Nina Camillone & Laura Jauch (Team 1)
CSCI335: Software Development
Dr. Roberts
Phase 1 Testing Plan

Function

For the instance methods, which are all fairly similar, we can feed the function something for which we already know the solution and check that the returned solutions match the expected solutions. For static methods evaluate and composedFunction, we can check that it returns the same value as a known function. To check that constant works, we can feed the returned function several values (ex. zero, a positive and a negative) to make sure that it always equals the expected value. To test the methods that return a function, we can compare the returned function pointwise to the expected function using Function::evaluate(). The operator overload methods can be tested by passing function pointers that should return known functions to the operations as parameters and using Function::evaluate to compare the results of the returned function to what we expect or evaluating the 12norm on some mesh of their difference and making sure it is zero (or really, really close).

SpatialFilter

For the instance methods we check the points against a known filter to see if the function returns as expected. To test the static methods that return a SpatialFilterPtr we can compare the returned filters against known filters and use the instance methods to confirm equality pointwise.

BC

First we can check the getDirichletBC against a known boundary condition. Then we can use it to check addDirichlet when we try to impose certain boundary conditions. For getSpatiallyFilteredFunctionForDirichletBC, we can compare the function returned for a particular parameter with the expected boundary conditions. For bcsImposed, singlePointBC, and valueForSinglePointBC, we can have a known single point BC and check that the former functions return true and the latter returns the expected value. We can then use these to check addSinglePointBC. For vertexForSinglePointBC, we should test impositions both with and without specified vertices, which should return the vertex or -1 respectively. For imposeZeroMeanConstraint, we should test that variables with and without zero-mean constraints return true or false accordingly. Once this is working, we can use it to test whether addZeroMeanConstraint and removeZeroMeanConstraint are working. We can use a variable without a zero-mean constraint to addZeroMeanConstraint, at which point imposeZeroMeanConstraint should return true, and then pass the variable to reoveZeroMeanConstaint, when imposeZeroMeanConstraint should return false.