**Capstone Technical Document**

***INTERACTIVE VISUALIZATION IN HTML: SOFTWARE-FOCUSED DOCUMENTATION***

*(For guidance on how to edit the equations used in this application, please refer to the Architecture & Design section of this document.)*

## Overview

In the academic world, graphics are used in scientific papers to convey important information. In their current state, these graphics are limited in the amount of information they can display. Not only are they static (see **Figure 1**), but they are often limited to one parameter set, making it difficult for others interested in the topic to alter the models to fit their needs. Moreover, there is no easy way to obtain the raw data used to generate the graphic.

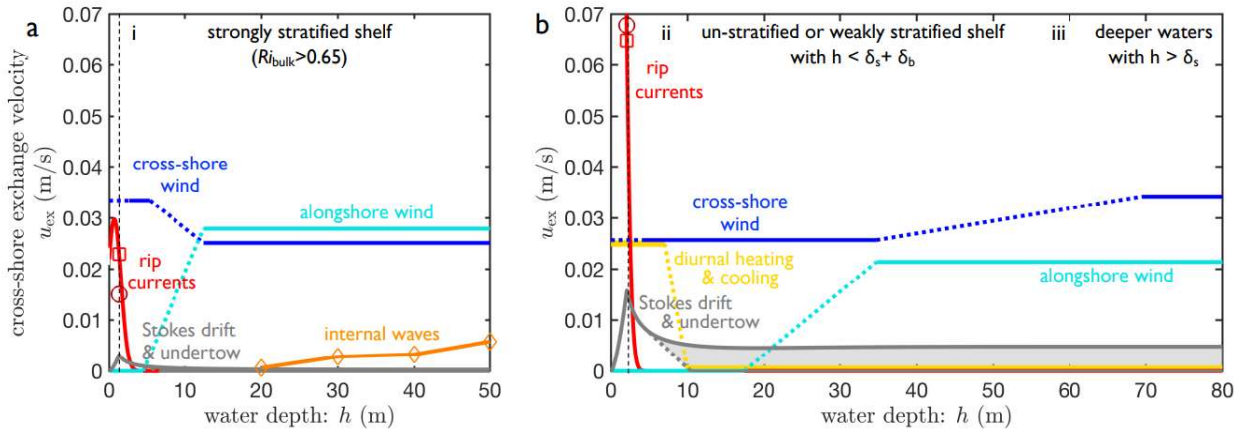


**Figure 1:** Example of the static graphs currently used in scientific works.

The goal of Interactive Scientific Visualization in HTML is to create a web-based application for scientific researchers and academic viewers that serves as a robust and reliable solution to the limitations of current scientific graphics. Our application will extend the current capabilities of these graphics by displaying data values, dynamically altering the graphic through custom parameter input, and by making the graphical data downloadable as a common file format. These extended capabilities will improve academic viewers' understanding of the complex models these graphics depict and will allow researchers to provide a new level of depth to their figures. These capabilities will also allow users to alter pre-existing models to fit their own needs. In terms of robustness and reliability, our application design is robust with respect to

the technological changes that will happen over time, and it is reliable in terms of security and accuracy. We also aim to make our application accessible across varying browsers and devices. Our goal is to support any closed form, real-valued equation expressed with standard algebraic and trigonometric functions on 95% of browsers.

## Requirements

The current limitations of graphics used in scientific papers range from a lack of interactivity to an inability to represent multiple datasets. These limitations affect two different types of users. The first type of user we consider is an academic viewer, henceforth referred to in this document as a **viewer**. Viewers are either students, academic professionals, or generally interested parties who will seek to use our application to either improve their understanding of the scientific models depicted or apply their own dataset to it. The second type of user that we consider is a publishing researcher, henceforth referred to in this document as a **researcher**. A researcher is an academic professional who is seeking to use our application to aid with the publication of their research. The main benefit of our application for a researcher is the ability to integrate their own complex models as equations into our framework. With these two types of users in mind, our product's goal is to overcome the limitations that these users face with static scientific graphics, while also withstanding technological changes over time, being robust and reliable, and providing additional capabilities to further improve the user's experience. The following lists outline both the nonfunctional and functional requirements for our product.

