GMRES Project

Applied Linear Algebra

Duc Phan

March 12, 2019

Abstract

This project objective is to create a set of tools to that mainly help with operating and manipulating on sparse matrices in compressed sparse row (CSR) format. The library I chose to implement cover most of the basic operations, which will be discussed later in this paper, for *Vectors*, *Dense Matrices* and *CSR Matrices*. The focus of this project and paper is the Generalized Minimal Residual (GMRES) method/algorithm. This paper is an overview of the tool set mentioned above, the implementation of GMRES and some statistics produced by GMRES.

1. Introduction

This project is implemented mostly in Java for the *Vector*, *Dense/CSR Matrix* operations and NodeJS for data and stats graphing.

Some libraries I used in my program are:

- Java:

```
//For parsing String into json
import com.google.gson.Gson;
//IO handler
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintStream;
import java.util.Scanner;
//Number handler - This will support number with a lot of decimals
import java.math.BigDecimal;
import java.math.MathContext;
import java.math.RoundingMode;
//Database
import java.util.LinkedList;
```

- NodeJS:

```
require('fs');         //For file reading/writing
require('path');       //For file searching
require('plotly');     //For graphing
```

The code and some sample results can be found at my GitHub repository:

https://github.com/ptmdmusique/Vector_Dense-CSR-Matrix_Operations

2. Acceptable Inputs

The program will be able to take and parse input of string type in multiple format:

- A string of numbers separated by single space for different columns and newline character ('\n') for different rows:

A vector or matrix can be constructed directly using:

1. the object's constructor *Vector(String input)* or *Matrix(String input)* or *CSRMatrix(String input)*.
2. the *TakeInput(String input)* method.

Example:

```
//This will create a vector with entries 1, 2, 3, 4, 5
Vector myVector = new Vector("1 2 3 4 5");

/* This will create a 2x4 matrix:
   1 2 3 4
   5 6 7 8
    (extra spaces after the last and before the first numbers of each row
can lead to bugs!)
*/
Matrix myMatrix = new Matrix("1 2 3 4\n5 6 7 8");
CSRMatrix myCSRMatrix = new CSRMatrix("1 2 3 4\n5 6 7 8");
```

- Matrix Market Exchange Format:

Before constructing a vector/matrix, a file from Matrix Market Exchange must go through a parsing process. Two support methods for this purpose (can be found in *Main*) are:

1. *static String ReadCSRFromFile(String fileName)* or
2. *static String ReadVectorFromFile(String fileName)*

The functions will then parse the input from file into a String with which we can constructed a Vector, Matrix or CSRMatrix using the method mentioned above.

The default folder to store the input file is: *./bigMatricess*

The path can be changed via *static String inputPath* declared in Main.

3. Data structures:

1) **General**:

All data will be stored using *BigDecimal* data type with the default precision of 50. I chose to use this data structure since a lot of files from Matrix Market contain really small number, which require high precision data structure to store and manipulate.

BigDecimal library from Java is an immutable and arbitrary-precision signed decimal numbers, meaning it can hold, theoretically, infinite number of decimals. However, it is a huge trade-off between performance and storage.

2) **Vector**:

Vector class will store its data in a *linear array* of *BigDecimal.* Vector class are both row and column major. With the correct context, the program will be able to detect automatically whether the specified Vector is a column vector or a row vector.

For example: *myVector.InnerProduct(Vector parm)* will return the inner product of myVector and parm correctly without getting transpose of any of them.

3) **Dense Matrix**:

Matrix class will store its data in a *linear array* of row *Vector*.

4) **CSR Matrix**:

CSRMatrix class will store its data in a standard CSR from, which includes a *linear array* of *data,* a *linear array* of row info and a integer colSize which stores the number of column of the matrix.

The *data* array is an array of *Data* which includes the *BigDecimal data* (value of the entry) and *int col* (column of the entry).

