pySailingVLM Documentation

Zuzanna Wieczorek, Grzegorz Gruszczyński

Jun 09, 2023

CONTENTS

The beginning of fluid dynamics has started in acient Greece, when Archimedes investigated the fluid statics and bouancy. From that time, many well-konwn mathematicians and engineers have contributed to defining how air and gases behave. The equations they have constructed are extremely complex and difficult to calculate by hand when the geometry of airfoil is complicated. The rapid improvement of computers' power in 1970s has led to developing complex flow desings. Over the past few decades Computational Fluid Dynamics (CFD) has been improved dramatically.

There are many methods, that are used in Computational Fluid Dynamics. Some of them are Finite Element Method (FEM), Volume of Fluid (VOF) and Lattice Boltzmann Method (LBM). They provide good and accurate results but require high amount of computational resources. To overcome this problem, low fidelity tools can be used, like Vortex Lattice Method (VLM). Using VLM is benefical in the early conceptual design phase [NJS+21], when many different desings should be tested. Thanks to fast computations of aerodynamics forces and pressures, engineers can quickly overview modelled design performance.

In this work, we present the first open source Python package which implements Vortex Lattice Method for initial aerodynamic analysis of upwind sails. Thanks to its light weight requirements, the software can be immediately installed and execuded locally or accessed by cloud environment such as Google Collab. Additionally, package users can define own sail geometries and use pySailingVLM inside custom scripts which makes creating a set of dynamics very convenient.

Keywords: Vortex Lattice Method (VLM), initial sail analysis, yacht engineering, Python Package

Inside documentation:

Part I

Description

CHAPTER

ONE

THEORY

The Vortex Lattice Method (VLM) is a numerical method used in computational fluid dynamics. VLM models a surface on aircraft as infinite vortices to estimate the lift curve slope, induced drag, and force distribution. The VLM is the extension of Prandtl's lifting-line theory that is capable of computing swept and low aspect ratio wings. [SHR+19].

1.1 Introduction

Vortex Lattice Method is build on the Potential flow theory. The viscous effects, drag and flow separation in VLM is sufficient for aproximatting ideal flow seen in nature.

1.2 Potential flow theory

Potenttial Flow Theory treats external flows around bodies as invicid and irrotational [Tec05] .The viscous effects are limitted to a thin layer next to the body (boundary layer). Becouse of this and separation phenomena such fluid can be modelled only with small angle of attack. In irrotational flow fluid particles are not rotating.

In Potential Flow Theory, because velocity must satisfy the conservation of mass equation, the Laplace Equation is qoverning:

$$\nabla^2 \phi = 0$$

where ϕ is a potentinal function defined as continous function which satisfies the conservation of mass and momentum (incompressible, inviscid and irrational flow).

CHAPTER

TWO

VORTEX LATTICE METHOD

Vortex Lattice Method allows the computation of induced wind. VLM is a panel method where wing or other configuration is modelled by a large number of elementary, quadrilateral panels lying either on the actual aircraft surface, or on some mean surface, or combination thereof. To satisfy boundary conditions VLM uses following elements attached to each panel:

- source - a point from which fluid issues and flows radially outward such that the continuity equation is satisfied everywhere but at the singularity that exists at the source's center
- sink - a negative source
- doublet - singularity resulting when a source or a sink of equal strength are made to approach each other such that the product of their strangths and their distance apart remains constant at a preselected finite value in the limit as a distance between them approaches zero
- vortex - an element that generates a circulation, or tangential motion, around its origin

Such singularities are defined by specifying functional variation across the panel and its value is set by determinating strength parameters. Such parameters are known after solving appropriate boundary condition equations. When singularity strengts are determinated, the pressure, velocity can be computed[JJB09].

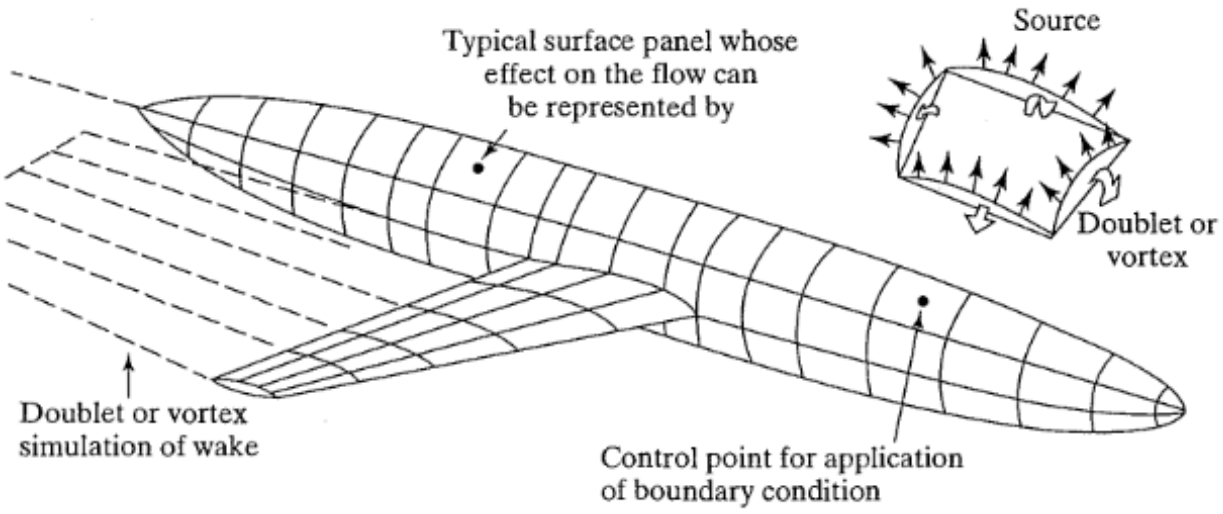


Fig. 2.1: Representation of an airplane flowfield by panel (or singularity) methods. Figure taken from [JJB09] (Fig. 7.22 page 350).

2.1 Vortices

The vortex theorems for inviscid incompressible flows has been developed by German scientist Hermann von Helmholtz (1821-1894). The theory is based on the following assumptions [JK01]:

- Flow is inviscid and incompressible
- The strength of vortex filament is constant along its length
- A vortex filament cannot start or end in a fluid (it must form or closed path or extend to infinity)
- The fluid that forms a vortex tube continues to form a vortex tube and the strength of the vortex tube remains constant as the tube moves about.

2.1.1 Vortex element

Vortex element in 2D is presented on figure 2.2. It is created by placing a rotating cylindrical core in two dimentional flowfield. If the cylinder has a radius R and a constant angular velocity of ω_y, then its motion results in a flow with circular streamlines [JK01].

The tangential velocity induced is described by equation:

$$q_\theta(r) = \frac{\Gamma}{2\pi \cdot r}$$

where r is described as distance from the vortex core.

If a vortex is located in free-strem with uniform velocity u_∞ then the total velocity at the distance r can be writtes as $\overrightarrow{u_\infty} + \overrightarrow{q_\theta(r)}$ [AH13].

2.1.2 Vortex segment

The velocity induced by a straight vortex segement (figure 2.3) in three dimentions with given vortex strength Γ at point $P(x, y, z)$ is qiven by equation:

$$\overrightarrow{w_{1,2}} = \frac{\Gamma}{4\pi} \frac{\overrightarrow{r_1} \times \overrightarrow{r_2}}{|\overrightarrow{r_1} \times \overrightarrow{r_2}|^2} \left(\frac{\overrightarrow{r_1}}{||\overrightarrow{r_1}||} - \frac{\overrightarrow{r_2}}{||\overrightarrow{r_2}||} \right) \cdot \overrightarrow{r_0} = \vec{\nu}\,\Gamma \tag{2.1}$$

where

$$\overrightarrow{r_0} = \overrightarrow{r_1} - \overrightarrow{r_2}$$

and

$$\begin{aligned} \overrightarrow{r_0} &= (x_2, y_2, z_2) - (x_1, y_1, z_1) \\ \overrightarrow{r_1} &= P - (x_1, y_1, z_1) \\ \overrightarrow{r_2} &= P - (x_2, y_2, z_2) \end{aligned}$$

In case of a semi-infinite vortex line, the point 2 is infinitely far away thus formula reads [PS00]:

$$\overrightarrow{w_{1,2}} = \frac{\Gamma}{4\pi} \frac{\overrightarrow{u_\infty} \times \overrightarrow{r_1}}{||\overrightarrow{r_1}||(||\overrightarrow{r_1}|| - \overrightarrow{u_\infty} \cdot \overrightarrow{r_1})} = \vec{\nu}\,\Gamma \tag{2.2}$$

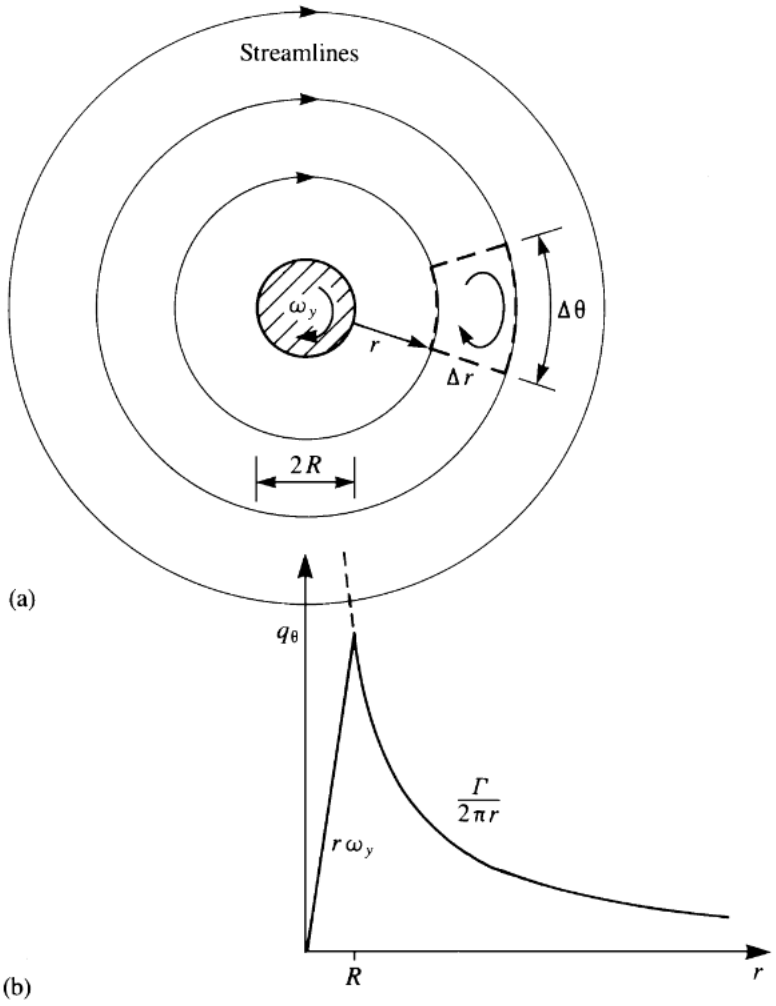


Fig. 2.2: Two dimentional flowfield around a cylindrical core rotating as a rigid body. Figure taken from [JK01] (Fig. 2.11 page 34).

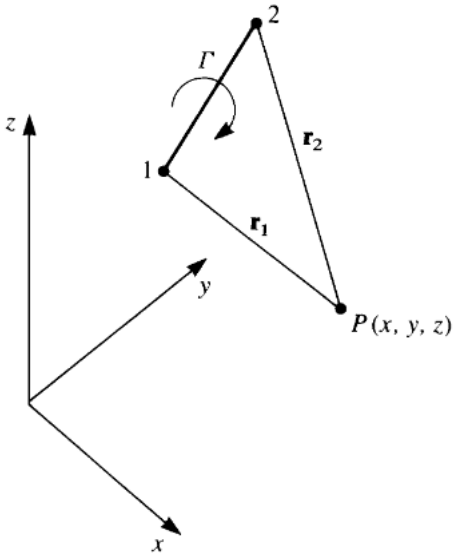


Fig. 2.3: Vortex segment in three dimentions. Figure taken from [JK01] (Fig. 2.16 page 40).