**Nonfunctional -**

- Application must be completely client-side with no outside dependencies
- Must support any closed form, real-valued equation expressed with standard algebraic and trigonometric functions
- Must run on 95% of browsers (Chrome, Safari, Firefox, etc)
- Must complete calculations in under 1 second
- Must handle invalid input gracefully
- Must withstand security issues
- UI must be easy to understand
- Graphs must display data in an understandable manner
- Must have two graph panels with their own set of parameter inputs
- System design must withstand technological changes so that our application is relevant and accessible for up to 5+ years

- Code design must follow code best practices that make it easy to understand and maintain, so users can alter the code to fit their needs.
- Code design must be flexible to allow for users to change the equations used in graph data generation

**Functional -**

- Users should be able to use input boxes for valid parameter input.
- ‘Apply Parameters’ button will accept user input from parameters input boxes and generate valid graphs based upon them.
- All parameters must be sanitized before generating graphs.
- Application will throw an error when attempting to apply invalid inputs
- ‘Reset Parameters’ button will reset the parameter input fields to default values and reload the graph with these default parameters
- Show/Hide equation selection should allow users to add or remove specified equations from being displayed on the interactive graph.
- The graph must display values when the user hovers the mouse over certain points on the Cartesian plot. For example, if the user hovers over the point (1,1), the website should display the x and y values for the various equations that intersect that point.
- Graphs will display all equations accurately using data generated with the specified parameters.
- Each of the lines on the displayed graph should be distinguishable from one another by its attributes, such as color or line type.
- The graph should include a key to distinguish equations based on line attributes.
- X-Value generation must provide accurate corresponding y-values for all listed equations
- ‘Download CSV’ button should convert the data backing the graph into a formatted .csv file.

## Architecture and Design

We will begin this section by discussing our current UI design. At the top of the webpage is explanatory text that provides the user context for how to use the application. Directly below this text is a set of input boxes where the user can input custom parameters. The 'Reset Parameters', 'Apply Parameters', and 'Download Graph Data as CSV' buttons can be found below the parameter input boxes, as shown in **Figure 2.**

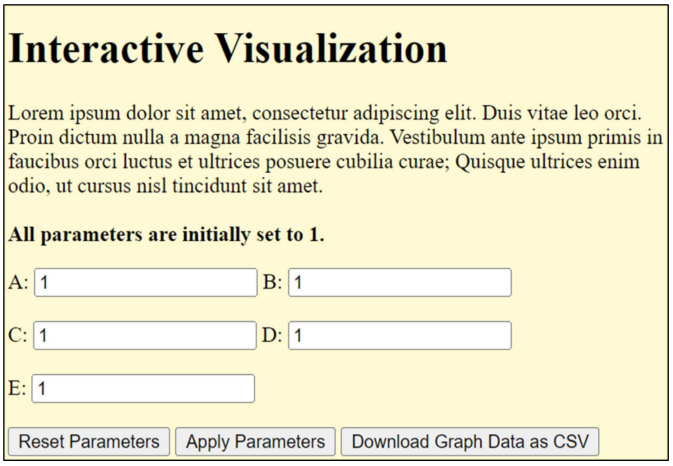


**Figure 2:** The explanatory text, parameter input boxes, and several useful buttons

The interactive graph lies below the custom parameter input and explanatory text. The interactive graph consists of the legend on the top, the scales on either axis, and the plotted lines on a cartesian plane, with unique colors to distinguish one equation from another. A user can hover over each line to see the values of the plotted points at various coordinates. A user can also click on any box in the legend to either hide or show the respective equation on the graph. These features can be seen in **Figure 3.**

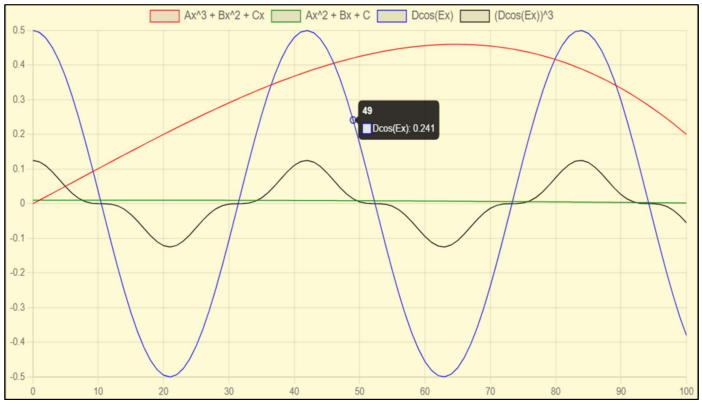


**Figure 3:** Interactive Graph with mouse-over feature shown

The 'Y-Value Generation' feature is anchored to the right hand side of the page. The input box accepts a floating point value that represents the value on the x-axis that a user would like to query. A user may

