TENLAB Reference Manual: A MATLAB-like Language for Prototyping with Tensors

Y. Cem Sübakan, ys2939
M. K. Turkcan, mkt2126
Dallas Randal Jones, drj2115

March 7, 2016

1 Introduction

TENLAB is a MATLAB-like numerical computation language specifically aimed at prototyping tensor-utilizing machine learning or numerical computation algorithms. **TENLAB** is designed to be extremely easy to use and work with, using a natural and extremely streamlined syntax whilst retaining enough versatility to address the needs of researchers or hackers who fiercely crave instant gratification.

2 Motivation

Thanks to the rapidly increasing hype around machine learning the race between machine learning researchers for novel breakthroughs is more vicious than ever before, creating a demand for extremely particular and prototype-friendly libraries. Whereas specialized libraries or workflows often exist for adjusting and combining well-known models and methods, research often requires the ability to code new models using recently-discovered methodologies in very short amounts of time whilst avoiding the soul-crushing process of the debugging of implementation-specific hacks that necessarily plague existing machine learning libraries.

TENLAB is an attempt at creating such a prototyping-friendly language for implementing numerical methods utilizing tensors. For our purposes, tensors are practically multidimensional extensions of matrices. Applications utilizing tensors usually require working with a number of different types of products and operators whose implementations require nontrivial and confusing alterations in the behavior of the original programming languages. **TENLAB** is designed to be more intuitive to use, facilitating the conversion of mathematical ideas to working prototypes.

2.1 Lexical Conventions

TENLAB closely mirrors MATLAB for the sake of simplicity. Main differences: No cell types, { } tokens have a different meaning. Only a single data type available to the user:

tensors. Tensor elements themselves are always double, and are considered tensors themselves during assignment i.e. everything is hidden to the end user.

{ } tokens are for tensor products and dimension shifting. Dimension shifting itself is possible through tensor multiplication but inefficient due to data structures and thus has its own operation.

2.1.1 Whitespace

Whitespace is normally ignored; however, if no `;` token has been detected by the `end-of-line` token and if the line is not terminated by the `...` token, a `;` token is added before `end-of-line` during compilation.

2.1.2 Comments

Comments start with the % symbol and continue until the end of the `end-of-line` symbol.

```
% A basic comment.
```

2.1.3 Compile-Time Error Fixing

Unmatched `( )` `[ ]` symbols are automatically paired during compilation (if obviously possible i.e. at the end of the line).

2.1.4 Statement Terminator

The `;` token is used to mark the end of statements.

2.1.5 Identifiers

Following the classical manner, identifiers are sequences of letters and digits; the first character of an identifier needs to be a letter.

2.2 Keywords

Keywords are restricted to special use and have particular use cases:

```
function
if
while
for
end
```

2.3 Data Types

The only type supported is called `tensor`. Tensors represent multidimensional arrays. Elements of tensors are floating point numbers with double precision. Strings do exist for users' convenience, but are converted to the corresponding numerical tensors during compilation.

2.3.1 Assignment and Indexwise Assignment of Tensors

In addition to a MATLAB/NumPy-like assignments of form

```
ID = tensor_list;
```

that is, for example:

```
a = 5;              % Create a 1D tensor with a single element
b = [5];            % Same as the line above
c = [[5,4],[4,3]]; % Create a 2D tensor i.e. a matrix with
                    % elements [5,4;4,3]
```

TENLAB allows one to only assign to a subset of indices, but only if both sides of the assignment have the same dimensionality.

```
d([[5,4,3],[5,4,4]]) = [5,4]; % Set d[5,4,3]=5 and set
                              % d[5,4,4]=4
```

In MATLAB fashion, indices begin counting from 1, not 0. Whereas tensors are abstracted as nested lists for users' convenience, under the hood they are one dimensional. In the list format, elements can be separated using ,'s or whitespace ' '.

2.4 Conversions

Tensors are converted into integers internally when certain operators, specifically, equality operators would like them to be.

2.5 Expressions

The most standard expressions supported are the ones common to popular and sane languages.

