Homework 2: Horseshoe Vortex Method

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

220024 - Aerodynamics, UPC ESEIAAT

Dated: January 2016

1. Efecto del alargamiento en el $c_{L,\alpha}$

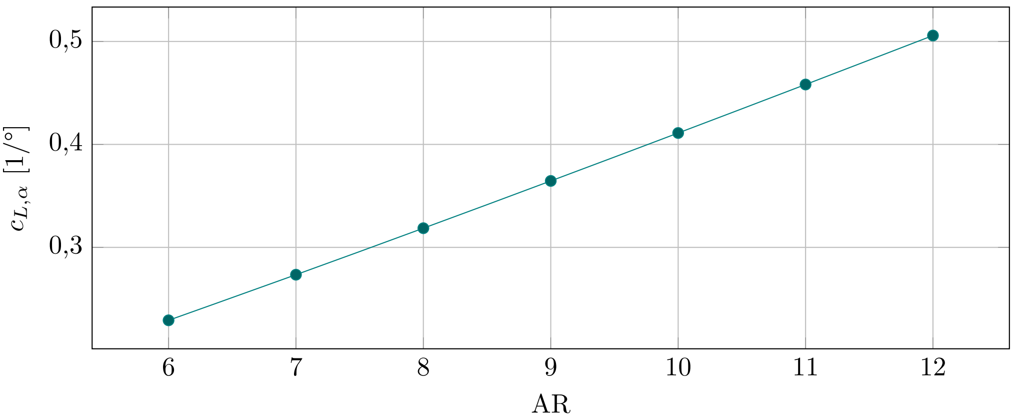

Figura 1: Caption

2. Efecto de la flecha en el $c_{L,\alpha}$ y el x_{ac}

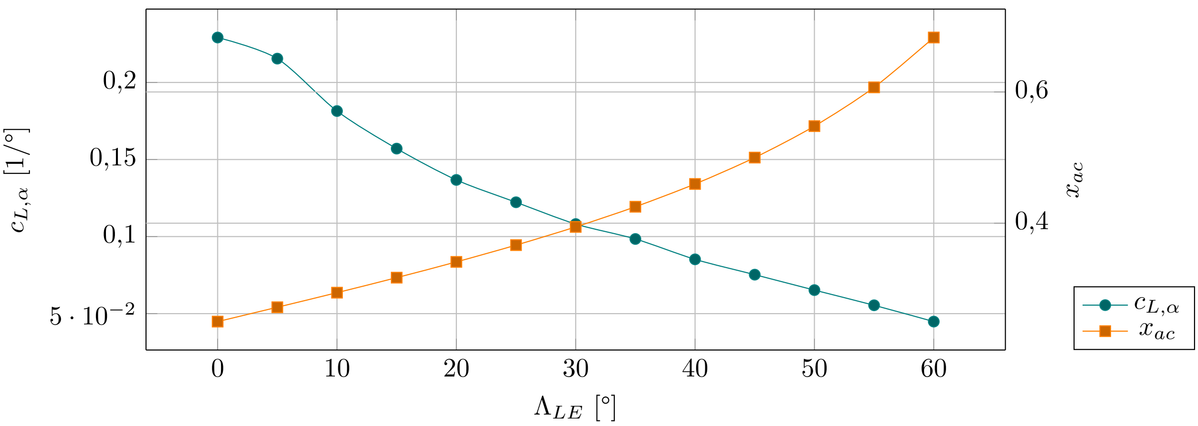

Figura 2: Caption

3. Efecto del estrechamiento en la distribución de sustentación local

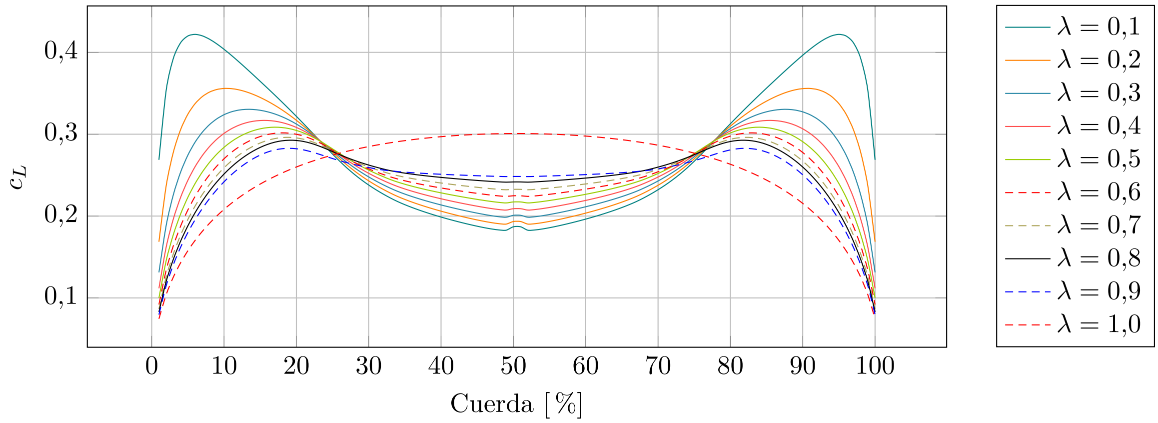

Figura 3: Caption

4. Efecto del estrechamiento y la flecha en el factor de eficiencia de Oswald

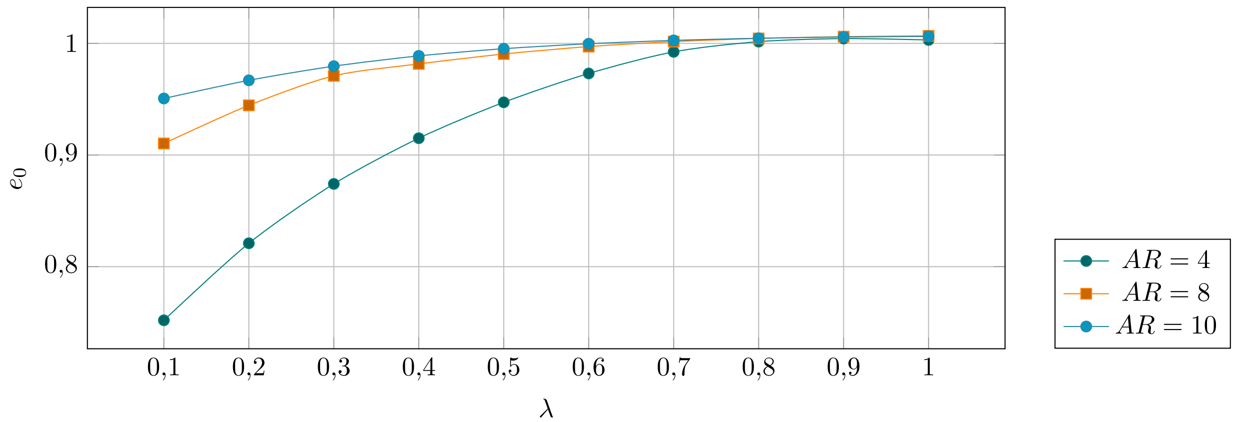

Figura 4: Caption

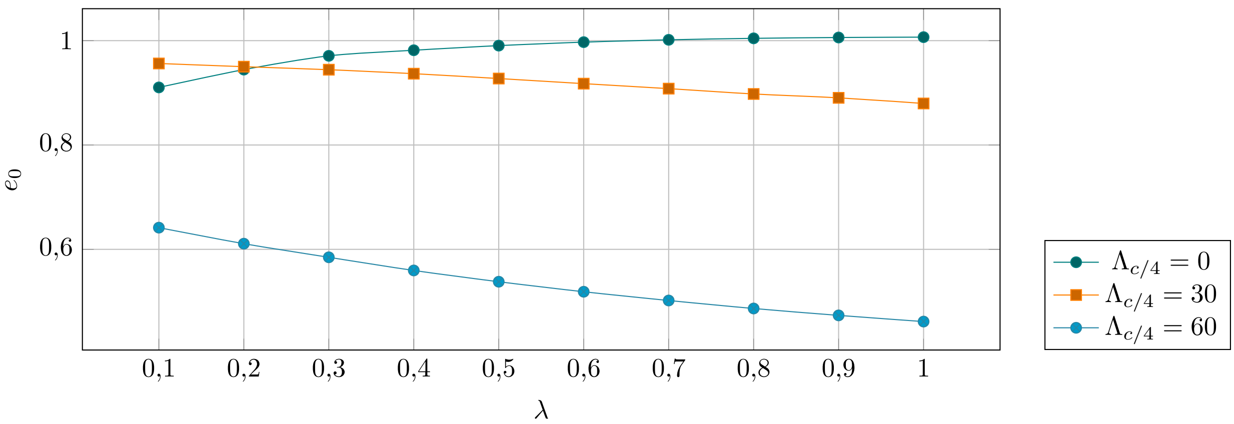

Figura 5: Caption

5. Apéndice: Código

```
function [ midPoints, controlPoints, bounded_nodes, trailing_nodes, panelAngles,
    panelAreas ] = wing_discretization(aspectRatio, taperRatio, quarterChordSweep,
    angleOfAttack, wingTipTwist, nPanels)
  %wing_discretization: Returns a vector of points along the quarter chord line of the
  %wing, each centered along the y axis of every panel. Also returns the control points
  %at 3c/4 and the angle of every panel after applying angle of attack and twist.
  midPoints = zeros(nPanels, 3);
