Capytaine: waves and floating bodies

CMSC6950 Course Project - 2021 Spring

Ruixin Song, 202093805

June 16, 2021

1 Introduction

Capytaine[1] is a python-based package reproduces an open-sourced software Nemoh, which is a Boundary Element Method (BEM) solver in the Computational Fluid Dynamics (CFD) domain. Applied to python, Capytaine reinforces the solver by utilizing the computational flexibility in python and combining with other efficient tools in the python scientific ecosystem such as VTK, NumPy, xarray, etc.

More specifically, Capytaine focuses on solving the problems of the water waves interacting with the floating bodies. It supports the computation of added mass, radiation damping, diffraction force and Froude-Krylov force for rigid bodies. For the theoretical part, it can perform Green Function and Kochin Function calculation inherited from Nemoh's core functionalities. Aside from these, Capytaine implemented 3D animation of the floating bodies motion and free surface elevation, which can function with the Visualization Toolkit (VTK).

With the features in Capytaine, the project defines several computational problems and uses the functionalities in the package to solve them, then extracts subset data from the results and makes them visible with Matplotlib and VTK.

2 Computational tasks

This project defines three computational tasks integrated in two separate scripts: `radiation.py` and `wave.py`. The following part is an overview for each computational task in the script.

1. Task 1 in `radiation.py`
   Set radiation problems with different set of variables: water depth, liquid density, frequency, acceleration of gravity, and calculate the added mass and radiation damping for each variable group.

The goal of task1 is to find the trend of the added mass and radiation damping through the change of a single physical variable. In each processing unit, only change the target physical variable without modifying the others. For example, if we want to figure out how water depth will affect the added mass and radiation damping, we will keep default values for the other variables: water density, wave frequency and the gravity, etc.

2. Task 2 in `wave.py`
   Set a diffraction problem with a sphere and a list of wave frequency. Solve the problem with the BEM solver and calculate the wave length, wave number and the wave period with different frequency values. The goal of task 2 is to find the correlation of the frequency, wave length, wave number and the period.

3. Task 3 in `wave.py`
   Generate the second geometry: a cylinder, along with the sphere in Task2. Calculate the influence matrices between the two floating bodies with customized green function. The goal of task3 is to see how one floating body might affect the other when they are positioned closely.

After defining the computational tasks, we should be able to explore in the Capytaine package to find the solutions.

3 Solutions

Capytaine has several functionalities to address the computational tasks we have defined in section2. Here introduce these functions and explain how they address our issues.

1. `RadiationProblem()` and `DiffractionProblem()`
   The two problem functions in Capytaine are derived from the LinearPotentialFlowProblem class, help to define the computational fluid problems. The parameters in the functions involving free surface, sea bottom, wave frequency, acceleration of gravity, have default values complied to the environment in Earth that can be applied to most of the computational cases. In task1, the series of problems are set by `RadiationProblem()`, with modifying the parameter groups. I customized 4 variable groups: a list of water depth(sea bottom), a list of omega (wave frequency), a list of g (acceleration of gravity), and a list of rho (liquid density). When performing the computation for each group, the remaining parameters in the problem function are kept with default values.
   In task2, I defined the problem by the `DiffractionProblem()` since I want to explore the diffraction force and Froude-Krylov force features in it. But things are not going smoothly, the computation result of diffraction force and Froude-Krylov force are stored in complex128 data format, that is performed in complex numbers. Without background in hydro-mechanics, I was not be able to figure out how to deal with the two kinds of forces

in complex numbers. Therefore, I simplified the problem to retrieve the calculation results of wave period, wave length and wave number, after using the problem solver `BEMSolver()`.

2. `BEMSolver()` and `HierarchicalToeplitzMatrixEngine()`
   Derived from Nemoh, the `BEMSolver()` in Capytaine is a solver for BEM problems. It has two parameters: green function and engine type, which can be customized by users. In my computational tasks1 and task2, I use the `BEMSolver()` with the `HierarchicalToeplitzMatrixEngine()` instead of the default `BasicMatrixEngine()`. After calculation, the solver function will return an Xarray data set, in which all the parameters and calculation results are inclusive.

