MathLight: Lightweight MATLAB implementation Language Reference Manual

Boya Song (bs3065),
Chunli Fu (cf2710),
Mingye Chen (mc4414),
Yuli Han (yh2986)

1. Motivation & Introduction

MathLight is a lightweight implementation of MATLAB. As a programming language, it allows basic matrix manipulations and common statistics computations. Our goal is to make it as an easy, fast and flexible language and the basic syntax is similar to C/C++.

In order to make it powerful and swift, we add the matrix data type since it's widely used in the areas of scientific computations. Also, many arithmetic operators and relational operators are designed for matrix manipulations such as matrix multiplication, transpose, equality test and so on. We implemented rich built-in functions to make it user-friendly: statistics computations such as mean, median, variance and more are supported, matrix computations like determinant, rank, trace, eigenvalues are prepared for developers to use. For I/O library, it supports CSV/XLS loading for the matrix.

2. Data Types

There are five kinds of basic data types in our language. The declaration of them are similar to statically-typed language like C and JAVA.

Type names	Description
int	32-bit signed integer
double	64- bit double precision float-point number
bool	8-bit logical value
string	string data
matrix	one or two dimensional matrix data with double type
void	no type

3. Lexical Conventions

There are six kinds of tokens: identifiers, keywords, constants, strings, expression operators, and other separators. In general, blanks, tabs, newlines, and comments as described below are ignored except as they serve to separate tokens. At least one of these characters is required to separate otherwise adjacent identifiers, constants, and certain operator-pairs.

3.1 Identifiers

In our language, identifiers uniquely defines an object. Identifiers must start with lowercase letter, followed by any combinations of numbers, letters and underscores.

3.2 Keywords

The following identifiers are reserved for use as keywords, and may not be used otherwise:

int double bool string matrix void True False
if else for while continue break func return PI

3.3 Constants

Numbers, strings are constants. Besides, predefined values used for quick access in mathematic computation, such as PI, are also constants.

3.4 Strings

A string is a sequence of characters surrounded by double quotes ‘‘ " ’’.

3.5 Expression Operators

3.5.1 Arithmetic operators

Operators	Description
+	Addition(int, double, matrix)
-	Subtraction(int, double, matrix)
*	Multiplication(int, double, matrix)
/	Division(int, double, matrix)

^	Power(int, double)
||	Absolute value(int, double)
.*	Element-wise multiplication for matrix
./	Element-wise division for matrix
.^	Element-wise power for matrix
‘	Transpose for matrix

The multiplication and division of the matrix are defined as that in the linear algebra. The dot is a special operator for matrix in our language. The dot means the operator is applied to each elements in the matrix.

3.5.2 Relational operators

Operators	**Description**
>	Greater than (int, double)
<	Small than(int, double)
<=	Smaller than or equal to (int, double)
>=	Greater than or equal to (int, double)
!=	Not equal to (int, double, matrix)
==	Equal to (int, double, matrix)

Two matrix are identical if they have the same size and the elements at the corresponding positions are the same.

3.5.3 Logical operators

Operators	**Description**
!	Not(boolean)
&&	And(boolean)
||	Or(boolean)

3.5.4 Other operators

Operators	Description
[]	Concatenate matrices horizontally, eg: [matrix a, matrix b] Concatenate matrices vertically, eg: [matrix a; matrix b]

3.6 Comments

In MathLight, the comments are similar to that in C language. We use /* to start a comment and */ to end it. Anything between them is ignored and comments cannot be nested. Besides, we also can use // to start a comment in a single line.

4. Syntax Notation

4.1 Expression

4.1.1 Primary Expression

Primary expressions are most basic expressions, which make up complex expressions. It includes constants, identifiers and expressions in parentheses.

4.1.2 Complex Expression

Expression	Explanation or Example
expression[expression]	slicing, such as a[1+2, 2*2] (a is an initialized 2d matrix)
expression(argument list)	function calls
identifier = expression	assignment

4.2 Declaration

4.2.1 Data Type Declaration and Initialization

Declarations and initializations for basic data types are similar to that in C++ as follow.

decl-specifiers declarator-list

The declarators in the declarator-list contain the identifiers being declared. The decl-specifiers consist of at most one type-specifier

Declaration	**Initialization**
int a;	int a = 0;
double a;	double a = 1.0;
bool a;	bool a = false;
string a;	string a = “hello world!”;
matrix a;	matrix a = [1, 2, 3, 4; 1, 2, 3, 4]; matrix a = InitMatrix(2, 4); (default value: 0)

Specially, for matrix, the dimension is set to be 2 and the data type of elements in the matrix is double by default.

We use the square brackets to initialize a matrix with values, eg, matrix a = [1, 2, 3, 4; 1, 2, 3, 4];. The different rows of a matrix is separated by semicolon. While the different entries in one row is separated by comma. There is only one data type in one matrix. We can use syntax like a[1, 2] to get the value which locate at the intersection of the second row and the third column. We can also use syntax like a[1, 2 : 3] to retrieve part of the matrix to get a new matrix.

