Homework 1: Discrete Vortex Method

Isaac Gibert, Ian Martorell, Sara Piñeiro, Esteban Ruiz, and Eduard Sulé

220024 - Aerodynamics, UPC ESEIAAT

Dated: December 2016

1 Verificación

Para poder verificar la solución DVM utilizaremos la teoría de perfiles delgados para calcular los valores de referencia exactos para un perfil NACA 2212 con un ángulo de ataque igual a 4°.

Partimos de la ecuación para perfiles delgados, que es la siguiente:

$$\frac{1}{2\pi}\int_0^\pi \frac{\gamma(\theta)\sin\theta}{\cos(\theta)-\cos(\theta_0)}d\theta = U_\infty(\alpha - \frac{dz}{dx})$$

Aplicamos el desarrollo de Glauert:

$$A_0 = \alpha - \frac{1}{\pi}\int_0^\pi \frac{dz}{dx}d\theta \qquad A_n = \frac{2}{\pi}\int_0^\pi \frac{dz}{dx}\cos(n\theta)d\theta$$

La circulación será:

$$\Gamma = cu_\infty[A_0\pi + A_1\frac{\pi}{2}]$$

Aplicando la ley de Kutta ($l = \rho_\infty u_\infty \Gamma$) y la definición $c_l = \frac{l}{q_\infty c}$ obtenemos la siguiente expresión:

$$c_l = \pi(2A_0 + A_1)$$

La línea media viene definida por las siguientes ecuaciones:

$$z_m = \frac{f}{p^2}(2px - x^2) \qquad 0 \le x \le p$$

$$z_m = \frac{f}{1-p^2}(1 - 2p + 2px - x^2) \qquad p < x \le 1$$

Derivando respecto x, obtenemos expresiones que podemos usar en el cálculo de A_0 y A_1:

$$\frac{dz}{dx} = \frac{2f}{p^2}(p - x) \qquad 0 \le x \le p$$

$$\frac{dz}{dx} = \frac{2f}{(1-p)^2}(p - x) \qquad p < x \le \pi$$

Haciendo el cambio $x = \frac{1}{2}(1 - \cos\theta)$, con los datos $p = 0.2$, $f = 0.02$, $\alpha = 4°$, y sabiendo que en la discontinuidad $x = p \to \theta = \arccos(0.6)$, obtenemos:

$$A_0 = 0.052213 \qquad A_1 = 0.09799$$

$$c_l = 0.6359$$

A continuación, utilizando nuestro programa para obtener el valor de c_l para diferentes números de paneles y aplicando las distribuciones uniforme y *full cosine* obtenemos:

Paneles	$c_{l,uniforme}$	$c_{l,fullcosine}$	$Error_{uniforme}[\%]$	$Error_{fullcosine}[\%]$
4	0.5735	0.5772	9.8189	9.2386
8	0.5947	0.603	6.4774	5.1797
16	0.6073	0.615	4.5007	3.2808
32	0.6154	0.6199	3.2281	2.5115
64	0.6205	0.6223	2.4189	2.1418
128	0.6224	0.6234	2.1242	1.9648
256	0.6234	0.624	1.9607	1.8761

Figure 1: Valores obtenidos para el c_l usando una distribución uniforme y *full cosine*

Observamos en primer lugar que no se obtienen mejoras significativas a través del uso de la distribución *full cosine*. Sin embargo, optamos por usarla de todas maneras ya que los resultados que obtenemos se acercan más a los teóricos, por poco que sea.

Tomando la distribución *full cosine*, vemos que a partir de 128 paneles se obtiene un error en el c_l inferior al 2%, que consideramos aceptable, por lo que utilizaremos este número de paneles en nuestros siguientes cálculos.