2.2 The Kutta Condition

The Kutta Condition states that at small angle of attack the flow leaves the sharp trailing edge of an airfoil smoothly and the velocity is finite there [JK01]. Because of this the normal component of velocity, from both sides of the airfoil, must vanish. Circulation at trailing edge can be expressed by equation:

$$\gamma_{T.E.} = 0$$

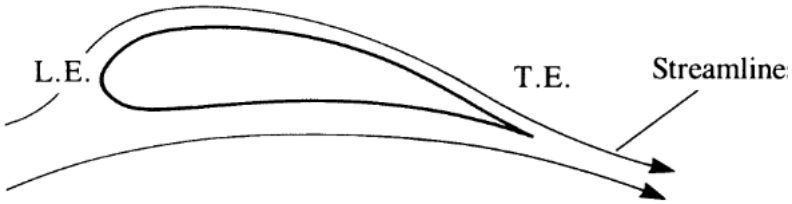


Fig. 2.4: Flow near cusped trailing edge. Figure taken from [JK01] (Fig. 4.12 page 89)

2.3 Lifting surface

The surface of the object is divided into panels (gray rectanges on figure 2.5). The total amount of them is a product of number of panels spanwise and chordwise. Vortex element is associated to each one of them (pink rectangles).

There are two types of vortex elements:

- vortex ring
- vortex horseshoe

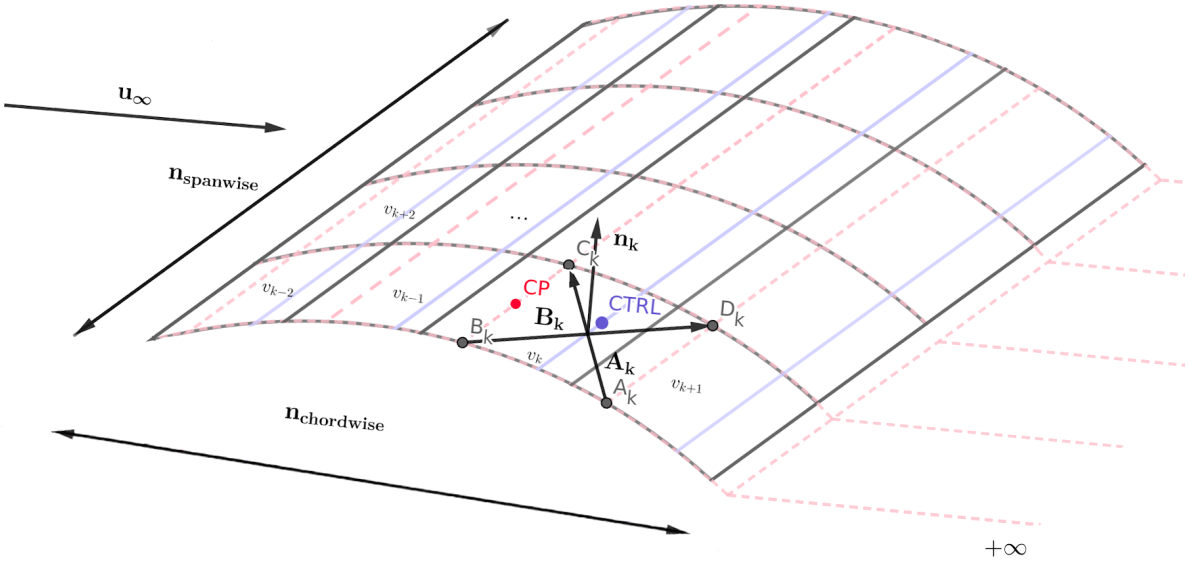


Fig. 2.5: Nomenclature of the lattice elements. Figure created by author.

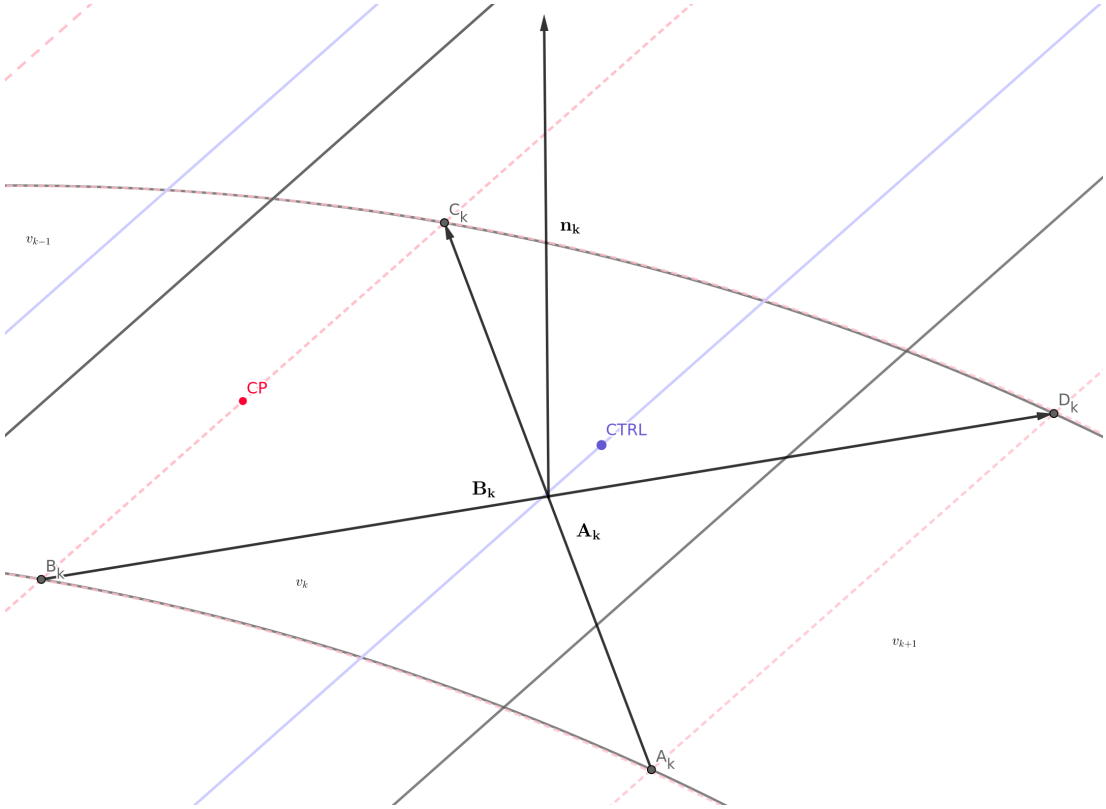


Fig. 2.6: Nomenclature of the vortex ring. Figure created by author.

Vortex ring (figure 2.6) is created by four finite segments according to equation (2.1). Horseshoe vortex is attached to the trailing edge of lifting surface. It consists of three finite and two semi-infinite segments following equation (2.1) and (2.2) (see figure 2.7). By placing vortex at the quarter chord line of the the two-dimensional Kutta condition is satisfied along the chord. Also, along the wing trailing edges, the trailing vortex of the last panel row must be canceled to satisfy the three-dimensional trailing-edge condition (see figure 2.8).

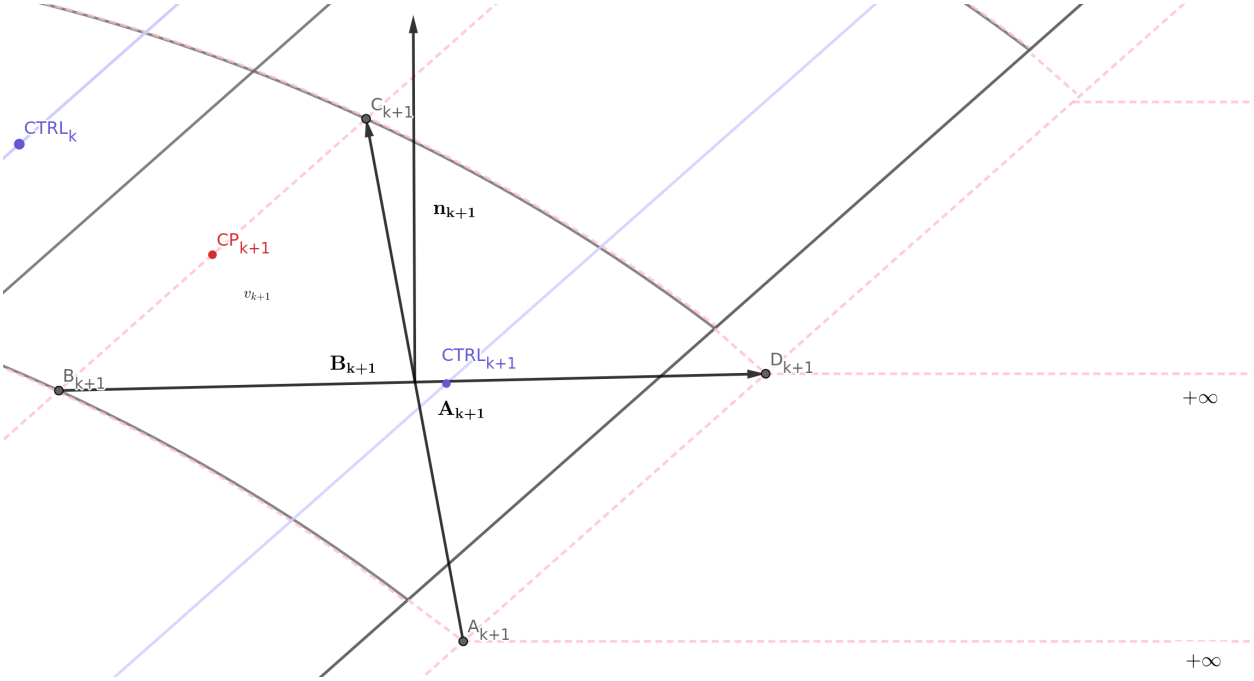


Fig. 2.7: Nomenclature of the vortex horseshoe. Figure created by author.

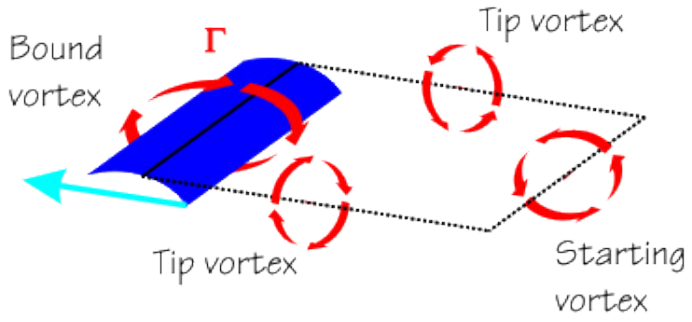


Fig. 2.8: The total amount of circulation is preserved. Figure 6 from [Gru21].

When thin surface is angled to a free-stream $\overrightarrow{u_\infty}$, the aerodynamic force is being generated at center of pressure. This point is located at the $1/4$ of a panel chord and at $1/2$ of the panel span (from panel leading edge). To fulfill no flow through the surface, the control point is defined at $3/4$ of the chord from the panel leading edge and in the middle of the span.

Vortex vertices B_k and C_k are placed at $1/4$ of panel chord (v_k), A_k and D_k at $1/4$ of the next panel (v_{k+1}). The panel opposite corner points define two vectors $\overrightarrow{A_k}$ and $\overrightarrow{B_k}$, and their vector product will point in the direction of $\overrightarrow{n_k}$ (normal vector).

