# Test Approach and Strategy

## Development and Testing Approach

To ensure good code quality, extensive testing initiatives were undertaken throughout this assignment. As the group were new to software development, we researched best practice development methods and attempted to implement a number of them throughout the project. The methods we used were Test Driven Development, Pair Programming and Continuous Integration. We found that these methods helped us to consistently improve our code and ensure no regression occurred over time.

## Test Driven Development

### Unit Testing

As the team are new to software development, at the beginning of this assignment we were not aware of the test driven development principle. After much research however we eventually adopted it in this project. For unit testing, our team used NoseTests, an extension to the python unit test framework to make testing easier, as our unit testing framework and built test cases for our initial functions. From this point, as we wrote functions, test cases were written in parallel and this helped ensure they were developed correctly. Nosetests also allowed the full suite of test cases to be run at any point, so as functions changed over time, we were able to capture any mistakes made or rewrite the unit tests if the functions now took new inputs or created new outputs. As our test suite grew, we ensured to run the suite before pushing to our master repository on Github. This ensured quality code only was sent to this main shared repository.

A detailed list of the unit test cases and results can be found in Appendix A. A small number of functions were excluded from unit testing. This was mainly due to the fact that the functions prompted a number of responses from the command line and we found this very difficult to unit test. All of these functions however were tested during functional testing to ensure their quality.

### Functional Testing

In conjunction with unit testing, we also created a number of functional test scenarios to ensure the program functioned as expected for a number of given inputs and outputs. These test scenarios were run towards the end of the project and tested the program as a whole, ensuring the correct inputs were processed correctly and incorrect inputs were captured early and communicated to the user. A detailed list of the functional test cases and results can be found in Appendix B.

### Pair Programming

In conjunction with our test approach, we conducted a number of pair programming sessions throughout the development of our project. Pair programming is an agile software development technique where two programmers work together at the same workstation. One programmer works as the driver, writing the code, while the second programmer observers and navigates, giving instant feedback to the driver.

Our project team implement this technique by meeting up in an on campus syndicate room with one member putting his laptop screen on a projector for the whole group to see. During these sessions, specific goals were laid out and were worked on by the group where one member wrote code and the other members gave instantaneous feedback. This technique allowed us to solve a number of problems we encountered and sped up our development process considerably.

### Continuous Integration

Throughout the development of this program the project team utilised Github as our code repository. This allowed the group to practice continuous integration, a software engineering technique, where each team member merged their code with the master at least once a day. This reduced the effort for the team members in dealing with merge conflicts, allowed for early detection of code problems and gave an accurate picture to the team on what section of the code each member was working on.

# Personal Thoughts – Andy

I found this assignment very useful to both put into practice what we have learned in class as well as improve on my development and testing skills. Firstly, putting into practice the theory we learned in class, personally, is the best way to get a full understanding of the content. Learning to break down piece by piece how to create the Black Scholes matrix, and then to build the SOR algorithm to solve it made it very clear to me how and why this is done and the complexity involved. Secondly, before undertaking this assignment I had very limited version control and testing skills. I was aware of both concepts and why they were useful but had no knowledge as to how to use them myself. This assignment forced our team to develop these skills fast and I now feel a lot more comfortable with both which will be vital for my career ahead.

# Discuss mathematical issues

The main mathematical issues we encountered with the SOR algorithm were the strict input matrix parameters that are required for the algorithm to successfully converge. To ensure convergence, the input matrix had to diagonally dominant or not diagonally dominant but with eigenvalues with absolute values less than 1. In both cases, no zeros could be present on the matrix diagonal also. These strict parameters restricted the number of matrices we could solve using our program.

To capture these parameters, we perform a number of input checks on the matrix before it is run through the algorithm. For small matrices, this is a very quick way to ensure the matrix is correctly conditioned for the algorithm. As matrices grow larger however, and with initial conditions unknown, while these checks will still capture ill conditioned matrices, the computational time will grow and may lead to wasted time on matrices that our program cannot solve.

# Would scaling be helpful?

Scaling?