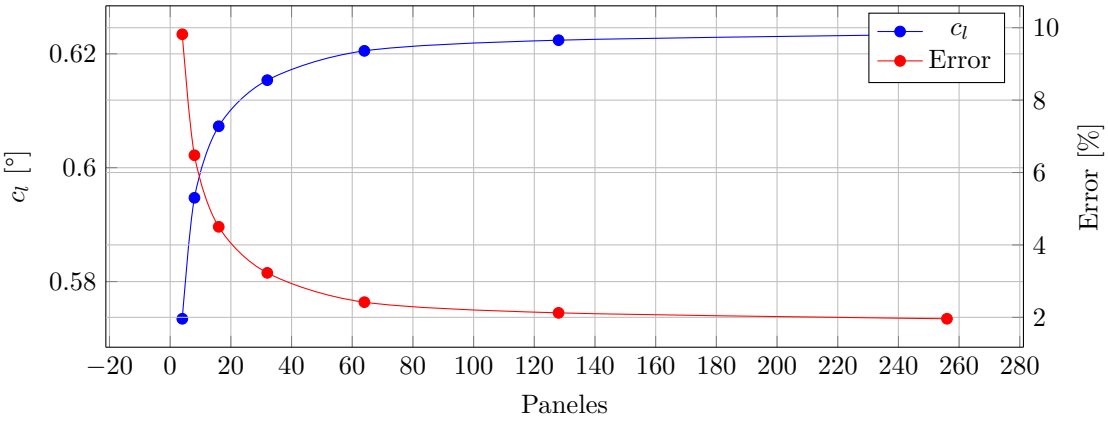

Figure 2: Distribución uniforme

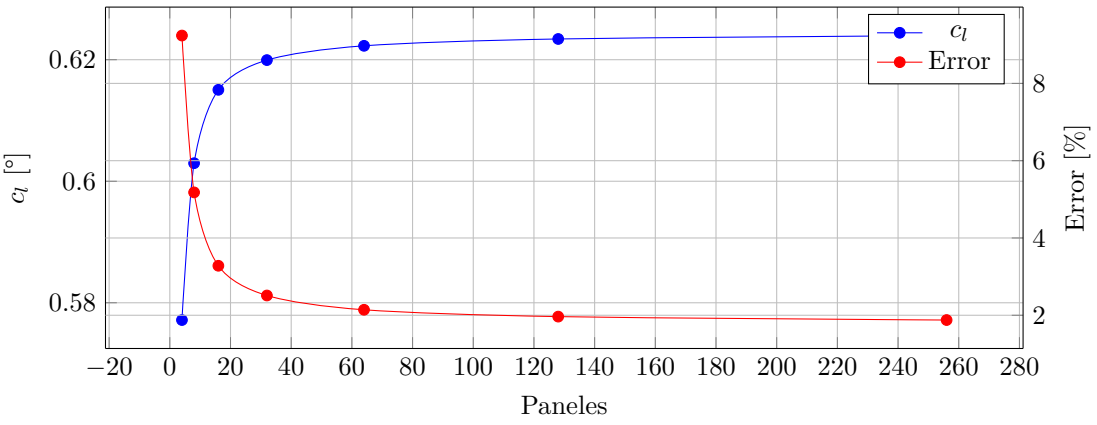

Figure 3: Distribución *full cosine*

En cuanto al flap, lo escogemos de tal forma que se sitúe en $x = 0.8$. Hacemos variar el ángulo de deflexión del flap y graficamos el ángulo de sustentación nula con respecto de éste, obteniendo la siguiente recta.

