Vortex Lattice Method Classwork

Facultad de Ingeniería Aeronáutica, UPB

Tomás Atencia
Juan Pablo Orrego
Pablo Lorduy

Abstract

This report presents the development and implementation of the Vortex Lattice Method (VLM) for the calculation of aerodynamic characteristics, such as lift and circulation, on wings with varying geometries. Unlike the traditional Lifting Line Theory (LLT), which is limited to high aspect ratio wings, the VLM allows for the analysis of wings with complex shapes, such as delta and arrow wings. The Python programming language was employed to create a numerical model capable of handling the wing geometry, control points, and horseshoe vortices distribution. The model was tested using a flat wing example, and results show the versatility of VLM in calculating aerodynamic forces without requiring prior airfoil data. The implementation is scalable, allowing the number of control points to be adjusted to refine the accuracy of the solution.

1 Introduction

This report explores the application of VLM using a Python-based numerical model, designed to compute lift characteristics for a flat wing surface with symmetrical airfoils or flat plates. By distributing horseshoe vortices across the wing and solving the induced velocities through the Biot-Savart law, VLM provides a more generalized approach for determining aerodynamic forces. The results obtained through this method, while neglecting viscous effects, provide accurate insights into the pressure-driven forces on the wing.

2 Mathematical model employed

The Vortex Lattice Method (VLM) is a tool used to calculate lift characteristics on wings of complex geometry, such as low aspect ratio wings, arrow wings and delta wings. Unlike the Load Line Theory (LLT), which is limited to straight wings with a high aspect ratio, the VLM allows varied wing shapes to be addressed. However, some considerations must be considered when analyzing the wing in question:

- The wing is treated as a flat surface, over which vortices are distributed. The flow is assumed to be potential, i.e., there are no viscous effects, and the wing generates only aerodynamic forces derived from pressure.
- The wing is represented by a vortex grid, where each vortex line represents a carrying vortex like the one used in the LLT, distributed on the wing geometry. These vortices are aligned in the flow direction (x-axis) and along the span (y-axis). Unlike the LLT, these vortices do not contain other vortices inside but are side by side.
- The induced velocities at the wing surface and in the wake behind the wing are calculated using the Biot-Savart law, which relates the force of a vortex to the induced velocity along a point in the flow.

3 Python Code

To obtain numerical results of the lift and Cl of the wing, Python has been used with the help of some libraries such as Pandas, Numpy, Math and Matplolib. Unlike LLT, in this method it is not necessary to obtain aerodynamic data of the airfoil previously, because it is assumed that we are working with symmetrical airfoils or a flat plate, so it was not necessary to use other software such as X-Foil or aerodynamic tables.

In order to use the VLM calculation model, it is first necessary to obtain the wing geometry, the x,y coordinates of the control points and the horseshoe vortices. Therefore, the code will be divided into 3 sections: Wing geometry, numerical solution using VLM and graphs and results. Something to keep in mind is that this method depends on the number of vortices (and therefore, control points) that we want to analyze in the wing, so although we based on an example in the course slides that used 4 control points (and as we will see later, the workshop only asked for 4 control points), everything in the code was put in terms of n (control points), so theoretically, we can increase that number as much as we want.

All code contains comments indicating the present section and actual thoughts from October 18 at approximately 1:00 am.

3.1 Geometry input data

To start, we import the necessary libraries and enter the geometrical data that define the wing geometry, define the n, and then we obtain the remaining values.

```
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

```
#VLT by: TOMY

#----------------------------------
#Parámetros
S = 0.2
AR = 5
Landa_w = 1
A_LE = 45 #deg

##########################
n = 4
##########################

b = math.sqrt(S*AR)
c_root = 2*S/(b*(1+Landa_w))
c_tip = c_root*Landa_w
A_c4 = A_LE #Deg
A_3c4 = A_LE #Deg

#----------------------------------
```

Subsequently, the existing information is used to obtain the coordinates of the four points that define the geometry of the wing and the points of the c/4 and 3c/4 line.