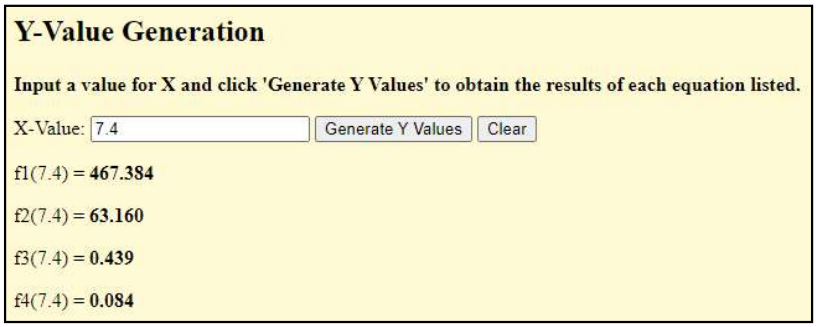


**Figure 4:** Y-Value Generation

press the ‘Generate Y Values’ button to generate and display the corresponding Y-Values for the given X-Value for each equation included in the interactive graph. A user may also click the ‘Clear’ button to clear both the generated Y-values and the X-value in the input box. The purpose of this feature is for users to obtain the value of each function at a specific X-value, specifically one that is difficult to obtain from interacting with the graph.

The major system components in the architecture of our application are the Webpage & supporting files, the Chart.js Library, and the Javascript files. The Webpage & supporting files include the main HTML file for the webpage, the Contributors page, the CSS stylesheet, and the User Documentation. The Webpage & supporting files interact with the User and the JavaScript files, and acts as a bridge between them to process the User’s requests. A simple architectural diagram can be seen in **Figure 5**.

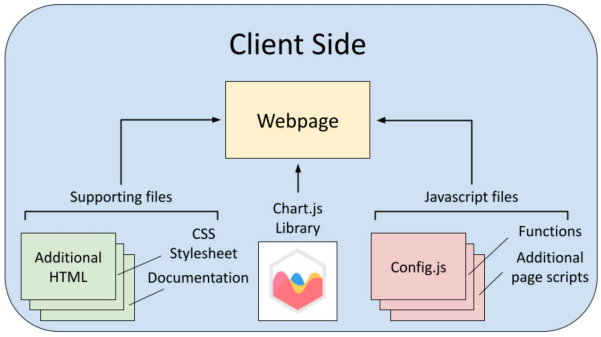


**Figure 5:** Architecture Diagram for *Interactive Visualization in .html*

The most important piece of our architecture is the Javascript files. These files provide the bulk of the functionality of the page. The following is a list of the files, the important aspects they cover, and how to edit them to represent new equations.

- ***functions.js:*** This file is where the equations to be plotted are defined. It includes one function per equation, where every equation must take the parameters it uses and an x value as arguments. *To integrate new/different equations*, the editor must add/remove functions to represent the needed equations.
- ***init_data.js*:** This file is used to set up the data arrays with default parameters. *To integrate new/different equations*, an editor must add/remove data arrays (one per equation), define the default parameter values, and edit the data generation with the new data arrays and their respective function call.
- ***config.js:*** This file includes the configuration required to set up the chart for the Chart.js library. It includes the equation names, the line colors, the data assignment, and the font size. *To integrate new/different equations*, an editor must add/delete a ‘dataset’ to/from the list of datasets. Each dataset includes the name of the equation, the line color, and the corresponding data array

assignment. The font size can be adjusted in the “options” section near the bottom. Each chart displayed will require a corresponding configuration. If the number of charts displayed must be changed, ensure that the assignment of the chart configuration at the bottom of the file is present for each chart intended to be displayed.

- ***page_methods.js*:** This file contains methods that are more or less directly related to the functionality of the webpage itself. The functions are as follows:
    - resetForm() is called when the ‘Reset Parameters’ button is pressed, and it resets the parameter input boxes to the default value provided in the button HTML element. The method then updates the chart with these values. Nothing must be done to this function when integrating new/different equations.
    - updateChart() is called when either the ‘Apply Parameters’ or ‘Reset Parameters’ buttons are pressed. This method reads in each parameter, validates the input, generates new data from these parameters, and then updates the chart with the new data. *To integrate new/different equations*, an editor must:
        - add/remove lines to read in each parameter
        - add/remove validation for parameter values
        - add/remove data arrays to empty
        - add/remove data arrays with their respective functions for data generation
        - add/remove update calls for however many charts are currently displayed on the webpage
    - generateY() reads in the Y-Value Generation input and produces output for each function using the input value. *To integrate new/different equations*, the string generated must be edited to include the number of functions required.
    - download_csv_file() uses the current data to create a CSV representation that is then downloaded. *To integrate new/different equations*, the equation names and the number of data arrays used must be changed.

