Blake Hensel
Laura Jauch
Kyle Kent
Conner de la Cruz

Phase II: Testing Plan

Our goal is to assure accurate and functional code by writing a variety of test cases to model the variety of user experience. In most cases, the test strategy is to build two identical forms, run one of them through our program's version of the code and the other through the version of the code within the test case and compare the results from our code with the actual solution.

testSaveLoad: Test cases for the save and load classes. A form is created and saved to the form object. Elements of the form object are retrieved and compared to the original form to assure that it was stored properly. The form is then saved and loaded and it's energy error, number of elements and poly order are compared to the original form to make sure they are equal.

testRefine: Test cases for refinement. A form is created and then refined within the test case. The same form is then fed through our refine class. Elements of the form are compared to make sure that our refine class did the proper calculations. This process is applied to automatic and manual p-Refine and h-Refine for both Stokes and Navier-Stokes equations.

testFunParser: Test cases for the function parser. Accurately parsing function input is essential to program performance. Here several strings are fed to the function parser and evaluated. The results are compared to the correct answer to the function to assure correctness.

testInput: Test input and solver classes. These tests model input into the program (particularly the solver class) and check that the information is properly applied in the construction of a form. Input functionalities are also used in checking manual refinement (cells to refine).

testPlot: Test plot class. Create forms and run one through our plotting code. Compare the actual solution with what our code produced, checking that the points that were generated match and the values mapped to each are equal and correct.