ANEXO:

casosPrueba.m

```
clc;
clear;

fprintf("-------------Consistente-------------\n");
w = [0.4 0.3 0.2 0.1]
M= zeros(4);
for i = 1:4
    for j = 1:4
        M(i,j) = w(i)/w(j);
    end
end

M
ic(M)

fprintf("-------------Metodo de la potencia-------------\n");
w0 = funciones(0, M) % potencia(M)
[errorInf, IndexMaxErr, errorFro, errorUno,  errorNoAciertos] = errores(w0, M)

fprintf("-------------Metodo de Minimos Cuadrados [Log]-------------\n");
w1 = funciones(1, M) % minCuadLog(M)
[errorInf, IndexMaxErr, errorFro, errorUno,  errorNoAciertos] = errores(w1, M)

fprintf("-------------Metodo de Minimos Cuadrados [Ponderado]-------------\n");
w2 = funciones(2, M) % minCuadPond(M)
[errorInf, IndexMaxErr, errorFro, errorUno,  errorNoAciertos] = errores(w2, M)

fprintf("-------------Metodo de Minimo Suma Desviaciones [Log]-------------\n");
w3 = funciones(3, M) % minSumDesvLog(M)
[errorInf, IndexMaxErr, errorFro, errorUno,  errorNoAciertos] = errores(w3, M)

fprintf("-------------Metodo de Minimo Suma Desviaciones [Ponderado]-------------\n");
w4 = funciones(4, M) % minSumDesvPond(M)
[errorInf, IndexMaxErr, errorFro, errorUno,  errorNoAciertos] = errores(w4, M)

%Dibujar pesos
figure();
c = categorical(["Potencia", "Min Cuadrado Log", "Min Cuadrado Pond", "Min Sum Desv Log", "Min Sum Desv
Pond"]);
bar(c, [w0';w1';w2';w3';w4']);
l=compose("A%d",(1:length(M)));
legend(l);

clear;
```

```
fprintf("-------------Consistente Varios Expertos-------------\n");
w = [0.4 0.3 0.2 0.1]
M= zeros(4);
for i = 1:4
    for j = 1:4
        M(i,j) = w(i)/w(j);
    end
end

M
ic(M)

fprintf("-------------Metodo de Minimos Cuadrados [Log]-------------\n");
w1 = funciones(1, M, M) % minCuadLog(M)
[errorInf, IndexMaxErr, errorFro, errorUno,  errorNoAciertos] = errores(w1, M, M)

fprintf("-------------Metodo de Minimos Cuadrados [Ponderado]-------------\n");
w2 = funciones(2, M, M) % minCuadPond(M)
[errorInf, IndexMaxErr, errorFro, errorUno,  errorNoAciertos] = errores(w2, M, M)

fprintf("-------------Metodo de Minimo Suma Desviaciones [Log]-------------\n");
w3 = funciones(3, M, M) % minSumDesvLog(M)
[errorInf, IndexMaxErr, errorFro, errorUno,  errorNoAciertos] = errores(w3, M, M)

fprintf("-------------Metodo de Minimo Suma Desviaciones [Ponderado]-------------\n");
w4 = funciones(4, M, M) % minSumDesvPond(M)
[errorInf, IndexMaxErr, errorFro, errorUno,  errorNoAciertos] = errores(w4, M, M)

%Dibujar pesos
figure();
c = categorical(["Min Cuadrado Log", "Min Cuadrado Pond", "Min Sum Desv Log", "Min Sum Desv Pond"]);
bar(c, [w1';w2';w3';w4']);
l=compose("A%d",(1:length(M)));
legend(l);

clear;
```

```
fprintf("-------------Consistente Incompleta-------------\n");
w = [0.4 0.3 0.2 0.1]
M= zeros(4);
for i = 1:4
    for j = 1:4
        M(i,j) = w(i)/w(j);
    end
end

%El experto no opina sobre la opcion 1
M(1,[3 4]) = 0;
M([3 4],1) = 0;

M
ic(M)

% M =
%
%         1     1.333         0         0
%      0.75    1.0000    1.5000    3.0000
%         0    0.6667    1.0000    2.0000
%         0    0.3333    0.5000    1.0000


fprintf("-------------Metodo de la potencia-------------\n");
w0 = funciones(0, M) % potencia(M)
[errorInf, IndexMaxErr, errorFro, errorUno,  errorNoAciertos] = errores(w0, M)

fprintf("-------------Metodo de Minimos Cuadrados [Log]-------------\n");
w1 = funciones(1, M) % minCuadLog(M)
[errorInf, IndexMaxErr, errorFro, errorUno,  errorNoAciertos] = errores(w1, M)

fprintf("-------------Metodo de Minimos Cuadrados [Ponderado]-------------\n");
w2 = funciones(2, M) % minCuadPond(M)
[errorInf, IndexMaxErr, errorFro, errorUno,  errorNoAciertos] = errores(w2, M)