2.3.1 Computations

To converse which claims that there should be of the k-th panel, the equation below is set:

If the velocity induced at k-th panel is $\overrightarrow{q_{ind}}$, no flow through the surface (boundary condition) is fulfield when:

$$(\overrightarrow{u_\infty} + \overrightarrow{w_k}) \cdot \overrightarrow{n_k} = 0 \tag{2.3}$$

After equation transformation:

$$\sum_j \overrightarrow{\nu_{kj}} \Gamma_j \cdot \overrightarrow{n_k} = -\overrightarrow{u_\infty} \cdot \overrightarrow{n_k} \tag{2.4}$$

where $\overrightarrow{\nu_{kj}}$ is defined as coefficient of proportionality of induced velocity $\overrightarrow{q_{ind}}$ at k-th control point by j-th vortex.

By expanding equation (2.3) and (2.4) the RHS coefficient vector can be computed:

$$RHS_k = -\overrightarrow{u_\infty} \cdot \overrightarrow{n_k}$$

In order to obtain gamma magnitude at k-th panel, the following set of algebraic equations must be solved:

$$\begin{bmatrix} a_{11} & \dots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mm} \end{bmatrix} \begin{bmatrix} \Gamma_1 \\ \vdots \\ \Gamma_m \end{bmatrix} = \begin{bmatrix} RHS_1 \\ \vdots \\ RHS_m \end{bmatrix} \tag{2.5}$$

where $m = n_{spanwise} \cdot n_{chordwise}$ and $a_{kj} = \overrightarrow{\nu_{kj}} n_k$

The aerodynamics force can be expressed according to the Kutta-Joukowski theorem as:

$$\overrightarrow{F_{areo}} = \rho \overrightarrow{u_\infty} \times \vec{b} \Gamma$$

Where $\vec{b}$ is span vector.

Discretizing, the aerodunamic force corresponding to the j-th section is:

$$\begin{aligned} \overrightarrow{F_j} = \rho \overrightarrow{V}_{app\_wind\_fs\_j} \times \overrightarrow{b_j} \Gamma_j = \\ \rho(\overrightarrow{V}_{app\_wind\_infs\_j} + \vec{q}_{ind\_j}) \times \overrightarrow{b_j} \Gamma_j = \\ \rho(\overrightarrow{V}_{app\_wind\_infs\_j} + \sum \overrightarrow{\nu_{jk}} \Gamma_k) \times \overrightarrow{b_j} \Gamma_j \end{aligned}$$

where b_j is a vector representing a finite vortex filement going through the center of pressure of the j-th section, $\overrightarrow{V}_{app\_wind\_infs\_j}$ is apparent wind velocity for an 'infinite sail' (without induced wind velocity) and $\overrightarrow{V}_{app\_wind\_fs\_j}$ is apparent wind velocity for a finite sail' (with induced wind velocity).

Lift is defined as the component of the aerodynamic force that is perpendicular to the flow direction ($\overrightarrow{u_\infty}$) and can is defined as a dot product of force and normal vector:

$$L_k = \overrightarrow{F_k} \cdot \overrightarrow{n_k}$$

Pressure at k-th panel:

$$p_k = \frac{L_k}{S_k}$$

Pressure coefficient at k-th panel can be defined as:

$$c_p = \frac{p_k}{\frac{1}{2} \rho \left|\left| \overrightarrow{u_{\infty_k}} \right|\right|^2}$$

CHAPTER

THREE

COORDINATE SYSTEM AND WIND

The surface of the geometry used is defined with coordinates in three dimensions (figure 3.1) with the x-axis in the yachts longitudinal (chordwise) direction positive backwards, the y-axis in the transverse direction positive to starboard and the z-axis in spanwise direction positive upwards.

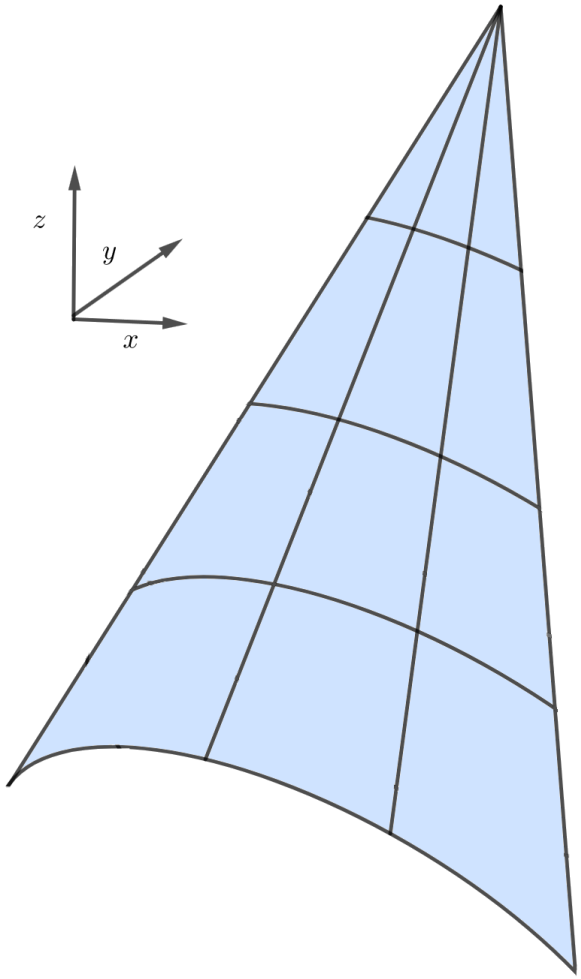


Fig. 3.1: Sail coordinate system. Figure by author.

A boat moves along x-axis with the $\overrightarrow{V}_{yacht}$ velocity (figure 3.2). The speed of yacht measured directly from the sailor point of view is desscribed by vectors $\overrightarrow{V}_{app\_wind\_infs}$ (without induced wind velocity $\overrightarrow{w}$) and $\overrightarrow{V}_{app\_wind}$ (with induced

wind velocity). An aerodynamic force is generated and is perpendicular to the apparent wind velocity. The profile of apparent wind is twisted (see figure 3.3).

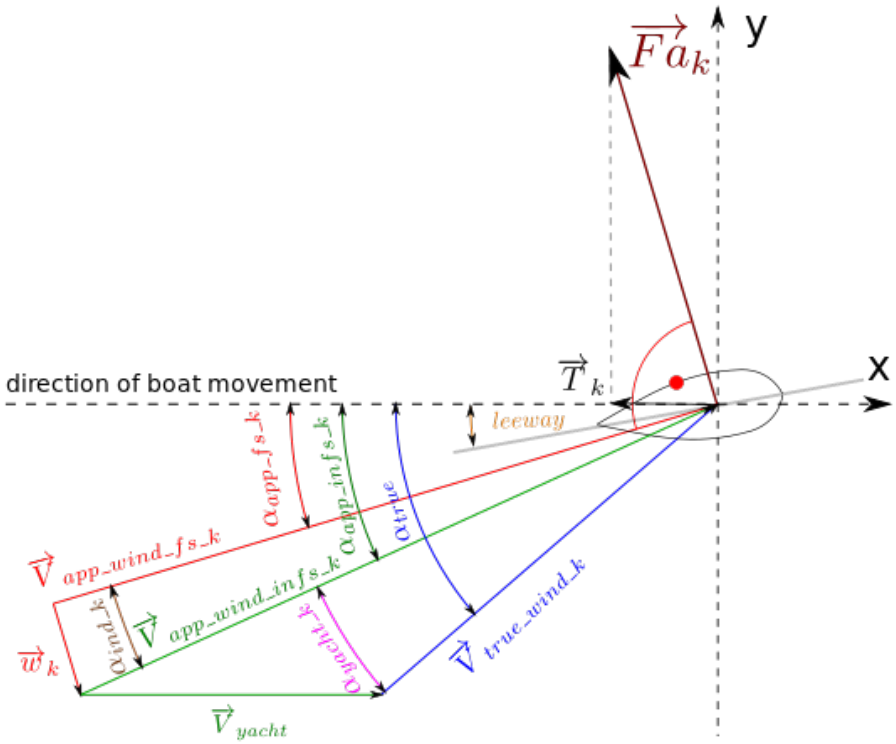


Fig. 3.2: Yacht coordinate system. Figure from [Gru21] p.18.

Table 3.1: Nomenclature

Name	Description
$\vec{V}_{yacht}$	angle between apparent wind and true wind
$\vec{V}_{app\_wind\_infs\_k}$	apparent wind velocity for an 'infinite sail' (without induced wind velocity) acting on k-th panel
$\vec{V}_{app\_wind\_fs\_k}$	apparent wind velocity for an 'finite sail' (with induced wind velocity) acting on k-th panel
$\vec{V}\_true\_wind\_k$	true wind velocity acting on k-th panel
$\vec{w}\_k$	induced wind velocity acting on k-th panel
$\alpha_{app\_infs\_k}$	angle between apparent wind acting on k-th panel for an 'infinite sail' (without induced wind velocity) and direction of boat movement (including leeway)
$\alpha_{app\_fs\_k}$	angle between apparent wind of a 'finite sail' acting on k-th panel (with induced wind velocity) and direction of boat movement (including leeway)
α_{ind_k}	angle between apparent wind acting on k-th panel of a 'finite sail' (induced wind velocity) and apparent wind of an 'infinite sail' (without induced wind velocity)
α_{yacht_k}	angle between apparent wind acting on k-th panel and true wind
α_{true}	angle between true wind acting on k-th panel and direction of boat movement with reference to course over ground (i.e. including leeway) [deg]
$\vec{F}_a$	aerodynamic force acting on k-th panel
$\vec{T}$	thrust force in direction of boat movement - Speed Over Ground (SOG)

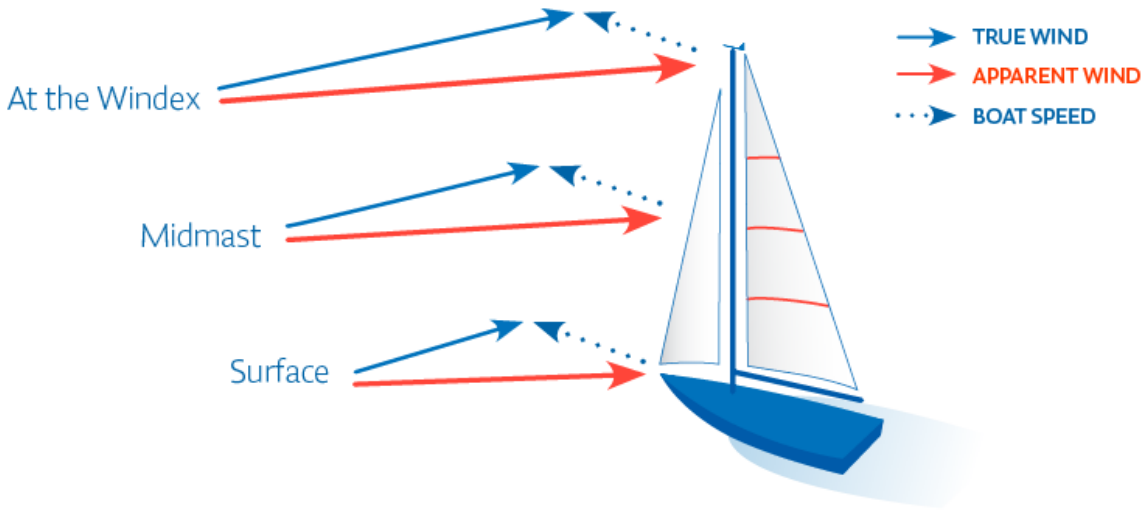


Fig. 3.3: Profile of apparent wind on sails [NS18].

