MatCV - Language Reference Manual

Abhishek Walia (aw3011),
Anuraag Advani (ada2161),
Rahul Kapur (rk2749),
Shardendu Gautam (sg3391)

Contents

1. Introduction

1.1 Motivation

MatCV is a programming language that aims at providing the programmers with a syntax that makes matrix manipulation easier and more intuitive. Since many fields, such as computer vision and machine learning use matrix operations extensively, our language introduces some constructs that allow beginners to get started easily. "Mat" in MatCV denotes that it deals with matrices and "CV" stands for Computer Vision. MatCV primarily focuses on matrix operations that will be more useful for computer vision related applications.

1.2 Description

MatCV will support primitive matrix operations such as transpose, inverse, determinant etc. We introduce a few constructs that will make looping over pixels, rows of a matrix, elements of a matrix, columns of a matrix as well as performing updates in these loops intuitive and more readable. Concatenation of matrices, creating matrices with zeros, read and display images, add and subtract pixels etc. are some other features that will be supported by the language.

2. Syntax

2.1 Data Types

MatCV will support following data types:

int	64 bit integers (32 bit integers will not be supported)
float	64 bit floating point numbers
boolean	True or False
matrix	m-by-n matrix which stores int/float type data
string	Stores sequence of UTF-8 characters

2.2 Operators

While considering operations between data types, we enforce some restrictions on the data types that can be used with each other. The operators we support are listed below:

Addition	+	Addition is supported between two matrices having the same dimensions. Addition of a matrix and scalar is not supported.
Subtraction	-	Subtraction is supported between two matrices having the same dimensions. Addition of a matrix and scalar is not supported.
Multiplication	*	Multiplication of two compatible matrices as well as multiplying a matrix and a scalar is supported.
Remainder	%	Remainder obtained upon integer division.
Exponent	^	a ^b Returns the value of a raised to power b
Division	/	Division of two compatible matrices as well as dividing a matrix and a scalar is supported.
Transpose	'	Transpose of a matrix is supported.
Assignment	=	We assign an appropriate RHS to an appropriate LHS where type promotion is supported.
Equality Check	==	Returns 1 if two 1x1 matrices are equal
Not Equal To	!=	Returns 1 if two matrices are not equal
Greater Than Operator	>	Compares the value of the elements in two 1x1 matrices
Greater Than or Equal To Operator	>=	Compares the value of the elements in two 1x1 matrices
Less Than Operator	<	Compares the value of the elements in two 1x1 matrices
Less Than Operator or Equal To	<=	Compares the value of the elements in two 1x1 matrices
AND	&&	Logical AND Operator
OR	||	Logical OR Operator

2.2.1 Operator Precedence

The precedence of our operators is the following from **Highest** to **Lowest** :

{ }, [ ]	Highest
!	
* , /, %	
+ , -	
< , > , <= , >=	
== , !=	
& &	
||	
=	Lowest

2.3 Comments

Multi - line and nested comments are supported:

```
/* This is a comment. Comments can be nested
and can be spread across multiple lines.
Comments have to be closed */
```

2.4 Keywords

MatCV will support following keywords:

row	used to iterate over the rows in a matrix
col	used to iterate over the columns in a matrix
ele	identifier to access each element in a matrix sequentially
var	declares a variable
const	modifies a variable to be immutable
if..else if..else	Supports standard conditional operations
for	loops over given elements
break	breaks out of loop
continue	returns control flow to the beginning of the loop
pixel	is a 1x3 matrix that is used to store RGB/YCrCb/HSV values corresponding to a pixel
exit	stops the program execution and returns control to the host environment
return	finish function execution and return value to the calling function

2.5 Identifiers

Identifiers in MatCV are alphanumeric and must must start with an alphabet.

