# Test Approach and Strategy

## Development and Testing Approach

To ensure good code quality, extensive testing initiatives were undertaken throughout this assignment. We used a number of best practice developer methods such as Test Driven Development, Pair Programming and Continuous Integration to consistently improve our code and ensure no regression occurred over time.

## Test Driven Development

### Unit Testing

A test driven approach was taken early on in this project. Our team used NoseTests, an extension to the python unit test framework to make testing easier, as our unit testing framework and built a number of test cases for our initial functions. We chose NoseTests as according to best practice unit testing, test cases must be able to run both independently and as part of a suite of test cases and NoseTests allows us to follow these practices. NoseTests also produces a report on all tests executed after completion, even if all tests passed which is another useful feature we felt was required.

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 can be found in Appendix XX.

### Functional Testing

In conjunction with unit testing, we also created a number of functional test scenarios to ensure the program was functioning as expected for a number of given inputs. 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 can be found in Appendix XX.

### Pair Programming

In conjunction with our unit test framework, 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.

# Appendix - 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** |  |
| **Expected Output** |  |
| **Actual Output** |  |

### 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 - Functional Test Cases**

**Case 1 - *A small diagonally dominant matrix A***

Input file - nas\_Sor1.in

Output filename - nas\_Sor.out

Input file contains a diagonally dominant 4 x 4 matrix with no 0s on the diagonal

Expected Result - nas\_Sor.out file with the following results:

* Stopping reason: x sequence convergence
* Max Number of Iterations: 100
* Number of Iterations: 31
* Machine Epsilon: 2.22044604925e-16
* X Sequence Tolerance: 1e-10
* Residual Sequence Tolerance: 1.8884963934515226e-10

Actual Results - Output file named nas\_Sor.out correctly created with the data outlined above. Test case successful.

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

Input file - nas\_Sor2.in

Output filename - nas\_Sor2.out

Expected result -

“The following errors were identified:  
  
-  There are zeros on the diagonal  
  
Please correct these errors and restart the program”

Actual results - Error message received on the command line stating “ The following errors were identified: - There are zeros on the diagonal Please correct these errors and restart the program” Test case successful.

**Case 3 - *A small matrix A such that***

***◦ A has no 0 on the main diagonal***

***◦ A is not diagonally dominant***

***◦ all of the eigenvalues of C have absolute value < 1.***

Input file - nas\_Sor3.in

Output file - nas\_Sor3.out

Expected result -

**Case 4 - *A small matrix A such that   
 ◦ A has no 0 on the main diagonal   
 ◦ A is not diagonally dominant  
 ◦ all of C tC’s eigenvalues2 have absolute value < 1.***

Input file -

Output file -

Expected result -

• A large (say n = 1000–10000) sparse diagonally dominant matrix.