```
#Superficie del ala
X_wing_s = np.zeros(5)
Y_wing_s = np.zeros(5)

X_wing_s[0] = 0
X_wing_s[1] = (b/2)*math.tan(math.radians(A_LE))
X_wing_s[2] = X_wing_s[1] + c_tip
X_wing_s[3] = c_root
X_wing_s[4] = 0
Y_wing_s[0] = 0
Y_wing_s[1] = b/2
Y_wing_s[2] = Y_wing_s[1]
Y_wing_s[3] = 0
Y_wing_s[4] = 0
```

```
#---------------------------------------------------------
#C/4
X_c4 = np.zeros(2)
Y_c4 = np.zeros(2)

X_c4[0] = c_root*1/4
X_c4[1] = (c_tip*1/4)+(math.tan(math.radians(A_LE))*b/2)
Y_c4[0] = 0
Y_c4[1] = b/2

#3C/4
X_3c4 = np.zeros(2)
Y_3c4 = np.zeros(2)

X_3c4[0] = c_root*3/4
X_3c4[1] = (c_tip*3/4)+(math.tan(math.radians(A_LE))*b/2)
Y_3c4[0] = 0
Y_3c4[1] = b/2
```

In order to obtain the position of the horseshoe vortices and the position of the control points located along the line that is 3/4 of the chord, two functions are defined to calculate the slope, the first function works with a dy/dx, but the second uses the formula that excel uses to calculate the slope, because negative slopes can occur that give error when applying dy/dx.

```
#-------------------------------------------------------------
#Pendiente
def pendiente(x1, y1, x2, y2):
    if x2 != x1:  #No tocar pls
        pendiente = (y2 - y1) / (x2 - x1)
    else:
        return None
    return pendiente

def pendiente1(x1, y1, x2, y2):
    if x2 != x1:  #No tocar pls
        pendiente = ((x2-x1)*(y2-y1))/((x2-x1)**2)
    else:
        return None
    return pendiente
```

Now, having this function, we can proceed to calculate the coordinates of the horseshoe vortices and the control points.

```
#Divisiones Horseshoe
X_HorseshoeDiv = np.zeros((n+1)*2)
Y_HorseshoeDiv = np.zeros((n+1)*2)

iter = 1
for i in range((n+1)*2):
    if i == 0:
        X_HorseshoeDiv[0] = X_c4[0]
        Y_HorseshoeDiv[0] = Y_c4[0]
    elif i == 1:
        X_HorseshoeDiv[1] = 1
        Y_HorseshoeDiv[1] = Y_c4[0]
    elif i == len(X_HorseshoeDiv)-2:
        X_HorseshoeDiv[i] = X_c4[1]
        Y_HorseshoeDiv[i] = Y_c4[1]
    elif i == len(X_HorseshoeDiv)-1:
        X_HorseshoeDiv[i] = 1
        Y_HorseshoeDiv[i] = Y_c4[1]
    elif i%2 == 0 and i!=0:
        Y_HorseshoeDiv[i] = ((b/2)/n)*iter
        if X_c4[0] == X_c4[1]:
            X_HorseshoeDiv[i] = c_root*1/4
        else:
            X_HorseshoeDiv[i] = (c_root*1/4) +
pendiente1(Y_c4[0],X_c4[0],Y_c4[1],X_c4[1])*Y_HorseshoeDiv[i]
        iter = iter+1
    elif i%2 == 1 and i!=1:
        X_HorseshoeDiv[i] = 1
        Y_HorseshoeDiv[i] = Y_HorseshoeDiv[i-1]


#Puntos de control
X_controlpoint = np.zeros(n)
Y_controlpoint = np.zeros(n)

for i in range(n):
    Y_controlpoint[i] = ((Y_HorseshoeDiv[2]-
Y_HorseshoeDiv[0])/2)*(((i+1)*2)-1)
```