  controlPoints = zeros(nPanels, 3);
  bounded_nodes = zeros(nPanels+1, 3);
  trailing_nodes = zeros(nPanels+1, 3);
  panelAngles = zeros(nPanels, 1);
  panelAreas = zeros(nPanels, 1);
  %Compute y, which is distributed linearly
  panelWidth = 1/nPanels;
  lastY = 0.5 - panelWidth/2;
  midPoints(:, 2) = linspace(-lastY, lastY, nPanels);
  controlPoints(:, 2) = linspace(-lastY, lastY, nPanels);
  bounded_nodes(:, 2) = linspace(-0.5, 0.5, nPanels+1);
  trailing_nodes(:, 2) = linspace(-0.5, 0.5, nPanels+1);
  %Compute some needed constants
  surfaceArea = 1/aspectRatio;
  chordRoot = 2*surfaceArea/(1+taperRatio);
  chordTip = taperRatio*chordRoot;
  %Find equation of sweep(y) = x(y) for quarter chord points
  sweepSlope = tand(90-quarterChordSweep);
  sweepOrd = -sweepSlope*0.25*chordRoot;
  %Find equation for twist, which has a zero y-intercept
  twistSlope = wingTipTwist/0.5;
  for i = 1:nPanels
    chord = chordRoot + (chordTip - chordRoot) / 0.5 * abs(midPoints(i, 2));
    panelAreas(i) = chord*panelWidth;
    panelAngles(i) = twistSlope * abs(midPoints(i, 2)) + angleOfAttack;
    %Calculate x position, if sweep is 90 degrees, the slope will be infinity
    if isinf(sweepSlope)
