Trix Project Proposal:
Matrix Manipulation Language

Andrew S. Hunter
July 29, 2015

1 Introduction

Trix is a Matlab-like language offering the mathematical tools to perform advanced computations. Scalar, vector and matrix calculations can be performed. A special feature of Trix is the ability to handle complex numbers. All of the standard arithmetic scalar operators are available, as well as many of the commonly used vector and matrix operators. Like the C programming language Trix has if-else decisions, while loops and for loops. It also uses functions, which behave similarly to those in Fortran and C.

2 Language Description

Trix is an imperative language used to perform numeric computations. It is a strongly typed language with types determined at compile time. It uses many of the traditional arithmetic and conditional operators as defined in the following sections. Along with the traditional operators, there are several pre-defined operations that execute commonly used algorithms. Like many languages, symbolic expressions can be used to represent the primitive types. The four primitive types are integers, single precision floating-point real numbers, single precision floating-point complex numbers and booleans. There is one derived type, the vector/matrix, which is a rectangular array with a minimum of one row and one column of the indicated primitive type. Types may be established with or without variable initialization.

2.1 Comments

The comment syntax is similar to that used in Matlab, however a comment requires two percent (%%) symbols at the start and finish of a comment, to distinguish from the single percent operator discussed later.

```
%% This is a comment %%
```

2.2 Types and Declarations

The primitive types are integers, single precision floating-point real numbers, single precision floating-point complex numbers and booleans.

`int` Integer number

`float`	Single precision floating-point real number
`bool`	True, false, 0 or 1
`cmplx`	Single precision floating-point complex number

There is one derived type, the vector/matrix.

`vm_type`	A derived type that is a rectangular array of the primitive *type*, where white space indicates a new column and the semicolon indicates a new row.

Trix is a strongly typed declarative language. Types may be declared with or without variable initialization.

```
int a = 1;
float b = 2.0;
bool c = true;
cmplx d = (1.5,2.0);
vm_float f = [.5 ; 1.0 ; 1.5]
```

Following the matrix variable with an open bracket and the row number, column number or the row-column pair can return matrix rows, matrix columns and elements, respectively.

```
vm_int B = A[R4] %% Return 4th row of matrix A %%
vm_float C = D(C1) %% Return 1st column of matrix D %%
cmplx E = F(2,3) %% Return 2nd row 3rd column %%
```

2.3 Operators

Traditional arithmetic operators. Any time the following operators are performed with a scalar and a vector or matrix, the operation is applied to every element.

=	Assignment
^	Exponentiation
*	Multiplication
/	Division
+	Addition
-	Subtraction
%	Remainder after division
++	Increment by 1
--	Decrement by 1

Relational operators. Relational operators applied to vectors and matrices of the same dimension return an Boolean vector or matrix of the same size.

==	Equality test
!=	Inequality test

>	Greater than test
<	Less than test
>=	Greater than or equal test
<=	Less than or equal test

Vector and matrix operators.

.*	Vector or matrix multiplication
.+	Vector or matrix addition
.-	Vector or matrix subtraction

2.4 Functions

Functions in Trix behave similarly to functions in Fortran or the C language, they receive one or more arguments and return one of the primitive or derived types. In Trix a function must return at least one value. Like the primitive types, the arguments and return value of a function must be declared. There is also a library of functions built into the language, described below. The function names contained in the built in library are reserved keywords and cannot be used as variable names.

Example function.

```
vm_int V1 = [ 1 2 3 ];
vm_int V2 = [ 4 ; 5 ; 6 ];

int vector_multiply (vm_int a, vm_int b) {
     int c = a[1,1] * b[1,1];
     int b = a[1,2] * b[2,1];
     int d = a[1,3] * b[3,1];

     int e = c+b+d;
     return e;
}
%% returns 32 %%
```

Mathematical functions. Any time the following functions are applied to a vector or a matrix, the operation is element-wise.

`sin()`	Sine
`cos()`	Cosine
`tan()`	Tangent
`asin()`	Arcsine
`acos()`	Arccosine
`atan()`	Arctangent
`abs()`	Absolute value or magnitude of a complex number
`log()`	Natural log
`log10()`	Base 10 logarithm

`conj()`	Complex conjugate

Vector and matrix functions. The following functions cannot be applied to integer, float or boolean types.

`Inv()`	Inverse of a matrix
`det()`	Determinant of a matrix
`tran()`	Transpose of a matrix
`kron(a,b)`	Kronecker tensor product of matrices a and b
`esum()`	Sum of all elements in a matrix or vector
`eprod()`	Product of all elements in a matrix or vector
`mag()`	Magnitude of a vector

Vector and matrix construction.

`zeros(n,m)`	Defines an n x m matrix of zeros
`ones (n,m)`	Defines an n x m matrix of ones
`eye(n,m)`	Defines an n x m matrix with ones in the main diagonal and zeros elsewhere

2.5 If-else Decision

The if-else structure is the same as C. To avoid ambiguity Trix requires parenthesis at all times. Braces help to improve clarity and reduce bugs in heavily nested code. The basic syntax structure is as follows.

```
if (expression) (
     statement
)
elseif (expression) (
     statement
)
else (expression) (
     statement
)
```

2.6 While Loops

The while loop in Trix is the same as C. Like the if-else, it requires parenthesis. The basic syntax structure is as follows.

```
while (expression) (
     statement
)
```

2.7 For loops

The for loop in Trix is the same as C. Like the while and if-else, parethesis are required. The basic syntax structure is as follows.

```
for (expr;expr;expr) (
     statement
)
```