```
    X_controlpoint[i] = 
pendiente(Y_3c4[0],X_3c4[0],Y_3c4[1],X_3c4[1])*Y_controlpoint[i]+((c_root*3/
4))
```

To use the VLM, we need the following points:

- xm, ym, which refer to the coordinates of each control point.
- x1n, y1n, x2n, y2n which refer to the coordinates of the horseshoe vortices.

We have already calculated these points, but for the sake of practicality, we are going to organize them into vectors with the correct nomenclature and print the data in a table. It is worth remembering that here we are only considering the right wing of the aircraft, but the control point is affected by all the horseshoe vortices, including those on the left wing. To solve this, other vectors and another table are created for the coordinates of these points and vortices on the left side of the aircraft.

However, when looking at the example already done in the course presentations, we noticed that when analyzing a point on the right wing with a vortex on the left wing, the same ym value is not used for the left wing (although it is known that we are calculating the right wing), since the wing is on the other side of the x-axis, and its y values are negative, in addition, the value of x1n that they were using was unknown. Empirically, it was discovered that if x1n was changed to x2n, and y1n to y2n, and vice versa in the data for the left wing, we obtain the correct values and only a correction with a minus sign needs to be made.

s = starboard (Right), p = port (Left)

```
#-----------------------------------------------------------
#TABLA
Panel = np.zeros(n)
xms = np.zeros(n)
yms = np.zeros(n)
x1ns = np.zeros(n)
y1ns = np.zeros(n)
x2ns = np.zeros(n)
y2ns = np.zeros(n)

xmp = np.zeros(n)
ymp = np.zeros(n)
x1np = np.zeros(n)
y1np = np.zeros(n)
x2np = np.zeros(n)
y2np = np.zeros(n)

for i in range(n):
    Panel[i] = i+1
    xms[i] = X_controlpoint[i]
    yms[i] = Y_controlpoint[i]
    x1ns[i] = X_HorseshoeDiv[i*2]
    y1ns[i] = Y_HorseshoeDiv[i*2]
    x2ns[i] = X_HorseshoeDiv[(i+1)*2]
    y2ns[i] = Y_HorseshoeDiv[(i+1)*2]


    xmp[i] = X_controlpoint[i]
    ymp[i] = -Y_controlpoint[i] #MAN, I HATE THE NEGATIVE
    x1np[i] = X_HorseshoeDiv[(i+1)*2] #IDK WHAT'S GOING ON
    y1np[i] = Y_HorseshoeDiv[(i+1)*2] #EMPIRICAL SOLUTION :)
    x2np[i] = X_HorseshoeDiv[i*2]
    y2np[i] = Y_HorseshoeDiv[i*2]

#DataFrame
tabla_s = pd.DataFrame({
    'Panel': Panel,
    'xm': xms,
    'ym': yms,
    'x1n': x1ns,
    'y1n': y1ns,
    'x2n': x2ns,
    'y2n': y2ns
})

tabla_p = pd.DataFrame({
    'Panel': Panel,
    'xm': xmp,
    'ym': ymp,
    'x1n': x1np,
    'y1n': y1np,
    'x2n': x2np,
    'y2n': y2np
})#By:Tomy

print(tabla_s)
print("-----------------------------------")
print(tabla_p)
```

3.2 VLM Solution

Here the following equation is executed for each control point on the wing, the negative is also included which gives the correct geometric sense to the circulation coefficients calculated with the horseshoe vortices of the port.

$$
\begin{aligned}
w_{m,n} = \frac{\Gamma_n}{4\pi} \Bigg\{ & \frac{1}{(x_m - x_{1n})(y_m - y_{2n}) - (x_m - x_{2n})(y_m - y_{1n})} \\
& \left[\frac{(x_{2n} - x_{1n})(x_m - x_{1n}) + (y_{2n} - y_{1n})(y_m - y_{1n})}{\sqrt{(x_m - x_{1n})^2 + (y_m - y_{1n})^2}} \right. \\
& \left. - \frac{(x_{2n} - x_{1n})(x_m - x_{2n}) + (y_{2n} - y_{1n})(y_m - y_{2n})}{\sqrt{(x_m - x_{2n})^2 + (y_m - y_{2n})^2}} \right] \\
& + \frac{1}{y_{1n} - y_m} \left[1 + \frac{x_m - x_{1n}}{\sqrt{(x_m - x_{1n})^2 + (y_m - y_{1n})^2}} \right] \\
& - \frac{1}{y_{2n} - y_m} \left[1 + \frac{x_m - x_{2n}}{\sqrt{(x_m - x_{2n})^2 + (y_m - y_{2n})^2}} \right] \Bigg\}
\end{aligned}
$$

