Language Reference Manual

Matlab **M**atrix **M**anipulation

Shenghao Jiang(sj2914)
Shikun Wang(sw3309)
Yixiong Ren (yr2344)

Date: Oct.15 2018

1. Introduction

“MMM” is a Matlab style matrix manipulation language that can be used to calculate matrix operations efficiently. Users are able to do matrix operations through both in-built and self-defined functions. Same operations can be applied to user defined struct to make code more concise and more efficient. Due to our implementation and optimization of basic matrix operations in Ocaml, the basic matrix operations will be fast. The application of this programming language ranges from image cropping, rotating, denoising, enhancement, edge detection, and color filtering. Users are able to define the struct and functions to process image more efficiently.

2. Data Types

2.1 Primitive Data Types

Type Name	Description
int	32-bit signed integer
float	64-bit float point number
bool	bool value for True/False

2.2 Structure and Advanced Data Types

2.2.1 Matrix

Matrix is a data container that can hold only floats. Users can access individual elements in the matrix, or slice the matrix to have a new matrix. Each matrix is regarded as an array and a tuple of integer. The array contains all the elements in the matrix left to right, top to bottom, and the tuple indicates the number of row and columns of the matrix.

2.2.2 Struct

Structure is a way to group different types of data. A struct can be initialized by:

```
struct name {
    type element1;
    type element2;
```

…
} /*No semicolon after the last element*/
Create a struct by:
struct img1 = name(arg1,arg2,...)
struct img2 = name(arg1,arg2,...)

Once the struct is defined, users can access the data inside struct by:
name.element1
name.element2

3. Lexical Conventions

3.1 Identifiers

An identifier of a variable consists of one or more characters where the leading character is a letter followed by a sequence letters, digits, and possibly underscores.

3.2 Keywords

Type Name	Description
if	conditional statement that follows the syntax if-elif-else
else	conditional statement for completing if
elif	if(cond){statement} elif(cond){statement} else{}
for	for-loop follows the syntax for(init;cond;inc)
while	while-loop follows the syntax while(cond){statement}
break	breaking the iteration for for/while loops
return	ending current execution and return some values
func	keyword for declaring a function follows syntax func name(arg1,arg2,..)
struct	keyword for declaring a struct that contains data, struct name{type1 ele1; type2 ele2}
matrix	keyword for declaring a matrix contains int, float, matrix name = [...]
int	keyword for declaring a integer, int name = ...
float	keyword for declaring a float, float name = ….

bool	keyword for declaring a bool, bool name = true
string	keyword for declaring a string, string name = “....”
true	boolean type constant
false	boolean type constant

3.3 Operations

3.3.1 Basic Operators

=	value assignment, a=b
+ - * /	arithmetic operators
-- ++	increment operators
!= == < > <= >=	comparison operators
&& || !	logical operators

3.3.2 Matrix Operators

+	addition between matrices
-	subtraction between matrices
*	dot product of two matrices
.*	element-wise matrix multiplication
./	element-wise matrix division
M[a][b]	select the ath row, bth column element
M[:][b], M[a][:]	select all the rows/columns with column/row index, return a matrix
M[a:b][c], M[a][b:c]	select row number a to b with column number c, return a matrix

3.4 Comments

inline comment style	// comments	can only comment one line
multiple comment style	/* comments */	can contain newline character inside

3.6 punctuator

Semicolons at the end of each statement perform no operation but signal the end of a statement. Statements must be separated by semicolons.

4. Syntax

4.1 Loops and Statements

4.1.1 Conditional Statements

There are three conditional statements: if, else.
The syntax of each of the statement is:
if (condition) {statements}
else {statements}
There should always be a **if** statement before the **else** statements.
The "{" and "}" besides the statements can be omitted.

4.1.2 While loop

The **While** loop syntax is: **while** (expr) {stats}
The "{" and "}" besides statements can be omitted, but each statement must separate by ";"
The loop condition expr can be an integer, a bool, and a comparison expression.

4.1.3 For loop

The For loop syntax is: for(init;cond;inc)
The ";" that separates each part of the for loop expression cannot be omitted.
The initial variable, conditional statement, and increment method should be a variable, comparison expression, and an expression that increments loop variable.

4.2 Expressions

4.2.1 Associativity Rules

Tokens (Priority from High to Low)	Associativity
**NEG ! **	Right - Left
*** /**	Left-Right
.* ./	Left-Right

++ --	Left-Right
+ -	Left-Right
> <= < >=	Left-Right
== !=	Left-Right
&&	Left-Right
||	Left-Right
.	Non-Associative
:	Non-Associative
,	Left-Right
Else	Non-Associative
NOELSE	Non-Associative
=	Right - Left
return	Non-Associative
;	Left-Right

4.3 Specifications

4.3.1 Data Type

There must be a data type specifier in front of each variable name. The data type specifier can be the following:
int name = val
float name = val
bool name = val

4.3.2 Matrix

The matrix specifier identifies a matrix variable.
Example:
matrix M[row,col]
row and col are integers and it initializes an matrix of zeros with number of rows and columns
matrix M = [1,2;3,4;5,6]
, is the identifier of elements in different columns, ; is to identify different rows.

4.3.3 Functions

Example:
func name (type arg1, <struct_name> arg2, ...)
When defining a function, user needs to declare the **func** keyword followed by the name of the function. In the parenthesis where the inputs are defined, each input argument is separated by ",". Input arguments include normal data type and the struct that user defined.
The function is called by
name (arg1, arg2, ...)
The type of inputs during each call must match the input types when function is defined.

5. Standard Library Functions

5.1 Built-in Functions

5.1.1 Matrix calculations

Function Name	Return Type	Description
height(matrix M)	int	return the number of rows in matrix
width(matrix M)	int	return the number of columns in matrix
sum(matrix M)	float	return the sum of all the elements in matrix
mean(matrix M)	float	return the average of all the elements in matrix
trans(matrix M)	matrix	transpose a matrix
eig(matrix M)	float	calculate eigenvalues of a matrix
inv(matrix M)	matrix	inverse a matrix
det(matrix M)	float	calculate the determinant of a matrix
cov(matrix kernel, matrix M)	matrix	apply a convolutional kernel to a matrix

5.2 Image I/O

Function Name	Description
imread(string filepath)	load a image by giving the file path, return a struct of matrix

save(Image image,string filepath)	save the image into the file path
print(string str)	print a string when called

6. Sample Program

The purpose of this program is to denoise the image by assigning the kernel to calculate the mean of the neighbors of every pixel. The Image type is a struct that contains one matrix which represents a black-and-white image. By calling some built-in functions such as height, width, and conv, the function will return the denoised image which is also Image type.

```
// Definition of Image
struct Image {
   matrix BW
}

func noiseDeduction(<Image> sampleImage, int numOfIterations){
   matrix copyImage = sampleImage.BW;
   height = height(copyImage);
   width = width(copyImage);
   matrix kernel = [1.0, 1.0, 1.0; 1.0, 1.0, 1.0; 1.0, 1.0, 1.0];
   int i;int r;int c;
   for(i=0;i<numOfIterations;i++){
      for (r=1;r<height-1;r++){
         for (c=1;c<width-1;c++){
            matrix neighbor = copyImage[r-1:r+1][c-1:c+1];
            neighbor = neighbor .* kernel;
            float averageNeighbor =  mean(neighbor);
            copyImage[r][c] = averageNeighbor;
         }
      }
      sampleImage.BW = copyImage;
   }
   return sampleImage;
}
```