# Appendix A - Unit Test Cases

Calculate\_residual.pyresidual()

|  |  |
| --- | --- |
| **Input** | val, col, rowstart, b(all originating from matrix in file nas\_Sor.in) and randomly generated x vector |
| **Expected Output** | 1.1532562570790883e-07 |
| **Actual Output** | 1.1532562570790883e-07 |

### convert\_to\_csr.py

#### con\_to\_csr()

|  |  |
| --- | --- |
| **Input** | Four separate values each for vector, matrix size and rowStart:  vector1 = np.array([13., 0., 0., 4.]) vector2 = np.array([4., 11., 0., 4.]) vector3 = np.array([7., 8., 20., 4.]) vector4 = np.array([1., 0., 1., 14.])  matrix\_size = 4  rowStart1 = 0 rowStart2 = 2 rowStart3 = 5 rowStart4 = 9 |
| **Expected Output** | Val = np.array([13]), np.array([4]), Col =[0, 3], RowStart = 2 Val = np.array([4]), np.array([11]), np.array([4]), Col = [0, 1, 3], RowStart = 5 Val = np.array([7]), np.array([8]), np.array([20]), np.array([4]), Col = [0, 1, 2, 3], RowStart = 9 Val = np.array([1]), np.array([1]), np.array([14]), Col = [0, 2, 3], RowStart = 12 |
| **Actual Output** | [np.array([13]), np.array([4])], [0, 3], 2)) [np.array([4]), np.array([11]), np.array([4])], [0, 1, 3], 5)) [np.array([7]), np.array([8]), np.array([20]), np.array([4])], [0, 1, 2, 3], 9)) [np.array([1]), np.array([1]), np.array([14])], [0, 2, 3], 12)) |

### create\_BS\_b.py

#### create\_BS\_b()

|  |  |
| --- | --- |
| **Input** | N = 20 X = 12 S\_max = 20 h = (S\_max/N) k = 5 sigma = 0.02 r = 0.3 |
| **Expected Output** | np.array([11., 10., 9., 8., 7., 6., 5., 4., 3., 2., 0., 0., 0., 0., 0., 0., 0., 0., 0.] |
| **Actual Output** | np.array([11., 10., 9., 8., 7., 6., 5., 4., 3., 2., 0., 0., 0., 0., 0., 0., 0., 0., 0.] |

### create\_BS\_matrix.py

#### create\_BS\_matrix()

|  |  |
| --- | --- |
| **Input** | Test Case 1: M = 2, k =5, r = 0.02, sigma = 0.3  Test Case 2: M = -2, k =5, r = 0.02, sigma = 0.3  Test Case 3: M = 5, k =5, r = 0.02, sigma = 0.3  Test Case 4: M = 5, k =’string’, r = 0.02, sigma = 0.3 |
| **Expected Output** | Test Case 1: SystemError("There must be at least 3 intervals")  Test Case 2: M = 2, k =5, r = 0.02, sigma = 0.3  Test Case 3:  Val = np.array([1.55, -0.275, -0.8, 2.9, -1., -1.875, 5.15, -2.175, -3.4, 8.3]  Col = np.array([0, 1, 0, 1, 2, 1, 2, 3, 2, 3]  rowStart = np.array([0, 2, 5, 8, 10])  Test Case 4: TypeError |
| **Actual Output** | Test Case 1: SystemError("There must be at least 3 intervals")  Test Case 2: M = 2, k =5, r = 0.02, sigma = 0.3  Test Case 3:  Val = np.array([1.55, -0.275, -0.8, 2.9, -1., -1.875, 5.15, -2.175, -3.4, 8.3]  Col = np.array([0, 1, 0, 1, 2, 1, 2, 3, 2, 3]  rowStart = np.array([0, 2, 5, 8, 10])  Test Case 4: TypeError |

### get\_bsm\_inputs.py

As this function prompts to the command line it was not unit tested. It was tested during functional testing.

### output\_bsm.py

As this function writes the Black Scholes matrix results to an output file it was not unit tested. It was fully tested during functional testing.