```python
#MATRIZ
Comp_s = np.zeros((n,n))
Comp_p = np.zeros((n,n))
Comp = np.zeros((n,n))

for m in range(n):
    for u in range(n):
        Comp_s[m,u] = (1/((xms[m]-x1ns[u])*(yms[m]-y2ns[u])-(xms[m]-
x2ns[u])*(yms[m]-y1ns[u])))*((((x2ns[u]-x1ns[u])*(xms[m]-x1ns[u])+(y2ns[u]-
y1ns[u])*(yms[m]-y1ns[u]))/(math.sqrt(((xms[m]-x1ns[u])**(2))+(yms[m]-
y1ns[u])**(2))))-(((x2ns[u]-x1ns[u])*(xms[m]-x2ns[u])+(y2ns[u]-
y1ns[u])*(yms[m]-y2ns[u]))/(math.sqrt(((xms[m]-x2ns[u])**(2))+(yms[m]-
y2ns[u])**2))))+(1/(y1ns[u]-yms[m]))*(1+((xms[m]-
x1ns[u])/(math.sqrt(((xms[m]-x1ns[u])**(2))+(yms[m]-y1ns[u])**(2)))))-
(1/(y2ns[u]-yms[m]))*(1+((xms[m]-x2ns[u])/(math.sqrt(((xms[m]-
x2ns[u])**(2))+(yms[m]-y2ns[u])**(2)))))

for m in range(n):
    for u in range(n):
        Comp_p[m,u] = (1/((xmp[m]-x1np[u])*(ymp[m]-y2np[u])-(xmp[m]-
x2np[u])*(ymp[m]-y1np[u])))*((((x2np[u]-x1np[u])*(xmp[m]-x1np[u])+(y2np[u]-
y1np[u])*(ymp[m]-y1np[u]))/(math.sqrt(((xmp[m]-x1np[u])**(2))+(ymp[m]-
y1np[u])**(2))))-(((x2np[u]-x1np[u])*(xmp[m]-x2np[u])+(y2np[u]-
y1np[u])*(ymp[m]-y2np[u]))/(math.sqrt(((xmp[m]-x2np[u])**(2))+(ymp[m]-
y2np[u])**2))))+(1/(y1np[u]-ymp[m]))*(1+((xmp[m]-
x1np[u])/(math.sqrt(((xmp[m]-x1np[u])**(2))+(ymp[m]-y1np[u])**(2)))))-
(1/(y2np[u]-ymp[m]))*(1+((xmp[m]-x2np[u])/(math.sqrt(((xmp[m]-
x2np[u])**(2))+(ymp[m]-y2np[u])**(2)))))
        Comp_p[m,u] = -Comp_p[m,u] #Another empirical correction, I'm sleepy

#SUMA DE STARBOARD Y PORT
for i in range(n):
    for j in range(n):
        Comp[i,j] = Comp_s[i,j]+Comp_p[i,j]

print("-------------------------------------")
print(Comp_s)
print("-------------------------------------")
print(Comp_p)
print("-------------------------------------")
print(Comp)
print("-------------------------------------")
```

Now, we can solve the system of equations by setting the coefficient vector as 1 and obtain the circulation values without considering (for now) the medium in which the wing operates.

```
#------------------------------------------------------------
#Solucion del sistema de ecuaciones


Coeficientes = np.zeros(n)
for i in range(n):
    Coeficientes[i] = -1

Sol = np.linalg.solve(Comp,Coeficientes)
print(Sol)
```

Once the system of equations has been solved, we can begin to consider the angle of attack of the profile and the environmental conditions.