      midPoints(i, 1) = 0.25*chord;
      bounded_nodes(i, 1) = 0.25*chord;
    else
      if midPoints(i, 2) > 0
        midPoints(i, 1) = (midPoints(i, 2) - sweepOrd) / sweepSlope;
        bounded_nodes(i, 1) = (bounded_nodes(i, 2) - sweepOrd) / sweepSlope;
      else
        midPoints(i, 1) = (midPoints(i, 2) + sweepOrd) / -sweepSlope;
        bounded_nodes(i, 1) = (bounded_nodes(i, 2) + sweepOrd) / -sweepSlope;
```

```
      end
    end
    trailing_nodes(i, 1) = bounded_nodes(i, 1) + 20;
    %Correction due to panel angle
    midPoints(i, 1) = midPoints(i, 1) + chord * (1 - cosd(panelAngles(i)));
    %Calculate z position
    midPoints(i, 3) = sind(panelAngles(i)) * chord;
    %Calculate control point position
    controlPoints(i, 1) = midPoints(i, 1) + chord/2 + (1 - cosd(panelAngles(i)))/4;
    controlPoints(i, 3) = sind(panelAngles(i)) * chord/4;
  end
  %Last bounded and trailing node
  if isinf(sweepSlope)
    bounded_nodes(nPanels+1, 1) = 0.25*chord;
    trailing_nodes(nPanels+1, 1) = bounded_nodes(nPanels+1, 1) + 20;
  else
    bounded_nodes(nPanels+1, 1) = (bounded_nodes(nPanels+1, 2) - sweepOrd) / sweepSlope;
    trailing_nodes(nPanels+1, 1) = bounded_nodes(nPanels+1, 1) + 20;
  end
end
```