*The only other file that must be edited for the purpose of changing the equations displayed in the application is the main HTML file for the webpage.* An editor may change the explanatory

text to describe the depicted model in the paragraph element labeled “Description”. To change the parameter input boxes, an editor must add/remove both a label element and an input element corresponding to the respective parameter. Lastly, to edit the number of charts displayed, an editor must add/remove the canvas element with its respective chart ID.

A more detailed flow of how the User applies custom parameters can be seen in **Figure 6.** First, a User will input the custom parameters into the parameter input boxes and press the ‘Apply Parameters’ button. This button press calls the updateChart() method. It first will parse the parameters and check if they are valid. If they are invalid, the script will alert the user, and the chart data will not be updated. If the parameters are valid, it will generate new data based on the equations listed and the new parameters provided. It will then reconfigure the chart to include the newly generated data and pass this new configuration to the Chart.js library. The Chart.js library will consequently update the graph.

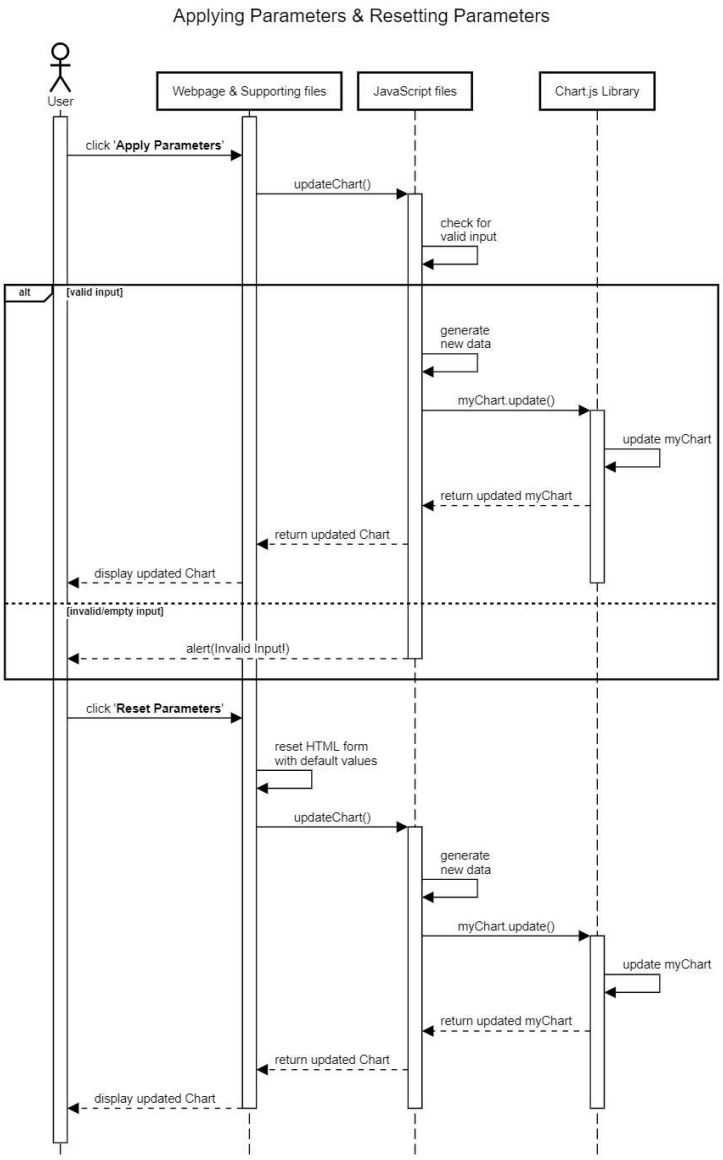


**Figure 6:** Sequence Diagram for Applying Parameters & Resetting Parameters

**Testing/Results**

There are many test cases required to ensure that the success criteria for this project are met. The following list outlines the testing requirements.