```
#Valor test de Lift
#=================================
alphatest = math.radians(2)     ############## ////////////// AoA
V = 6 #m/s
rho = 1.225
q = (1/2)*rho*V**2
#=================================

Sol_Sum = 0
for i in range(n):
    Sol_Sum = Sol_Sum + Sol[i]
ValueCond = 2*rho*(V**2)*4*math.pi*b*y2ns[0]
Ltest = ValueCond*Sol_Sum*alphatest

print(Sol_Sum)
print("------------------------------------")
print(Ltest, "N")
print("------------------------------------")

CLtest = Ltest/(q*S)
print(CLtest)

#-------------------------------------------------------------
#CL vs Angle (Here comes the good thing :D, I'm SOOOO SLEEEPY)
angles = 10 #Sin contar a=0 ###################################
alpha = np.zeros(angles+1)
Lift = np.zeros(angles+1)
CL = np.zeros(angles+1)

for i in range(angles+1):
    alpha[i] = i
    Lift[i] = ValueCond*Sol_Sum*math.radians(alpha[i])
    CL[i] = Lift[i]/(q*S)
```

3.3 Plots and Data

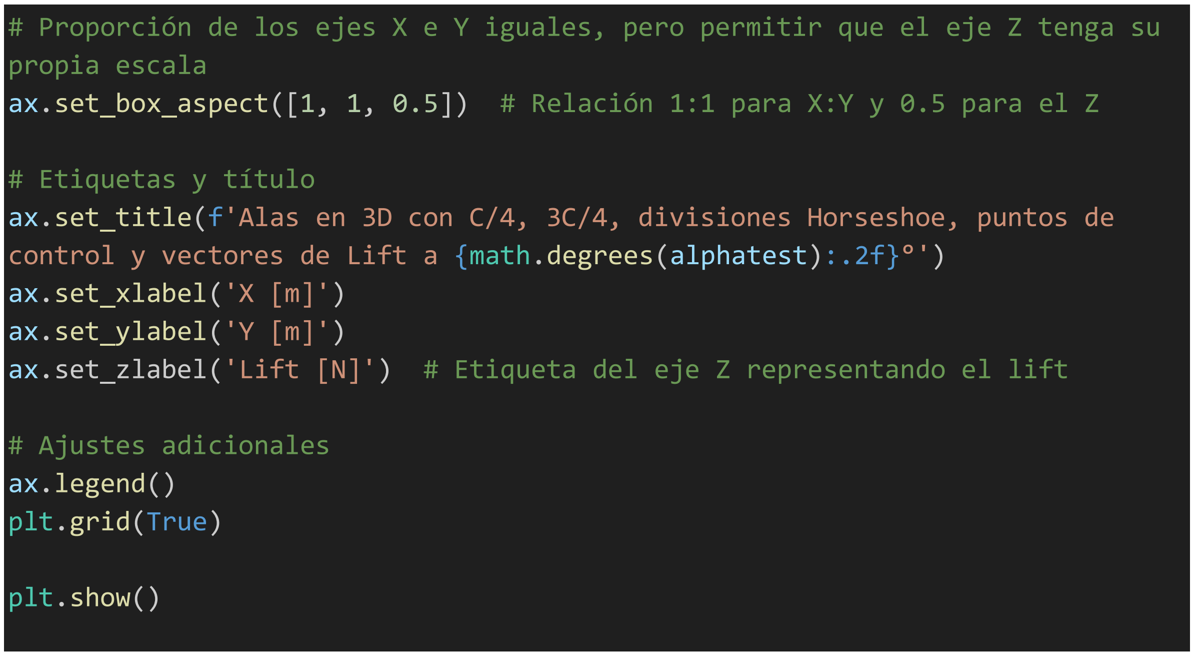
```
#Graficas (Gracias a CHATGPT :D)
# Parámetros del ala
```

```
X_wing_s = np.array([0, (b/2)*math.tan(math.radians(A_LE)),
(b/2)*math.tan(math.radians(A_LE)) + c_tip, c_root, 0])
Y_wing_s = np.array([0, b/2, b/2, 0, 0])

# Gráfica del ala sin puntos
plt.figure()
plt.plot(X_wing_s, Y_wing_s, '-', color='black', label='Contorno del
ala')  # Solo líneas

