ECE 4960
Programming Assignment – 2
Modular Testing in Sparse Matrix Solvers

BY:
Vishisht Tiwari (vmt28)
Deepak Agarwal (da475)

Goal: The goal of the assignment is to understand and implement different matrix solvers for both full and sparse matrices and implement as comprehensive testing methods as possible.

Code Structure:
The parent directory contains following files:

- Code Files:

1. Assignment2_Header.h: It is a global header file included in all code files, containing globally defined functions and other classes declataions.
   Class Global_Functions: Global class defined in the header file containing functions to create dynamically allocated matrix from an ar ray, create dynamic arrays, create full matrix from sparse etc. Exception handling check is added in all functions under a macro EXCEPTION_ HANDLING
   Class Matrix_Operations: Class declared having matrix operations (permute, scaling and product) for full and sparse matrices
   Class Jacobi: Class implementing iterative solver using Jacobi method on full and sparse matrices
   Class LoadMat1: Class implementing functions to load csv files and generate sparse matrix from it
2. Assignment2_Main.cpp: Main file loading the sparse matrix from the .csv files, performing Jacobi iterative solver and calculating the norm.
3. Matrix_Operations.cpp: Implements the class Matrix_Operations containing the three matrix operations (permute, scaling and product) for sparse and full matrice s, with exception handling under EXCEPTION_HANDLING macro.
4. Jacobi.cpp: Class implementing functions for iterative solver for full-matrix and row-compressed formats.
5. Load_Mat1.cpp: Implements functions for loading the .csv files in different arrays.

- Report documents:

1. Report.pdf and Report.docx: Documentation of code design and testing strategies
2. Readme.txt: File describing code structure, compilation command and testing platforms
3. Output_MacOS.txt: Log output on MacOS
4. Output_ECE_linux.txt: Log output on ECE linux machines

- Data:

1. CSV files for mat1

Testing Strategy:

The testing strategy for this assignment can be described as multi-phase incremental methodology where each operation was verified to be working properly, before proceeding to the next operation in the process. The process of iterative solver problem can be put into following multiple phases where each next phase depends on the correctness of all previous phases.

1. Get the matrix, either in full or row-compressed format
2. Convert the matrix in row-compressed format if it is in full format
3. Implement helper methods required for Jacobi implementation
4. Apply Jacobi method on it

Since it is known that each phase will function properly only when the previous step is tested and is assured to not have any bugs, each phase is associated with a norm calculation function. At each phase, a norm value is calculated and compared with an emperically set tolerance value. If the error goes out of bound, the execution doesn't proceed further. To achieve that, coherent testing methods were written as generically as possible so that any class function can use them whenever it wants to.

1. **Conversion testing:** The conversion of full matrix to row-compressed format is an important step and verified comprehensively with various types of matrices. Functions were tested using different ranks and sparsity of the matrices and comparing the resultant row-compressed with the original using norm function.
2. **Operations testing:** The three operations: permute, scaling and multiply are the most important functions for any matrix solver and verification of their functionality was of utmost importance. Again, norm calculation was done for each of the three operations between row-compressed and full matrices and was compared with a configurable tolerance value
3. **Jacobi testing:** Once the helper functions were verified to be working correctly, Jacobi method was tested by calculating the norm between the computed solution X and value of A*b.

Silent Features: The above methods were also supplemented with few more testing features

1. Exception Handling: A macro called EXCEPTION_HANDLING was added which can be enabled to explicitly check if any exception occurred
2. Confiurable Tolerance: A variable 'tolerance' was defined as extern in which can be set to any value by any class. This is the value against which different norm values are compared, and can be set according to the need of testing accuracy.