- **Accuracy Testing:** Unit tests and other small scale testing is required to show correctness for features included in the web page such as the data generation, the Y-value generation, Hide/Show Equations, and the 'Download CSV' feature.
- **Performance Testing:** Our web page must respond quickly when being used. Testing must be done to ensure that features respond within at most 1 second, including the Apply/Reset Parameters buttons, the Y-value generation, and the
- **Input Coverage Testing:** Automated parameter input testing for validity, known as fuzzing, will ensure that no invalid inputs will be accepted. We will use a fuzzing testing platform to run this testing. To be deemed as successful, we should see no faults in these fuzzing tests.
- **Interface Comprehension Testing:** Viewers will make use of our webpage in order to enhance their understanding of complex equations. To ensure that the viewer's user experience is seamless, we must conduct testing to receive feedback about the comprehensiveness of our application. To do so, a group of students will be asked to interface with our application via a set of tasks. These tasks will include all possible features of our application. We will then ask the students to fill out a response survey that will ask them about their experience, specifically how easy the application was to use, what aspects were confusing, and how the application improved their understanding of the models depicted. This feedback will be collected and reviewed to further improve our user interface and additional features.
- **Researcher Usability:** A researcher must be able to integrate their own equations in order to make use of our product for their own research models. This means that our code structure must be easily adaptable and our user documentation must be comprehensive. We plan to test this usability by surveying researchers in a similar manner to the Interface Comprehension Testing, except that these users will be instructed to integrate provided equations into our framework. The survey feedback will provide us with information about what aspects of our organization and documentation are confusing and what aspects are clear. We will then adjust our organization and user documentation accordingly.

- **Cross-Browser Functionality:** We need to test against multiple browser types to ensure that our users can make fluent use of our web application. Different browsers have their own unique way of rendering and displaying pages, so we must ensure that our page runs with all of its functionality on 95% of browsers.
- **Compatibility with Different Devices:** It is crucial for our web application to be accessible on both desktop and mobile devices because it enables a wider audience to use and engage with the application. By ensuring that the web application is accessible on both platforms, users can seamlessly switch between devices without losing any functionality or compromising their experience. This form of testing is fairly easy for us to conduct ourselves. Among the members of our group, we have a variety of devices that would cover what is being used by our target audience. By testing our application on our own different devices, we should be able to ensure consistent usability and user experience

## Discussions & Conclusion

The most important goal of this project is to provide an application that allows researchers to provide a greater level of depth to their scientific models while enhancing viewer's comprehension of these models. The following features address the status of our application:

**Current Completed Features:**

- Custom Parameter Input with input validation
- 2 Interactive Graph panels
- 'Reset Parameters' button & functionality
- 'Download Graph Data as CSV' button & functionality
- Y-Value Generation with input validation
- Hide/Show Equations
- User Documentation that provides comprehensive instruction on how to interface with the application

In its current state, Interactive Visualization in HTML provides functionality that can improve a viewer's comprehension of complex scientific models and serve as a robust solution for present issues with static graphs for researchers. The application can support any closed form, real-valued equation expressed with standard algebraic and trigonometric functions. The

features listed above fulfill a majority of the requirements for our project. Another critical requirement that our application fulfills is that it must be designed to be entirely client-side, with no outside dependencies. The purpose of this is to prevent the page from becoming obsolete in the future. Furthermore, the application's code has been written with best practices, and is organized in a logical manner. Along with providing user documentation, this allows for researchers to integrate their own equations to our application.

In the future, we plan to fully incorporate the oceanographic functions provided by our sponsor into the application. We also plan to begin usability testing and comprehension testing with graduate students provided by our sponsor. We will gather feedback from this testing and make necessary changes to our webpage and source code to improve user experience for each user group. Additionally, we will conduct performance and accuracy testing to ensure the application meets these requirements. Cross browser functionality testing will ensure that our application runs on 95% of browsers, fuzzing testing will ensure input is sanitized, and compatibility testing with different devices will ensure usability across different platforms. Finally, we must also obtain the MIT Licence to be able to consider the project open source.

## **Appendices**

*Summary of work:*

- Jimmy
    - Integrate CSV download button
    - Integrate displaying simple graph using framework
    - Display graph using parameter inputs
    - Integrate parameter sanitization
    - Organized code into logical sections
    - Integrate Y-value generation
- Beau
    - Worked on the front end using html
    - Webpage skeleton design
    - Researched potential frameworks
    - Worked with Jason in advanced equation comprehension provided by sponsor
        - Created a comprehensive list of advanced equations
    - Architecture Diagram
    - CSS Stylesheet

- Mitch
    - Integrate CSV download button
    - Display graph using parameter inputs
    - Parameter input testing
    - Organize code into separate files
    - Integrate Y-value generation
- Jason
    - Worked on the front end using html
    - Researched potential frameworks and how to implement chart.js
    - Wrote author link/document
    - Basic web skeleton elements
    - Worked with Beau to comprehend and organize advanced equations
    - CSS Stylesheet

*User Documentation:*

This document will be used for our user documentation.

*Resources*

- Our GitLab, which was used for project organization, can be found here.
- Our code repository, hosted on GitHub, can be found here.
- Our web application can be accessed here.
- Helpful documentation for Chart.js can be found here.
- Helpful documentation for understanding MatLab can be found here.