Using Ocean-Going Robots to Observe Wave Conditions: CS

Trevyn C Exley

Oregon State University

CS 463-400 Spring 2022

William Pfeil

May 23, 2022

Table of Contents

Introduction

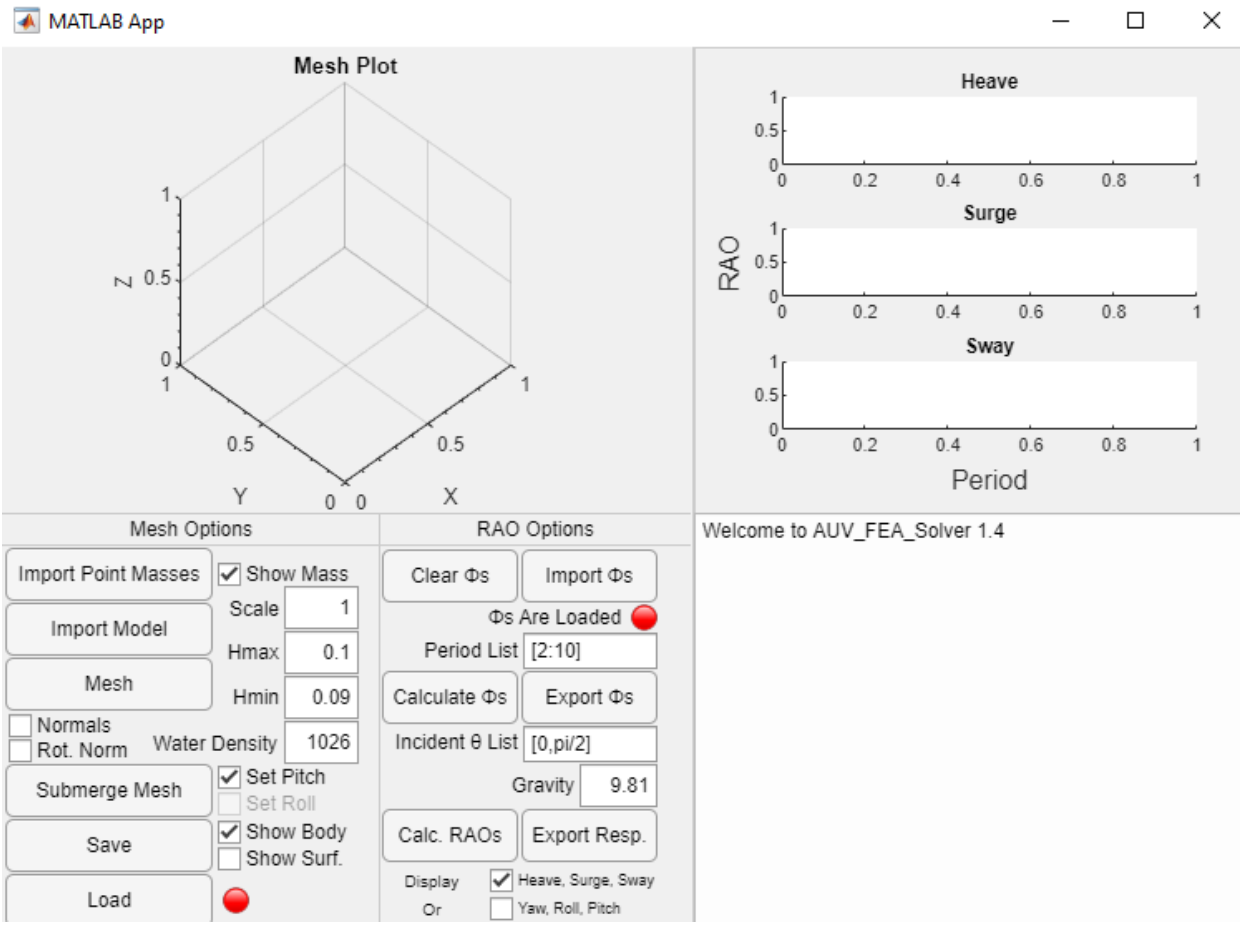
This paper is a report detailing what the project is, how to use and interact with it, and how to continue development with it. The User Guide gives an overview of the project and its features. Then, the System Description describes the main components of the application and how they work together. Next, the Development Software section lists which software I used for what. Finally, Project Continuance lists modifications that can be made to the application to improve it and how to go about doing that.