### get\_filename.py

#### check\_CM\_args()

As this function prompts to the command line it was not unit tested. It was tested during functional testing.

#### check\_file\_exists()

|  |  |
| --- | --- |
| **Input** | a = 'nas\_Sor.in' b = 'nas\_Sor' c = '/nas\_Sor10.in' |
| **Expected Output** | True, False, False |
| **Actual Output** | True, False, False |

#### con\_filename()

|  |  |
| --- | --- |
| **Input** | a = 'sample\_inputs/nas\_Sor2.in'  b = 'Input\_descriptions.txt'  c = 'san\_Ros.ni'  d = 'sample\_inputs/nas\_Sor3.in'  e = 'input.txt' |
| **Expected Output** | "sample\_inputs/nas\_Sor2.in", True  "Input\_descriptions.txt", True  “san\_Ros.ni", False  "sample\_inputs/nas\_Sor3.in", True  "input.txt", False |
| **Actual Output** | "sample\_inputs/nas\_Sor2.in", True  "Input\_descriptions.txt", True  “san\_Ros.ni", False  "sample\_inputs/nas\_Sor3.in", True  "input.txt", False |

### import\_mtx.py

#### import\_mtx()

|  |  |
| --- | --- |
| **Input** | Test Case 1: 'sample\_inputs/sample\_mtx.mtx', 'nas\_Sor.out'  Test Case 2: 'sample\_inputs/s3dkt3m2.mtx', 'nas\_Sor.out' |
| **Expected Output** | Test Case 1:  val = np.array([1,2,3,4,5,6,7,8,9])  col = np.array([0,1,2,0,1,2,0,1,2])  rowStart = np.array([0,3,6,9])  Test Case 2:  SystemExit("Unable to import the .mtx file. Please check it and try again") |
| **Actual Output** | Test Case 1:  val = np.array([1,2,3,4,5,6,7,8,9])  col = np.array([0,1,2,0,1,2,0,1,2])  rowStart = np.array([0,3,6,9])  Test Case 2:  SystemExit("Unable to import the .mtx file. Please check it and try again") |

#### get\_mtx\_b()

As this function prompts to the command line it was not unit tested. It was tested during functional testing.

### input\_checks.py

#### csr\_input\_checks()

|  |  |
| --- | --- |
| **Input** | val1 = np.array([13.0, 4.0, 4.0, 11.0, 4.0, 7.0, 8.0, 20.0, 4.0, 1.0, 1.0, 14.0])  val2 = np.array([13.0, 4.0, 4.0, 11.0, 4.0, 7.0, 8.0, 4.0, 1.0, 1.0, 14.0])  val3 = np.array([3.0, 4.0, 4.0, 11.0, 4.0, 7.0, 8.0, 20.0, 4.0, 1.0, 1.0, 14.0, 15.0])  col1 = np.array([0, 3, 0, 1, 3, 0, 1, 2, 3, 0, 2, 3])  col2 = np.array([0, 3, 0, 1, 3, 0, 1, 3, 0, 2, 3, 4])  col3 = np.array([0, 3, 0, 3.5, 3, 0, 1, 2, 3, 0, 2, 3])  rowStart1 = np.array([0, 2, 5, 9, 12])  rowStart2 = np.array([0, 2, 5, 8, 10])  rowStart3 = np.array([1, 2, 5, 9, 12])  rowStart4 = np.array([0, 2.5, 5, 9, 12])  rowStart5 = np.array([0, 2, 5, 9, 13])  b1 = np.array([2, 3, 4, 5])  b2 = np.array([2, 3, 4, 5, 6])  Test Case 1: val1, col1, rowStart1, b1  Test Case 2: val1, col1, rowStart1, b2  Test Case 3: val1, col2, rowStart1, b1  Test Case 4: val1, col1, rowStart2, b1  Test Case 5: val1, col1, rowStart3, b1  Test Case 6: val1, col1, rowStart4, b1  Test Case 7: val3, col1, rowStart5, b1  Test Case 8: val1, col3, rowStart1, b1 |
| **Expected Output** | Test Case 1: **[**]  Test Case 2: Number of columns in matrix is not the same as the "  "number of rows in Vector b"  Test Case 3: “Uneven number of rows and columns"  Test Case 4: “Last entry of RowStart vector is equivalent to the "  "nth entry of val + 1"  Test Case 5: “First entry of RowStart vector is not 0"  Test Case 6: “RowStart vector contains non-integer entries"  Test Case 7: “Value and column vectors do not have the same number "  "of entries"  Test Case 8: "Column vector contains non-integer entries", "Uneven number of rows and columns" |
| **Actual Output** | Test Case 1: **[**]  Test Case 2: Number of columns in matrix is not the same as the "  "number of rows in Vector b"  Test Case 3: “Uneven number of rows and columns"  Test Case 4: “Last entry of RowStart vector is equivalent to the "  "nth entry of val + 1"  Test Case 5: “First entry of RowStart vector is not 0"  Test Case 6: “RowStart vector contains non-integer entries"  Test Case 7: “Value and column vectors do not have the same number "  "of entries"  Test Case 8: "Column vector contains non-integer entries", "Uneven number of rows and columns" |

