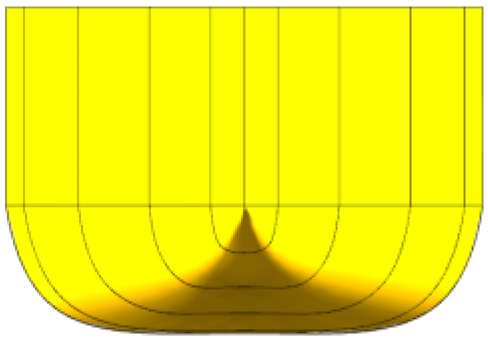


# The YDEOS Project

*Yacht Design and Engineering Open Source*

*v. 2020*

# Project info

October 15th, 2020

# Contents

Conventions:

Some `code` or `symbol` in a sentence

**library name**

```
code
```

| Column header #1 | Column header #2 | Column header #3 | Column header #4 |
|---|---|---|---|
| Table content | | | |
| | | | |

http://www.anylink.com

# 1. Introduction

## 1.1. Vision

The YDEOS project is a collection of software libraries and executables that help with yacht and ship design and engineering.

## 1.2. Scope

The following topics are intended to be covered:

- units conversions (**ydeos_units**)
- forces modeling (**ydeos_forces**)
- hydrodynamics and resistance estimates using textbook and publication formulae and regressions (**ydeos_hydrodynamics**)
- aerodynamic sails and windage models (**ydeos_aerodynamics**)
- upright and heeled/trimmed hydrostatics
- 2D foil sections generation
- sailing yacht velocity prediction
- sailing yacht polars handling and utilities
- offsets from CAD
- 2D structured meshing
- ship and yacht related CFD cases generators
- automated sail shape analysis

## 1.3. Governance

The governance model for 2020 / 2021 is a BDFL model (https://en.wikipedia.org/wiki/Benevolent_dictator_for_life)

# 2. Architecture

## 2.1. Technology

The initial developments of the YDEOS project will use Python 3 and its scientific stack (Numpy, Scipy, etc …). The scientific ecosystem + ease of development combination of Python is unrivaled; yet, some parts of the project may later have to be re-written using other technologies for performance reasons.

The targeted Python 3 versions are >= 3.7.

## 2.2. Decisions

### 2.2.1. Functional Core, Imperative Shell

Functional Core, Imperative Shell: development will follow this philosophy with domains modeled as functionally as possible.

The SRP (Single Responsability Principle) should be respected by functions. If you have to use ‘then’ or ‘and’ to describe what a function does, it may need refactoring.

Shells may be developed in imperative style, with an optional Service Layer if orchestration logic starts to creep into domain logic or into the shell developments.

Potential shells are:

- CLI (Command Line Interface)
- GUI (Graphical User Interface)
- Web entry points
- CAD Workbench

### 2.2.2. Pipes and filters

Keep in mind that, to make the YDEOS project globally more useful, the output of executables should be designed as the input of others. Create common formats between projects whenever it makes sense.

### 2.2.3. Side effects

Side effects coding should be separated into ‘what’ and ‘how’ parts.

### 2.2.4. Dependencies

Each project should be as self-contained as possible. Whenever possible, there should be no dependency on any other internal project.

Depending on the Python scientific stack is OK. Depending on PythonOCC for geometry stuff is OK.

### 2.2.5. Parallelism

Any function taking typically more than 10 seconds to complete should have the option to be parallelized.

**ydeos_parallel** may be used or improved if necessary to capture parallelization concerns.

### 2.2.6. Feedback

Any function taking more than 2 seconds to complete should report its progress. In a CLI shell, a library like **tqdm** should be used to give the user a sense of progress. A progress bar should be used in a GUI.

### 2.2.7. Graphical User Interfaces

TkInter, wxPython are recommended. PyQt is okay.

## 2.3. Integration & Delivery

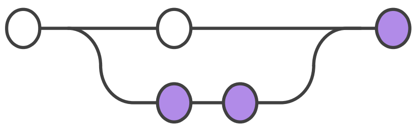
### 2.3.1. Tests

Use **pytest** and **hypothesis**. **hypoyhesis** is a Python implementation of Haskell's **QuickCheck** and helps with testing edge cases properly.

The absolute lower limit of tests coverage is 80 % at project/repo level.

### 2.3.2. Version control

Use the Git Feature Branch Workflow



The master branch should never contain broken code.

### 2.3.3. CI / CD

CI should be handled at each internal project level and at the YDEOS project level (end-to-end scenarii). The online CI tool is **Travis CI**.

Test coverage is handled with **coverage** locally and coveralls.io (Travis CI sending coverage results to coveralls.io) online.

Code quality is handled with **bandit**, **prospector** (gathers many code quality checks: pylint, pep8, pyroma, vulture …) locally and codacy.com online.

CD will be handled at the YDEOS project level.

CD will create (upon tests success):

- an install script for Linux and Windows for the whole YDEOS project software
- a Docker image that contains the whole YDEOS project software

# A. Project checklist

## A.1. Structure

```
/<project>
  .gitignore
  requirements.txt
  LICENSE
  README.rst
  setup.py
  /bin
  /docs
  /tests
  /<same name as project>
    __init__.py (with text art and version, author, license info)
```

## A.2. Code

Type hints on all functions

PEP8 with a 120 characters line length limit

Numpy style docstrings

## A.3. Testing

Unit tests using pytest

## A.4. CI

Travis CI

Code quality