fprintf("-------------Metodo de Minimo Suma Desviaciones [Log]-------------\n");
w3 = funciones(3, M) % minSumDesvLog(M)
[errorInf, IndexMaxErr, errorFro, errorUno,  errorNoAciertos] = errores(w3, M)

fprintf("-------------Metodo de Minimo Suma Desviaciones [Ponderado]-------------\n");
w4 = funciones(4, M) % minSumDesvPond(M)
[errorInf, IndexMaxErr, errorFro, errorUno,  errorNoAciertos] = errores(w4, M)

%Dibujar pesos
figure();
c = categorical(["Min Cuadrado Log", "Min Cuadrado Pond", "Min Sum Desv Log", "Min Sum Desv Pond"]);
bar(c, [w1';w2';w3';w4']);
l=compose("A%d",(1:length(M)));
legend(l);

clear;
```

```
fprintf("-------------NO Consistente-------------\n");
M = [1.0000   0.1429    0.1429    0.2000;
    7.0000    1.0000    0.5000    0.3333;
    7.0000    2.0000    1.0000    0.1111;
    5.0000    3.0000    9.0000    1.0000]

ic(M)

fprintf("-------------Metodo de la potencia-------------\n");
w0 = funciones(0, M) % potencia(M)
[errorInf, IndexMaxErr, errorFro, errorUno,  errorNoAciertos] = errores(w0, M)

fprintf("-------------Metodo de Minimos Cuadrados [Log]-------------\n");
w1 = funciones(1, M) % minCuadLog(M)
[errorInf, IndexMaxErr, errorFro, errorUno,  errorNoAciertos] = errores(w1, M)
fprintf("-------------Metodo de Minimos Cuadrados [Ponderado]-------------\n");
w2 = funciones(2, M) % minCuadPond(M)
[errorInf, IndexMaxErr, errorFro, errorUno,  errorNoAciertos] = errores(w2, M)

fprintf("-------------Metodo de Minimo Suma Desviaciones [Log]-------------\n");
w3 = funciones(3, M) % minSumDesvLog(M)
[errorInf, IndexMaxErr, errorFro, errorUno,  errorNoAciertos] = errores(w3, M)

fprintf("-------------Metodo de Minimo Suma Desviaciones [Ponderado]-------------\n");
w4 = funciones(4, M) % minSumDesvPond(M)
[errorInf, IndexMaxErr, errorFro, errorUno,  errorNoAciertos] = errores(w4, M)

%Dibujar pesos
figure();
c = categorical(["Potencia", "Min Cuadrado Log", "Min Cuadrado Pond", "Min Sum Desv Log", "Min Sum Desv
Pond"]);
bar(c, [w0';w1';w2';w3';w4']);
l=compose("A%d",(1:length(M)));
legend(l);
```

```
fprintf("-------------NO Consistente Incompleta Varios Expertos-------------\n");
M1 = [1    0    1/7    1/5;
     0    1    1/2    1/3;
     7    2     1      1/9;
     5    3     9      1]

M2 = [1  0  1/3 1/9;
      0  1   0  1/8;
      3  0   1  1/9;
      9   8  9   1]

M3 = [1 1/3 0 0;
      3  1  1/2 1/5;
      0  2   1  0;
      0  5   0  1]

%Tomamos M3 traspuesta
ic(M1)
ic(M2)
ic(M3')

fprintf("-------------Metodo de Minimos Cuadrados [Log]-------------\n");
w1 = funciones(1, M1, M2, M3') % minCuadLog(M)
[errorInf, IndexMaxErr, errorFro, errorUno,  errorNoAciertos] = errores(w1, M1, M2, M3')

fprintf("-------------Metodo de Minimos Cuadrados [Ponderado]-------------\n");
w2 = funciones(2, M1, M2, M3') % minCuadPond(M)
[errorInf, IndexMaxErr, errorFro, errorUno,  errorNoAciertos] = errores(w2, M1, M2, M3')

fprintf("-------------Metodo de Minimo Suma Desviaciones [Log]-------------\n");
w3 = funciones(3, M1, M2, M3') % minSumDesvLog(M)
[errorInf, IndexMaxErr, errorFro, errorUno,  errorNoAciertos] = errores(w3, M1, M2, M3')

fprintf("-------------Metodo de Minimo Suma Desviaciones [Ponderado]-------------\n");
w4 = funciones(4, M1, M2, M3') % minSumDesvPond(M)
[errorInf, IndexMaxErr, errorFro, errorUno,  errorNoAciertos] = errores(w4, M1, M2, M3')

%Dibujar pesos
figure();
c = categorical(["Min Cuadrado Log", "Min Cuadrado Pond", "Min Sum Desv Log", "Min Sum Desv Pond"]);
bar(c, [w1';w2';w3';w4']);
l=compose("A%d",(1:length(M)));
legend(l);

%Dibujar digrafo
grafo(M1, M2, M3')
```