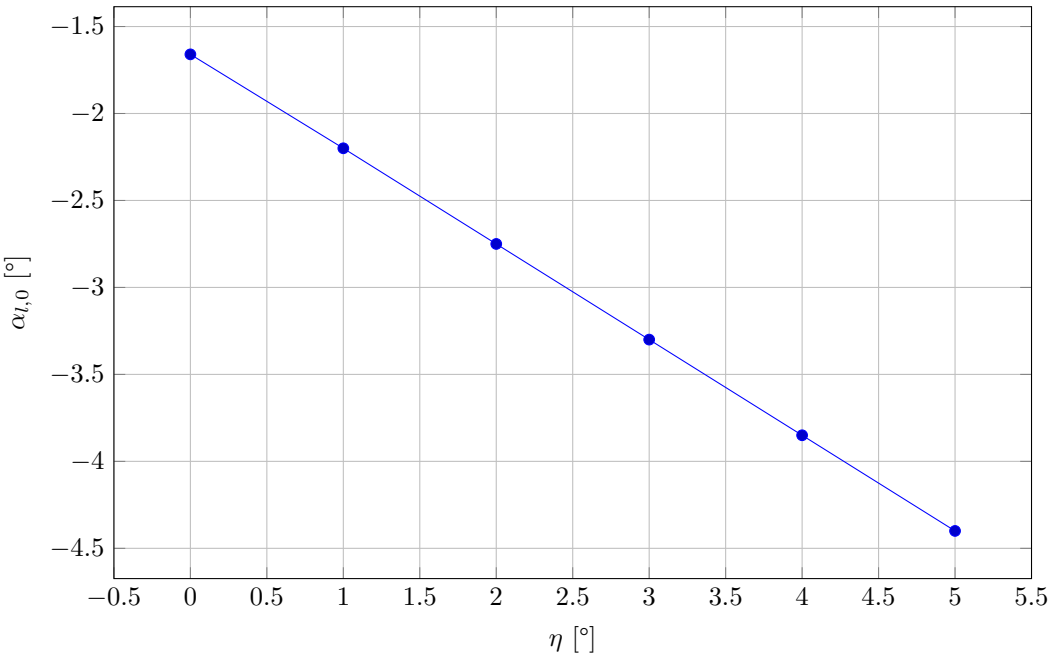

Figure 4: Eficiencia del flap

El pendiente de la curva resultante nos da el factor de eficiencia:

$$\frac{\delta \alpha_{l,0}}{\delta \eta} = -0.548$$

2 Validación

Basándonos en el análisis anterior y usando el mismo perfil NACA, calculamos la pendiente de sustentación, el ángulo de ataque de sustentación nula ($\alpha_{l,0}$) y el coeficiente de momento respecto del centro aerodinàmico ($c_{m,LE}$):

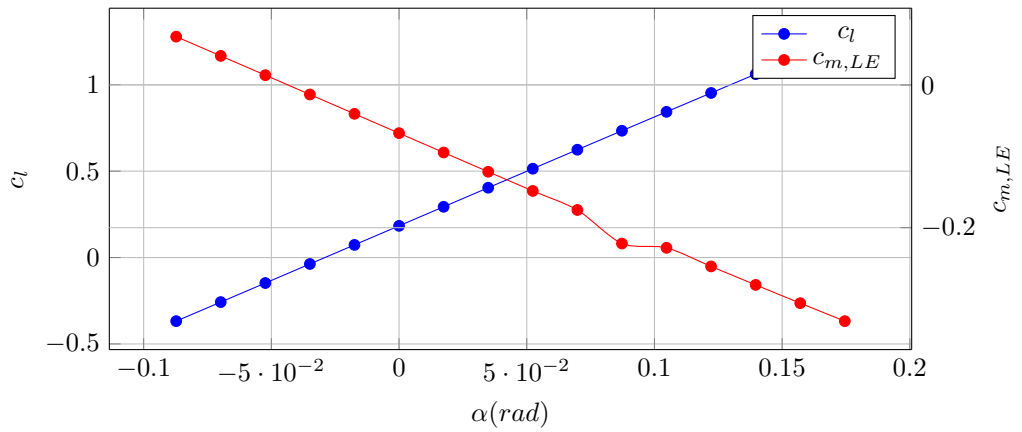

Figure 5: Distribución *full cosine*

Del gráfico obtenemos los siguientes resultados:

$$c_{l,\alpha} = 6.294 \approx 2\pi = 6.2832$$

$$\alpha_{l,0} = -0.0280 rad$$

$$c_{m,0} = -0.0436$$

La pendiente de sustentación es prácticamente igual a 2π, y además el cociente entre la pendeinte de la curva de coeficiente de momento y el coeficiente de sustentación resulta ser:

$$\frac{1.537}{6.294} = 0.244$$

Esto es aproximadamente igual a 0.25, es decir, $\frac{1}{4}$ de la cuerda, posición en la que se encuentra el centro aerodinámico. Con estos datos opdemos concluir que la teoria de perfiles delgados es acertada.

Del *NACA report 460* obtenemos los siguientes resultados:

$$c_{l,\alpha} = 5.2$$

$$\alpha_{l,0} = -0.035 rad$$

$$c_{m,0} = -0.03$$

3 Discusión

Como se puede ver en las gráficas que aparecen a continuación, al aumentar la curvatura máxima y al retrasar la posición en la que se encuentra, podemos observar que las rectas de sustentación se desplazan hacia arriba, mientras que las de los momentos bajan, manteniéndose su pendiente constante. Esto se traduce en un aumento del $c_{l,0}$, en una reducción del $\alpha_{l,0}$ y en un incremento

negativo de $c_{m,0}$. No obstante, estos incrementos son muy pequeños cuando retrasamos la posición de la combadura máxima. Cuando aumentamos la curvatura máxima, aumentamos la velocidad máxima que alcanzará el flujo en el extradós, creando un pico de succión mayor, y aumentando la diferencia de presiones, por lo que obtenemos una sustentación más grande. Esto también aumenta el momento que actúa sobre el ala debido a que estamos aumentando el módulo de la fuerza que lo provoca. Por otro lado, si retrasamos el punto de combadura máxima, estaremos variando el brazo de palanca que provocará el momento.