Or we can define a matrix with size, such as matrix a = InitMatrix(2, 4);. We use parentheses to define a matrix with size and initialize it with zeros by default.

4.2.2 Function Declaration

The user-defined function could be declared as below:

```
func T name(T arg, ...) {
        statements
}
```

It specifies the return type, arguments name and type and function name. For example:

```
func int add_num(int a, int b)  {
        return a + b;
}
```

If nothing is returned, the return type is void.

4.3 Statements

4.3.1 Expression statements

Each expression statement includes one expression as in section 4.1, which is usually an assignment or a function call. In MathLight, every expression is followed by a semicolon:

```
expression;
```

4.3.2 Compound statements

A compound statement is a sequence of statements enclosed by braces as follow:

```
{
        statement1;
        statement2;
}
```

If a variable is declared in a compound statement, then the scope of this variable is limited into this statement.

4.3.3 Control Flows

4.3.3.1 Condition statements

The definition of a if statement is like below:

```
if  (condition(statement)) {
        Statements
}
else if (condition(statement)) {
        Statements
}
else {
        Statements
}
```

Statement following *if/ else if* (condition) must be boolean statement, or can be evaluated to boolean.

4.3.3.2 Loop statements

The definition for a for loop is like below:

```
for ( initializer(statement); condition(statement); iterator(statements)) {
        Statements
}
```

Here, initializer sets the initial state, condition is a boolean value controls whether continuing executing statements, and iterator is executed after each iteration. For example:

```
for (int a = 1 ; a < 3; a = a + 1) {
        print(a)
}
```

or

```
for (int name = begin : end) {
        Statements
}
```

which is equivalent to

```
for (int name = begin; name < end; name = name + 1) {
        Statements
}
```

The definition for a while loop is like below:

```
while (condition(statement)) {
        Statements
}
```

Here, condition is a boolean expression, which controls whether continuing executing the statements. For example:

```
while (a > 0) {
        a = a - 1
}
```

We can use **break** statement and **continue** statement to jump out from a loop or jump to the beginning of a loop.

5 Built-in Functions

Function names	Description

InitMatrix(int x) / InitMatrix(int x, int y)	Initialize 1D or 2D matrix with default value
fill(matrix m, double a)	Set all elements in matrix m to a.

log(int a)
log(double a)
	Compute the logarithm of base 2.

mean(matrix a) mean(matrix a, int dim)	Compute the mean value of a matrix. M = mean(a) returns the mean of the elements of a along the first array dimension whose size does not equal 1. M = mean(a,,dim) returns the mean along dimension dim.
var(matrix a) var(matrix a, int dim)	Compute the variance number of a matrix. M = var(a) returns the variance of the elements of a along the first array dimension whose size does not equal 1. M = var(a,dim) returns the variance along dimension dim.
med(matrix a) med(matrix a, int dim)	Compute the median number of a matrix. M = med(a) returns the median of the elements of a along the first array dimension whose size does not equal 1. M = med(a,dim) returns the median along dimension dim.
max(matrix a) max(matrix a, int dim)	Compute the maximum number of a matrix. M = max(a) returns the maxinum of the elements of a along the first array dimension whose size does not equal 1. M = max(a,dim) returns the maximum along dimension dim.
min(matrix a) min(matrix a, int dim)	Compute the minimum number of a matrix. M = min(a) returns the minimum of the elements of a along the first array dimension whose size does not equal 1. M = min(a,dim) returns the minimum along dimension dim.
inv(matrix m)	Get the Inverse matrix of a given a matrix
det(matrix m)	Compute the determinant of a matrix
rank(matrix m)	Get the rank of a matrix
tr(matrix m)	Compute the trace of a matrix
eigvalue(matrix m)	Compute the eigenvalues of a matrix
eigvec(matrix m, matrix v)	Compute the eigenvectors of a matrix given eigenvalues
size(matrix m)	Get the size of a matrix (return: 1*2 size matrix)
norm(matrix m) norm(matrix m, int p)	get the Euclidean norm of matrix m get the p-norm of matrix m

det(matrix m)	get the determinant of matrix m

6. Standard Library Functions

Function names	**Description**

print(int a)
print(double a)
print(boolean a)
print(matrix a)
print(string a)
	The matrix will be printed row by row.

scatter(matrix x, matrix y)	Draw a scatter plot picture based on an array of position[x, y]
polyfit(matrix x, matrix y, int n)	Returns the coefficients for a polynomial p(x) of degree n that is a best fit (in a least-squares sense) for the data in y
polyval(matrix x, matrix coef)	Evaluates the polynomial p at each point in x.
plot(matrix x, matrix y)	Creates a 2-D line plot of the data in y versus the corresponding values in x.
load(string filename, boolean header)	Load a csv/xls file as a matrix