funciones.m

```
function w = funciones(metodo, varargin)
 % Juntar las matrices de los expertos
    M = varargin{1};
    for i = 2 : nargin-1
        M = [M; varargin{i}];
    end

if nargin == 1
    [~, w, ~] = potencia(M);
else
    if metodo == 0 && nargin == 2
        [~, w, ~] = potencia(M);
    elseif metodo == 0 && nargin >= 2
        fprintf("Número erróneo de argumentos\n");
    elseif metodo == 1
        w = minCuadLog(varargin, nargin-1);
    elseif metodo == 2
        w = minCuadPond(varargin, nargin-1);
    elseif metodo == 3
        [w, ~, ~] = minSumDesvLog(varargin, nargin-1);
    else
        [w, ~, ~] = minSumDesvPond(varargin, nargin-1);
    end
end

return
```

potencia.m

```
function [lambda,x,iter] = potencia(A,tol,nmax,x0)
% Calcula el mayor (abs) autovalor lambda de A y un autovector asociado x
n=size(A,1);

if nargin == 1
    tol = 1e-06; % Tolerancia
    x0=rand(n,1); % Vector de arranque
    nmax=100; % NÂº mÃ¡x iteraciones
end

x0=x0/norm(x0); x1=A*x0;
lambda=x0'*x1;
err = tol*abs(lambda) + 1;
iter=0;

while err > tol*abs(lambda) && abs(lambda) ~= 0 && iter <= nmax
    x=x1; x=x/norm(x);
    x1=A*x; lambda_new=x'*x1;
    err = abs(lambda_new - lambda);
    lambda=lambda_new;
    iter = iter + 1;
end

x = x / sum(x);
end
```

minCuadLog.m

```
function w = minCuadLog(E, n)

H = [];
b = [];


for i = 1:n
    M = E{i};

    s=size(M);
    long = (s(1)*s(2))-s(1);

    HAux = zeros([long,s(2)]);
    bAux = zeros([long,1]);

    k = 1;
    for i = 1:s(1)
        for j = 1:s(2)
            if (i ~= j)
                notZero = (log(M(i,j)) ~= -Inf);
                % Generar H
                HAux(k,i) = notZero;
                HAux(k,j) = -1*notZero ;
                % Generar b
                if (notZero)
                    bAux(k) = log(M(i,j));
                else
                    bAux(k) = 0;
                end
                k = k+1;
            end
        end
    end
    H = [H; HAux];
    b = [b; bAux];
end
cond(H'*H)
% The result h is 1 if the test rejects the null hypothesis at the 5% significance level,
% that the data in vector x is from a population with a normal distribution,
% using the Anderson-Darling test, or 0 otherwise.
[h,p,adstat,cv] = adtest(exp(b))
% Resolver con minimos cuadrados
v=H\b;
% Deshacer el cambio de logaritmo.
w=exp(v)
% Normalizar
w=w/sum(w);
return
```