wing_discretization.m

```
function coordinates = rectangular_horseshoe(midPoint, panelAngle, nPanels)
  panelWidth = 1 / nPanels;
  coordinates = zeros(4, 3);
  %Points b, c, a, d
  coordinates(2, :) = [ midPoint(1) midPoint(2)-panelWidth/2 midPoint(3) ];
  coordinates(3, :) = [ midPoint(1) midPoint(2)+panelWidth/2 midPoint(3) ];
  coordinates(1, :) = [ midPoint(1)+20 coordinates(2,2) -sind(panelAngle)*20 ];
  coordinates(4, :) = [ midPoint(1)+20 coordinates(3,2) -sind(panelAngle)*20 ];
end
```

rectangular_horseshoe.m

```
function inducedVelocity =  compute_induced_velocity(xa, xb, xp, circulation)
  %Cross products
  x = (xp(2)-xa(2))*(xp(3)-xb(3)) - (xp(3)-xa(3))*(xp(2)-xb(2));
  y = -(xp(1)-xa(1))*(xp(3)-xb(3)) + (xp(3)-xa(3))*(xp(1)-xb(1));
  z = (xp(1)-xa(1))*(xp(2)-xb(2)) - (xp(2)-xa(2))*(xp(1)-xb(1));
  d = x*x + y*y + z*z;
  r1 = sqrt((xp(1)-xa(1))*(xp(1)-xa(1)) + (xp(2)-xa(2))*(xp(2)-xa(2)) + (xp(3)-xa(3))*(xp
    (3)-xa(3)));
  r2 = sqrt((xp(1)-xb(1))*(xp(1)-xb(1)) + (xp(2)-xb(2))*(xp(2)-xb(2)) + (xp(3)-xb(3))*(xp
    (3)-xb(3)));
  %We set the induced velocity to zero if r1, r2 or their cross product is less
  %than a small constant, to avoid dividing by zero
  if d<(10^-6) || r2<(10^-6) || r1<(10^-6)
    inducedVelocity = [ 0; 0; 0 ];
  else
    ror1 = (xb(1)-xa(1))*(xp(1)-xa(1)) + (xb(2)-xa(2))*(xp(2)-xa(2)) + (xb(3)-xa(3))*(xp
    (3)-xa(3));
    ror2 = (xb(1)-xa(1))*(xp(1)-xb(1)) + (xb(2)-xa(2))*(xp(2)-xb(2)) + (xb(3)-xa(3))*(xp
    (3)-xb(3));
    com = (circulation/(4*pi*d))*((ror1/r1)-(ror2/r2));
    inducedVelocity = [x*com; y*com; z*com];
  end
end
```

compute_induced_velocity.m

```
function angle = quarter_chord_sweep(leadingEdgeSweep, aspectRatio, taperRatio)
  angle = atand(tand(leadingEdgeSweep)-(1/aspectRatio)*((1-taperRatio)/(1+taperRatio)));
end
```