It is worth adding that the induced wind causes extra drag (see figure 3.4)

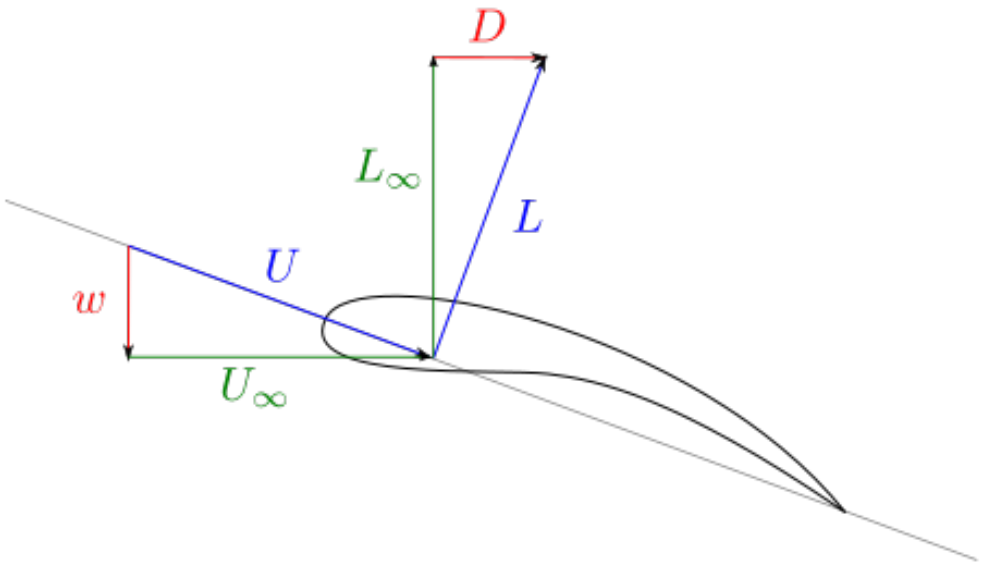


Fig. 3.4: Extra drag generated by induced wind. Figure 6 from [Gru21].

CHAPTER

FOUR

TECHNOLOGY

The main purpose of the thesis was to optimize the existing code for initial sail analysis. Orginally the code was implementing many panel objects. Each one of them had many atributes like panel cooridinates, force, pressure, pressure coefficient, span and normal vector. When the size of lattice was increasing, the huge number of objects inside pySailingVLM slowed down the program. To optimize code, the Numba - JIT compiler that translates Python code and NumPy into machine code which enable parallel coalcuations, was applied. Becouse Numba do not understand custom objects like panel, the code was rewritten using two memory layout approaches.

4.1 Memory layout

Array of Structures (AoS) and Structure of Arrays (SoA) are layouts arranging a sequence of records in memory. Structure of Arrays (SoA) layout splits elements of a record into separate units allowing access these elements in parallel. The AoS is the opposite layout in which data for different fields is interleaved. Comparison between conceptual layout and memory layout is shown on figure 4.1.

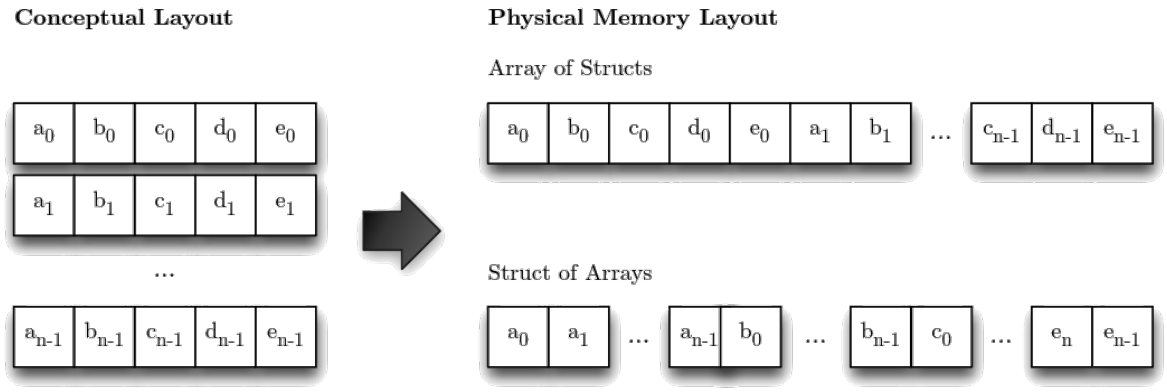


Fig. 4.1: Comparison of the Structure of Arrays (SoA) and Arrray of Structure (AoS). Figure 2 from [PHW+13]

The AoS approach ensures that the five values a_i, b_i, c_i, d_i, e_i are next to one another in memory, providing good cache utilisation. The SoA approach ensures that these values are split into five separate units, allowing access to corresponding elements in parallel [PHW+13].

Simple values like pressure coefficients, forces, pressure acting on each panel were arranged into simple NumPy arrays (SoA). Panel coorditnates, span and normal vectors was rearranged into nested ones (AoS). Figure 4.2 shows data layout implemented.

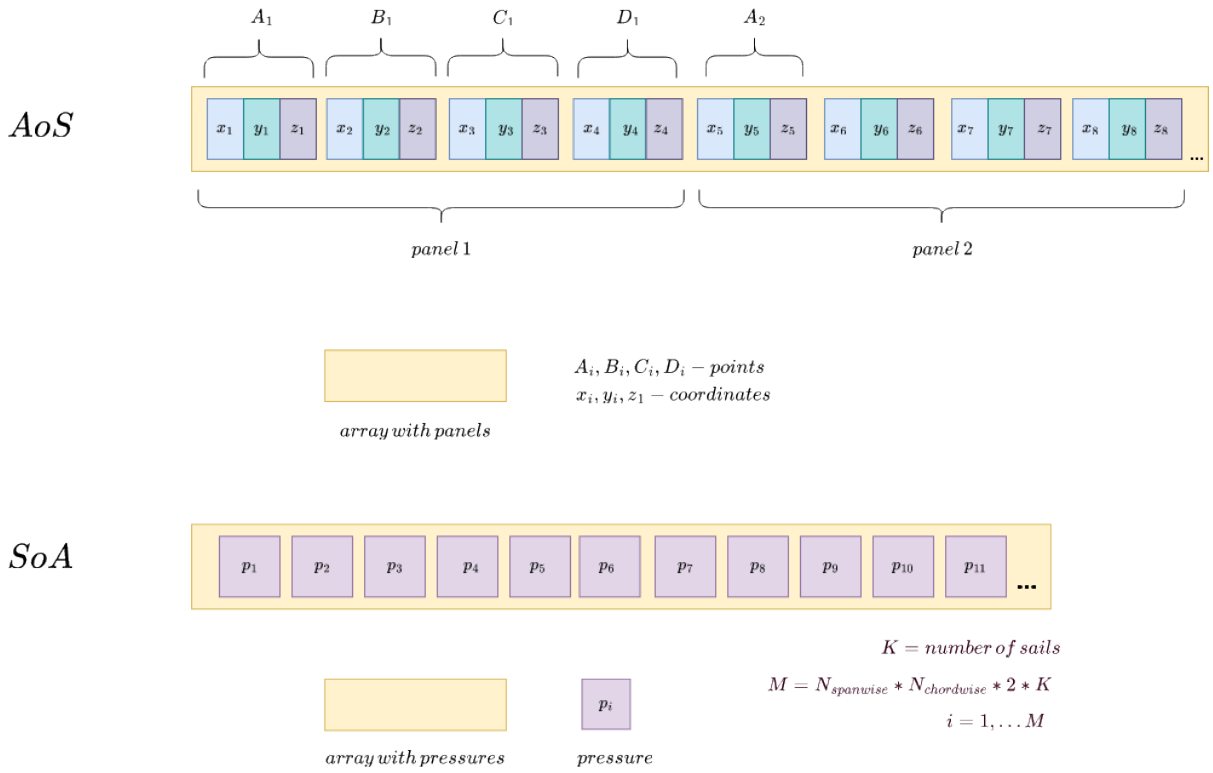


Fig. 4.2: Data layout implemented in pySailingVLM. Figure created by author.

4.2 Benchmarks

To benchmark pySalingVLM, the time comparison tests were conducted. Three approaches was summaries in the table 4.1: objective code, 'non-objective', objective with Numba and 'non-objective' compiled with Numba. The tests were carried out on a laptop with the following parameters: AMD Ryzen 7 4800H with 8 CPU cores (16 threads), base clock: 2.9 GHz, max boost: 4.2GHz, 32 GB RAM. For 1600 panels more then thirty times acceleration was obtained.

Table 4.1: Time execution comparison between different approaches of implementing pySailingVLM depending on sail shape.

No. panels	objective [s]	'non-objective' [s]	objective + Numba [s]	'non-objective' + Numba [s]
100	16.51	20.0	1.71	1.04
400	269.13	301.39	21.81	10.41
600	593.85	647.96	48.44	22.51
1600	4325.78	4458.05	320.54	143.69
2400	no data	no data	724.70	320.12
3600	no data	no data	1653.52	706.62

According to time results, using Numba is much more efficient when applied to the 'non-objective' code. In order to explain such behaviour the SIMD term should be investigated.

4.2.1 SIMD and vectorization

SIMD stands for Single Instruction Multiple Data. Instead of performing a single instruction on every single data, SIMD uses wider data-width for similar computational operations [AS20]. A comparison between simple scalar operation and a SIMD computation is depicted in figure 4.3.

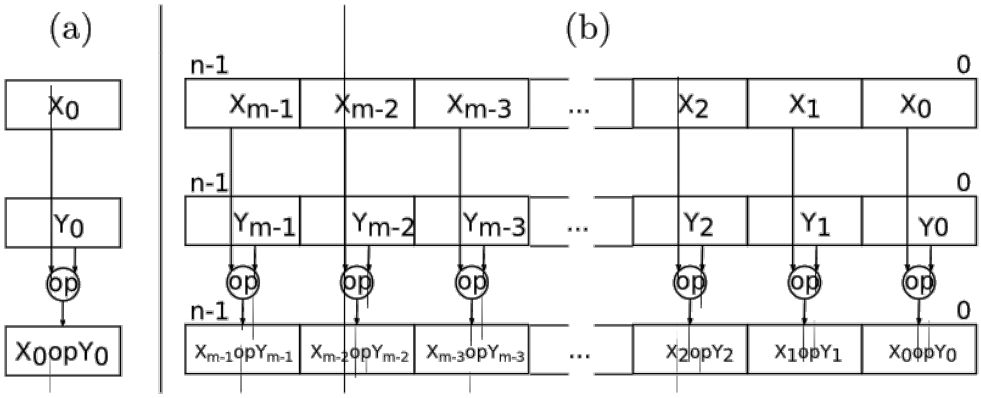


Fig. 4.3: Simple scalar operation (a) and a SIMD computation (b). Figure 1 from [AS20].

Vectorization with AOS in memory data layout requires multiple load/shuffle/insert or gather instructions. Because of the reduced CPU frequency in SIMD mode its improvements are not sufficient. Increase in vector width demands more instructions for vector construction. A properly aligned memory data layout for vectorization which Numba uses, needs Structure of Arrays (SOA). It provides SIMD compatible memory accesses and results in efficiency and speedup [WP16].

Array of Structure memory layout is more appropirate when no vectorization is used because proessing data are next to one another in memory. As a consequence, 'non-objective' code without Numba is slower than objective one (table 4.1).

4.3 Other improvements

Apart from code optimizations, the pySailingVLM has gained more features. The possibility to calculate cambered sailis and visualize pressure coefficients on colormap was introduced. Code has been packaged and now is available at Python Package Index (PyPI). It can be run locally from command line or in a cloud using Jupyter Notebook. pySailingVLM can be executed in a user defined script, which make it easy to calculate and compare many sailing cases.

Part II

Getting Started

CHAPTER

FIVE

INSTALLATION

5.1 For Users

pySailingVLM package is available at PyPI, to install it do:

```
pip install pySailingVLM
```

5.2 For Developers

If you would like to dive into pySailingVLM code and test it on your own please do the folowing steps:

- Clone git project

```
git clone #TODO wsadzic tutaj link
```

- Go into project

```
cd #TODO project
```

- There is no requirements.txt file, for installing dependencies install a project in editable mode (i.e. setuptools "develop mode") from a local project path:

```
pip install -q -e .
```

5.2.1 Create package

Building package requires setup.cfg and pyproject.toml files located in main tree of pySailingVLM project. If you do not have the latest pip build package do:

```
pip install --upgrade build
```

Then:

```
pip install build
```

Build and install

```
python3 -m build
```

```
pip install dist/pySailingVLM-VERSION.tar.gz
```

CHAPTER

SIX

USAGE

Note: First usage of pySailingVLm will produce numba warining. This behaviour is correct. Warining messages will disappear during second run of program.

