David Golub (drg2112)
Carmine Elvezio (ce2236)
Muhammad Ali Akbar (maa2206)
Ariel Deitcher (ayd2102)
Computer Science 4115
Project Proposal

LAME: Linear Algebra Made Easy

We propose to develop a language with built-in support for linear algebra and matrix operations. The language will provide functionality similar to MATLAB from The MathWorks, Inc. However, the syntax will be similar to C, C++, or Java.

Our language will provide four primitive data types, `scalar`, `vector`, `matrix`, and `string`. These data types will do as their names suggest. A `scalar` will hold a double-precision floating point number. A `vector` or a `matrix` will store a one- or two-dimensional array of double-precision floating point values. Vectors will be declared using the array declaration syntax from C, C++, and Java. Matrices will be declared using commas to separate columns and semicolons to separate rows. For example, the code

```
matrix A = {
    1, 2, 3;
    4, 5, 6;
    7, 8, 9
};
```

will correspond to the matrix

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

in the standard notation from linear algebra. Individual elements of a matrix variable will be able to be access and modified using the syntax `A[i, j]` for the element in the *i*th row and the *j*th column. Keywords will be provided to obtain the dimensions of a matrix.

Operators will be provided for addition, subtraction, multiplication, division, and exponentiation. There will also be a pair of concatenation operators to support combining pairs of matrices to form larger matrices. The operators will be interpreted appropriately for various combinations of operand types, as long as they are mathematically meaningful. Combinations of operand types that are not mathematically meaningful, such as division of two matrices, will yield a compiler error. When matrix or vector operations are performed, a check will be done at runtime to ensure that the dimensions are compatible. If the check fails, a runtime error will be thrown.

The language will follow the imperative paradigm and will provide constructs for variable assignment, decisions, loops, function calls, and basic I/O. Programs will be compiled to bytecode and then interpreted from the bytecode.

Code Example

```
/*
 * Rodrigues – find a rotation matrix about a given vector by
 *             a given angle using Rodrigues's formula
 */
matrix Rodrigues(vector omega, scalar theta)
{
    matrix omegahat = CrossProductMatrix(omega);
    matrix identity = {
       1, 0, 0;
       0, 1, 0;
       0, 0, 1
    };
    return identity + omegahat * sin(theta) +
           omegahat ^ 2 * (1 – cos(theta));
}
```