quarter_chord_sweep.m

```
function [ cL, cLY, cDi ] = HVM(aspectRatio, taperRatio, quarterChordSweep, angleOfAttack,
    wingTipTwist, horseshoeShape, nPanels)
  %HVM: Computes the lift coefficient of a wing using the Horseshoe Vortex Method
  %horseshoeShape: can be 'rectangular' or 'trapezoidal'
  density = 1.25;
  freestreamVelocity = [ 1 0 0 ];
  %Perform wing discretization
  [ midPoints, controlPoints, bounded_nodes, trailing_nodes, panelAngles, panelAreas ] =
    wing_discretization(aspectRatio, taperRatio, quarterChordSweep, angleOfAttack,
    wingTipTwist, nPanels);
  %Initialize variables
  influenceCoefficients = zeros(nPanels, nPanels);
  RHS = zeros(nPanels, 1);
  for i = 1:nPanels
    midPoint = [ midPoints(i, 1) midPoints(i, 2) midPoints(i, 3) ];
    normalUnitVector = [ sind(panelAngles(i)) 0 cosd(panelAngles(i)) ];
    for j = 1:nPanels
      midPoint = [ midPoints(j, 1) midPoints(j, 2) midPoints(j, 3) ];
      if strcmp(horseshoeShape, 'rectangular')
        horseshoe = rectangular_horseshoe(midPoint, panelAngles(j), nPanels);
      else
        horseshoe = [ trailing_nodes(j,:); bounded_nodes(j,:); bounded_nodes(j+1,:);
    trailing_nodes(j+1,:)];
      end
      inducedVelocity = zeros(3,1);
      for k = 1:3
        inducedVelocity = inducedVelocity + compute_induced_velocity(horseshoe(k,:),
    horseshoe(k+1,:), controlPoints(i,:), 1);
      end
      influenceCoefficients(i, j) = dot(inducedVelocity, normalUnitVector);
    end
    RHS(i) = -dot(freestreamVelocity, normalUnitVector);
  end
  circulation = influenceCoefficients \ RHS;
  %Compute lift
  lift = zeros(nPanels, 1);
  for i = 1:nPanels
    lift(i) = density * freestreamVelocity(1) * circulation(i) / nPanels;
  end
  wingLift = sum(lift);
  %Compute lift coefficient
  cL = 2 / freestreamVelocity(1) * aspectRatio * sum(circulation/nPanels);
  %Compute local lift distribution
  cLY = 2*circulation./panelAreas/nPanels/freestreamVelocity(1);
  %Compute moment
  momentLE = zeros(nPanels);
  for i = 1:nPanels
    momentLE(i) = lift(i) * midPoints(i, 1) * cosd(panelAngles(i));
  end
  chordRoot = 2/aspectRatio/(1+taperRatio);
  geometricChord = (2/3)*chordRoot*((1+taperRatio+taperRatio^2)/(1+taperRatio));
  cMLE = ((-2)/(freestreamVelocity(1)/aspectRatio*geometricChord))*sum(momentLE);
  [ alpha_i local_drag cDi ] = compute_cdi(nPanels, midPoints, panelAngles, circulation,
    1/aspectRatio);
end
```

HVM.m

```
aspectRatios = 6:12;
taperRatio = 1;
quarterChordSweep = 0;
wingTipTwist = 0;
horseshoeShape = 'rectangular';
nPanels = 100;
% Initialize output vector
cLAlphas = [];
for aspectRatio = aspectRatios
```

```
    %Compute cL for -2 and 2 degrees so we can draw a line
    [ cL1 ] = HVM(aspectRatio, taperRatio, quarterChordSweep, -2, wingTipTwist, horseshoeShape, nPanels);
    [ cL2 ] = HVM(aspectRatio, taperRatio, quarterChordSweep, 2, wingTipTwist, horseshoeShape, nPanels);
    cLAlphas = [ cLAlphas; (cL2-cL1)/4 ];
end

csvwrite('data/hw2_1.csv', [ aspectRatios' cLAlphas ]);
```