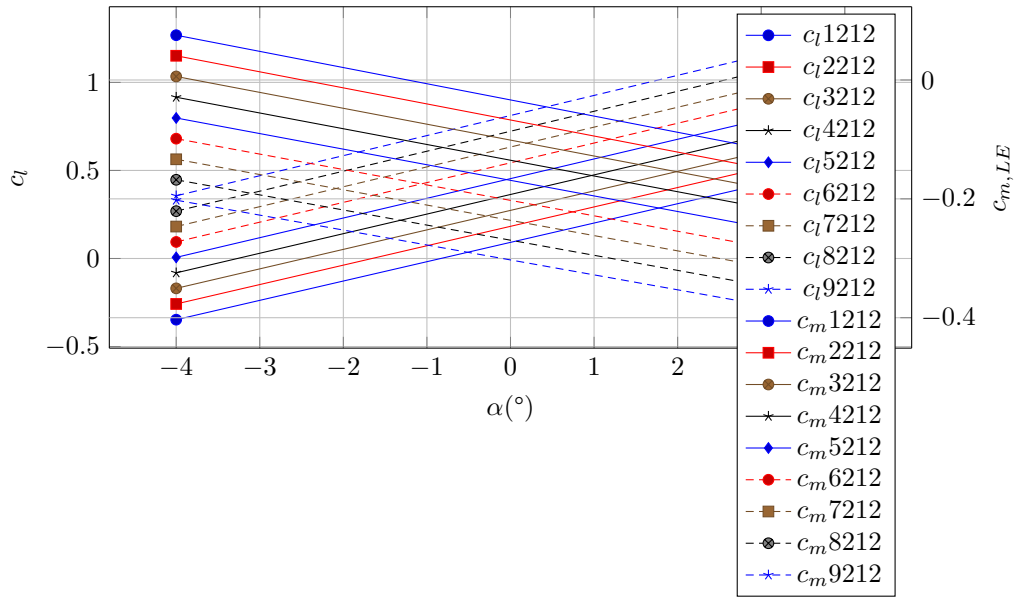

Figure 6: c_l y $c_{m,0}$ manteniendo p fija y variando f

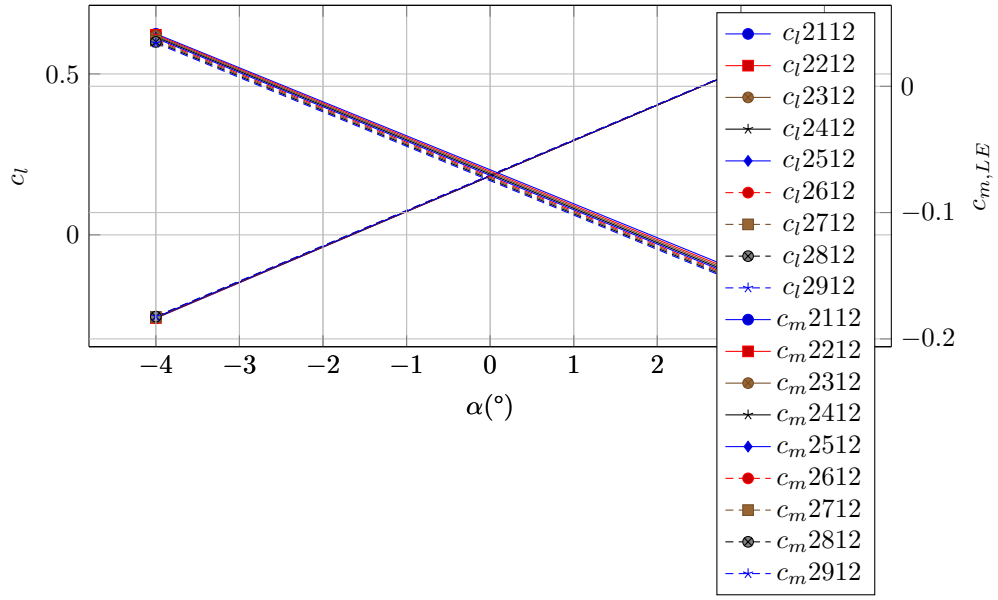

Figure 7: c_l y $c_{m,0}$ manteniendo f fija y variando p

4 Apéndice: Código

```
function camberLine = airfoil(NACA, flapPosition, flapAngle, nPanels, distribution)
% AIRFOIL  Define the camber line of an airfoil with an optional flap
%
%   airfoil(NACA, flapPosition, flapAngle, nPanels, distribution)
%
%  NACA: NACA 4-digit designation
%   flapPosition: flap position as a fraction of 1, use 1 for no flap
%   flapAngle: flap angle in degrees
%   nPanels: number of panels
%   distribution: panel distribution along the X axis, can be 'linear' or 'fullCosine',
    defaults to 'linear'

% Parse input data
maxCamber = str2double(NACA(1))/100;
maxCamberPos = str2double(NACA(2))/100;
nPoints = nPanels + 1;
% Prepare the result vectors
X = zeros(1, nPoints);
Z = zeros(1, nPoints);
if ~strcmp(distribution, 'fullCosine')
    % distribute control points linearly
    X = linspace(0, 1, nPoints);
end

for i = 1:nPoints
    if strcmp(distribution, 'fullCosine')
      % Calculate X according to the number of panels
      X(i) = (1/2) * (1-cos(pi*(i-1)/nPanels));
    end
    % Calculate Z using NACA equations for a cambered 4-digit airfoil
    if (X(i) <= maxCamberPos)
        Z(i) = (maxCamber/maxCamberPos^2)*(2*maxCamberPos*(X(i))-(X(i))^2);
    else
        Z(i) = (maxCamber/(1-maxCamberPos)^2)*(1-2*maxCamberPos+2*maxCamberPos*(X(i))-(X(i
    ))^2);
    end
    % Rotate by flapAngle degrees if this panel is part of the flap
    if (X(i) >= flapPosition)
        Z(i) = Z(i) - tand(flapAngle) * (X(i) - flapPosition);
    end
end

camberLine = [X; Z];
```

airfoil.m

```
function [ circulation, cl, cmLE ] = DVM(camberLine, angleOfAttack)

nPanels = size(camberLine, 2) - 1;
vortices = zeros(2, nPanels);
controlPoints = zeros(2, nPanels);
% We need to at least compute the vortices' positions beforehand,
% or not all of them be defined in the inner loop
for i = 1:nPanels