minCuadPond.m

```
function w = minCuadPond(E, n)

H = [];
b = [];

for i = 1:n
    M = E{i};

    s = size(M);
    long = (s(1)*s(2))-s(1);

    HAux = zeros([long,s(2)]);

    % Generar b
    bAux=zeros([long,1]);
    %bAux(long+1) = 1;
    b = [b; bAux];

    k = 1;
    for i = 1:s(1)
        for j = 1:s(2)
            if (i ~= j)
                % Generar
                %(M(i,j) == 0) devuelve 1 o 0 si queremos descartar
                %información que el experto no ha dado
                HAux(k,i) = (M(i,j) ~= 0);
                HAux(k,j) = -M(i,j);
                k=k+1;
            end
        end
    end

    H = [H; HAux];
end
cond(H'*H)
H = [H;  ones(1, s(1))];
b = [b; 1];

% Resolver con minimos cuadrados
w=H\b;

% Normalizar
w=w/sum(w);

return
```

minSumDesvLog.m

```
function [w, n, p] = minSumDesvLog(E, n)

Aeq = [];
beq = [];

s = zeros(1, 2);

for i = 1:n
    M = E{i};

    s = size(M);
    longM = (s(1)*s(2))-s(1);

    % No hay desigualdades
    A = [];
    b = [];

    AeqAux = zeros(longM, s(2));
    beqAux = zeros(longM, 1);

    k=1;
    for i = 1:s(1)
        for j = 1:s(2)
            if (i ~= j)
                notZero = (log(M(i,j)) ~= -Inf);
                % Generar Aeq
                AeqAux(k,i) = notZero;
                AeqAux(k,j) = -1*notZero ;
                % Generar beq
                if (notZero)
                    beqAux(k) = log(M(i,j));
                else
                    beqAux(k) = 0;
                end
                k = k + 1;
            end
        end
    end


    Aeq = [Aeq; AeqAux];
    beq = [beq; beqAux];
end

long = size(Aeq, 1);

%Añadimos las metas a las ecuaciones
N = eye(long);
P = -1 * eye(long);

Aeq = [Aeq N P];

x = zeros(s(1) + long + long, 1);
f = [zeros(s(1),1); ones(long,1); ones(long,1)];
```

```
% Todos los elementos de x mayor o igual que 0
lb = zeros(length(x), 1);

ub = [];

x = linprog(f, A, b, Aeq, beq, lb, ub);

% Coger de x la parte que nos interesa
v = x(1:s(1));
% Deshacer el cambio logaritmico
w = exp(v);
% Normalizar
w = w/sum(w);

n = x(s(1)+1:s(1)+long);
p = x(s(1)+long+1:s(1)+2*long);
end
```

minSumDesvPond.m

```
function [w, n, p] = minSumDesvPond(E, n)

Aeq = [];
beq = [];

s = zeros(1, 2);

for i = 1:n
    M = E{i};

    s = size(M);
    longM = (s(1)*s(2))-s(1);

    % No hay desigualdades
    A = [];
    b = [];

    AeqAux = zeros(longM, s(2));

    k = 1;
    for i = 1:s(1)
        for j = 1:s(2)
            if (i ~= j)
                % Generar Aeq
                AeqAux(k,i) = (M(i,j) ~= 0);
                AeqAux(k,j) = -M(i,j);
                k = k + 1;
            end
        end
    end

    Aeq = [Aeq; AeqAux];
end

long = size(Aeq, 1);
```

```
%Añadimos las metas a las ecuaciones
N = eye(long);
P = -1 * eye(long);
Aeq = [[Aeq N P]; [ones(1, s(1)) zeros(1, 2*long)]];

x = zeros(s(1) + long + long, 1);
f = [zeros(s(1),1); ones(long,1); ones(long,1)];

beq = zeros([long+1,1]);
beq(long+1) = 1;

% Todos los elementos de x mayor o igual que 0
lb = zeros(length(x), 1);

ub = [];

x = linprog(f, A, b, Aeq, beq, lb, ub);

% Coger de x la parte que nos interesa
w = x(1:s(1));

% Normalizar
w = w/sum(w);

n = x(s(1)+1:s(1)+long);
p = x(s(1)+long+1:s(1)+2*long);
end
```

