COSC346 Assignment 1 Matrix/Vector Library and test suite for Matrix/Vector Library

Ashley Midgley
ID: 7776496
Kahurangi Cassidy
ID: 6118539

Part 1 – Matrix/Vector Classes

The Matrix Vector Library consists of the following classes: Complex.swift, Fraction.swift, Interface.swift, Matrix.swift, MatrixTest.swift, TestMatrixVector.swift, Vector.swift, VectorTest.swift and main.swift.

The Complex class allows the creation of a complex number, which has a real and imaginary part. The class provides methods to be able to perform arithmetic on the complex numbers as well as using operators and scalars. The same goes for the Fraction class, which allows for the creation of fraction objects and the ability to perform arithmetic using fractions.

Interface defines all of the protocols we implemented to build the Matrix and Vector classes. There are 3 protocols for each of the structures: The basic protocols (Basic Matrix, BasicVector); which create the basic structure for the data type, the arithmetic protocols (MatrixArithmetic, VectorArithmetic) which define Matrix and Matrix operations, Matrix and scalar operations for the Matrix class as well as Vector and Vector operations, Vector and scalar operations for the Vector class and lastly the transformation protocols (MatrixToVector, VectorToMatrix) which provide methods that covert certain cases of each to the other class type.

We implemented the Matrix class first as we realized that if we could implement all the protocols for the Matrix, it would be a pretty straight forward process to implement the Vector class using inheritance, as a Vector could be created as a single row matrix. We used a 2D generic array to store the values of the Matrix. In the initializer we initialized the array and filled it with values that correspond the initialized generic types. Using this data structure to store the values of the Matrix made it easy for us to access specific values like in the subscript but also allowed for easy iteration through the structure when we needed to apply operations to all values like in the arithmetic functions. Using generics helped reduced code as we would have had to implement methods for 5 different data types otherwise and it allowed us to store data in a single generic 2D array.

Once we implemented the Matrix class, we created the Vector in the initializer by calling the parent initializer with the row argument as 1 and the column argument equaling the size argument for the vector. We could them implement pretty much all of the basic structure and arithmetic by overriding the Matrix parent methods and returning the vectorview of this Matrix value.

Part 2 – Test Suite

The TestMatrixVector class is split into two classes: MatrixTest and VectorTest.

MatrixTest and VectorTest are very similar in structure. In order to cover all cases, we included Int, Double, Fraction and Complex data type cases for each test. We chose not to include the float data type as we assume that if all the cases work for Int and Double data types, Float should work as well.

The tests covered all the methods that can be used on a Matrix or Vector. This includes all operations and values that are defined by the protocols for both classes.

In order to know that the results for the test are correct, the use of a string array is used. This is why the Matrix and Vector classes have a method for converting their contents to a string array. We used a String array as we could not compare generic types and we did not want to define 5 different methods to compare the output with the results.

We used a 2D Boolean array to store the results of the testing. The rows of the array correspond to each test and the columns correspond to the cases for each test. Each test was split into a function that returned a Boolean array of cases. True if the case passed and false if the case failed. This was then appended to the array in the runTests function. We decided to keep the array as dynamic and not set a fixed size so that it was easy for a user to add or remove tests at their discretion.

The TestMatrixVector class stores two global 2D Boolean arrays for the testing of the matrices and vectors. We split the Matrix and Vector testing into the separate classes to provide a layer of encapsulation between the user and the testing and also to break up the code making it easier for a user to understand.

In the main method we accessed the results in the 2D arrays and used the data to display to the user whether the cases had passed or failed for each test in a format easy for the user to understand. We also calculated how many tests had been performed for each class and how many of these tests had passed or failed.