groupproject: matrices

Connor Baker
Rae Bouldin

November 28, 2016
Version 0.6a

Contents

1 Summary of Problem Specification

1.1 Abstract

Write a program that reads two 3×3 matrices from file and computes the sum and product of the two matrices. Then, find the transpose, cofactor matrix, and determinant of the two resultant matrices. Then, find the inverse of the sum matrix, and multiply it by the the first column of the product matrix. Finally, compute the standard deviation of the diagonal elements of the two originally inputted matrices. All input and output should be stored in files.

1.2 Assumptions

The basic assumptions for this program are that no one is trying very hard to break the algorithms. Everything has been built around the knowledge that the methods will take in an perform processing on 3×3 matrices. Should this prove to be false, then the program will not work.

It is also assumed the the matrices read from file are integers. If they are decimals, the program will not work.

2 Formulae

2.1 Determinant

The determinant of a 3×3 matrix is most readily computed by row reducing to a triangular matrix, and taking the product of the main diagonal. However, failing that, one can calculate the determinant by doing cofactor expansion. Though a horribly inefficient algorithm for larger matrices, it gets the job done. For a 3×3 matrix A, its determinant, $\det(A)$, can be computed using placeholder values as follows:

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

$$\det(A) = a\begin{vmatrix} e & f \\ h & i \end{vmatrix} - b\begin{vmatrix} d & f \\ g & i \end{vmatrix} + c\begin{vmatrix} d & e \\ g & h \end{vmatrix}$$

$$\det(A) = a(ei - fh) - b(di - fg) + c(dh - eg) \tag{1}$$

As such, we can calculate the determinant of a 3×3 matrix by expanding across the top row. This yields the approach implemented in `determinant()`, detailed in Section 4.4.

2.2 Transpose

Using the same approach as above, we can easily compute the transpose of a matrix. It involves creating a matrix filled with placeholder values, calculating the transpose matrix by hand, and tracking the positions of the elements in the matrix. For a 3×3 matrix A, let its transpose be A^T. Then:

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

$$A^T = \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix} \tag{2}$$

This yields the approach implemented in `transpose()`, detailed in Section 4.5.

2.3 Matrix Addition

The approach used to create an algorithm for matrix addition is similar to that used above in finding the determinant. We again create matrices full of placeholder values, and track them as we perform the operation. For two 3×3 matrices A and B, the process is as follows:

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \qquad B = \begin{bmatrix} j & k & l \\ m & n & o \\ p & q & r \end{bmatrix}$$

$$A + B = \begin{bmatrix} a+j & b+k & c+l \\ d+m & e+n & f+o \\ g+p & h+q & i+r \end{bmatrix} \tag{3}$$

This yields the approach implemented in `add()`, detailed in Section 4.6.

2.4 Matrix Multiplication

The approach used to create an algorithm for matrix multiplication is similar to that used above in finding the determinant. We again create matrices full of placeholder values, and track them as we perform the operation. For two 3×3 matrices A and B, the process is as follows:

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \qquad B = \begin{bmatrix} j & k & l \\ m & n & o \\ p & q & r \end{bmatrix}$$

$$AB = \begin{bmatrix} aj + bm + cp & ak + bn + cq & al + bo + cr \\ dj + em + fp & dk + en + fq & dl + eo + fr \\ gj + hm + ip & gk + hn + iq & gl + ho + ir \end{bmatrix} \tag{4}$$

This yields the approach implemented in `multiply()`, detailed in Section 4.7.

2.5 Cofactor Matrix

The approach used to calculate the cofactor matrix is identical to what was done above. It involves creating a matrix filled with placeholder values, calculating the cofactor matrix by hand, and tracking the positions of the elements in the matrix. For a 3×3 matrix C, let its cofactor matrix be C'. Then:

$$C = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

$$C' = \begin{bmatrix} +\begin{vmatrix} e & f \\ h & i \end{vmatrix} & -\begin{vmatrix} d & f \\ g & i \end{vmatrix} & +\begin{vmatrix} d & e \\ g & h \end{vmatrix} \\ \\ -\begin{vmatrix} b & c \\ h & i \end{vmatrix} & +\begin{vmatrix} a & c \\ g & a \end{vmatrix} & -\begin{vmatrix} a & b \\ g & h \end{vmatrix} \\ \\ +\begin{vmatrix} b & c \\ e & f \end{vmatrix} & -\begin{vmatrix} a & c \\ d & f \end{vmatrix} & +\begin{vmatrix} a & b \\ d & e \end{vmatrix} \end{bmatrix}$$

$$C' = \begin{bmatrix} ei - fh & fg - di & dh - eg \\ ch - bi & ai - cg & bg - ah \\ bf - ce & cd - af & ae - bd \end{bmatrix} \tag{5}$$

This yields the approach implemented in `cofactor()`, detailed in Section 4.9.

2.6 Inverse of a Matrix

The inverse of a matrix A, denoted A^{-1} can be calculated as follows:

$$A^{-1} = \frac{1}{\det(A)} * A^T \tag{6}$$

Which is very doable using Equations (1) and (2). This yields the approach implemented in `inverse()`, detailed in Section 4.10.