hw2_1.m

```
aspectRatio = 6;
taperRatio = 1;
leadingEdgeSweeps = 0:5:60;
wingTipTwist = 0;
horseshoeShape = 'rectangular';
nPanels = 100;
%Initialize output vectors
cLAlphas = [];
aerodynamicCenters = [];
for leadingEdgeSweep = leadingEdgeSweeps
    %Although unnecessary for a unitary taper ratio, we calculate the quarter chord sweep angle
    quarterChordSweep = quarter_chord_sweep(leadingEdgeSweep, aspectRatio, taperRatio);
    %Compute cL for -2 and 2 degrees so we can draw a line
    [ cL1 ] = HVM(aspectRatio, taperRatio, quarterChordSweep, -2, wingTipTwist, horseshoeShape, nPanels);
    [ cL2 ] = HVM(aspectRatio, taperRatio, quarterChordSweep, 2, wingTipTwist, horseshoeShape, nPanels);
    cLAlphas = [ cLAlphas; (cL2-cL1)/4 ];
    aerodynamicCenters = [ aerodynamicCenters; 0.25 + tand(quarterChordSweep)/6*(1+2*taperRatio)/(1+taperRatio)];
end

csvwrite('data/hw2_2.csv', [ leadingEdgeSweeps' cLAlphas aerodynamicCenters ]);
```

hw2_2.m

```
aspectRatio = 5;
taperRatios = 0.1:0.1:1;
quarterChordSweep = 0;
wingTipTwist = 0;
horseshoeShape = 'rectangular';
nPanels = 100;
%Initialize output vectors
panels = (1:nPanels)';
cLYs = [];
for taperRatio = taperRatios
    %Compute cL for -2 and 2 degrees so we can draw a line
    [ cL1 ] = HVM(aspectRatio, taperRatio, quarterChordSweep, -2, wingTipTwist, horseshoeShape, nPanels);
    [ cL2 ] = HVM(aspectRatio, taperRatio, quarterChordSweep, 2, wingTipTwist, horseshoeShape, nPanels);
    cLAlpha = (cL2-cL1)/4;
    %Find angle of attack for a 0.25 lift coefficient
    alpha = 0.25/cLAlpha;
    %Find local lift distribution for computed angle of attack
    [ cL, cLY ] = HVM(aspectRatio, taperRatio, quarterChordSweep, alpha, wingTipTwist, horseshoeShape, nPanels);
    cLYs = [ cLYs cLY ];
end

csvwrite('data/hw2_3.csv', [ [ -1 taperRatios ]; panels cLYs ]);
```

hw2_3.m

```
aspectRatios = [ 4 8 10 ];
taperRatios = 0.1:0.1:1;
quarterChordSweep = 0;
wingTipTwist = 0;
horseshoeShape = 'rectangular';
nPanels = 100;
% Initialize output vector
oswaldFactors = [];
for aspectRatio = aspectRatios
  oswaldFactorsColumn = [];
  for taperRatio = taperRatios
    [ cL, cLY, cDi ] = HVM(aspectRatio, taperRatio, quarterChordSweep, -2, wingTipTwist, horseshoeShape, nPanels);
    oswaldFactorsColumn = [ oswaldFactorsColumn; cL^2/cDi/pi/aspectRatio ];
  end
  oswaldFactors = [ oswaldFactors oswaldFactorsColumn ];
end

csvwrite('data/hw2_4_aspect.csv', [ [ -1 aspectRatios ]; taperRatios' oswaldFactors ]);
```

hw2_4_aspect.m

```
aspectRatio = 8;
taperRatios = 0.1:0.1:1;
quarterChordSweeps = [ 0 30 60 ];
wingTipTwist = 0;
horseshoeShape = 'rectangular';
nPanels = 100;
% Initialize output vector
oswaldFactors = [];
for quarterChordSweep = quarterChordSweeps
  oswaldFactorsColumn = [];
  for taperRatio = taperRatios
    [ cL, cLY, cDi ] = HVM(aspectRatio, taperRatio, quarterChordSweep, -2, wingTipTwist, horseshoeShape, nPanels);
    oswaldFactorsColumn = [ oswaldFactorsColumn; cL^2/cDi/pi/aspectRatio ];
  end
  oswaldFactors = [ oswaldFactors oswaldFactorsColumn ];
end

csvwrite('data/hw2_4_sweep.csv', [ [ -1 quarterChordSweeps ]; taperRatios' oswaldFactors ]);
```

hw2_4_sweep.m

```
hw2_1;
hw2_2;
hw2_3;
hw2_4_aspect;
hw2_4_sweep;
```

shia_lebouf.m