3. `engine.build_matrices()`
   The `build_matrices()` feature in the `engine` class can perform the calculation of influence matrices of two given floating body meshes. The important parameters need to be set when calling the `build_matrices()` are two mesh objects and the green function. In task3, I use a `BasicMatrixEngine()` and call the `build_matrices()` function with the two floating body objects "sphere" and "cylinder" to calculate the influence matrices. After processing, the function returns two matrices S and K.

After the solutions complete the computational tasks, the results can be assembled by the `assemble_dataset()` function in Capytaine, which produces an Xarray data set. Then these Xarray data sets can be saved as NetCDF format files and passed to the visualization tasks. For the complex number in the computation results, for example the diffraction forces after performing solution to `DiffractionProblem()`, can use `separate_complex_values` in the `capytaine.io.xarray` module.

4 Visualization

All the visualization work are doing in the scripts `plot_radiation.py` and `plot_wave.py`. The following part will explain the data showing in each plot.

1. Derived from the computation task1, figure 1 displays the variation of added mass and the radiation damping with each of the variable groups. The corresponding code can be found in the `plot_radiation.py`. From the plotting result, we may find the trend of the added mass and radiation damping with the growth of water depth, liquid density, wave frequency and the acceleration of gravity.

2. Figure 2 from computational task2 plots the correlation of wave numbers, wave frequency, wave length and wave period. From the plot we may easily find the wave number is positively correlated with wave frequency, and negatively correlated with wave period and wave length. The corresponding code can be found in the script `plot_wave.py`. The detailed plot can be viewed in Matplotlib by rotating the plot object.

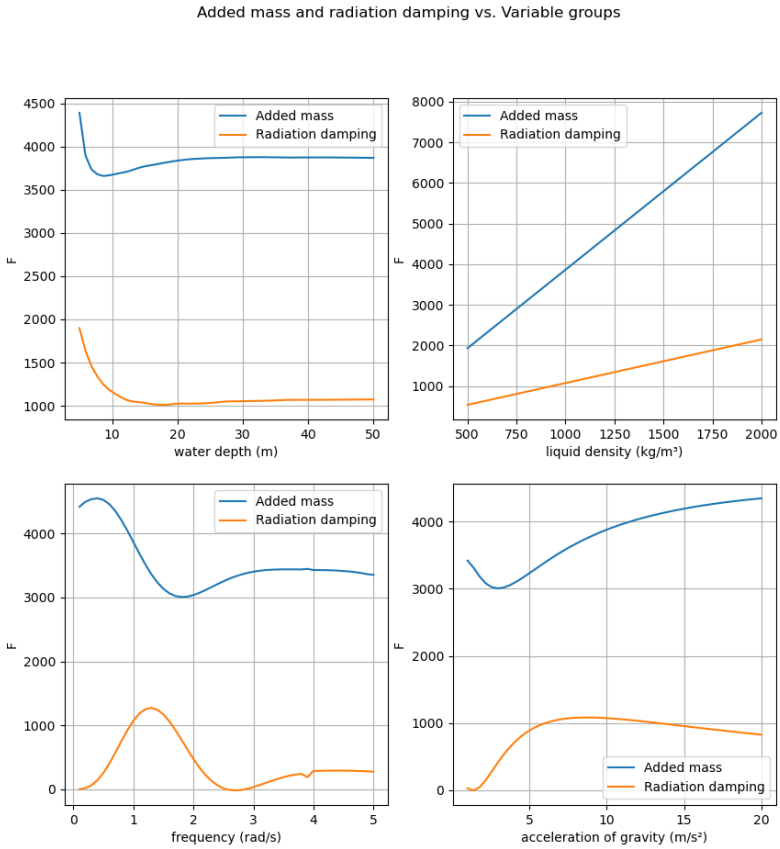

Figure 1: added mass and radiation damping

3. The data used in figure 3 is a subset of results in task2, the wave frequency. In the original plan, I want to use the free surface elevation data to generate the contour plot. However the data in the elevation array is one-dimension and hard to decoded in Matplotlib since they are values matching with the free surface normal. Therefore, I use sine() function to simulate the water waves, but some defect exists: the real wave amplitude decays with propagation distance, and this part might be fixed with exponent function.