2.7 Standard Deviation (Sample Based)

Standard deviation (as is taught in the developmental math courses here at NOVA) is the square root of variance. It is calculated as follows (and implemented in `standardDeviation()`, detailed in Section 4.11), with standard deviation denoted with sigma (σ), data points x_i, and mean mu (μ):

$$\sigma = \sqrt{\text{variance}}$$

$$\mu = \frac{x_i + ... + x_k}{k}$$

$$\text{variance} = \frac{x_i - \mu^2 + ... + x_k - \mu^2}{k}$$

$$\sigma = \sqrt{\frac{x_i - \mu^2 + ... + x_k - \mu^2}{k}} \tag{7}$$

3 `Main.class`

This class exists solely as a way to use `Matrix.class`. It creates objects that are instances of that class, and then performs operations using methods from that class on the objects of that class. It passes arguments (such as files to read and or write from/to) to the methods so that processing can happen.

4 Matrix.class

4.1 Matrix()

```
public Matrix()
public Matrix(double matrix[][])
```

`Matrix()` is the constructor. It is overloaded. The no-arg constructor sets the default matrix size to 3×3. The argumented constructor gives the flexibility of specifying different sizes of matrix.

4.2 copy()

```
public Matrix copy()
```

This method uses a `for` loop to create a copy of a matrix. This method returns a `Matrix`.

4.3 readMatrixFromFile()

```
public static Matrix readMatrixFromFile(String filename) throws
 ↪  IOException
```

The method `readMatrixFromFile()` reads a two-dimensional array of integers from file. Since the constructor is of type `double`, the values read in are automatically converted to `double` as well. This method accepts an argument in the form of a file name or path. This method is called from `Main.class` for the sake of writing matrices to file. This method returns a `Matrix`.

4.4 determinant()

```
public double determinant()
```

Using Equation (1) from Section 2.1, this method calculates the determinant of a matrix. This method returns a `double`.

4.5 transpose()

```
public Matrix transpose()
```

Using Equation (2) from Section 2.2, this method calculates the transpose of a matrix. This method returns a `Matrix`.

4.6 `add()`

```
public Matrix add(Matrix addend)
```

Using Equation (3) from Section 2.3, this method calculates the sum of two matrices. This method returns a `Matrix`.

4.7 `multiply()`

```
public Matrix multiply(Matrix multiplicand)
```

Using Equation (4) from Section 2.4, this method calculates the product of two matrices. This method returns a `Matrix`.

4.8 `multiplyByColumn()`

```
public double[] multiplyByColumn(Matrix column)
```

This method uses a variation of Equation (4) from Section 2.4 to calculate the product of a 3×3 matrix and a 3×1 column vector. This method returns a `double[]`.

4.9 `cofactor()`

```
public Matrix cofactor()
```

Using Equation (5) from Section 2.5, this method calculates the cofactor matrix of a matrix. This method returns a `Matrix`.

4.10 `inverse()`

```
public Matrix inverse()
```

Using Equation (6) from Section 2.6, this method calculates the inverse of a matrix (in part by using `determinant()` and `transpose()`). This method returns a `Matrix`.

4.11 `standardDeviation()`

```
public double standardDeviation(Matrix second)
```

Using Equation (7) from Section 2.7, this method calculates the standard deviation of the main diagonals of two columns. This method returns a `double`.

4.12 printMatrixToConsole()

```
public static void printMatrixToConsole(Matrix matrix)
```

The `printMatrixToConsole()` method consists of a `for` loop and a `System.out.println` holding a row of our 3×3 matrix (the two-dimensional array). It uses the speedup trick described in Section 5.1.

4.13 printMatrixToFile()

```
public static void printMatrixToFile(Matrix matrix, String
 ↪  filename) throws IOException
public static void printMatrixToFile(double matrix[], String
 ↪  filename) throws IOException
```

The `printMatrixToFile()` method consists of output streams (a `FileWriter`, `BufferedWriter`, and a `PrintWriter`) `for` loop and a `System.out.println` holding a row of our 3×3 matrix (the two-dimensional array). It is overloaded and prints either a `Matrix` (the first method) or a `double[]` (the second method) which holds a column vector (3×1). It uses the speedup trick described in Section 5.1.

4.14 printNumberToFile()

```
public static void printNumberToFile(double a, String filename)
 ↪  throws IOException
```

The `printNumberToFile()` method consists of output streams (a `FileWriter`, `BufferedWriter`, and a `PrintWriter`) and simply prints to file the integer passed in through the method signature.

5 Notes

5.1 A Note About the Methods

By tailoring our methods for the specific order of matrix that we operate on (3×3), we eliminate the need for method calls (specifically `matrix.length`), as well as the need for nested `for` loops. In this case, we should observe a speedup of 900% (as the compiler does not unroll loops, we are effectively doing one ninth of the iterations that we would normally do) in all methods that process arrays.

References

http://download.java.net/java/jdk9/docs/api/