Example warning:

/home/user/miniconda3/envs/sv_build_test_2/lib/python3.10/site-packages/numba/core/lowering.py:107: NumbaDebugInfoWarning: Could not find source for function: <function __numba_array_expr_0x7fbf333f1780 at 0x7fbf33361b40>. Debug line information may be inaccurate.

6.1 Jupyter Notebook

For Jupyter Notebook examples see Usage subsection.

6.2 Command line

6.2.1 Input file

In order to run pySailingVLM from command line you must provide a variable.py file. Example file is shown below. Modify it for your needs.

```
import os
import numpy as np
import time

mgirths =  np.array([0.00, 1./8, 1./4, 1./2, 3./4, 7./8, 1.00])
jgirths = np.array([0.00, 1./4, 1./2, 3./4, 1.00])

output_args = {
    'case_name': os.path.basename(__file__),  # get name of the current file
    'case_dir': os.path.abspath(''), # get dir of the current file
    'name': os.path.join("results_example_jib_and_mainsail_vlm", time.strftime("%Y-%m-
↪%d_%Hh%Mm%Ss")),
    'file_name': 'my_fancy_results', # name of xlsx excel file
}

solver_args = {
    'n_spanwise':  15,  # No of control points (above the water) per sail,␣
↪recommended: 50
```

(continues on next page)

(continued from previous page)

```
    'n_chordwise': 10, # No of control points (above the water) per sail,␣
↪recommended: 50
    'interpolation_type': "spline",  # either "spline" or "linear"
    'LLT_twist': "real_twist",  # defines how the Lifting Line discretize the sail␣
↪twist.
}

conditions_args = {
    'leeway_deg': 5.,     # [deg]
    'heel_deg': 10.,      # [deg]
    'SOG_yacht': 4.63,    # [m/s] yacht speed - speed over ground (leeway is a␣
↪separate variable)
    'tws_ref': 4.63,      # [m/s] true wind speed
    'alpha_true_wind_deg': 50.,   # [deg] true wind angle (with reference to course␣
↪over ground) => Course Wind Angle to the boat track = true wind angle to centerline␣
↪+ Leeway
    'reference_water_level_for_wind_profile': -0.,  # [m] this is an attempt to␣
↪mimick the deck effect
    # by lowering the sheer_above_waterline
    # while keeping the wind profile as in original geometry
    # this shall be negative (H = sail_ctrl_point - water_level)
    'wind_exp_coeff': 0.1428,  # [-] coefficient to determine the exponential wind␣
↪profile
    'wind_reference_measurment_height': 10.,  # [m] reference height for exponential␣
↪wind profile
    'rho': 1.225,  # air density [kg/m3]
    'wind_profile': 'exponential', # allowed: 'exponential' or 'flat' or 'logarithmic'
    'roughness': 0.05, # for logarithmic profile only
}

rig_args = {
    'main_sail_luff': 12.4,  # [m]
    'jib_luff': 10.0,  # [m]
    'foretriangle_height': 11.50,  # [m]
    'foretriangle_base': 3.90,  # [m]
    'sheer_above_waterline': 1.2,#[m]
    'boom_above_sheer': 1.3, # [m],
    'rake_deg': 92. ,  # rake angle [deg]
    'mast_LOA': 0.15,  # [m]
    'sails_def': 'jib_and_main', # definition of sail set, possible: 'jib' or 'main'␣
↪or 'jib_and_main'
}
# INFO for camber:
# First digit describing maximum camber as percentage of the chord.
# Second digit describing the distance of maximum camber from the airfoil leading␣
↪edge in tenths of the chord.
main_sail_args = {
    'girths' : mgirths,
    'chords': np.array([4.00, 3.82, 3.64, 3.20, 2.64, 2.32, 2.00]),
    'centerline_twist_deg': 12 * mgirths + 5,
    'camber': 5*np.array([0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01]),
    'camber_distance_from_luff': np.array([0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]),
}

jib_sail_args = {
    'centerline_twist_deg': 15. * jgirths + 7,
```

(continues on next page)

(continued from previous page)

```
    'girths': jgirths,
    'chords': np.array([3.80, 2.98, 2.15, 1.33, 0.5]),
    'camber': 5*np.array([0.01, 0.01, 0.01, 0.01, 0.01]),
    'camber_distance_from_luff': np.array([0.5, 0.5, 0.5, 0.5, 0.5]), # starting from␣
 ↪leading edge
}

# REFERENCE CSYS
# The origin of the default CSYS is located @ waterline level and aft face of the mast
# The positive x-coord: towards stern
# The positive y-coord: towards leeward side
# The positive z-coord: above the water
# To shift the default CSYS, adjust the 'reference_level_for_moments' variable.
# Shifted CSYS = original + reference_level_for_moments
# As a results the moments will be calculated around the new origin.

# yaw_reference [m] - distance from the aft of the mast towards stern, at which the␣
 ↪yawing moment is calculated.
# sway_reference [m] - distance from the aft of the mast towards leeward side. 0 for␣
 ↪symmetric yachts ;)
# heeling_reference [m] - distance from the water level,  at which the heeling moment␣
 ↪is calculated.
csys_args = {
    'reference_level_for_moments': np.array([0, 0, 0]),  # [yaw_reference, sway_
 ↪reference, heeling_reference]
}

# GEOMETRY OF THE KEEL
# to estimate heeling moment from keel, does not influence the optimizer.
# reminder: the z coord shall be negative (under the water)
keel_args={
    'center_of_lateral_resistance_upright': np.array([0, 0, -1.0]),  # [m] the␣
 ↪coordinates for a yacht standing in upright position
}
```

6.2.2 Run script

To run script with variables.py located in working directory do:

```
pySailingVLM
```

If variables.py is located in different directory, you must sepecify its location by providing additional option for pySailingVLM script:

```
pySailingVLM --dvars path_to_foler_with_variables.py
```

More information is available inside script help:

```
pySailingVLM --help
```

6.2.3 Output

pySailingVLM produces output files: data in xlsx file, matplotlib figures and pressure coefficient colormap in html extention (interactive plot). It is saved in the location specified in varaibles.py.

6.3 Cambered jib and main example

In cell below insert your initial parameters. If some of them are not required for your model, simply pass 0 value (for numbers). Some parameters are necessary only for specific cases like roughness (used by package when logarithmic profile is set) and they are omitted during computation.

More information can be found in code comments below.

```
# varaibles.py for jupyter
import os
import numpy as np
import time

mgirths =  np.array([0.00, 1./8, 1./4, 1./2, 3./4, 7./8, 1.00])
jgirths = np.array([0.00, 1./4, 1./2, 3./4, 1.00])

output_args = {
    'case_name': 'my_case_name',  # get name of the current file
    'case_dir': os.path.abspath(''), # get dir of the current file
    'name': os.path.join("results_example_jib_and_mainsail_vlm", time.strftime("%Y-%m-
 ↪%d_%Hh%Mm%Ss")),
    'file_name': 'my_fancy_results', # name of xlsx excel file
}

solver_args = {
    'n_spanwise':  10,  # No of control points (above the water) per sail,␣
 ↪recommended: 50
    'n_chordwise': 5, # No of control points (above the water) per sail, recommended:␣
 ↪50
    'interpolation_type': "spline",  # either "spline" or "linear"
    'LLT_twist': "real_twist",  # defines how the Lifting Line discretize the sail␣
 ↪twist.
}

conditions_args = {
    'leeway_deg': 5.,     # [deg]
    'heel_deg': 10.,      # [deg]
    'SOG_yacht': 4.63,    # [m/s] yacht speed - speed over ground (leeway is a␣
 ↪separate variable)
    'tws_ref': 4.63,      # [m/s] true wind speed
    'alpha_true_wind_deg': 50.,   # [deg] true wind angle (with reference to course␣
 ↪over ground) => Course Wind Angle to the boat track = true wind angle to centerline␣
 ↪+ Leeway
    'reference_water_level_for_wind_profile': -0.,  # [m] this is an attempt to␣
 ↪mimick the deck effect
    # by lowering the sheer_above_waterline
    # while keeping the wind profile as in original geometry
    # this shall be negative (H = sail_ctrl_point - water_level)
    'wind_exp_coeff': 0.1428,  # [-] coefficient to determine the exponential wind␣
 ↪profile
```

(continues on next page)

(continued from previous page)

```
    'wind_reference_measurment_height': 10.,  # [m] reference height for exponential␣
 ↪wind profile
    'rho': 1.225,  # air density [kg/m3]
    'wind_profile': 'exponential', # allowed: 'exponential' or 'flat' or 'logarithmic'
    'roughness': 0.05, # for logarithmic profile only
}

rig_args = {
    'main_sail_luff': 12.4,  # [m]
    'jib_luff': 10.0,  # [m]
    'foretriangle_height': 11.50,  # [m]
    'foretriangle_base': 3.90,  # [m]
    'sheer_above_waterline': 1.2,#[m]
    'boom_above_sheer': 1.3, # [m],
    'rake_deg': 92. ,  # rake angle [deg]
    'mast_LOA': 0.15,  # [m]
    'sails_def': 'jib_and_main', # definition of sail set, possible: 'jib' or 'main'␣
 ↪or 'jib_and_main'
}
# INFO for camber:
# First digit describing maximum camber as percentage of the chord.
# Second digit describing the distance of maximum camber from the airfoil leading␣
 ↪edge in tenths of the chord.
main_sail_args = {
    'girths' : mgirths,
    'chords': np.array([4.00, 3.82, 3.64, 3.20, 2.64, 2.32, 2.00]),
    'centerline_twist_deg': 12 * mgirths + 5,
    'camber': 5*np.array([0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01]),
    'camber_distance_from_luff': np.array([0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]),
}

jib_sail_args = {
    'centerline_twist_deg': 15. * jgirths + 7,
    'girths': jgirths,
    'chords': np.array([3.80, 2.98, 2.15, 1.33, 0.5]),
    'camber': 5*np.array([0.01, 0.01, 0.01, 0.01, 0.01]),
    'camber_distance_from_luff': np.array([0.5, 0.5, 0.5, 0.5, 0.5]), # starting from␣
 ↪leading edge
}

# REFERENCE CSYS
# The origin of the default CSYS is located @ waterline level and aft face of the mast
# The positive x-coord: towards stern
# The positive y-coord: towards leeward side
# The positive z-coord: above the water
# To shift the default CSYS, adjust the 'reference_level_for_moments' variable.
# Shifted CSYS = original + reference_level_for_moments
# As a results the moments will be calculated around the new origin.

# yaw_reference [m] - distance from the aft of the mast towards stern, at which the␣
 ↪yawing moment is calculated.
# sway_reference [m] - distance from the aft of the mast towards leeward side. 0 for␣
 ↪symmetric yachts ;)
# heeling_reference [m] - distance from the water level,  at which the heeling moment␣
 ↪is calculated.
csys_args = {
```

(continues on next page)

(continued from previous page)

```
    'reference_level_for_moments': np.array([0, 0, 0]),  # [yaw_reference, sway_
 ↪reference, heeling_reference]
}

# GEOMETRY OF THE KEEL
# to estimate heeling moment from keel, does not influence the optimizer.
# reminder: the z coord shall be negative (under the water)
keel_args={
    'center_of_lateral_resistance_upright': np.array([0, 0, -1.0]),  # [m] the␣
 ↪coordinates for a yacht standing in upright position
}
```

