MatCab: Matrix Manipulation Language

Author:
Cheng Xiang (cx2142)
Yu Qiao (yq2145)
Ran Yu (yr2239)
Tianchen Yu (ty2275)

Professor:
Stephen A. Edwards

October 31, 2012
## Contents

1 Introduction ................................................. 2

2 Lexical Conventions ........................................ 2
   2.1 Comments .............................................. 2
   2.2 Identifiers ............................................ 2
   2.3 Keywords .............................................. 2
   2.4 Constants ............................................. 2
      2.4.1 Numeric Constants ................................. 3
      2.4.2 String Literal Constants ....................... 3
      2.4.3 Vector Constants ................................ 3
      2.4.4 Matrix Constants ................................. 3

3 Syntax Notations ........................................... 4

4 Types ..................................................... 4
   4.1 Atomic Types ......................................... 4
   4.2 Compound Types ...................................... 4

5 Expressions ................................................ 4
   5.1 Primary Expressions ................................... 4
   5.2 Unary operators ....................................... 5
   5.3 Binary operators ..................................... 5
   5.4 Other operators ....................................... 6
   5.5 I/O Expressions ...................................... 6

6 Declarations .............................................. 6

7 Statements ............................................... 7

8 Program Definition ....................................... 7

9 How Scope Rules Work in MatCab .......................... 8

10 Example .................................................. 8
1 Introduction

The MatCab is a programming language that simplifies and accelerates matrix manipulations. It is a C-style language that particularly aims at easier and faster matrix computing for programmers. It provides intuitive matrix related operators and will take the advantage of GPU to speed up the program run-time performance. This manual is inspired by the C reference manual [Dennis M. Ritchie, 1973] and CLAM reference manual [Jeremy, Robert, Kevin and Yongxu, 2011].

2 Lexical Conventions

There are six kinds of tokens: comments, identifiers, keywords, constants, strings, and operators. Comments, blanks, tabs, tabs, new lines and are ignored. The token is recognized until the parser sees a separator.

2.1 Comments

A line that begins with a pound sign (#) is comments. The compiler will ignore the comments. Only single-line comment is supported.

#This is a line of comment.

2.2 Identifiers

An identifier is used to distinguish one variable or function to another. It is a sequence of letters and numbers, and it must start with a letter. The identifiers are case-sensitive, i.e. “move” and “Move” are different variables.

\[
\text{Identifier} \rightarrow [a-zA-Z]/([a-zA-Z][0-9])^*
\]

2.3 Keywords

A keyword is reserved and cannot be used as an identifier.

\[
\text{Keyword} \rightarrow \text{int} | \text{char} | \text{float} | \text{rowvec} | \text{colvec} | \text{matrix} | \text{return} | \text{break} | \text{continue} | \text{if} | \text{else} | \text{for} | \text{while} | \text{entry} | \text{true} | \text{false} | \text{import} | \text{export}
\]

2.4 Constants

There are four kinds of constants in Matcab: numeric constants, string literal constants, vector constants, and matrix constants.
2.4.1 Numeric Constants

A numeric constant can be an integer constant, a float constant. The integer is a series of numbers. The float constant contains an integer part, a decimal point, a fraction part and a character ‘f’.

\[
\text{Numeric Constant} \rightarrow \text{Integer Constant} \mid \text{Float Constant}
\]

\[
\text{Integer Constant} \rightarrow [0-9]+
\]

\[
\text{Float Constant} \rightarrow [0-9]+ . [0-9]+ 'f'
\]

2.4.2 String Literal Constants

A string literal constant is anything inside a pair of double quote marks.

\[
\text{String Literal} \rightarrow "(any character)"
\]

2.4.3 Vector Constants

A vector constant can be a row vector or a column vector. A pair of square brackets demarcates them both. A row vector is ended with a character ‘r’ while a column vector is ended with a character ‘c’. For example, [3,4,5]r is a row vector.

\[
\begin{align*}
\text{Vector Constant} & \rightarrow \text{Row Vector Constant} \mid \text{Column Vector Constant} \\
\text{Row Vector Constant} & \rightarrow \text{open_square_bracket} \text{close_square_bracket} 'r' \\
\text{Column Vector Constant} & \rightarrow \text{open_square_bracket} \text{Numeric Constant,}+ \text{ Numeric Constant close_square_bracket} 'c'
\end{align*}
\]

2.4.4 Matrix Constants

A matrix constant begins with an open square bracket and ends with a close square bracket, just like the vector constant. It can be filled with several row vectors, column vectors, or numbers. For example, [(1,2,3);(4,5,6);(7,8,9)] gives a 3*3 matrix with values specified; [c1;c2;c3] gives a matrix with column vector specified; [r1;r2;r3] gives a matrix with row vector specified.

\[
\begin{align*}
\text{Matrix Constants} & \rightarrow \text{open_square_bracket} \text{comma separated numbers} \ast \text{close_square_bracket}
\end{align*}
\]
3 Syntax Notations

Because MatCab supports both number and matrix operations, we use uppercase letter and lowercase letter to distinguish them in the following discussion - Any uppercase character denotes a matrix, and any lowercase denotes a number.

4 Types

There are two types can either be an atomic type or a compound type.