User Guide

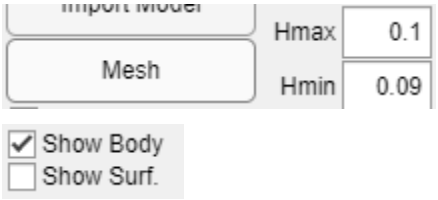
AUV_FEA is an application that allows users to get the wave characteristics of a floating body. Specifically, the user can export the body's responses to waves of multiple frequencies from multiple angles. Additionally, users can view the wave characteristics through plots of the response amplitude operators (RAO).

Features

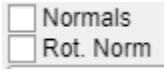
1. The first feature a user might see is a functioning GUI shown below. It even has a console in the bottom right where buttons can give the user feedback.



2. By first importing an STL and TXT file representing the floating body's shape and mass distribution (estimated as a list of point masses: m,x,y,z) the user can create and view a triangle mesh of the object's surface using the Mesh, Hmin, and Hmax buttons. Now the Mesh Plot in the top left will either show the entire model or just the part of it submerged below z = 0 depending on if Show Body or Surf. is chosen respectively.



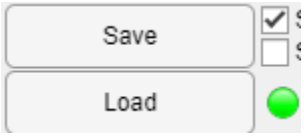
3. Similarly, a user can toggle the normals for all six degrees of freedom using the Normals and Rot. Normals checkboxes.



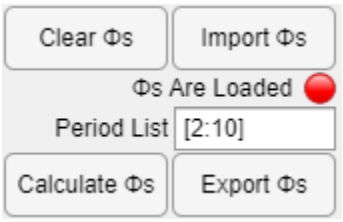
4. After creating a mesh and specifying the Water Density the user can then use the Submerge Mesh button to have the app automatically find the position where the body

floats at neutral buoyancy. However, this position only considers the object's depth and pitch (if the Set Pitch box is checked).

5. If the body's mesh is positioned beneath the waterline (z = 0) and the point masses have been loaded, then the red light next to Load will turn green. So long as the light is green, the user can use the Save button to save their estimation of the floating body. Later, this save file can be restored using the Load button.



6. A subsequent feature of the Save button is that users can use third-party software to re-mesh the submerged surface by changing the SurfaceMesh.STL file created.
7. Under the RAO Options section, the user can tell the machine to calculate the velocity potentials on the submerged body's surface for a list of periods with Calculate Φs. The separation of the Φ calculation allows the user to perform multiple disjoint calculations on the same floating body's mesh. This is achieved through the use of the Export and Import buttons that allow results to be shared between instances of the app. Also, any Φs currently loaded are not overwritten when calculating for new periods. To clear the Φs a user must press the Clear Φs button or import new Φs from a file (two saved Φ files cannot be combined in the app). The separation of the Φ calculation also allows the user to quickly get the RAOs or responses of the body with different incident angles and gravitational forces.



System Description

The system is split into a MATLAB application file called "AUV_FEA.mlapp" and supporting MATLAB ".m" function files named in camelCase. The function files contain most of the actual math and algorithms, while the app file is filled with callbacks and helper functions to use those files. There is also an additional folder called tests that contains a small unit test suite for most of the functions. These can be run with the command "runtests" from inside the folder

to check that a change didn't break anything. Inside this folder, there are also the Gaussian and Green comparison test files. These can be used to find input combinations for the Green function and Gaussian quadrature from "velocityPotential.m" respectively.

The core of the system is spent creating a system of damped and coupled harmonic oscillators when subjected to external forces. The matrices themselves are defined in "addedMassAndDampingMatrices.m", "bodyInertiaMatrix.m", "hydrostaticRestoringMatrix.m", and "excitingForces.m". The first and last of these are dependent on the output of "velocityPotential.m" which itself uses a Gaussian quadrature to compute the integrals of the "greenFunctionAndPartialXI.m" (as in the "greenFunction.m" and its directional derivative). Finally, the Green function uses the Adaptive Simpson's Quadrature algorithm ("asq.m") to solve its integral.