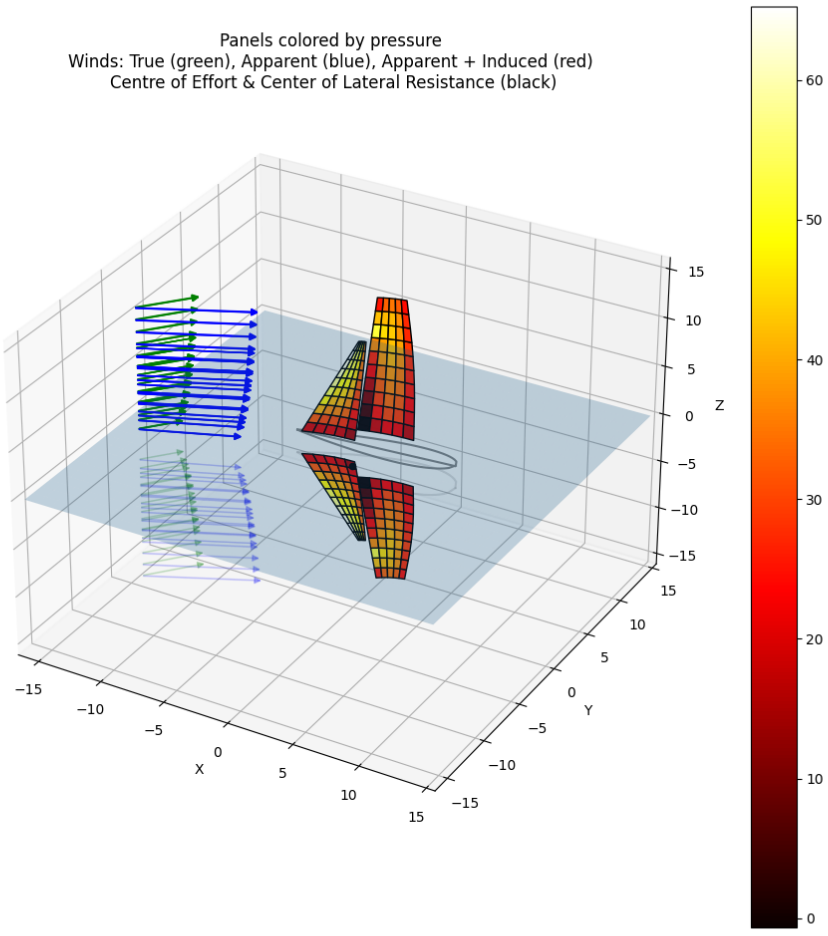
```
import shutil
from pySailingVLM.rotations.csys_transformations import CSYS_transformations
from pySailingVLM.yacht_geometry.hull_geometry import HullGeometry
from pySailingVLM.results.save_utils import save_results_to_file
from pySailingVLM.solver.panels_plotter import display_panels_xyz_and_winds
from pySailingVLM.results.inviscid_flow import InviscidFlowResults
from pySailingVLM.solver.vlm import Vlm
from pySailingVLM.runner.sail import Wind, Sail
from pySailingVLM.runner.container import Output, Rig, Conditions, Solver, MainSail,␣
 ↪JibSail, Csys, Keel
from pySailingVLM.solver.panels_plotter import plot_cp
```

```
out = Output(**output_args)
conditions = Conditions(**conditions_args)
solver = Solver(**solver_args)
main = MainSail(**main_sail_args)
jib = JibSail(**jib_sail_args) # j2 = JibSail(centerline_twist_deg= 15. * jgirths + 7,
 ↪ girths= jgirths) set centerline_twist_deg and girths, rest is default

csys = Csys(**csys_args)
keel = Keel(**keel_args)
rig = Rig(**rig_args)

csys_transformations = CSYS_transformations(
    conditions.heel_deg, conditions.leeway_deg,
    v_from_original_xyz_2_reference_csys_xyz=csys.reference_level_for_moments)


w = Wind(conditions)
s = Sail(solver, rig, main, jib, csys_transformations)
sail_set = s.sail_set
hull = HullGeometry(rig.sheer_above_waterline, rig.foretriangle_base, csys_
 ↪transformations, keel.center_of_lateral_resistance_upright)
myvlm = Vlm(sail_set.panels, solver.n_chordwise, solver.n_spanwise, conditions.rho, w.
 ↪profile, sail_set.trailing_edge_info, sail_set.leading_edge_info)

inviscid_flow_results = InviscidFlowResults(sail_set, csys_transformations, myvlm)
inviscid_flow_results.estimate_heeling_moment_from_keel(hull.center_of_lateral_
 ↪resistance)
```

Cell below displays computations and saves integrals to output file.

```
%matplotlib widget
print("Preparing visualization.")
display_panels_xyz_and_winds(myvlm, inviscid_flow_results, myvlm.inlet_conditions,␣
↪hull, show_plot=True) # add show_apparent_induced_wind=True for apparent + induced␣
↪wind
df_components, df_integrals, df_inlet_IC = save_results_to_file(myvlm, csys_
↪transformations, inviscid_flow_results, s.sail_set, out.name, out.file_name)
```

```
Preparing visualization.
```



Lets see our integrals :)

```
print(f"-----------------------------------------------------------------")
print(f"Notice:\n"
```

(continues on next page)

(continued from previous page)

```
  f"\tThe forces [N] and moments [Nm] are without profile drag.\n"
  f"\tThe the _COG_ CSYS is aligned in the direction of the yacht movement (course␣
↪over ground).\n"
  f"\tThe the _COW_ CSYS is aligned along the centerline of the yacht (course over␣
↪water).\n"
  f"\tNumber of panels (sail s.sail_set with mirror): {s.sail_set.panels.shape}")

df_integrals
```

```
------------------------------------------------------------
Notice:
        The forces [N] and moments [Nm] are without profile drag.
        The the _COG_ CSYS is aligned in the direction of the yacht movement␣
↪(course over ground).
        The the _COW_ CSYS is aligned along the centerline of the yacht (course␣
↪over water).
        Number of panels (sail s.sail_set with mirror): (200, 4, 3)
```

	Quantity	Value
0	F_jib_total_COG.x	-279.764768
1	F_jib_total_COG.y	678.500639
2	F_jib_total_COG.z	-50.156361
3	F_main_sail_total_COG.x	-354.636548
4	F_main_sail_total_COG.y	1057.983557
5	F_main_sail_total_COG.z	-202.903407
6	F_sails_total_COG.x	-634.401316
7	F_sails_total_COG.y	1736.484196
8	F_sails_total_COG.z	-253.059767
9	F_sails_total_COW.x	-783.331797
10	F_sails_total_COW.y	1674.584632
11	F_sails_total_COW.z	-253.059767
12	M_jib_total_COG.x	-3579.994204
13	M_jib_total_COG.y	-1661.671677
14	M_jib_total_COG.z	-798.223020
15	M_keel_total_COG.x	-1753.879249
16	M_keel_total_COG.y	-628.593251
17	M_keel_total_COG.z	83.462722
18	M_keel_total_COW.x (heel)	-1692.419697
19	M_keel_total_COW.y (pitch)	-779.061913
20	M_keel_total_COW.z (yaw - JG sign)	-83.462722
21	M_keel_total_COW.z (yaw)	83.462722
22	M_main_sail_total_COG.x	-9443.296080
23	M_main_sail_total_COG.y	-2994.708741
24	M_main_sail_total_COG.z	2404.207819
25	M_sails_total_COG.x	-13023.290285
26	M_sails_total_COG.y	-4656.380418
27	M_sails_total_COG.z	1605.984799
28	M_sails_total_COW.x (heel)	-12567.902440
29	M_sails_total_COW.y (pitch)	-5773.716022
30	M_sails_total_COW.z (yaw - JG sign)	-1605.984799
31	M_sails_total_COW.z (yaw)	1605.984799
32	M_total_COG.x	-14777.169534
33	M_total_COG.y	-5284.973669
34	M_total_COG.z	1689.447521
35	M_total_COW.x (heel)	-14260.322137

(continues on next page)

(continued from previous page)

```
36                M_total_COW.y (pitch)  -6552.777935
37        M_total_COW.z (yaw - JG sign)  -1689.447521
38                  M_total_COW.z (yaw)   1689.447521
```

Compute aerodynamic parameters:

```
sails_Cxyz = myvlm.get_Cxyz(w, 1.0)
print(f"Cxyz for {rig.sails_def}")
for idx, c in enumerate(sails_Cxyz):
    print(f"C[{idx}]: {c}")
```

```
Cxyz for jib_and_main
C[0]: [-2.02304539  4.90639904 -0.36269254]
C[1]: [-1.33774067  3.99086794 -0.76538118]
```

Make model plot in 2D colored by pressure coefficients:

```
from pySailingVLM.solver.panels_plotter import plot_cp
plot_cp(sail_set.zero_mesh, myvlm.p_coeffs, out.name)
```

Thats all. Experiment and play with this code on your own.

6.4 Sweep cambered main sail

```
# varaibles.py for jupyter
import os
import numpy as np
import time

half_wing_span = 8
sweep_angle_deg = 5.
chord_length = 4
AoA_deg = 8.
mgirths =  np.array([0.00, 1./8, 1./4, 1./2, 3./4, 7./8, 1.00])
mchords = np.array([chord_length]* len(mgirths))


output_args = {
    'case_name': 'my_case_name',  # get name of the current file
    'case_dir': os.path.abspath(''), # get dir of the current file
    'name': os.path.join("results_example_jib_and_mainsail_vlm", time.strftime("%Y-%m-
↪%d_%Hh%Mm%Ss")),
    'file_name': 'my_fancy_results', # name of xlsx excel file
}

solver_args = {
    'n_spanwise':  5,  # No of control points (above the water) per sail,␣
↪recommended: 50
    'n_chordwise': 5, # No of control points (above the water) per sail, recommended:␣
↪50
```

(continues on next page)

(continued from previous page)

```
    'interpolation_type': "linear",  # either "spline" or "linear"
    'LLT_twist': "real_twist",  # defines how the Lifting Line discretize the sail␣
 ↪twist.
}

conditions_args = {
    'leeway_deg': 0.,     # [deg]
    'heel_deg': 0.,      # [deg]
    'SOG_yacht': 1.,    # [m/s] yacht speed - speed over ground (leeway is a separate␣
 ↪variable)
    'tws_ref': 1.0,      # [m/s] true wind speed
    'alpha_true_wind_deg': AoA_deg,    # [deg] true wind angle (with reference to␣
 ↪course over ground) => Course Wind Angle to the boat track = true wind angle to␣
 ↪centerline + Leeway
    'reference_water_level_for_wind_profile': -0.,  # [m] this is an attempt to␣
 ↪mimick the deck effect
    # by lowering the sheer_above_waterline
    # while keeping the wind profile as in original geometry
    # this shall be negative (H = sail_ctrl_point - water_level)
    'wind_exp_coeff': 0.,  # [-] coefficient to determine the exponential wind profile
    'wind_reference_measurment_height': 10.,  # [m] reference height for exponential␣
 ↪wind profile
    'rho': 1.,  # air density [kg/m3]
    'wind_profile': 'flat', # allowed: 'exponential' or 'flat' or 'logarithmic'
    'roughness': 0.05, # for logarithmic profile only
}


rig_args = {
    'main_sail_luff': half_wing_span / np.cos(np.deg2rad(sweep_angle_deg)),  # [m]
    'jib_luff': 10.0,  # [m]
    'foretriangle_height': 11.50,  # [m]
    'foretriangle_base': 3.90,  # [m]
    'sheer_above_waterline': 1.2,#[m]
    'boom_above_sheer': 1.3, # [m],
    'rake_deg': 90. + sweep_angle_deg,  # rake angle [deg]
    'mast_LOA': 0.,  # [m]
    'sails_def': 'main', # definition of sail set, possible: 'jib' or 'main' or 'jib_
 ↪and_main'
}


# INFO for camber:
# First digit describing maximum camber as percentage of the chord.
# Second digit describing the distance of maximum camber from the airfoil leading␣
 ↪edge in tenths of the chord.
main_sail_args = {
    'girths' : mgirths,
    'chords': mchords,
    'centerline_twist_deg': 0*mgirths,
    'camber': 10*np.array([0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01]),
    'camber_distance_from_luff': np.array([0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]),
}


jgirths = np.array([0.00, 1./4, 1./2, 3./4, 1.00])
```