4. Figure 4 depict the influence matrices of two floating bodies, which is derived from the computational task3. The matrices are not square because the given mesh sizes are different. The corresponding code can be found in `plot_wave.py`

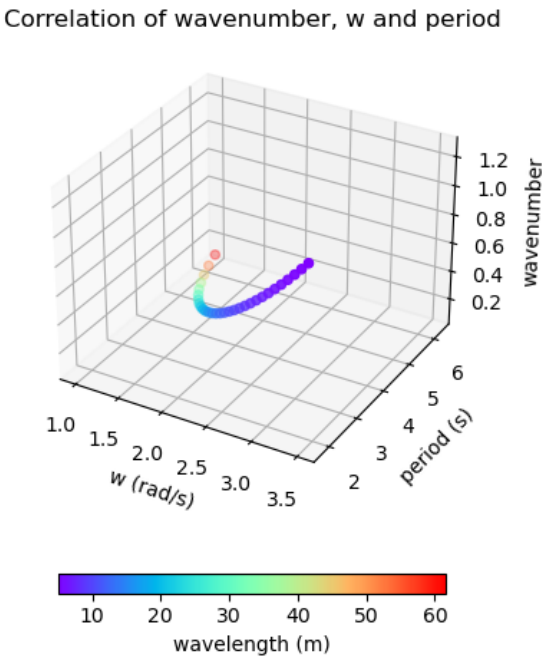

Figure 2: Correlation of wavenumber, frequency, wavelength and period

5 Workflow I/O

Figure 5 depicts the workflow of the project.