Development Software

- MATLAB - Used version R2021b to develop the entire app and its functions. Originally chosen for the wrong reasons, it still turned out a useful language for manipulating matrices. However, a remake might use c/c++ for better performance.
- Parallel Computing Toolbox - Used to run a single for loop in the "velocityPotential.m" file in parallel. This loop is a massive time killer; therefore, I recommend allocating as many cores and hyperthreads as available on the system to your default parallel pool.
- Partial Differential Equation Toolbox - Only used for its functions to import an STL file as a DiscreteGeometry and then generateMesh().
- Autodesk Fusion360 - I used Fusion360 to create STL files for the test bodies I wanted to model. The process is straightforward, but make sure that the object's position lines up with its point masses and is close to the $z = 0$ plane. Also, the object's length should be orthogonal to $y = 0$ with the center of mass (COM) at a lower x value than the center of volume if you want the app to find the correct pitch (where the COM sits below rather than above the center of buoyancy (COB)).

Project Continuance

Before I talk about new features or modifications to the existing app, the most obvious continuation of the project is developing an app that can use the output from this one. In his paper, "Assessment of Sea Wave Spectra Using a Surfaced Glider," A. Alvarez makes it seem like the responses (also called transfer function) of the floating body can be used to find directional wave spectra [1]. I can't say for sure, but the relevant part of the paper would be equation 14 in section 3.3. The responses are currently exported as a file with the following properties:

1. The first line has the number of angles followed by each angle in the order it appears and padding zeros. (All separated by commas)

2. The second line has the number of periods followed by each period in the order it appears and padding zeros. (All separated by commas)
3. Every 6 lines afterward are the response for each of the 6 degrees of freedom for the corresponding angle where each direction of freedom (x,y,z,rx,ry,rz) is on a new line.

Each individual line has the response for corresponding periods separated by commas.
It might also be useful to view the "ExportResponseButtonPushed()" callback inside the application file.

Next, the "SubmergeMeshButtonPushed()" callback can use some updates. The first consideration is that it only tries to reduce the spacing between the COM and COB when ideally a submersible will keep its COM below the COB while surfaced. So trying to find the angle that reduces the distance and moves the COM below the COB is ideal. Since most objects will have multiple positions where they can be neutrally buoyant even with the condition that COM is below COB, it would be useful to distinguish between them to find the most stable one. This might be done by finding one buoyant position and then trying again from an angle offset by 180 degrees. Once multiple positions are found they would then need to be compared based on their stability (probably a function of the distance between COM and COG). Once all of this is done with just pitch and height (x and y pos are irrelevant), then you might look into finding the correct roll of the object. Currently, the algorithm works by finding the depth where p*V=M ("moveDepth.m") for each pitch and using this to tell how close the pitch is to a neutral position. Likely, the roll angle will increase this by an order of magnitude where the pitch and depth will each need to be found for every tiny change in the roll. A good start to making this all faster would be to have the depth close in on the correct value faster. It currently looks for each decimal in base 10 until it has the depth down to 6 digits. However, it would be much faster to go by orders of 2 instead. For each depth, it could half the dz check if it needs to go up or down, then half the dz again adding it to either the new or old depth. It might also be useful to give the function a starting dz value. Currently, it always starts at 1m, but for a brand new system, the body might be 10m away from the waterline while one near the correct pitch might only be 0.001m from the correct depth. On another note, the change in pitch works just like dz but by 10s of degrees at a time and will similarly benefit from a log2 behavior.

The Adaptive Simpsons Quadrature ("asq.m") is the function where the CPU spends most of its time during a velocity potential calculation. Therefore, any performance boost here can improve runtimes drastically. One such boost can come from changing the implementation from a recursive one to an iterative one. This is mostly because MATLAB takes way too long in function calls. Just removing a single parameter from the function boosts its efficiency. So, it should make sense that getting rid of them entirely will be the fastest. Someone has already made an iterative ASQ function and confirms that it is faster, however, I couldn't figure out how to use their implementation for our purposes. That function might still be an option, but most likely the iterative version will need to be built from scratch for this project. Also, it will most likely be faster to pass the 3 values to be used in a built-in function (as I have done in "asq.m") rather than passing an anonymous function like the one required for the other iterative ASQ.