### Condition.py

#### Condition()

|  |  |
| --- | --- |
| **Input** | row = np.array([0, 0, 1, 2, 2, 2]) col = np.array([0, 2, 1, 0, 1, 2]) data = np.array([1, 2, 3, 4, 5, 6]) row = row.astype(float) col = col.astype(float) data = data.astype(float) n = 3 |
| **Expected Output** | 26.4622102617 |
| **Actual Output** | 26.4622102617 |

### Optimise\_w.py

#### Op\_w()

|  |  |
| --- | --- |
| **Input** | list = [0.6, 0.7, 0.79, 0.7, 0.8]  w = 1.2 |
| **Expected Output** | 1.2 |
| **Actual Output** | 1.2 |

### raw\_input\_checks.py

#### read\_raw\_inputs()

|  |  |
| --- | --- |
| **Input** | Test Case 1: 'sample\_inputs/nas\_Sor2.in', 'nas\_Sor.out'  Test Case 2: 'sample\_inputs/nas\_Sor3.in', 'nas\_Sor.out' |
| **Expected Output** | Test Case 1: False  Test Case 2: SystemExit("First line contains non-digit entries. Please fix and try "  "again") |
| **Actual Output** | Test Case 1: False  Test Case 2: SystemExit("First line contains non-digit entries. Please fix and try "  "again") |

### read\_input.py

#### read\_inputs()

|  |  |
| --- | --- |
| **Input** | 'sample\_inputs/nas\_Sor.in', 'nas\_Sor.out' |
| **Expected Output** | val = np.array([12, 1, 4, 11, 3, 7, 8, 16, 1, 3])  col = np.array([0, 3, 0, 1, 3, 0, 1, 2, 0, 3])  rowStart = np.array([0, 2, 5, 8, 10])  vector\_b = np.array([1, 2, 3, 4]) |
| **Actual Output** | val = np.array([12, 1, 4, 11, 3, 7, 8, 16, 1, 3])  col = np.array([0, 3, 0, 1, 3, 0, 1, 2, 0, 3])  rowStart = np.array([0, 2, 5, 8, 10])  vector\_b = np.array([1, 2, 3, 4]) |

### solve\_sor.py

#### sor()

|  |  |
| --- | --- |
| **Input** | val = [13.0, 4.0, 4.0, 11.0, 4.0, 7.0, 8.0, 20.0, 4.0, 1.0, 1.0, 14.0]  col = np.array([0, 3, 0, 1, 3, 0, 1, 2, 3, 0, 2, 3])  rowStart = np.array([0, 2, 5, 9, 12])  b = np.array([1, 2, 3, 4])  n = 4  maxits = 100  w = 1.2  x = np.array([-0.00979992, 0.08289098, 0.06390363, 0.28184973])  e = 2.22044604925e-16  tol = 1e-10 |
| **Expected Output** | np.array([-0.00979992, 0.08289098, 0.06390363, 0.28184973])  np.array([5])  np.array([1.040731881863575e-10]) |
| **Actual Output** | np.array([-0.00979992, 0.08289098, 0.06390363, 0.28184973])  np.array([5])  np.array([1.040731881863575e-10]) |

