuales (mi)

    v=zeros(1,EQ);
    q0=qi;
    for j=1:EQ
        q0(j,:)=q0(j,:)/max(q0(j,:));
        for i=1:nd
            v(j)=v(j)-m*(q0(j,:)');
        end
    end

end

function v=energia(s)
    % energia de deformacion y energia disipada

    lambda=s(1); % factor de carga
    M=Mp*s(2:2+NPR-1); % momentos reales - (sistematizar)
    q=Mp*Lp/(vE*Iz)*s(2+NPR:2+2*NPR-1); % giros acumulados
    en la RPs

    v=0;
    for i=1:nb
        ij=barras{i}{1};
        Li=norm(nodos{i+1}{1}-nodos{i}{1});
        vEi=material{barras{i}{2}(1)}{1};
        Izi=perfil{barras{i}{2}(2)}{1};
        Ma=M(ij(1)); Mb=M(ij(2));
        ma=Ma; mb=Mb;
        p0=barras{i}{3};
        v=v+Li/(6*vEi*Izi)*(ma*(2*Ma+Mb)+mb*(Ma+2*Mb)+...
        lambda*p0*Li^2*(ma+mb+Ma+Mb)/4+(lambda*p0)^2*Li^4/20);
    end
    for i=1:nd
        v=v+M(i)*q(i);
    end

    v=-v;

end

function [c,ceq]=restricciones(s)

    lambda=s(1); % factor de carga
    M=Mp*s(2:2+NPR-1); % momentos reales - (sistematizar)
    q=Mp*Lp/(vE*Iz)*s(2+NPR:2+2*NPR-1); % giros acumulados
    en la RPs
    ceq=zeros(1,NPR);

    % aplicar el Principio de los Desplazamientos
    Virtuales (PDV)

```

```

ceq(1:EQ)=PDV(di); % ecuaciones de equilibrio

% aplicar PFV sistematico
ceq(EQ+1:NPR)=PFV(mi); % GH ecuaciones de
compatibilidad

c=zeros(1,NPR);

for i=1:NPR
    c(i)=-M(i)*q(i);
end

function v=PDV(di)
% aplica el PDV -> obtener EQ ecuaciones de equilibrio

v=zeros(1,EQ);
for j=1:EQ
    desp=zeros(2,nd);
    desp=reshape(di(j,:,:),[2,nd]);
    for i=1:nd % nudos
        v(j)=v(j)-M(i)*qi(j,i);
        kk=nodos{i}{2};
        if kk{1}~='Fx'

v(j)=v(j)+lambda*(desp(1,i)*kk{1}+desp(2,i)*kk{2});
            end
        end
        for i=1:nb % barras
            ij=barras{i}{1};
            Li=norm(nodos{i+1}{1}-nodos{i}{1});
            kk=(nodos{i+1}{1}-nodos{i}{1})/Li;
            lx=kk(1);
            ly=kk(2);
            p0=barras{i}{3};

            a=desp(1,ij(1)); % horizontal
            b=(desp(1,ij(2))-desp(1,ij(1)))/Li;
            d=Li*(a+b*Li/2); % integral del
desplazamiento

            v(j)=v(j)+lambda*ly*p0*d;

            a=desp(2,ij(1)); % vertical
            b=(desp(2,ij(2))-desp(2,ij(1)))/Li;
            d=Li*(a+b*Li/2); % integral del
desplazamiento

            v(j)=v(j)-lambda*lx*p0*d;
        end
    end

end

function v=PFV(mi)
% aplica el PFV -> obtener GH ecuaciones de
compatibilidad

```

```

v=zeros(1,GH);
for j=1:GH
    for i=1:nb
        ij=barras{i}{1};
        Li=norm(nodos{i+1}{1}-nodos{i}{1});
        vEi=material{barras{i}{2}(1)}{1};
        Izi=perfil{barras{i}{2}(2)}{1};
        Ma=M(ij(1)); Mb=M(ij(2));
        ma=mi(j,ij(1)); mb=mi(j,ij(2));
        p0=barras{i}{3};

v(j)=v(j)+Li/(6*vEi*Izi)*(ma*(2*Ma+Mb)+mb*(Ma+2*Mb)+lambda*p0*Li^2
*(ma+mb)/4);

        end
        for i=1:NPR
            v(j)=v(j)+mi(j,i)*q(i);
        end
    end

end

end

end

function v=energiaDisipada(s)
    % energia de deformacion y energia disipada

    q=zeros(1,NPR); % giros acumulados en la RPs
    q(RPs)=s(:);

    v=0;
    for i=1:NPR
        v=v+M(i)*q(i);
    end

end

function [c,ceq]=restricciones2(s)

    q=zeros(1,NPR); % giros acumulados en la RPs
    q(RPs)=s(:);

    ceq=zeros(1,GH);

    % aplicar PFV sistematico
    ceq=PFV(mi); % GH ecuaciones de compatibilidad

    c=zeros(1,NPR);

    for i=1:NPR
        c(i)=-M(i)*q(i);
    end

```

```

function v=PFV(mi)
    % aplica el PFV -> obtener GH ecuaciones de
    compatibilidad

    v=zeros(1,GH);
    for j=1:GH
        for i=1:nb
            ij=barras{i}{1};
            Li=norm(nodos{i+1}{1}-nodos{i}{1});
            vEi=material{barras{i}{2}(1)}{1};
            Izi=perfil{barras{i}{2}(2)}{1};
            Ma=M(ij(1)); Mb=M(ij(2));
            ma=mi(j,ij(1)); mb=mi(j,ij(2));
            p0=barras{i}{3};

            v(j)=v(j)+Li/(6*vEi*Izi)*(ma*(2*Ma+Mb)+mb*(Ma+2*Mb)+lambda*p0*Li^2
            *(ma+mb)/4);
        end
        for i=1:NPR
            v(j)=v(j)+mi(j,i)*q(i);
        end
    end

end

end
end
end

```