Another useful modification to the app would be the ability to choose between different Gaussians for different projects. For rectangular elements, A. Alvarez used a 4th order Gaussian Quadrature. However, for triangular elements, I found that a 3rd, 4th, and 6th order all work well. The current implementation in "veclotiyPotential.m" uses the 3rd order as it only uses 3 points compared to the 6 and 12 points of the other two, but out of the three, each higher-order gives more accurate results. Also, the actual class I used to test different Gaussians is in the "tests" folder and called "gaussianComparisonTests.m". Inside this class is a function called "getGaussian()" that returns a list of points and weights to test for each different Gaussian I tested. The good ones are the 1st, 8th, and 10th elements of these lists respectively (the 5th element is another variation of the 3rd order and it works well but slightly worse than the other). These three Quadratures are all shown in "Quadrature Formulas in Two Dimensions" by Meredith Charles [2]. The feature itself would just need a way to perform a different Gaussian depending on whichever option the user has chosen in the app. This would be especially useful if the user could find the approximate area of the object's natural frequency and only use the larger Gaussians there and the lower order ones for the rest. Only around the object's natural frequency did the higher-order Gaussians substantially change the output.

The final improvements I can think of are for the Green function. Inside "greenFunctionAndPartialXINormal.m" there is an epsilon value for calculating the derivative. If this value is too small or too large then the result is garbage. The "greenFunction.m" has an epsilon for isolating the integral's singularity, and a value that estimates the evaluation of the integral to infinity. The smaller the value around the singularity the more accurate the result until a tipping point where the value becomes garbage. As for the infinite variable, it can't ever be smaller than K but doesn't need to be much larger either since usually the exponential part drives the answer to 0. The exception for this is at $z = 0$ where the exponential is 1 and the Bessel curve continues to oscillate forever. In this case, the best solution might be to set the infinite estimate to one of the zeros of the function. Then the approximate result would be the average value of this and the value obtained when adding the area from that zero to the next. However, the logic behind this might take too long so be careful (maybe use an if case to only use when $z = 0$). There is one more relevant constant set in ASQ that is now the epsilon value for when the algorithm converges. This value is the one that changes the runtime of the whole process the most. The current values I have I decided on after running my "greenComparisonTests.m" class and looking at the results in excel for their convergence. Each of the variables listed above interacts with each other and effect when the overall result converges. A feature useful to the user might allow them to choose between different presets for all the variables with the ASQ epsilon being the major one. The user might also like to know how the different presets might affect their results. The current layout I have set seems to be the fastest one that is most likely to converge for triangular elements with points 0.5-0.0001 apart, but this could be refined for sure.

Conclusion

To wrap this all up, it was a fun project. I ended up learning a lot about finding, reading, and using research papers. I also learned a lot about the Finite Element Method, various estimation algorithms, a bit of new math, and even some hydrodynamics. Maybe most importantly, I got experience working with the project partners to achieve a goal and practice with the documentation required. The project itself can still be modified in many useful ways and would greatly benefit from the extra accuracy and reliability they would bring. But the project does currently achieve its goal, and I call that a win.

Citations

[1] A. Alvarez, "Assessment of Sea Wave Spectra Using a Surfaced Glider," Deep Sea Research Part I: Oceanographic Research Papers, vol. 102, pp. 135–143, 2015. [Online]. Available: https://openlibrary.cmre.nato.int/bitstream/handle/20.500.12489/844/CMRE-PR-2019-104.pdf?sequence=1&isAllowed=y

[2] Charles, M. (2018, April 20). *[PDF] quadrature formulas in two dimensions math finite element method Section 001, spring 2010 - free download PDF*. Quadrature Formulas in Two Dimensions. Retrieved May 24, 2022, from https://silo.tips/download/quadrature-formulas-in-two-dimensions-math-finite-element-method-section-001-spr