autoMATic

-- A language for matrix manipulation that *might* not suck™ --

COMS4115 Programming Languages & Compilers
Instructor: Richard Townsend
Team PLTeam | Spring 2018

INITIAL PROPOSAL
FEB 2ND, 2018

Team Members:

Tester/Warlock	Nelson Gomez - ng2573@columbia.edu
Language Guru/Monk	Ivy Chen - ic2389@columbia.edu
Manager/Dungeon Master	Jimmy O'Donnell - jo2474@columbia.edu
~~Ghost~~	~~Gregory Yap - gy2228@columbia.edu~~
Sys Architect/Druid	Nguyen Chi Dung - ncd2118@columbia.edu

Contents

Description

autoMATic is a language for matrix manipulation that *might* not suck. Matrices are single or multi-dimensional arrays that can be evaluated with arithmetic operators, or matrix-specific operators. Our language takes inspiration from Python (coupled with numpy), MATLAB, and Julia to provide support for data processing with a straightforward syntax.

We also allow general language features such as standard primitives (int, double, boolean, strings), control flow with C/Java-like syntax, and function definition using the *fun* keyword. Types are dynamically inferred and we opt for whitespaces to delimit statements.

Applications of our language range from standard algorithms, programs that perform intensive linear algebra tasks, as well as image processing tasks and machine learning.

Language Features

Syntax Constraints

- Comments begin with //
- Surround code blocks with curly braces { … }
- Parentheses are allowed for disambiguation
- Indentation/whitespaces **are** semantically significant

Types

Types are inferred.

Data Types	**Documentation**	**Declaration**
int	Integer value	X = 7
double	Floating point value	X = 7.
boolean	Boolean value (true/false)	X = False
string	A string, strings are immutable	X = “hello world”
matrix	Single or multi-dimensional array of ints/doubles/boolean values	X = zeros(2,2) X = [[2,2];[2,2]]

Arithmetic Operators

Matrix operations require matrices to have compatible dimensions. Ie. Element operators allow an (n x m) matrix and an int/double, another (n x m) matrix, or a 1-dimensional vector (broadcasted) as arguments.

Operator	**Description**	**Example**
+, ++	Plus operator works the same way as in	X = [1,2,3] Y = X + [4,5,6]

	MATLAB. Plusplus operator works same as in C.	// Y = [5,7,9] X++ // X = [2,3,4]
-, --	Minus operator works the same way as in MATLAB. Minusminus operator works same as in C.	X = [4,5,6] Y = [1,2,3] - X // Y = [-3,-3,-3] Y-- // Y = [0,1,2]
=	Equals operator sets the value of the left variable equal to the value of the right side	X = [1,2,3]
*	Multiply operator works the same as in MATLAB, requires matrix dimensions to be compatible.	X = [1,2,3]' Y = X * [4,5,6] // Y = [2]
**	Exponentiation operator	X = [1,2,3] X = X ** 2 // [1,4,9]
.*	Elementwise multiplication multiplies matrices A and B element by element.	X = [1,2,3] Y = [4,5,6] Z = X .* Y // Z = [4,10,18]
/	Division operator works the same as in MATLAB.	X = [1,2,3] Y = X / [4,5,6] // Y = 0.4156
./	Element-wise division divides elements of matrix A by the corresponding element of matrix B.	X = [1,2,3] Y = X ./ [2,2,2] // Y = [0.5,1.,1.5]
'	Transpose operator transposes a matrix	X = [1,2,3] Y = X' // Y = [[1],[2],[3]]
%	Yields the remainder	X = [1,2,3]

	from the division of the first expression by the second.	Y = X % 2 // Y = [1,0,1]

Relational Operators

Relational operators work on both scalars and matrices. For matrices, relational operators perform elementwise comparison and require compatible dimensions.

Operator	**Description**	**Example**
<	Less than	A = [2,3] B = [3,4] A < B // true
>	Greater than	A = [8,5] B = [3,4] A > B // true
<=	Less than or equal to	A = [2,3] B = [2,4] A <= B // true
>=	Greater than or equal to	A = [2,4] B = [2,4] A >= B // true
==	Equality check	A = [2,4] B = [2,4] A == B // true
!=	Inequality check	A = [2,4] B = [3,3] A != B // true

Logical Operators

Logical

Keywords	**Description**	**Example**
and	Logical AND	A = 5 B = True (A and B) // true

not	Negation	A = 5 B = True (A and not B) // false
or	Logical OR	A = 5 B = True (A or not B) // true

Keywords

Control flow syntax is similar to C or Java.

Keywords	**Description**
if	if (condition) {...}
else	else {...}
elif	elif (condition) {...}
for	for (i = 0; i < 10; i++)
while	while i < 10 {...}
fun	Functions
return	Returns result of a function
~~please~~	~~Denotes the end of a statement~~

Sample Program

GCD Algorithm

```
// Computes the GCD according to Euclid's algorithm
fun GCD(a, b) {
      while (a > 0) {
            c = b % a
            b = a
            a = c
      }
      return b
}
```

Matrix Determinant

```
// Computes the determinant using expansion by minors
fun det(A) {
      res = 0
      for (i = 0; i < A(0:).size(); i++) {
            res = res + (-1)**i * det(minor(A, 0, i))
      }
      return res
}
```

Maybe

These are features we'll "maybe" include, some of which are just amusing to think about:

- Membership keyword: in
- Linear Algebra Standard Library:
    - Inverse
    - Determinant
    - Convolutions
- Type annotations
- Does LLVM optimize for vector operations or is that something we have to do?
- ARE FUNCTIONS FIRST CLASS?? What if we have a matrix of functions?
- Ivy: Can we do ML for ML?
- Define a "maybe" keyword that terminates statements. It would allow us to evaluate statements probabilistically (may or may not succeed). We will employ TalMalkin as an oracle.
- User-defined objects, structs
- MAKE OUR LANGUAGE EXTENSIBLE
    - M E T A P R O G R A M M I N G
- Emojis
    - As keywords
    - As operators
    - As a primitive type with no use in particular
    - As a syntax requirement