# Añadir líneas de C/4 y 3C/4
plt.plot(X_c4, Y_c4, '--', color='blue', label='C/4')  # Línea discontinua
azul para C/4
plt.plot(X_3c4, Y_3c4, '--', color='red', label='3C/4')  # Línea discontinua
roja para 3C/4

# Añadir divisiones Horseshoe
for i in range(0, len(X_HorseshoeDiv), 2):
    plt.plot([X_HorseshoeDiv[i], X_HorseshoeDiv[i+1]],
             [Y_HorseshoeDiv[i], Y_HorseshoeDiv[i+1]],
             '--', color='blue')  # Mismo color y estilo que C/4

# Añadir puntos de control
plt.scatter(X_controlpoint, Y_controlpoint, color='green', zorder=5,
label='Puntos de control')  # Puntos verdes

# Ajustes de la gráfica
plt.fill(X_wing_s, Y_wing_s, 'lightblue', alpha=0.6)  # Sombreado azul claro
plt.title('Contorno del ala con C/4, 3C/4, divisiones Horseshoe y puntos de
control')
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
plt.grid(True)
plt.axis('equal')
plt.legend()

fig, axs = plt.subplots(1, 2, figsize=(12, 5))

#------------------------------------------------------------------
# Graficar Lift vs Alpha
axs[0].plot(alpha, Lift, 'b-', marker='o', label='Lift vs Alpha')
axs[0].set_title('Lift vs Alpha')
axs[0].set_xlabel('Alpha [°]')
```

```
axs[0].set_ylabel('Lift [N]')
axs[0].grid(True)
axs[0].legend()

# Graficar Cl vs Alpha
axs[1].plot(alpha, CL, 'r-', marker='o', label='Cl vs Alpha')
axs[1].set_title('Cl vs Alpha')
axs[1].set_xlabel('Alpha [°]')
axs[1].set_ylabel('Cl')
axs[1].grid(True)
axs[1].legend()
#By: Tomy
# Ajustar la presentación de las gráficas
plt.tight_layout()

#---------------------------------------------------------------------------
----
# Definir las coordenadas en el espacio 3D (Z se mantiene en cero)
Z_wing_s = np.zeros(len(X_wing_s))  # El ala está en el plano Z=0
Z_c4 = np.zeros(len(X_c4))
Z_3c4 = np.zeros(len(X_3c4))
Z_HorseshoeDiv = np.zeros(len(X_HorseshoeDiv))
Z_controlpoint = np.zeros(len(X_controlpoint))

# Calcular el lift en cada punto de control
Lift_vectors = np.zeros(n)
for i in range(n):
    Lift_vectors[i] = (ValueCond/2) * Sol[i] * alphatest

#'''''''''''''''''''''''''''''''''''''''''''''''''''''''#"#"###"#"#"#"#""
# Crear figura en 3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Ajustar las coordenadas Z para que el ala esté en Z=0
Z_wing_s = np.zeros(len(X_wing_s))
Z_c4 = np.zeros(len(X_c4))
Z_3c4 = np.zeros(len(X_3c4))
Z_HorseshoeDiv = np.zeros(len(X_HorseshoeDiv))
Z_controlpoint = np.zeros(len(X_controlpoint))

# Graficar contorno del ala derecha (en Z=0)
```

```
ax.plot(X_wing_s, Y_wing_s, Z_wing_s, '-', color='black', label='Contorno
del ala derecha')

# Graficar línea de C/4 de la ala derecha (en Z=0)
ax.plot(X_c4, Y_c4, Z_c4, '--', color='blue', label='C/4 derecha')

# Graficar línea de 3C/4 de la ala derecha (en Z=0)
ax.plot(X_3c4, Y_3c4, Z_3c4, '--', color='red', label='3C/4 derecha')

# Graficar divisiones Horseshoe de la ala derecha (en Z=0)
for i in range(0, len(X_HorseshoeDiv), 2):
    ax.plot([X_HorseshoeDiv[i], X_HorseshoeDiv[i+1]],
            [Y_HorseshoeDiv[i], Y_HorseshoeDiv[i+1]],
            [Z_HorseshoeDiv[i], Z_HorseshoeDiv[i+1]],
            '--', color='blue')