4. Operations:

This toolkit provides functions for several basic vector and matrix operations:

1) Vectors:

- Vector *Add(BigDecimal parm)*: return a vector with its entries equal the sum of the old data and the parameter/input

```
//Example:
myVector.Add(BigDecimal.valueOf(5));
//will add all entries of myVector with 5
```

- *Vector Add(BigDecimal parm)*: add the original vector and the parm vector up and return the result.

```
//Example:
myVector.Add(anotherVector);
//will add myVector with another Vector
```

- *Vector Scale(BigDecimal parm):* scale all entries of the current vector with parm and return the result.

```
//Example:
myVector.Scale(BigDecimal.valueOf(-1));
//will multiply all entries of myVector with -1
```

- *BigDecimal InnerProduct(Vector parm)*: calculate inner product of the two vector.

```
//Example:
myVector.InnerProduct(anotherVector);
//will calculate the inner product of myVector and anotherVector
```

- *Matrix Multiply(Vector parm)*: multiply 2 vector with the original vector as column vector on the left and parm as the row vector on the right and return the result matrix.

```
//Example:
myVector.Multiply(anotherVector);
//will multiply myVector with anotherVector
```

- *Vector Normalize()*: return a new vector which is the normalized version of the current vector.

```
//Example:
myVector.Normalize();
//will return a unit vector based on myVector
```

- *void Copy(Vector parm)*: replace all entries with the data from parm.

```
Example:
myVector.Copy(anotherVector);
//will copy data from anotherVector to myVector
```

- *boolean Equal(Vector parm)*: check if two vectors are equal.

```
//Example:
myVector.Equal(anotherVector);
//will return true if anotherVector has the same entries as myVector
```

- *void TakeInput(String input):* process an input string and convert it to a corresponding vector

```
//Example:
myVector.TakeInput("1 2 3 4 5");
//will produce a vector with entries: 1, 2, 3, 4, 5
```

- *void Print():* print all the data of the current vector

```
//Example:
myVector.Print();
//will print myVector as a column vector
```

2) Matrix:

- Matrix *Add(Matrix parm)*: return a matrix with its entries equal the sum of the old matrix and the parameter/input

```
//Example:
myMatrix.Add(newMatrix);
//will add all entries of the corresponding entries of newMatrix
```

- *Matrix Multiply(Matrix parm)*: multiply 2 matrix and return the result. I used the naïve $O(n^3)$ algorithm.

```
//Example:
myMatrix.Multiply(anotherMatrix);
//will multiply myMatrix with anotherMatrix
```

- *Matrix Multiply(Vector parm):* multiply the current matrix with a vector on the right side.

```
//Example:
myMatrix.Multiply(aVector);
//will multiply myMatrix with aVector (myMatrix * aVector)
```

- *Matrix AugmentVectorAtEnd(Vector parm)*: augment the specified Vector at the end of the matrix.

```
//Example:
myMatrix.AugmentVectorAtEnd(anotherVector);
//will augment anotherVector (column vector) at the end of myMatrix
```

- *LinkedList<Matrix> LUFactorization():* return the L and U matrix from the result of LU Factorization. This method will be able to rotate the rows of the current matrix until it produces the a LU factor. However, if after trying all permutation and there is no result, the method will return null.
  This method is not fully tested, especially with big or complicated matrices.

```
//Example:
myMatrix.LUFactorization();
//will return the LU factorization of myMatrix
```

- *LinkedList<Matrix> QRFactorization()*: return the Q and R matrix from the result of QR Factorization. This method used the principle of Gram-Schmidt process to produce the correct QR factorization. This method supports any matrix.

```
//Example:
myMatrix.QRFactorization();
//will return the QR Factorization of myMatrix
```

- *Vector BackwardSubstitution(Vector rhs)*: perform BackwardSubtition and return the result vector. This only works for upper triangular matrix.

```
//Example:
myMatrix.BackwardSubstitution(b);
//will perform backward substitution on myMatrix.x = b and return x
```