### Value\_checks.py

#### value\_tests():

|  |  |
| --- | --- |
| **Input** | val1 = np.array([13.0, 4.0, 4.0, 11.0, 4.0, 7.0, 8.0, 4.0, 1.0, 1.0, 14.0])  col1 = np.array([0, 3, 0, 1, 3, 0, 1, 3, 0, 2, 3])  rowStart1 = np.array([0, 2, 5, 8, 11]) |
| **Expected Output** | SystemExit("There are zeros on the diagonal") |
| **Actual Output** | SystemExit("There are zeros on the diagonal") |

### vector\_norm.py

#### vectornorm()

|  |  |
| --- | --- |
| **Input** | Test Case 1: np.array([1, 1, 1, 1, 1])  Test Case 2: np.array([-1, 3, 5, -7]) |
| **Expected Output** | Test Case 1: sqrt(5)  Test Case 2: sqrt(84) |
| **Actual Output** | Test Case 1: sqrt(5)  Test Case 2: sqrt(84) |

### write\_output.py

#### output\_text\_file()

As this function writes the SOR algorithm results to an output file it was not unit tested. It was fully tested during functional testing.

# Appendix B - Functional Test Cases

## Case 1 - A small diagonally dominant matrix A

|  |  |
| --- | --- |
| Input file | Small\_diag\_dom1.in - contains a diagonally dominant 4 x 4 matrix with no 0s on the diagonal |
| Output file | Small\_diag\_dom1.out |
| Expected Result | Small\_diag\_dom1.out file with the following results:   * Stopping reason: x sequence convergence * Max Number of Iterations: 100 * Number of Iterations: 36 * Machine Epsilon: 2.22044604925e-16 * X Sequence Tolerance: 1e-10 * Residual Sequence Tolerance: 1.0137282758487921e-10 * X vector: -1.39963413362e-11 0.5 0.500000000011 0.499999999993 |
| Actual Output | Small\_diag\_dom1.out file with details above created. Test Successful. |

## Case 2 - A small matrix A having a 0 on the diagonal

|  |  |
| --- | --- |
| Input file | nas\_Sor\_zero\_on\_diag.in - contains a 3 x 3 matrix with one 0 on the diagonal |
| Output file | nas\_Sor\_zero\_on\_diag.out |
| Expected Result | Error printed to the terminal: There are zeros on the diagonal  nas\_Sor\_zero\_on\_diag.out file with the following results:   * Stopping reason: Zero on diagonal * Max Number of Iterations: 100 * Number of Iterations: 0 * Machine Epsilon: 2.22044604925e-16 * X Sequence Tolerance: 1e-10 * Residual Sequence Tolerance: 0 |
| Actual Output | Error printed successfully.  nas\_Sor\_zero\_on\_diag.out file with details above created. Test Successful. |

## Case 3 - A small matrix A (such that A has no 0 on the main diagonal, A is not diagonally dominant and all of the eigenvalues of C have absolute value < 1)

|  |  |
| --- | --- |
| Input file | not\_diag\_dom\_evls\_less\_than.in - contains a non-diagonally dominant 2 x 2 matrix with all eigenvalues with absolute value less than 1 |
| Output file | not\_diag\_dom\_evls\_less\_than.out |
| Expected Result | not\_diag\_dom\_evls\_less\_than.out file with the following results:   * Stopping reason: X Sequence Convergence * Max Number of Iterations: 100 * Number of Iterations: 23 * Machine Epsilon: 2.22044604925e-16 * X Sequence Tolerance: 1e-10 * Residual Sequence Tolerance: 4.589902317696708e-12 |
| Actual Output | not\_diag\_dom\_evls\_less\_than.out file with details above created. Test Successful. |