# Graficar puntos de control de la ala derecha (en Z=0)
ax.scatter(X_controlpoint, Y_controlpoint, Z_controlpoint, color='green',
label='Puntos de control derecha', zorder=5)

# Añadir vectores de lift de la ala derecha
for i in range(n):
    ax.quiver(X_controlpoint[i], Y_controlpoint[i], Z_controlpoint[i],
              0, 0, Lift_vectors[i], color='orange', label='Lift vector
derecha' if i == 0 else "", arrow_length_ratio=0.1)

# Graficar contorno del ala izquierda (espejado respecto al eje Y)
X_wing_s_left = X_wing_s
Y_wing_s_left = -Y_wing_s
X_c4_left = X_c4
Y_c4_left = -Y_c4
X_3c4_left = X_3c4
Y_3c4_left = -Y_3c4
X_HorseshoeDiv_left = X_HorseshoeDiv
Y_HorseshoeDiv_left = -Y_HorseshoeDiv
X_controlpoint_left = X_controlpoint
Y_controlpoint_left = -Y_controlpoint

# Graficar ala izquierda
ax.plot(X_wing_s_left, Y_wing_s_left, Z_wing_s, '-', color='black',
linestyle='--', label='Contorno del ala izquierda')
```

```python
ax.plot(X_c4_left, Y_c4_left, Z_c4, '--', color='blue', linestyle='--',
label='C/4 izquierda')
ax.plot(X_3c4_left, Y_3c4_left, Z_3c4, '--', color='red', linestyle='--',
label='3C/4 izquierda')

# Graficar divisiones Horseshoe de la ala izquierda
for i in range(0, len(X_HorseshoeDiv_left), 2):
    ax.plot([X_HorseshoeDiv_left[i], X_HorseshoeDiv_left[i+1]],
            [Y_HorseshoeDiv_left[i], Y_HorseshoeDiv_left[i+1]],
            [Z_HorseshoeDiv[i], Z_HorseshoeDiv[i+1]],
            '--', color='blue')

# Graficar puntos de control de la ala izquierda
ax.scatter(X_controlpoint_left, Y_controlpoint_left, Z_controlpoint,
color='green', marker='^', label='Puntos de control izquierda', zorder=5)

# Añadir vectores de lift de la ala izquierda
for i in range(n):
    ax.quiver(X_controlpoint_left[i], Y_controlpoint_left[i],
Z_controlpoint[i],
              0, 0, Lift_vectors[i], color='orange', linestyle='--',
label='Lift vector izquierda' if i == 0 else "", arrow_length_ratio=0.1)

# Añadir sombreado para la ala derecha
ax.plot_trisurf(X_wing_s, Y_wing_s, Z_wing_s, color='lightblue', alpha=0.6,
linewidth=0)

# Añadir sombreado para la ala izquierda
ax.plot_trisurf(X_wing_s, Y_wing_s_left, Z_wing_s, color='lightblue',
alpha=0.6, linewidth=0)

# Calcular el mayor rango entre X e Y para establecer proporciones iguales
max_range = max(max(abs(X_wing_s)), max(abs(Y_wing_s)), b/2)

# Ajustar el rango de los ejes X y Y para que tengan el mismo tamaño
ax.set_xlim([-max_range, max_range])
ax.set_ylim([-max_range, max_range])

# Ajustar el rango del eje Z para ser el doble del lift máximo
ax.set_zlim(0, 2 * max(Lift_vectors))
```



```
# Proporción de los ejes X e Y iguales, pero permitir que el eje Z tenga su
propia escala
ax.set_box_aspect([1, 1, 0.5])  # Relación 1:1 para X:Y y 0.5 para el Z

# Etiquetas y título
ax.set_title(f'Alas en 3D con C/4, 3C/4, divisiones Horseshoe, puntos de
control y vectores de Lift a {math.degrees(alphatest):.2f}°')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_zlabel('Lift [N]')  # Etiqueta del eje Z representando el lift

# Ajustes adicionales
ax.legend()
plt.grid(True)

plt.show()
```