(continues on next page)

(continued from previous page)

```
jib_sail_args = {
    'centerline_twist_deg': 0*(10+5)  + 0*15. * jgirths,
    'girths': jgirths,
    'chords': 0* np.array([3.80, 2.98, 2.15, 1.33, 0.5]),
    'camber': 0*np.array([0.01, 0.01, 0.01, 0.01, 0.01]),
    'camber_distance_from_luff': np.array([0.5, 0.5, 0.5, 0.5, 0.5]), # starting from␣
 ↪leading edge
}

# REFERENCE CSYS
# The origin of the default CSYS is located @ waterline level and aft face of the mast
# The positive x-coord: towards stern
# The positive y-coord: towards leeward side
# The positive z-coord: above the water
# To shift the default CSYS, adjust the 'reference_level_for_moments' variable.
# Shifted CSYS = original + reference_level_for_moments
# As a results the moments will be calculated around the new origin.

# yaw_reference [m] - distance from the aft of the mast towards stern, at which the␣
 ↪yawing moment is calculated.
# sway_reference [m] - distance from the aft of the mast towards leeward side. 0 for␣
 ↪symmetric yachts ;)
# heeling_reference [m] - distance from the water level,  at which the heeling moment␣
 ↪is calculated.
csys_args = {
    'reference_level_for_moments': np.array([0, 0, 0]),  # [yaw_reference, sway_
 ↪reference, heeling_reference]
}

# GEOMETRY OF THE KEEL
# to estimate heeling moment from keel, does not influence the optimizer.
# reminder: the z coord shall be negative (under the water)
keel_args={
    'center_of_lateral_resistance_upright': np.array([0, 0, -1.0]),  # [m] the␣
 ↪coordinates for a yacht standing in upright position
}
```

```
import shutil
from pySailingVLM.rotations.csys_transformations import CSYS_transformations
from pySailingVLM.yacht_geometry.hull_geometry import HullGeometry
from pySailingVLM.results.save_utils import save_results_to_file
from pySailingVLM.solver.panels_plotter import display_panels_xyz_and_winds
from pySailingVLM.results.inviscid_flow import InviscidFlowResults
from pySailingVLM.solver.vlm import Vlm
from pySailingVLM.runner.sail import Wind, Sail
from pySailingVLM.runner.container import Output, Rig, Conditions, Solver, MainSail,␣
 ↪JibSail, Csys, Keel

from pySailingVLM.solver.panels_plotter import plot_cp
```

```
import numpy as np
from pySailingVLM.solver.coefs import get_vlm_Cxyz

C_results = []
a_vlm_results = []
```

(continues on next page)

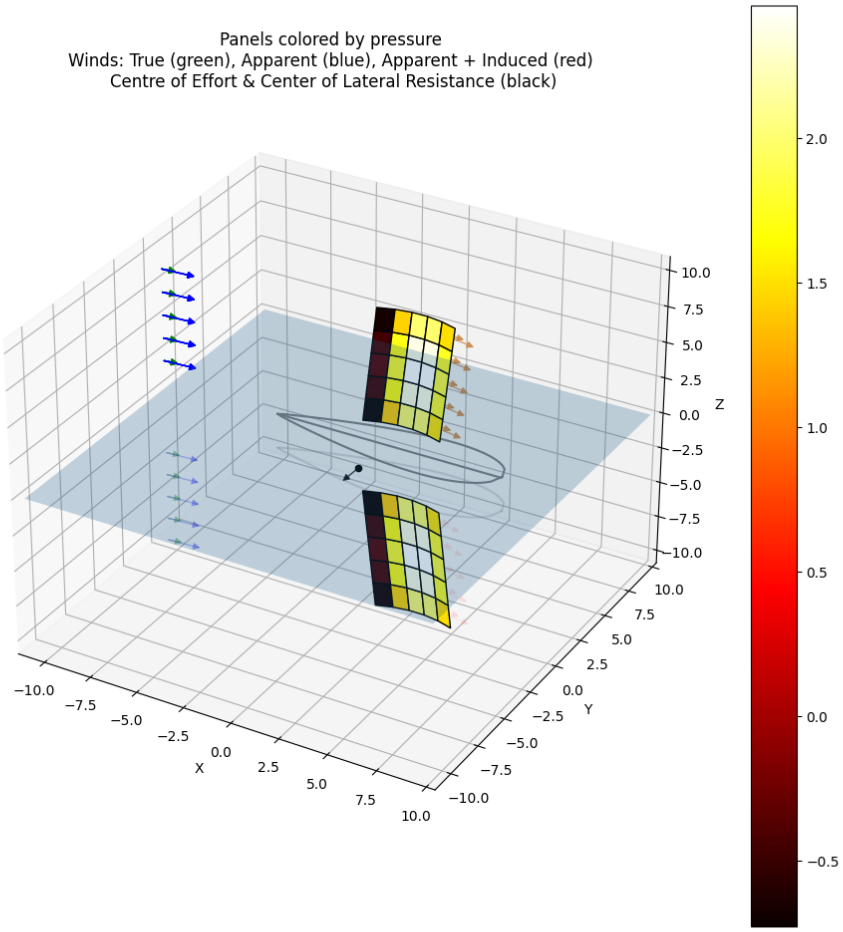
(continued from previous page)

```
out = Output(**output_args)
conditions = Conditions(**conditions_args)
solver = Solver(**solver_args)
main = MainSail(**main_sail_args)
jib = JibSail(**jib_sail_args)
csys = Csys(**csys_args)
keel = Keel(**keel_args)
rig = Rig(**rig_args)
csys_transformations = CSYS_transformations(conditions.heel_deg, conditions.leeway_
↪deg, v_from_original_xyz_2_reference_csys_xyz=csys.reference_level_for_moments)


w = Wind(conditions)
s = Sail(solver, rig, main, jib, csys_transformations)
sail_set = s.sail_set
myvlm = Vlm(sail_set.panels, solver.n_chordwise, solver.n_spanwise, conditions.rho, w.
↪profile, sail_set.trailing_edge_info, sail_set.leading_edge_info)

height = 1.0
sails_Cxyz = myvlm.get_Cxyz(w, height)

# enumerate through sails
# in this example we have only main
print(f"Cxyz for {rig.sails_def}")
for idx, c in enumerate(sails_Cxyz):
    print(f"C[{idx}]: {c}")

hull = HullGeometry(rig.sheer_above_waterline, rig.foretriangle_base, csys_
↪transformations, keel.center_of_lateral_resistance_upright)
inviscid_flow_results = InviscidFlowResults(sail_set, csys_transformations, myvlm)
inviscid_flow_results.estimate_heeling_moment_from_keel(hull.center_of_lateral_
↪resistance)
```

```
Cxyz for main
C[0]: [ 0.06028863  2.83686792 -0.00527457]
```

```
%matplotlib widget
print("Preparing visualization.")
display_panels_xyz_and_winds(myvlm, inviscid_flow_results, myvlm.inlet_conditions,␣
↪hull, show_plot=True, show_apparent_induced_wind=True) # add show_apparent_induced_
↪wind=True for apparent + induced wind
df_components, df_integrals, df_inlet_IC = save_results_to_file(myvlm, csys_
↪transformations, inviscid_flow_results, s.sail_set, out.name, out.file_name)
```

```
Preparing visualization.
```




```
print(f"------------------------------------------------------------------")
print(f"Notice:\n"
  f"\tThe forces [N] and moments [Nm] are without profile drag.\n"
  f"\tThe the _COG_ CSYS is aligned in the direction of the yacht movement (course␣
↪over ground).\n"
  f"\tThe the _COW_ CSYS is aligned along the centerline of the yacht (course over␣
↪water).\n"
  f"\tNumber of panels (sail s.sail_set with mirror): {s.sail_set.panels.shape}")

df_integrals
```

```
------------------------------------------------------------
Notice:
        The forces [N] and moments [Nm] are without profile drag.
        The the _COG_ CSYS is aligned in the direction of the yacht movement␣
↪(course over ground).
```

(continues on next page)

(continued from previous page)

```
        The the _COW_ CSYS is aligned along the centerline of the yacht (course␣
↪over water).
        Number of panels (sail s.sail_set with mirror): (50, 4, 3)
```

	Quantity	Value
0	F_main_sail_total_COG.x	0.988981
1	F_main_sail_total_COG.y	46.536280
2	F_main_sail_total_COG.z	-0.086525
3	F_sails_total_COG.x	0.988981
4	F_sails_total_COG.y	46.536280
5	F_sails_total_COG.z	-0.086525
6	F_sails_total_COW.x	0.988981
7	F_sails_total_COW.y	46.536280
8	F_sails_total_COW.z	-0.086525
9	M_keel_total_COG.x	-46.536280
10	M_keel_total_COG.y	0.988981
11	M_keel_total_COG.z	0.000000
12	M_keel_total_COW.x (heel)	-46.536280
13	M_keel_total_COW.y (pitch)	0.988981
14	M_keel_total_COW.z (yaw - JG sign)	-0.000000
15	M_keel_total_COW.z (yaw)	0.000000
16	M_main_sail_total_COG.x	-302.514742
17	M_main_sail_total_COG.y	6.902970
18	M_main_sail_total_COG.z	137.446498
19	M_sails_total_COG.x	-302.514742
20	M_sails_total_COG.y	6.902970
21	M_sails_total_COG.z	137.446498
22	M_sails_total_COW.x (heel)	-302.514742
23	M_sails_total_COW.y (pitch)	6.902970
24	M_sails_total_COW.z (yaw - JG sign)	-137.446498
25	M_sails_total_COW.z (yaw)	137.446498
26	M_total_COG.x	-349.051021
27	M_total_COG.y	7.891951
28	M_total_COG.z	137.446498
29	M_total_COW.x (heel)	-349.051021
30	M_total_COW.y (pitch)	7.891951
31	M_total_COW.z (yaw - JG sign)	-137.446498
32	M_total_COW.z (yaw)	137.446498

```
plot_cp(sail_set.zero_mesh, myvlm.p_coeffs, out.name)
```

CHAPTER

SEVEN

VALIDATION

To validate result obtained from pySailingVLM two different wings were tested: swept Bertin wing and rectangular plat one.

7.1 Rectangular flat plate

Using pySailingVLm a rectangular flat plate was modeled with a span of 10 units and a constant chord of 1 unit. The geomery of sail is discretized into 16 spanwise and 8 chordwise panels. The free-stream strength is 1 unit and the angle of attack is 10°. Then the lift coefficient slope $C_{L,\alpha}$ was compared with [AH13] and analitically derived results in the table below.

Table 7.1: Results from pySailingVLM code of the rectangular flat plate of a slope

	Analitical	[AH13]	pySailingVLM
$C_{L,\alpha}$	4.896	4.786	4.846

?????????????????????? dac cp plot z magisterki??????????????????????????????

7.1.1 Results

The results obtained by pySailingVLM are close to analitical and also [AH13] data. It should be noticed that analytically derived results assumes a span efficiency factor of 0.8.

?????????????????????????? DAC wzor gdzies??????????????????

7.2 Sweep wing

Swept wing by Bertin [JJB09], is a flat plate with a 45° leading edge sweeping angle. The span of wing is 1 unit and a chord of 0.2 unit. Model was discretized into 4 spanwise panels and 1 chordwise panel. The free stream is aligned with the x-axis and has a magnitude of 1 units.

Table 7.2: Comparison between swept wing by Bertin and pySailingVLM results.