errores.m

```
function [errorInf, IndexMaxErr, errorFro, errorUno, errorNoAciertos] = errores(w,varargin)

datos_conocidos_total = 0;
errorInf = [];
IndexMaxErr = [];
errorFro = [];
errorUno = [];
errorNoAciertos = [];

M = [];
W = [];

% Construccion W
WAux = zeros(size(varargin{1}));
for i = 1:size(w)
    for j = 1:size(w)
        WAux(i,j) = w(i)/w(j);
    end
end
```

```
% Errores de cada experto
for k = 1:nargin


    if k == nargin
        E = M;
        WAux = W;
    else
        E = varargin{k};

        % Contar Os de M
        datos_conocidos = 0;
        for m = 1:length(E)
            for n = 1:length(E)
                if E(m,n) ~= 0
                    datos_conocidos = datos_conocidos + 1;
                end
            end
        end
    end


    WAux2 = WAux .* zerosM(E);
    % Matriz de residuos
    R = abs(E-WAux2) ;

    % Norma Infinito: residuo maximo y su indice
    [errorInfAux, I] = max(R(:));
    errorInf = [errorInf errorInfAux/datos_conocidos];
    [I_row, I_col] = ind2sub(size(R),I);
    IndexMaxErr = [IndexMaxErr [I_row, I_col]];

    % Norma Frobenius
    errorFroAux = norm(R,'fro');
    errorFro = [errorFro errorFroAux/datos_conocidos];

    % Norma 1
    errorUnoAux = sum(sum(R));
    errorUno = [errorUno errorUnoAux/datos_conocidos];

    % Norma "ErrorRel"
    % errorErrRelAux = norm(R./M, 'fro'); Puede dividir por 0
    % errorErrRel = [errorErrRel errorErrRelAux/datos_conocidos];

    % Norma "No aciertos"
    [m, n] = size(E);
    errorNoAciertosAux = 0;
    for i = 1:m
        for j = i:n
            if (E(i,j) > 0)
                if w(i) > w(j) && E(i,j) < 1
                    errorNoAciertosAux = errorNoAciertosAux + 1;
                elseif (w(i) == w(j) && E(i,j) ~= 1) || (w(i) ~= w(j) && E(i,j) == 1)
                    errorNoAciertosAux = errorNoAciertosAux + 0.5;
                end
            end
        end
    end
```

```
    errorNoAciertos = [errorNoAciertos errorNoAciertosAux/datos_conocidos];

    % Si solo nos pasan una matriz
    if (nargin == 2)
        break;
    end

    M = [M; E];
    W = [W; WAux2];
    datos_conocidos_total = datos_conocidos_total + datos_conocidos;

end

if nargin > 2
    figure();
    l=compose("E%d",(1:nargin-1));
    % Pie
    tiledlayout(1, 3);
    ax1 = nexttile;
    pie(ax1, errorInf(1:end-1)./sum(errorInf(1:end-1)))
    legend(l)
    title('Error Inf')

    ax2 = nexttile;
    pie(ax2,errorFro(1:end-1)./sum(errorFro(1:end-1)))
    legend(l)
    title('Error Frobenius')

    ax3 = nexttile;
    pie(ax3,errorUno(1:end-1)./sum(errorUno(1:end-1)))
    legend(l)
    title('Error 1')
end

end
```

ic.m

```
function res = ic(M)
%Autovalor dominante
lambda = max(eig(M));
m = length(M);
%Calculo del indice de consistencia
res = (lambda - m) / (m - 1);
return
```

zerosM.m

```
function mBin = zerosM(M)
    mBin=(M~=0);
return
```