Final Project Report

Kristopher Mensing

December 4, 2016

The proposed project title is Basic Linear Algebra Project (BLAP).

This C++ application will provide a data structure representing 2-Dimensional matrices and a variety of linear algebra operations, including:

- Solving a square linear system
- Square Matrix Inversion
- Determinants for square matrices
- Dot products for vectors in Rn space
- Cross products for vectors in R3 Space
- Matrix Multiplication
- Scalar Matrix Multiplication

The intended user for this project is someone who needs to implement basic linear algebra into a larger application, or someone who wants to do basic linear algebra operations from the command line.
The problem that this project is trying to solve is performing sundry tasks required to solve linear algebraic systems, as well as the variety of matrix operations that may need to be performed in the process.
The technologies that I will need to complete this project will be 2-Dimensional arrays (likely implemented for signed integers, signed floats, and signed double floats. I will also need to implement a rudimentary input/output file system (likely CSV or text files) and a command line user interface.

1 Use Case Analysis

Type of user: Someone solving large linear system

First, the user (outside of the program) formulates their linear system in terms of a system of equations.

Then, the user creates a matrix of numbers in a CSV file, with each line representing a row in the matrix, with the columns delimited by commas. There will need to be on file with the left hand side of the equations, e.g. the $\vec{A}$ matrix, and another with a $\vec{B}$ matrix, such that

$$\vec{A}\vec{x} = \vec{B} \tag{1}$$

After creating these text files, the user will run the program. The program will prompt the user, requesting the user to input what they would like to do from the following list:

1. Solve a Linear System
2. Invert a Matrix
3. Find the Determinant of a nxn Matrix
4. Convert the matrix to another form (e.g. upper triangular, lower triangular, RREF)
5. Find the dot product of 2 vectors in $\mathbb{R}^n$ space
6. Find the cross product of 2 vectors in $\mathbb{R}^3$ space
7. Matrix Multiplication
8. Scalar Matrix Multiplication

The user will select the first option by typing "1" and pressing enter. The user will be prompted to select a matrix $\vec{A}$ from a text file, and then a matrix $\vec{b}$ from a text file. The user will be prompted for a directory and filename that they would like their solution matrix $\vec{x}$ to be placed. Then the program will run and either print that the operation was successful, or provide feedback on why the operation failed.

2 Data Design

The data the program is really about is matrices and the elements within the matrices. Matrices can be rectangular, mxn, with $m \geq 1$ and $n \geq 1$. There is also the possibility of a zero matrix $\vec{0}$, where all the elements are 0. This proves important in matrix multiplication, so will definitely take some thought. The data will extend C++ arrays, and will support serialization (at least via textfiles, but perhaps binary as well). The elements of the arrays will be either signed integers, signed floats, or signed double floats. The data will not need to be aggregated into larger structures, but it is possible that the user will want to perform batch operations in the future, so extensibility will be kept in mind when creating the libraries.

3 UI Design

When the program is first run from the command line, the program will prompt the user like this:

```
What would you like to do?
1. Solve a Linear System
2. item Invert a Matrix
3. Find the Determinant of a $nxn$ Matrix
4. Find the dot product of 2 vectors in $\mathbb{R}^n$ space
5. Matrix Multiplication
6. Scalar Matrix Multiplication
7. Find the cross product of 2 vectors in $\mathbb{R}^3$ space
8. Exit
```

Upon selecting options 1-8, the user will be presented with a screen like this:

```
What is the path of the input matrix (e.g. CSV textfile)?
-
```

Where the user will enter the path of the textfile that they want to perform the operation. And then the user will be prompted with the output screen.

At the conclusion of each operation, the program will return to the main screen.

Upon selection of option 9, the program will terminate.

4 Algorithm

The primary algorithm I'll be implementing is one that inverts square matrices. Starting with a square input matrix $\vec{A}$ and a resultant column vector $\vec{b}$, it is possible to find the inverse $\vec{A^{-1}}$. Then, finding the solution of the system $\vec{x}$ is as simple as matrix multiplication

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & \dots & A_{1n} \\ A_{21} & A_{22} & A_{23} & \dots & A_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & A_{n3} & \dots & A_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{2}$$

Finding $\vec{A^{-1}}$ is possible for a square matrix as follows:

$$\frac{1}{det(\vec{A})} * \vec{M} = \vec{A^{-1}} \tag{3}$$

Where $\vec{M}$ is the matrix of minors. Then, finding the solution $\vec{x}$ is as simple as:

$$\vec{A^{-1}}\vec{A}\vec{x} = \vec{A^{-1}}\vec{b} \rightarrow \vec{I}\vec{x} \rightarrow \vec{x} \tag{4}$$

Where $\vec{x}$ is the desired solution vector.

The matrix of minors is simply as follows:

$$\begin{bmatrix} \begin{vmatrix} A_{22} & A_{23} \\ A_{32} & A_{33} \end{vmatrix} & \begin{vmatrix} A_{13} & A_{12} \\ A_{33} & A_{32} \end{vmatrix} & \begin{vmatrix} A_{12} & A_{13} \\ A_{22} & A_{23} \end{vmatrix} \\ \begin{vmatrix} A_{23} & A_{21} \\ A_{33} & A_{31} \end{vmatrix} & \begin{vmatrix} A_{11} & A_{13} \\ A_{31} & A_{33} \end{vmatrix} & \begin{vmatrix} A_{13} & A_{11} \\ A_{23} & A_{21} \end{vmatrix} \\ \begin{vmatrix} A_{21} & A_{22} \\ A_{31} & A_{32} \end{vmatrix} & \begin{vmatrix} A_{12} & A_{11} \\ A_{32} & A_{31} \end{vmatrix} & \begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} \end{bmatrix} \tag{5}$$

for a 3x3 matrix (and it expands accordingly for larger nxn matrices). Consequently I need to implement an algorithm that determines each matrix of cofactors for each cell, and then I need to define a recursive determinant algorithm for matrices larger than 2x2.