Project Aim

This project aims to use a Mutation Testing tool on a personal project to learn about the concept of Mutation Testing. For our project we have used the PIT Mutation Testing tool, which is the most popular mutation testing tool for Java projects.

Source Code Used

We have used a linear algebra library written in Java as our source code. The code can be found on GitHub (link). The code contains two classes: Vector and Matrix, which are extensively used in Linear Algebra and even Machine Learning. If you are familiar with NumPy library in Python, then this library provides a similar interface for Java. A decent amount of functionality is provided in these two classes **(around 1800 lines of code**).

Unit Testing using JUnit

Firstly, for the linearalgebra library, we wrote unit tests to test each function. Unit testing was done with JUnit4, which is the most popular unit testing tool for Java projects. The test files can be found in src/test/java/linearalgebra folder. The Matrix class is tested by MatrixTest.java and Vector class is tested by VectorTest.java.

What is Mutation Testing?

Mutation testing is a software testing technique designed to assess the effectiveness of a test suite by introducing small, artificial changes or "mutations" to the source code. These mutations represent common programming errors such as typos, logical mistakes, or incorrect operators. The objective is to evaluate the ability of the test suite to detect and identify these mutations.

The mutated code is created by making slight modifications to the original codebase. The test suite is then executed against both the mutated and original code. If the test suite successfully identifies and flags the mutations, it indicates that the tests are robust and capable of detecting potential bugs. On the other hand, if a mutation goes undetected, it suggests a gap in the test coverage.

Mutation testing gives insights into the quality of the test suite and helps developers enhance the overall reliability of their software by refining test cases that may have missed certain types of defects.

A testcase can either weakly kill a mutant or strongly kill a mutant:

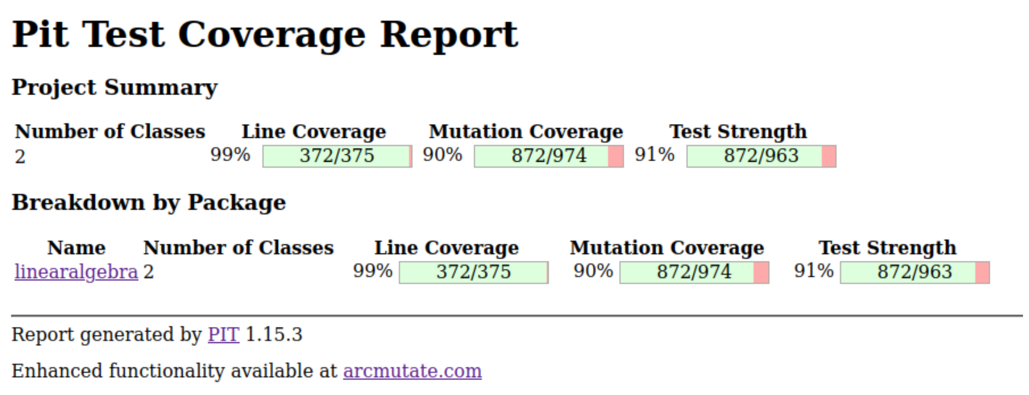
Weak killing: In week killing, the **memory state of the program** after the execution of the mutated statement is different from the memory state of the program when the statement was not mutated and executed. The output of the original and mutated version is the same in this case.

Strong killing: In strong killing, the **output of the program** on a test case when a statement was mutated and not mutated is different.

Our aim here is to make robust testcases that can strongly kill all the mutants.

Mutation Coverage Report

Overall, we have been able to achieve mutation coverage of 90% across all classes.



Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
2	99% 372/375	90% 872/974	91% 872/963

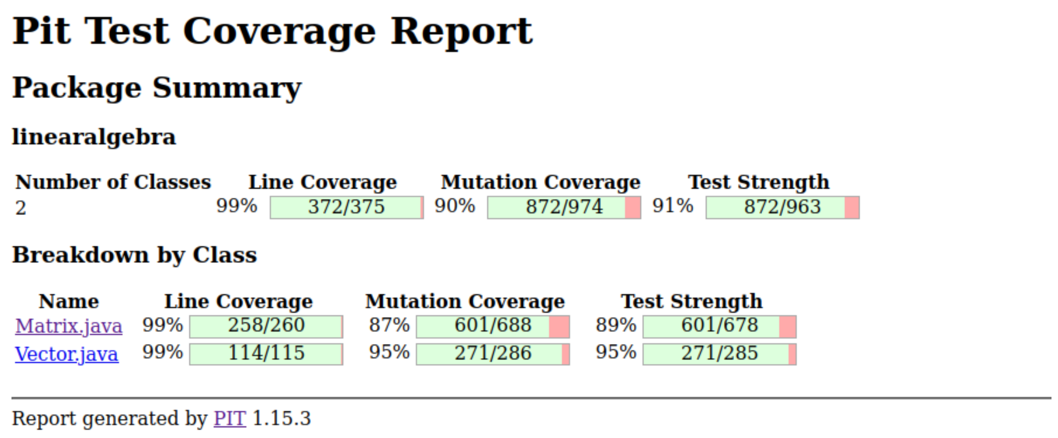
Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage	Test Strength
linearalgebra	2	99% 372/375	90% 872/974	91% 872/963

Report generated by PIT 1.15.3

Enhanced functionality available at arcmutate.com

For the Vector class, the mutation coverage is 95%, and for the Matrix class, mutation coverage is 87%.



Pit Test Coverage Report

Package Summary

linearalgebra

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
2	99% 372/375	90% 872/974	91% 872/963

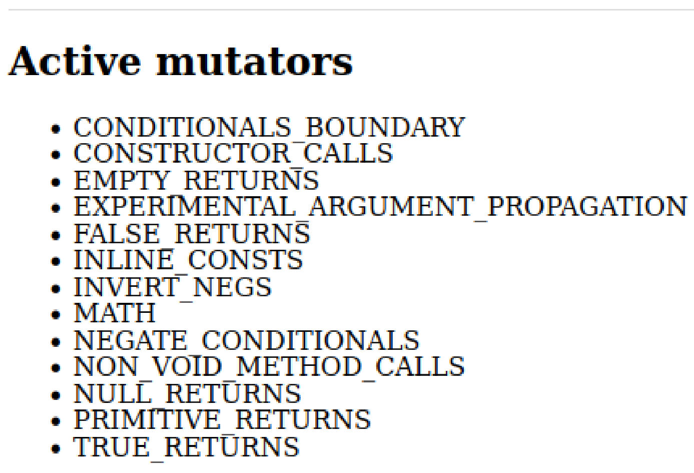
Breakdown by Class

Name	Line Coverage	Mutation Coverage	Test Strength
Matrix.java	99% 258/260	87% 601/688	89% 601/678
Vector.java	99% 114/115	95% 271/286	95% 271/285

Report generated by PIT 1.15.3

Mutation Operators Used

The following mutation operators were used by us:



Active mutators

- CONDITIONALS_BOUNDARY
- CONSTRUCTOR_CALLS
- EMPTY_RETURNS
- EXPERIMENTAL_ARGUMENT_PROPAGATION
- FALSE_RETURNS
- INLINE_CONSTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS
- NON_VOID_METHOD_CALLS
- NULL_RETURNS
- PRIMITIVE_RETURNS
- TRUE_RETURNS

In this, the following operators work at the unit level:

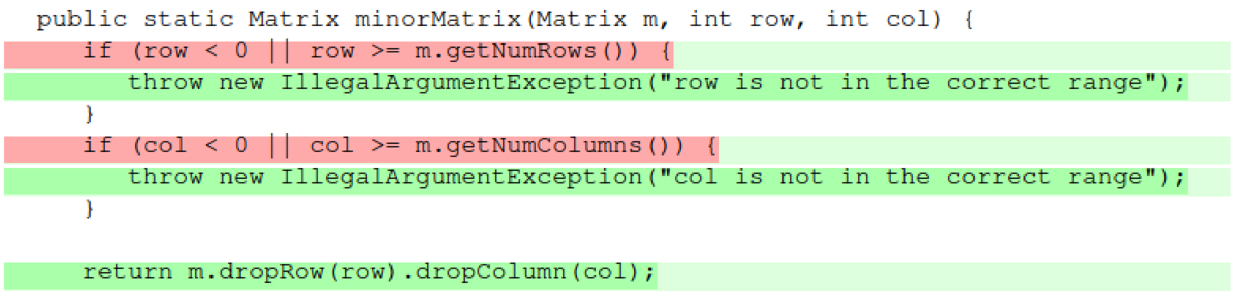
- CONDITIONALS_BOUNDARY
- INLINE_CONSTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS

And the following operators work at the integration level:

- CONSTRUCTOR_CALLS
- EMPTY_RETURNS
- EXPERIMENTAL_ARGUMENT_PROPAGATION
- FALSE_RETURNS
- NON_VOID_METHOD_CALLS
- NULL_RETURNS
- PRIMITIVE_RETURNS
- TRUE_RETURNS

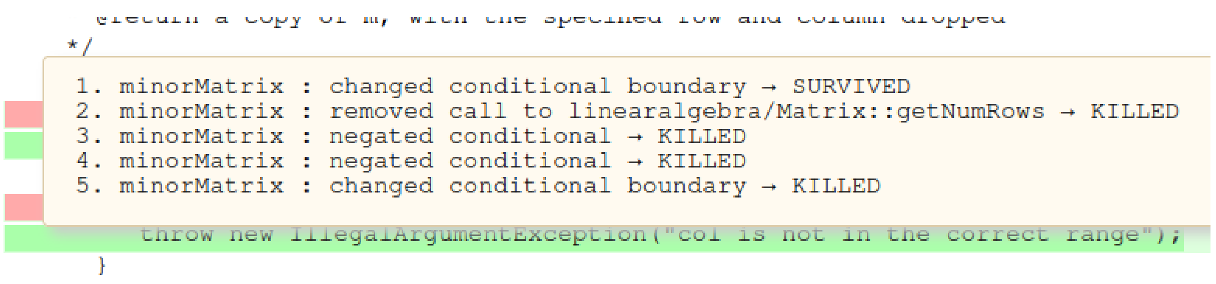
Bugs Found by Mutation Testing

Absence of boundary tests: Mutation testing is used to test our test cases. We found out that changing boundary conditions was not killing our mutant, and the reason for this was that there were no test cases in our test suite that tested the boundary conditions. To give a concrete example, there were no test cases in our test suite that tested minorMatrix() method where parameter row was equal to number of rows. This edge condition should have been in the test suite to begin with, and mutation testing helped us find out that we were missing this test case.



```
public static Matrix minorMatrix(Matrix m, int row, int col) {
    if (row < 0 || row >= m.getNumRows()) {
        throw new IllegalArgumentException("row is not in the correct range");
    }
    if (col < 0 || col >= m.getNumColumns()) {
        throw new IllegalArgumentException("col is not in the correct range");
    }

    return m.dropRow(row).dropColumn(col);
```



```
  @return a copy of m, with the specified row and column dropped
 */
  1. minorMatrix : changed conditional boundary → SURVIVED
  2. minorMatrix : removed call to linearalgebra/Matrix::getNumRows → KILLED
  3. minorMatrix : negated conditional → KILLED
  4. minorMatrix : negated conditional → KILLED
  5. minorMatrix : changed conditional boundary → KILLED
        throw new IllegalArgumentException("col is not in the correct range");
    }
```

References

1. To setup maven:
    a. https://maven.apache.org/guides/getting-started/
    b. https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html
2. To setup PIT in maven:
    a. https://pitest.org/quickstart/maven/
3. List of mutation operators in PIT:
    a. https://pitest.org/quickstart/mutators/
4. JUnit Asserts API reference:
    a. https://junit.org/junit4/javadoc/4.13/org/junit/Assert.html