    length = sqrt((camberLine(1, i+1) - camberLine(1, i))^2 + (camberLine(2, i+1) -
    camberLine(2, i))^2);
    angle = atand((camberLine(2, i+1) - camberLine(2, i)) / (camberLine(1, i+1) -
    camberLine(1, i)));
    vortices(1, i) = camberLine(1, i) + 0.25*length*(cosd(angle));
    vortices(2, i) = camberLine(2, i) + 0.25*length*(sind(angle));
    controlPoints(1, i) = camberLine(1, i) + 0.75*length*(cosd(angle));
    controlPoints(2, i) = camberLine(2, i) + 0.75*length*(sind(angle));
end

% We can now calculate the circulation
```

```
influenceCoefficients = zeros(nPanels, nPanels);
RHS = zeros(nPanels, 1);
for i = 1:nPanels
    % Find the unit normal vector at this control point
    d = zeros(2, 1);
    d(1) = (camberLine(1, i+1) - (camberLine(1, i)));
    d(2) = (camberLine(2, i+1) - (camberLine(2, i)));
    unitTangentVector = zeros(2, 1);
    unitTangentVector(1) = d(1) / sqrt(d(1)^2 + d(2)^2);
    unitTangentVector(2) = d(2) / sqrt(d(1)^2 + d(2)^2);
    unitNormalVector = zeros(2, 1);
    unitNormalVector(1) = - unitTangentVector(2);
    unitNormalVector(2) = unitTangentVector(1);
    for j = 1:nPanels
        % Compute the induced velocity at i due to a lumped tortex at j
        d(1) = controlPoints(1, i) - vortices(1, j);
        d(2) = controlPoints(2, i) - vortices(2, j);
        r = sqrt(d(1)^2 + d(2)^2);
        inducedVelocity = zeros(2, 1);
        inducedVelocity(1) = 1/(2*pi) * d(2)/r^2;
        inducedVelocity(2) = -1/(2*pi) * d(1)/(r)^2;
        % Compute the influence coefficients
        influenceCoefficients(i, j) = inducedVelocity(1) * unitNormalVector(1) +
    inducedVelocity(2) * unitNormalVector(2);
    end
    RHS(i) = -((cosd(angleOfAttack) * unitNormalVector(1)) + (sind(angleOfAttack) *
    unitNormalVector(2)));
end
circulation = influenceCoefficients \ RHS;
% Compute the lift coefficients
cl = 2 * sum(circulation);
% Compute the moment coefficient about the leading edge
sumArray = zeros(nPanels, 1);
for i = 1:nPanels
    sumArray(i) = circulation(i) * vortices(1, i) * cosd(angleOfAttack);
end
cmLE = -2 * sum(sumArray);
```

