Team 4 Testing Plan

Directory Structure

/TestAutomationTeam4

- /project
  - /src
  - /bin
  - /pom.xml?
- /scripts
  - /tests
    - runAllTests.sh
    - NelderMeadTestCase.sh
    - ...
  - /build
    - build.sh
- /testCasesExecutables
  - NelderMeadTestCase.java
- /temp
- /oracles
  - NelderMeadTestCaseOracle
- /docs
  - README.txt
- /reports
  - allTestsReport.txt

Testing Framework

runAllTests.sh under the directory "/scripts" will run all tests that rely on input from "/testCases." Each file under "/testCases" will contain csv-formatted inputs for each test case, and each test case will be checked against a corresponding oracle under "/oracle." Both the inputs and oracles for each test case are passed into the main methods of Java classes, each of which contains a single test case. There will also be a separate script for each test case, so any single test case can be run with a single script. These scripts will execute the Java classes that in turn write the results of the tests they run to a single file under "/results" with strings in the format "Test Case X has failed/passed." runAllTests.sh, for example, writes results to a file titles runAllTestsResults.txt. and will then check the file for failing tests to notify the user in the terminal. The results of the tests will then be displayed in the default browser. For the testing framework to run, the required classes from STEM will need to be compiled under /project, which will be taken care of via build.sh under /scripts.

Test Cases

- **org.eclipse.stem.analysis.automaticexperiment.NelderMeadAlgorithm.java**
- **Test class:**

Method	Inputs	Oracles
getMinimumFunctionValue ()	SimplexFunction (-3x + x^2 – y -xy + y^2), 0, 0.0, 9999999.0, , 1, 0.0, 9999999.0, (1.2, 1.8), (.5, .5)	-4
getMinimumErrorResult ()	SimplexFunction (-3x + x^2 – y -xy + y^2) 0, 0.0, 9999999.0, 1, 0.0, 9999999.0, (1.2, 1.8), (.5, .5)	2, 2
getMinimumErrorResult ()	SimplexFunction (-3x + x^2 – 5y -xy + y^2) 0, 0.0, 9999999.0, 1, 0.0, 9999999.0, (1.2, 1.6), (.1, .1)	2, 1

- **org.eclipse.stem.analysis.automaticexperiment.ElapsedTimeTest.java**

testAddincrement_long ()	currentTime	currentTime + 1day

- **org.eclipse.stem.analysis.automaticexperiment.STEMTimeTest.java**

testGetElapsedMilliseconds()	40	Valid, 3456000000

Resource Constraints

STEM currently sits in our TestAutomation directory structure at 5.8GB, far more memory than we can allow to end up in our final product. Much of the data on geographical locations can be discarded and allow us to cut down on our framework's memory usage. It will require that we configure dependencies differently and ensure they are locally present in memory before we compile and build, or else we will have to reconfigure Maven to build only with a fraction of the classes that are in STEM.

Dependencies

One of the main issues we are facing running our code in the terminal is the sheer number of dependencies required to actually run many parts of the code. Finding parts of the code that don't have an astronomical amount of dependencies proved interesting as some of the required packages to run code were needed for code that was extended several times. This made actually finding all of the required dependencies/packages the most frustrating part of trying to get some pieces of code to work. To avoid this issue we are going to have to be more selective in how we choose code to make into unit tests. The code we actually select must have limited dependencies to streamline that aspect of the test creation and allow us to test the code more thoroughly.