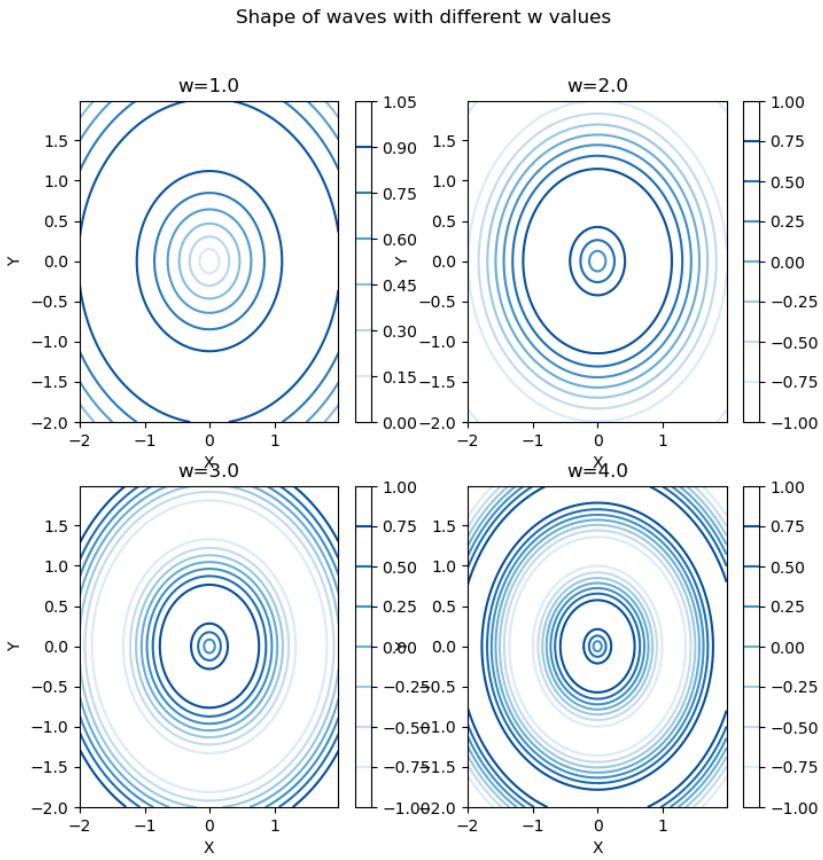

Figure 3: Wave shapes with different wave frequency

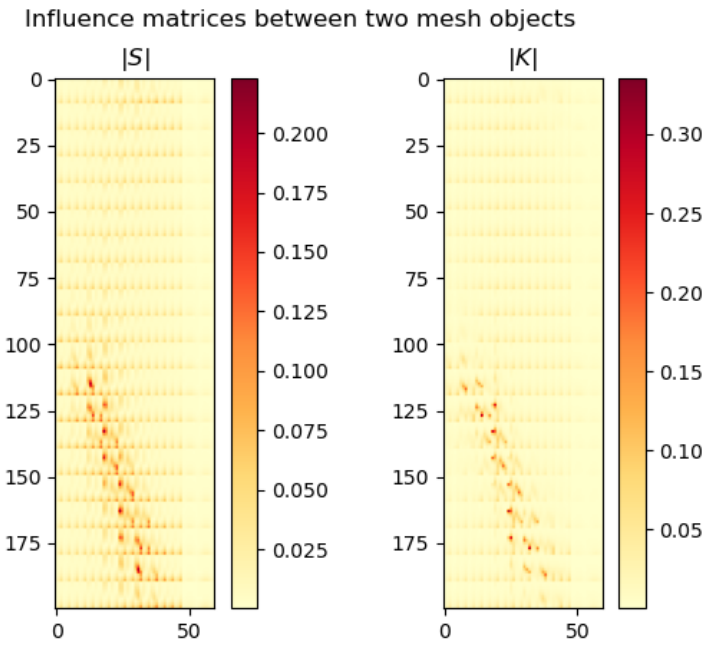

Figure 4: Influence matrices of two floating bodies

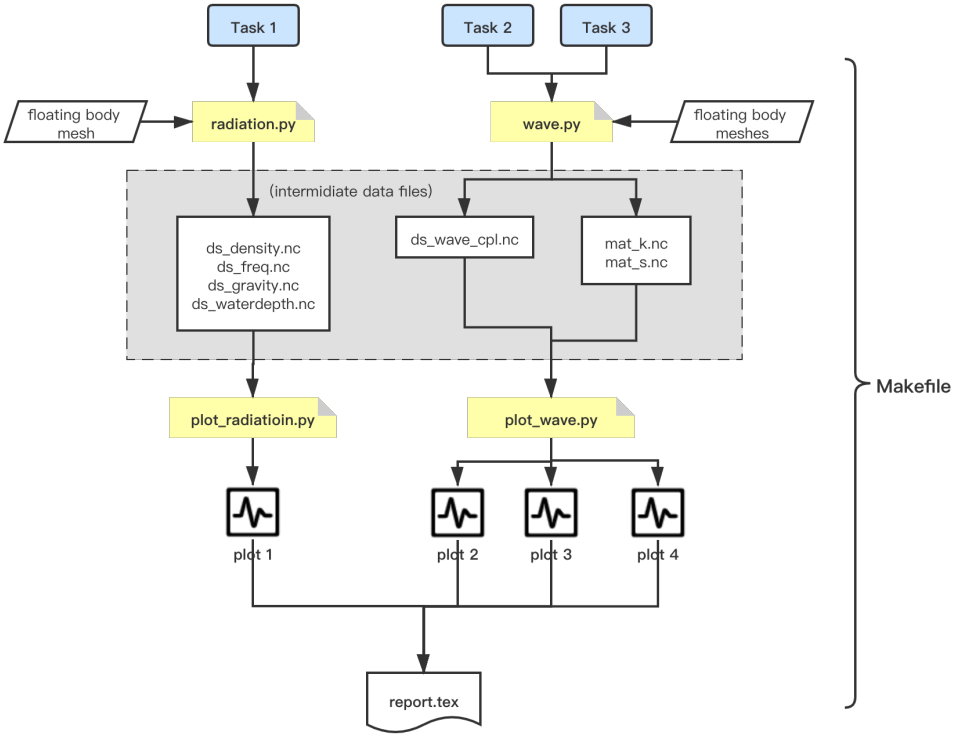

Figure 5: workflow of the project

References

[1] Matthieu Ancellin and Frédéric Dias. "Capytaine: a Python-based linear potential flow solver". In: *Journal of Open Source Software* 4.36 (2019), p. 1341. DOI: 10.21105/joss.01341. URL: https://doi.org/10.21105/joss.01341.