2.6 Library Functions

MatCV will provide some basic functions which can be extended to implement complicated functionality:

zeros(m,n)	returns a matrix containing only zeros of dimensions m x n
eye(m, n)	returns an identity matrix of dimensions m x n
inv(a)	computes the inverse of matrix A. Matrix inverse can also be computed by $\frac{1}{A}$
det(A)	returns the determinant of a matrix in float type
rank(A)	returns the rank of the matrix
readImage(imagePath)	reads an image from the given path
showImage(windowTitle,img)	shows the image in a new window with window title
sin()	Applies sine function to all elements of a matrix
cos()	Applies cos function to all elements of a matrix
round()	Rounds all elements of a matrix to the nearest int
abs()	Absolute function is applied to all elements of the matrix
ceil()	Ceiling function is applied to all elements of the matrix
floor()	Floor function is applied to all elements of the matrix
log()	log function is applied to all elements of the matrix

3. Structure

1. All statements in MatCV are terminated by a semi-colon (;).
2. There is no specific function like 'main' that serves as the entry point in the program. Execution begins from the first statement in the program.
3. Blocks of code used by functions, if-else, for loops etc. have to be enclosed within opening and closing braces i.e. { and }

4. Statements

The following are a few examples of MatCV statements:

4.1 Matrix Declaration

You can declare a new matrix using the following syntax:

```
A = {1,2; 3,4};
```

You can also declare a matrix of zeros of size 4x2 using

```
A = [4][2];
```

4.2 Printing

The **print** keyword can be used to print out information to the console. For example:

```
A = {1,2; 3,4};
print(A[0][1]);
```

Will print 2 to the console. Row and column indexing starts from 0 in our language.

4.3 Matrix Dimensions

Row size and column size are stored as attributes for a variable internally represented as a matrix. If A is a matrix of size 5x7 then: **print(A.rowSize);** will print value 5 and **print(A.colSize);** will print the number of columns, that is 7.

4.4 Matrix Operations

Primitive matrix operations such as addition, subtraction, multiplication, transpose, inverse etc. are also provided by the language. You can invert the matrix A using:

```
inverseOfA = 1/A;
```

Alternatively, we could have used the library function **inv** to find the inverse:

```
inverseOfA = inv(A);
```

4.5 Matrix Iteration

The proposed language provides an intuitive way to iterate over all elements of a matrix.

Keyword **ele** is used in the following fashion in order to iterate over all elements of matrix A. If you had the following matrix: A = {1,2; 3,4};

Then the following code adds 1 to each element in the matrix:

```
ele e:A{
  e = e + 1;
    }
```

After the execution of above loop, the matrix A would look like:

```
A = {2,3; 4,5};
```

Inside the loop, element **e** contains attributes rowNum and colNum, that can be used to find the position of the element in the matrix. For example, if the current element in the loop corresponds to (3,2) in the matrix, then print(e.rowNum) will print 3 to the console.

There are two more variations to the above loop. You can add var in front of the variable name, using which you can change the value of the variable but the change will not be reflected in the matrix:

```
A = {1, 2, 3; 4, 5, 6};

ele var e:A{
        e = e + 3;
        print(e);
}
```

The above example prints 4, 5, 6...9 but the matrix A will still remain {1, 2, 3; 4, 5, 6}.

The const keyword can be used instead of the var keyword, which will throw an compilation error when **e** is changed in the loop. This makes sure that the user does not change the matrix unintentionally:

```
A = {1, 2, 3; 4, 5, 6};

ele var e:A{
        e = e + 3;
}
```

4.6 Row and Column Iteration

We can also iterate through the rows and columns easily. Keyword **row** is used in the following fashion in order to iterate over all rows of matrix. This example negates all the odd rows of A:

```
row r:A{
    if(r.rowNum % 2 ==1) {
        r = r * (-1);
    }
}
```

We can similarly use the keyword **column** to access the columns of the matrix.

The iterators also have attributes, rowNum and columnNum which output the row and column number at which the iterator is currently operating on.

5. References

5.1 C Language Reference Manual

Dennis M. Ritchie, C Reference Manual, Murray Hill, Bell Telephone Laboratories, New Jersey 07974
Available at: https://www.bell-labs.com/usr/dmr/www/cman.pdf