4.1 Atomic Types

As a support or a complement language to other general programming languages, like C language, which focuses on matrix manipulation, MatCab supports a subset of arithmetic types defined in C programming language.

\begin{align*}
\text{char} \\
\text{int} \\
\text{float}
\end{align*}

4.2 Compound Types

MatCab supports compound types built upon atomic types.

\begin{align*}
\text{rowvec} \\
\text{colvec} \\
\text{matrix}
\end{align*}

Users cannot define new data types in MatCab.

5 Expressions

Here the definition of MatCab expressions follows the steps in C Language Reference Manual defining expressions.

5.1 Primary Expressions

An identifier, a constant, a string (an array of chars), an expression enclosed in parentheses, a primary expression followed by an expression in square brackets or a function call (a primary expression followed by parentheses containing possibly empty, as well as possibly comma separated list of expressions) could be a primary expression.
5.2 Unary operators

There are several kinds of unary operators in MatCab:

+ expression
- expression
! expression
evaluation ~
evaluation ~.
evaluation '
| expression |
\text{tr}(\text{expression})

! operator will make the entire expression of value 0, if the operand is of value other than 0; and will be of value 1 if the operand is of value 0.

~ and ~. operator performe matrix inverse calculation. As the convention of MatCab, if ~. operator is chosen by the programmer, the calculation will use GPU to accelerate.

' operator gives the transpose of the original expression.

| A | operator calculates the determinant of the given expression.
\text{tr}() operator calculates the trace of the given expression.

5.3 Binary operators

expression + expression
expression - expression
expression * expression
expression / expression
expression
expression *. expression
expression ** expression
expression == expression
expression || expression
expression && expression
*. operator will performe matrix multiplication using GPU acceleration. ==, || and && operators are used in flow control.

5.4 Other operators

Besides unary and binary operators, MatCab supports some useful matrix calculations using such operators:

\[ \text{submat( expression, expression, expression, expression, expression )} \]
\[ \text{sum( expression, expression, expression )} \]

submat operator gives out a submatrix of the original expression according to the parameters assigned to. sum operator gives the sum of the elements in a row or a column of a given expression.

More matrix operators are to be added along the development of MatCab.

5.5 I/O Expressions

MatCab take in the input and give out the output using I/O expressions defined within the language itself.

\[ \text{Matrix-identifier} = \text{Import ( filename )} \]
\[ \text{Export ( filename, matrix-identifier )} \]

6 Declarations

Declarations are used to associate an identifier and its data type. It tells the interpreter how to treat each identifier properly. All variables must be firstly explicitly declared before it can be used in a statement. Declarations could appear at any place of a MatCab program. It has the form:

declaration:
  Type-specifier declarator-list
declarator-list:
  First-declarator
  First-declarator, declarator-list
first-declarator:
  Declarator initializer
declarator:
  Identifier
Declarator ()
Declarator [ constant-expression ]

Initializer:
Constant
constant-expression-list

7 Statements

Statements end with semicolons in MatCab, just like what it is like in C language. Most of the time, they are expression statements. If needed, statements can be grouped together using curly bracket pair. For flow control purpose, if statements, while statements, for statements are supported in MatCab. Their grammar rules are exactly the same as C language.

8 Program Definition

A MatCab program consists of several definition statements and exactly one entry point. The entry point itself is a function declaration with a specified form:

Program → EntryPoint FunctionDecl*
EntryPoint → int entry () Statement
FunctionDecl → Type id FormalList
FormalList → Type id FormalRest*
FormalRest → , Type id
Type → AtomicType
   → CompoundType
   AtomicType → char
   → int
   → float
CompoundType → rowec<AtomicType>
   → columnvec<AtomicType>
   → matrix<AtomicType>
Statement → VarDecl*
   → Statement*
   → if ( Exp ) Statement else Statement
   → while ( Exp ) Statement
   → for ( Exp ; Exp ; Exp ) Statement*
   → id = Exp ;
   → id | Exp | = Exp ;
   → id = import (id);
→ export (id);
Exp → ( Exp )
  → | Exp |
  → Exp BinOp Exp
  → LhsUnaryOp Exp
  → Exp RhsUnaryOp
  → | Exp |
  → | tr ( exp) |
  → submat (Exp+)
  → sum (Exp+)
  → Constant
  → true
  → false
  → id
BinOp → +
  → -
  → *
  → /
  → %
  → *
  → **
LhsUnaryOp → +
  → -
  → !
RhsUnaryOp → ~
  → ~
  → '
VarDecl → Type id

9 How Scope Rules Work in MatCab

Identifiers are only valid in the scope that enclosed by the nearest curly bracket pair. If an identifier is declared out of any curly bracket pairs, then it is a global variable that accessible from anywhere of the program.

10 Example

```c
// Sample program Matrix Multiplication
int entry() {
```
int THRES = 100;
matrix A = import('matrix1.txt');
matrix B = import('matrix2.txt');
int ma, mb, na, nb;
ma = rows_count(A);
mb = rows_count(B);
na = columns_count(A);
b = columns_count(B);
if (na == mb)
{
    matrix C = new matrix(ma, nb);
    if (ma > THRES || na > THRES || nb > THRES)
        C = A * B;
    else
        C = A * B;
    export('result.txt', C);
}

References