	Bertin	pySailingVLM
$C_{L,\alpha}$	3.443	3.434

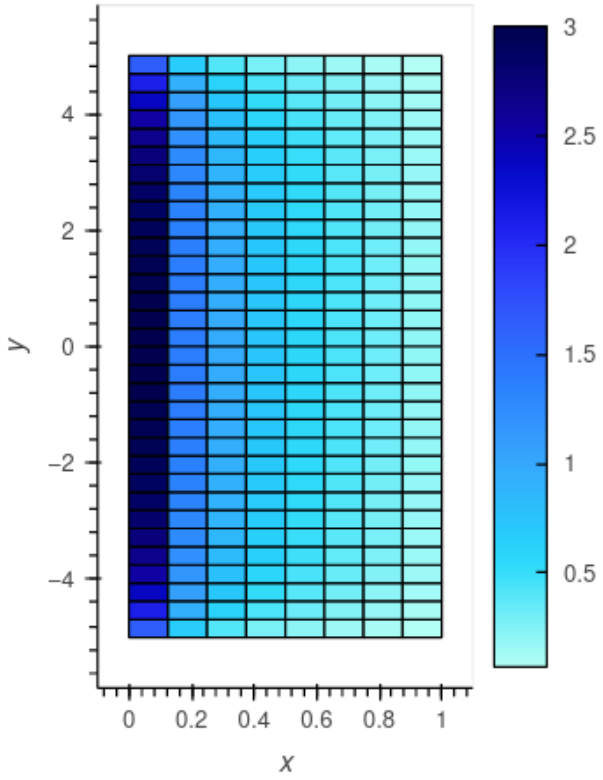


Fig. 7.1: Pressure coefficients for flat plate. Figure generated by pySailingVLM.

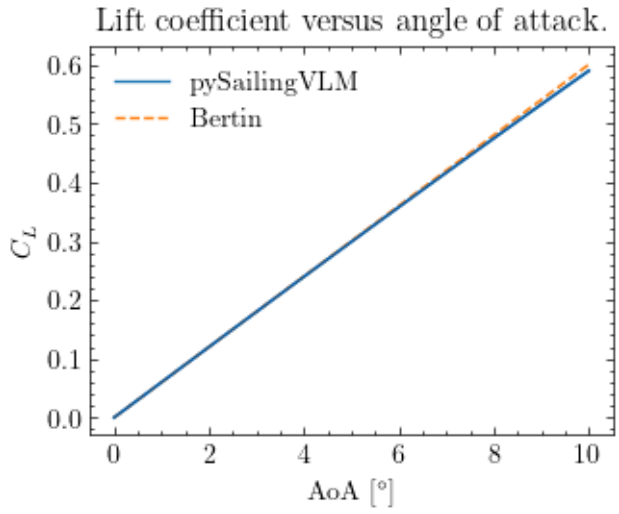


Fig. 7.2: Lift coefficient versus angle of attack. Figure generated by pySailingVLM.

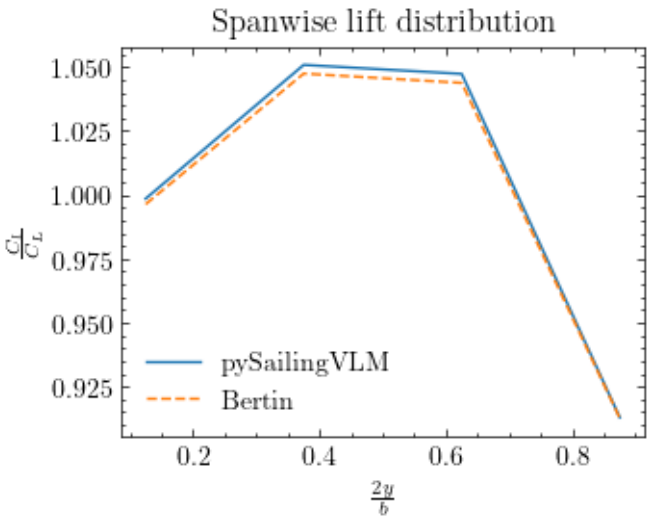


Fig. 7.3: Comparison of spanwise lift distributionfor the Bertin wing. Figure generated by pySailingVLM.

?????????????????????? TODO Dodatkowy wykres za Katzem: https://www.researchgate.net/publication/245355607_Calculation_of_tl
Fig4

7.3 Results

Results obtained by pySailingVLM are close to teoretical results taken from [JJB09]. When the angle of attack increases, the lift coefficient calculated by pySailingVLM differ from Bertin results (figure 7.2). It can be related with a small-angle approximation used in the Vortex Lattice Method, where the angle of attack and the angle of sideslip should be both small. ??

CHAPTER

EIGHT

FUTURE OUTLOOKS

Some improvements to pySailingVLM code can be conducted in order to decrease program time execution.

8.1 GPU

Apart form CPU programing, Numba supports CUDA GPU programming by directly compiling a restricted subset of Python code into CUDA kernels and device functions using the CUDA execution model. Kernels which are written in Numba appear to have direct access to NumPy arrays according to documantation [num]. The NumPy arrays are automatically shifted between the CPU and the GPU. To increase code performance, calcuations can be moved to GPU.

8.2 All code as SoA

Up to now, most of Python objects are represented as Structure of Arrays, except panels. To take full adventage of GPU speed up, panels can be rewriten into SoA. This memory layout is recommended for CUDA acceleration [WP16] due to parallel acces to data units.

8.3 Points dupllicate

Panels which are close to each other share same points (see figure 8.1). Current pySailingVLM implementation do not take into account this fact and there are duplicates of points coordinates in panel array. Reducing redundant points can improve the code efficiency.

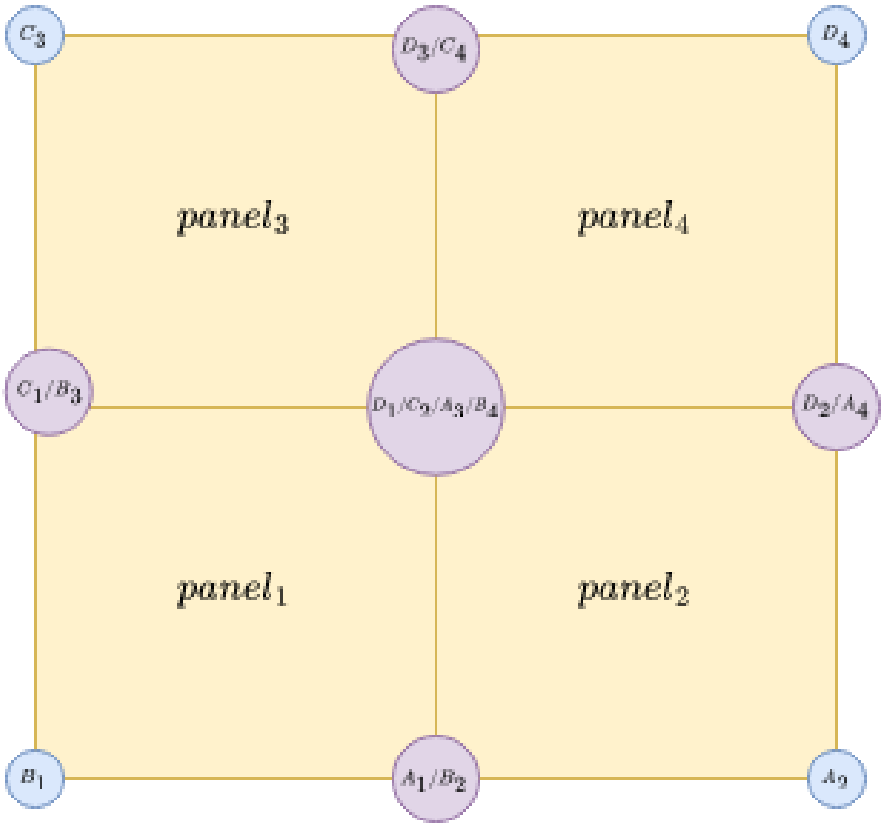


Fig. 8.1: Points on panels arragement. Figure created by author.

CHAPTER

NINE

CONCLUSIONS

pySailingVLM package created for initial aerodynamic analysis of upwind sails can be useful in early conceptual phase, when many different desings should be tested. Numba with 'non-objective' pySailingVLM code gives satisfying acceleration results in comparison to objective code. Data generated by this package is close to theory and other tools [AH13]. It has light weight requirements and can be run locally or in a cloud environment such as Google Collab. Apart from this, users can define own sail geometries and visualize obtained results.

CHAPTER

TEN

BIBLIOGRAPHY

BIBLIOGRAPHY

[num] Numba documentation [access: 08.06.2023]. URL: https://numba.pydata.org/numba-doc/latest/index.html.

[AH13] T.Larsson A. Helmstad. *An Aeroelastic Implementation for Yacht Sails and Rigs, Degree project in Naval Architecture*. KTH Engeneering Sciences, 2013.

[AS20] H. Amiri and A. Shahbahrami. Simd programming using intel vector extensions. *Journal of Parallel and Distributed Computing*, pages 83–100, 2020. URL: https://www.sciencedirect.com/science/article/pii/S074373151830813X, doi:10.1016/j.jpdc.2019.09.012.

[Gru21] G. Gruszczyński. Constrained optimization of an upwind sail set using fast solver. In *Lecture: 2.016 Hydrodynamics*. Interdisciplinary Center for Mathematical and Computational Modelling, University of Warsaw, 2021.

[JK01] A. Plotkin J. Katz. *Low-Speed Aerodynamics, Second Edition*. Cambridge University Press, 2001.

[JJB09] Russell M. Cummings John J. Bertin. *Aerodynamics for engineers, Fifth Edition*. Pearson Education International, 2009.

[NJS+21] Sheharyar Nasir, M Tariq Javaid, M. Usman Shahid, Asad Raza, Waseeq Siddiqui, and Shuaib Salamat. Applicability of vortex lattice method and its comparison with high fidelity tools. *Pakistan Journal of Engineering and Technology, PakJET*, 04(01):207– 211, 2021. URL: https://www.hpej.net/journals/pakjet/article/view/754/564, doi:DOI: 10.51846/vol4iss1pp207-211.

[NS18] Bill Gladstone North Sails. Twist explained. 2018. URL: https://www.northsails.com/sailing/en/2018/10/north-u-understanding-twist-by-bill-gladstone (visited on 2021-11-07).

[PHW+13] S.J. Pennycooka, S.D. Hammondb, S.A. Wright, J.A. Herdmanc, I. Millerc, and S.A. Jarvis. An investigation of the performance portability of opencl. *Journal of Parallel and Distributed Computing*, 73:439–1450, November 2013. URL: https://www.sciencedirect.com/science/article/abs/pii/S0743731512001669, doi:0.1016/j.jpdc.2012.07.005.

[PS00] W. F. Phillips and D. O. Snyder. Modern adaptation of prandtl's classic lifting-line theory. *JOURNAL OF AIRCRAFT*, 37(4):662–670, July - August 2000. URL: http://arc.aiaa.org, doi:DOI: 10.2514/2.2649.

[SHR+19] A. Septiyana, K. Hidayat, A. Rizaldi, M. L. Ramadiansyah, R. A. Ramadhan, P. A. P. Suseno, E. B. Jayanti, N. Atmasari, and A. Rasyadi. Analysis of aerodynamic characteristics using the vortex lattice method on twin tail boom unmanned aircraft. *7TH INTERNATIONAL SEMINAR ON AEROSPACE SCIENCE AND TECHNOLOGY - ISAST 2019*, 2226(April):536, 24-25 September 2019. URL: https://pubs.aip.org/aip/acp/article/2226/1/020003/814546/Analysis-of-aerodynamic-characteristics-using-the, doi:10.1063/5.0002337.

[Tec05] Prof. A.H. Techet. Potential flow theory. In *Lecture: 2.016 Hydrodynamics*. Massachusetts Institute of Technology, 2005. MIT. URL: https://web.mit.edu/2.016/www/handouts/2005Reading4.pdf.

[WP16] A. Wells and A. M. Prabha. Improve vectorization efficiency using intel simd data layout template (intel sdlt) [access: 08.06.2023]. In *INTEL HPC DEVELOPER CONFERENCE FUEL YOUR INSIGHT*.

Novenber 2016. URL: https://www.intel.com/content/dam/www/public/us/en/documents/presentation/improving-vectorization-efficiency.pdf.