DVM.m

```
function [ clArray, clErrorArray, cmLEArray, nPanels ] = verification_helper(distribution)
  clReference = 0.63590677;

  NACA = '2212';
  angleOfAttack = 4;
  flapPosition = 1;
  flapAngle = 0;

  clArray = [];
  clErrorArray = [];
  cmLEArray = [];
  nPanels = [];
  iPanels = 4;
  while (iPanels <= 256)
    camberLine = airfoil(NACA, flapPosition, flapAngle, iPanels, distribution);
    [ circulation, cl, cmLE ] = DVM(camberLine, angleOfAttack);
    clArray = [ clArray; cl ];
    clErrorArray = [ clErrorArray; (abs(cl-clReference))/clReference*100 ];
    cmLEArray = [ cmLEArray; cmLE ];
    nPanels = [ nPanels; iPanels ];
    iPanels = iPanels * 2;
  end
end
```

verification_helper.m

```
[ clLinear, clErrorLinear, cmLELinear, nPanels ] = verification_helper('linear');
[ clFullCosine, clErrorFullCosine, cmLEFullCosine, nPanels ] = verification_helper('
    fullCosine');

csvwrite('data/cl_convergence.csv', [ nPanels clLinear clFullCosine clErrorLinear
    clErrorFullCosine ]);
```

verification.m

```
NACA = '2212';
angleOfAttack = 4;
flapPosition = 1;
flapAngle = 0;
distribution = 'fullCosine';
nPanels = 128;

clArray1 = [];
clArray2 = [];
cmLEArray1 = [];
cmLEArray2 = [];
NACAArray = [];

% Fixing p
for i = 1:9
  NACA(1) = num2str(i);
  camberLine = airfoil(NACA, flapPosition, flapAngle, nPanels, distribution);
  % Compute cl and cmLE for -4 and 4 degrees so we can draw a line
  [ circulation, cl, cmLE ] = DVM(camberLine, -4);
  clArray1 = [ clArray1 cl];
  cmLEArray1 = [ cmLEArray1 cmLE ];
  [ circulation, cl, cmLE ] = DVM(camberLine, 4);
  clArray2 = [ clArray2 cl];
  cmLEArray2 = [ cmLEArray2 cmLE ];
  NACAArray = [ NACAArray str2double(NACA) ];
end

NACA = '2212';
% Fixing f
for i = 1:9
  NACA(2) = num2str(i);
  camberLine = airfoil(NACA, flapPosition, flapAngle, nPanels, distribution);
  % Compute cl and cmLE for -4 and 4 degrees so we can draw a line
  [ circulation, cl, cmLE ] = DVM(camberLine, -4);
  clArray1 = [ clArray1 cl];
  cmLEArray1 = [ cmLEArray1 cmLE ];
  [ circulation, cl, cmLE ] = DVM(camberLine, 4);
  clArray2 = [ clArray2 cl];
  cmLEArray2 = [ cmLEArray2 cmLE ];
  NACAArray = [ NACAArray str2double(NACA) ];
end

csvwrite('data/discussion.csv', [ NACAArray; clArray1; clArray2; cmLEArray1; cmLEArray2 ])
    ;
```

discussion.m