SML: Specialized Matrix Language
White Paper

Patty Ho
pph2103@columbia.edu
COMS W4115
Spring 2005
Columbia University

Introduction

Special mathematical functions and calculations can be quite difficult or cumbersome to implement in programming languages. For example, many matrix operations tend to require long lines of code involving multiple copies of the same matrix, nested loops, and two dimensional arrays. The driving force behind Specialized Matrix Language (SML) is to help alleviate these difficulties by giving users a programming language with built-in matrix functions as an abbreviated way to handle matrix operations.

Background

A matrix is an ordered set of numbers listed rectangular form. They have many uses in the field of mathematics, including the transformation of coordinates and the solution of linear systems of equations. There are many operations that are often applied when dealing with matrices. Some of the most commonly used matrix operations are: arithmetic (addition, subtraction, and scalar/matrix multiplication), inverse, determinant, and transpose.

An example of a matrix:

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 0 & 1 & 2 \end{bmatrix}$$

Goal

SML is a programming language intended for mathematicians and those who deal with matrices on a regular basis, whether they are experienced with programming or not. The main goal is to provide an easy-to-learn, high level, efficient, and convenient tool to assist with performing various matrix operations.

Simple

Since SML is not a general purpose language, its scope is rather limited, but it also means that it is not as complicated as those general purpose language. Designed specifically for performing various matrix operations, SML provides most of the common matrix functionalities using simple, intuitive, and easy-to-learn syntax. People who are already familiar with matrix operations should be able to understand and learn SML with ease, even those who has little or no programming experience.

High Level

Low level languages tend to be more difficult to understand and have very few helper functions or libraries to . To achieve the goal of being an intuitive and easy-to-use language, SML is designed to be a high level language. Therefore, users do not need to worry about low level details such as internal data representation or tedious algorithm implementations.

Interpreted

For convenience, SML allows commands to be evaluated and executed on the fly, which means that it is an interpreted language. This design gives the users the option of either executing commands one by one on the command line of the interpreter or supplying an SML program file as the argument for execution when calling the SML interpreter.

Portable

The portability of Java allows a program to run on any machine that has Java Virtual Machine on it. As a result, by using a Java interpreter, an SML program can also be easily ported to and executed on any machine that has JVM on it.

Robust

Programs written in SML are very robust because SML checks for possible problems early to avoid run-time errors. The simple design of the language, such as having only a couple of data types, also helps the robustness of the language by preventing any type casting or conversion problems. Any lexical, syntactic, or semantic error will be caught by the lexer, the parser, and the semantics checker. The SML interpreter will then check for any mathematical errors in terms of matrix operations, such as adding matrixes of different sizes, multiplying un-multipliable matrixes, invalid values for matrix, etc. Problems like these stop the execution and are reported back to the user via error messages.

Data Types

There are only two data types in SML – matrices and real numbers. Conveniently, there is no need to explicitly specify the data type, as the data type is recognized automatically by the existence or lack of existence of the "[" and "]" symbols in the initialization of the value. Furthermore, all real numbers are represented as floating point numbers to handle any potential decimal data and/or calculations.

Main Features

SML provides many matrix operations with easy built-in functions including:

1. Addition
2. Scalar Multiplication
3. Matrix Multiplication
4. Transpose
5. Inverse
6. Determinant
7. Concatenation
8. Identity Matrix
9. Gauss-Jordan Procedure

SML also provides a lot of helper functions such as setting a particular element, row, or column, getting a particular element, row, or column, and adding or removing a row or column.

Furthermore, SML also supports common programming control flow constructs in a slightly different syntax:

1. if … fi
2. elseif … fiesle
3. else … esle
4. for ... rof
5. while … elihw

Sample Code

Here is an example of how a simple SML program may look like (with explanations to each line of code on the right):

```
a = [1, 2, 3; 4, 5, 6; 7, 8, 9]     // matrix initialization
b = [1, 0, 0; 0, 1, 0; 0, 0, 1]     // matrix initialization
c = a + b                           // addition
b = 2 * b                           // scalar multiplication
d = a * b                           // matrix multiplication
print("a + b = ", c)                // print text "a + b = " followed by the value of c
print(b)                            // print the value of b
print("a * b = ", d, "(mat.)")      // print text "a * b = ", followed by the value of d, followed
                                    by the text "(mat.)"
trans(a)                            // compute & print transpose of matrix a
inv(a)                              // compute & print inverse of matrix a
det(a)                              // compute & print determinant of matrix a
print("We're outta here!")          // print text "We're outta here!"
```

Summary

SML is a specialized language that focuses only on matrix operations. It is not meant to be a general-purpose language. SML provides a fast and mechanical way to perform matrix operations and thus reduce potential "human calculation errors." Those who deal with matrix operations regularly but have little or no programming experience will find SML easy to learn and useful in helping with complex matrix operations.