## Case 4 - A small matrix A (such that A has no 0 on the main diagonal, A is not diagonally dominant and all of eigenvalues2 have absolute value < 1)

|  |  |
| --- | --- |
| Input file | not\_diag\_dom\_evls\_greater\_than.in - contains a non-diagonally dominant 3 x 3 matrix with all eigenvalues with absolute value greater than 1 |
| Output file | not\_diag\_dom\_evls\_greater\_than4.out |
| Expected Result | not\_diag\_dom\_evls\_greater\_than4.out file with the following results:   * Stopping reason: Divergence * Max Number of Iterations: 100 * Number of Iterations: 2 * Machine Epsilon: 2.22044604925e-16 * X Sequence Tolerance: 1e-10 * Residual Sequence Tolerance: |
| Actual Output | not\_diag\_dom\_evls\_greater\_than.out file with details above created. Test Successful. |

## Case 5 - A large sparse diagonally dominant matrix.

|  |  |
| --- | --- |
| Input file | Large\_matrix.in – contains a diagonally dominant 1000 x 1000 matrix |
| Output file | Large\_matrix.out |
| Expected Result | Large\_matrix.out file with the following results:   * Stopping reason: X Sequence Convergence * Max Number of Iterations: 100 * Number of Iterations: 33 * Machine Epsilon: 2.22044604925e-16 * X Sequence Tolerance: 1e-10 * Residual Sequence Tolerance: 1.7386968486732083e-08 * X vector with 1000 values |
| Actual Output | Large\_matrix.out file with details above created. Test Successful. |

## Case 6 - A large matrix with file format .mtx.

|  |  |
| --- | --- |
| Input file | Test Case 1: Sample\_mtx1.mtx – contains a large matrix in .mtx file format (94.8MB)  Test Case 2: Sample\_mtx1.mtx – contains a large matrix in .mtx file format (94.8MB) |
| Output file | Nas\_Sor\_mtx1.out  Nas\_Sor\_mtx1\_maxits200.out |
| Expected Result | Test Case 1: Nas\_Sor\_mtx1.out file with the following results:   * Stopping reason: Max Iterations Reached * Max Number of Iterations: 100 * Number of Iterations: 100 * Machine Epsilon: 2.22044604925e-16 * X Sequence Tolerance: 1e-10 * Residual Sequence Tolerance: 11091314.622381652 * X vector values   Test Case 2: Nas\_Sor\_mtx1.out file with the following results:   * Stopping reason: Max Iterations Reached * Max Number of Iterations: 200 * Number of Iterations: 200 * Machine Epsilon: 2.22044604925e-16 * X Sequence Tolerance: 1e-10 * Residual Sequence Tolerance: 8389635.509502143 * X vector values |
| Actual Output | Nas\_Sor\_mtx1.out file with details above created. Test Successful. |

## Case 7 – bsm.py default input tests

|  |  |
| --- | --- |
| Input file | Python bsm.py prompted at the command line  When prompted, default values are chosen:   * Stock Price Today: $40.00 * Strike Price: $42.00 * Days to Maturity: 90 * Risk Free Rate: 2% * Volatility: 30% |
| Output file | bsm\_solution.out |
| Expected Result | bsm\_solution.out file with the following results:   * Option Value: $0.51 * Stock Price Today: $40.00 * Strike Price: $42.00 * Days to Maturity: 90 * Risk Free Rate: 2% * Volatility: 30% |
| Actual Output | Bsm\_solution.out file with details above created. Test Successful. |

## Case 8 – bsm.py input tests

|  |  |
| --- | --- |
| Input file | Python bsm.py prompted at the command line  When prompted, default values are chosen:   * Stock Price Today: $54.00 * Strike Price: $60.00 * Days to Maturity: 60 * Risk Free Rate: 25% * Volatility: 15% |
| Output file | bsm\_solution.out |
| Expected Result | bsm\_solution.out file with the following results:   * Option Value: $0.24 * Stock Price Today: $54.00 * Strike Price: $60.00 * Days to Maturity: 60 * Risk Free Rate: 25% * Volatility: 15% |
| Actual Output | Bsm\_solution.out file with details above created. Test Successful. |