- *void TakeInput(String input):* process an input string and convert it to a corresponding matrix.

```
//Example:
myMatrix.TakeInput("1 2\n3 4");
/*will produce a 2x2 matrix
   1 2
   3 4
*/
```

- *void Print():* print all the data of the current matrix

```
//Example:
myMatrix.Print();
//will print myMatrix row by row
```

- *static boolean IsSymmetric(Matrix parm):* check if a matrix is symmetric or not.

```
//Example:
Matrix.IsSymmetric(myMatrix);
//will check for myMatrix symmetry.
```

3) CSRMatrix:

- Some general note about the algorithms I used in CSRMatrix traversal and manipulation.
  A lot of time I used a lookup array to travel the CSR's data array column wise instead of row wise in the matrix. I called it *rowCurCol*. The purpose is to help us avoid iterating through the array to find the next entry in the current row every single time.

For example: *rowCurCol[curRow]* = 5 means the index, on the *data* array of the *CSRMatrix*, of the current entry we are talking a look at on row *curRow* is 5. So if we want to print the next entry in the same row the next time we come back to that row, we only need to get the next index from *rowCurCol[curRow]* (which is 6) instead of traversing from the start index of the row.

This is a tradeoff between performance and space. However, the space we allocate for the array is only *Θ(numberOfRow)* for the entire matrix while the worst case of performance can reach up to *O(numberOfCol)* for each row traverse and *O(nnz)* for the entire array. Thus, I believe the lookup array is worth the tradeoff.

- Matrix GetMatrixForm*()*: return a matrix with its entries are the entries of the CSRMatrix.

```
//Example:
myMatrix.GetMatrixForm();
//will copy myMatrix and return it in regular matrix form.
```

- *Matrix Multiply(Matrix parm)*: multiply 2 matrix and return the result. I used the naïve $O(n^3)$ algorithm.

```
//Example:
myMatrix.Multiply(anotherMatrix);
//will multiply myMatrix with anotherMatrix
```

- *Matrix Multiply(Vector parm):* multiply the current matrix with a vector on the right side.

```
//Example:
myMatrix.Multiply(aVector);
//will multiply myMatrix with aVector (myMatrix * aVector)
```

- *Matrix AugmentVectorAtEnd(Vector parm)*: augment the specified Vector at the end of the matrix.

```
//Example:
myMatrix.AugmentVectorAtEnd(anotherVector);
```

```
//will augment anotherVector (column vector) at the end of myMatrix
```

- *LinkedList<Matrix> LUFactorization():* return the L and U matrix from the result of LU Factorization. This method will be able to rotate the rows of the current matrix until it produces the a LU factor. However, if after trying all permutation and there is no result, the method will return null.
  This method is not fully tested, especially with big or complicated matrices.

```
//Example:
myMatrix.LUFactorization();
//will return the LU factorization of myMatrix
```

- *LinkedList<Matrix> QRFactorization()*: return the Q and R matrix from the result of QR Factorization. This method used the principle of Gram-Schmidt process to produce the correct QR factorization. This method supports any matrix.

```
//Example:
myMatrix.QRFactorization();
//will return the QR Factorization of myMatrix
```

- *Vector BackwardSubstitution(Vector rhs)*: perform BackwardSubtition and return the result vector. This only works for upper triangular matrix.

```
//Example:
myMatrix.BackwardSubstitution(b);
//will perform backward substitution on myMatrix.x = b and return x
```

- *void TakeInput(String input):* process an input string and convert it to a corresponding matrix.

```
//Example:
myMatrix.TakeInput("1 2\n3 4");
/*will produce a 2x2 matrix
    1 2
    3 4
*/
```

- *void Print():* print all the data of the current matrix

```
//Example:
myMatrix.Print();
//will print myMatrix row by row
```

- *static boolean IsSymmetric(Matrix parm):* check if a matrix is symmetric or not.

```
//Example:
Matrix.IsSymmetric(myMatrix);
```

```
 //will check for myMatrix symmetry.
```

5.