2.5.1 Arithmetic and Comparison Operators

All arithmetic operations are elementwise; however, tensors with only a single element are automatically replicated along the singleton dimensions. Precedence of these expressions are as expected.

```
'+'   : Elementwise addition
'-'   : Elementwise subtraction
'*'   : Elementwise multiplication
'/'   : Elementwise division
'=='  : Elementwise equal to
'!='  : Elementwise not equal to
'<'   : Elementwise smaller than
'<='  : Elementwise smaller than or equal to
'>'   : Elementwise greater than
'>='  : Elementwise greater than or equal to
```

2.5.2 Tensor Products

In addition to the aforementioned expressions, **TENLAB** features a powerful implementation of the tensor product.

```
{list} ID{list}*ID{list}
```

All lists should have the same length. Lists are sets of numbers separated by ,'s or whitespace ' '. Elements of the first list are either 0 or 1. The second and the third lists contain the corresponding indices over which the tensor product will be taken. For the indices corresponding to the 0's of the first list, the diagonalization argument is applied and a sum is taken over the other indices.

Implementation and Explanation: Let $A, B \in \mathbb{R}^{L_1,L_2,L_3}$. Say, we want to the following tensor product:

$$X_{i_1,i_2} = \sum_{j_1,j_2} A_{i_1,j_1,j_2} B_{i_2,j_1,j_2}$$

The statement for this operation in our language is the following:

```
X = {1 1} A{2 3}.B{2 3}
```

Here's the pseudo-code of how we can compile this statement into C code:

```
// Set L1, L2, L3 here.
// Allocate the memory for X, as a zeros tensor

for(i1=0;i1<L1;i1++) {
  for(i2=0; i2<L1; i2++ ) {
    for(j1=0; j1<L2; j1++) {
      for(j2=0; j2<L3; j2++) {
        X[i1][i2] = X[i1][i2] + A[i1][j1][j2]*B[i2][j1][j2];
      }
    }
  }
}
```

Here, there are two types of for loops. The outer most two for loops fills in the array X. The inner most two for loops, matches the second and third dimensions of A and B. Now, let's suppose we would like to do the following with the same tensors A and B:

$$X_{i_1,i_2,i_3} = \sum_{j_1} A_{i_1,j_1,i_3} B_{i_2,j_1,i_3}$$

This operation corresponds to the following statement in **TENLAB** :

```
X = {1 0} A{2 3}.B{2 3}
```

The corresponding for loops in C format are as follows:

```
//Set L1, L2, L3 here.
//Allocate the memory for X, initialize it as a zeros tensor

for(i1=0;i1<L1;i1++) {
 for(i2=0; i2<L1; i2++ ) {
  for(i3=0; i3<L3; i3++) {
   for(j1=0; j1<L2; j1++) {
    X[i1][i2][i3] = X[i1][i2][i3] + A[i1][j1][i3]*B[i2][j1][i3];
   }
  }
 }
}
```

One of the challenges is to figure out the limits of each for loop, as they change according to the statement.

2.6 Tensor Shifts

Shifting dimensions of tensors is possible through

```
ID{list},{list}
```

in which the first list contains the indices to be shifted and the second contains the amounts by which they should be shifted.

2.7 Statements

2.7.1 Expression Statements and Compound Statements

Expression Statement: Similar to C, classical statements are expression statements.

Compound Statement: A group of statements could be chained back to back, but only when expected by conditional statements `if`, `while` or `for`. Collectively we can refer those as `statement lists`.

2.7.2 Conditional Statements

Conditional statements are supported, in its most basic sense, through the `if/else` keywords. The `if` keyword could be used without an `else` like

```
if ( expression );
        statement_list;
end;
```

or it can be used in conjunction with an `else` as follows:

```
if ( expression );
        statement_list;
else;
        statement_list;
end;
```

Finally, the use of the `elseif` keyword:

```
if ( expression-1 );
        statement_list;
elseif ( expression-2 );
        statement_list;
else;
        statement_list;
end;
```

allows the chaining of two or more conditional statements.

2.7.3 While Statement

The `while` keyword defines a loop statement that resembles

```
while ( expression );
        statement_list;
end;
```

and as in most other languages the loop will continue as long as the value of the `expression` is nonzero.

2.7.4 For Statement

The `for` statement is in the MATLAB fashion:

```
for assignment;
        statement_list;
end;
```

The statements in `statement_list` run through for each element assigned during `assignment`.

2.8 Function Definitions

While different than the usual MATLAB syntax, functions are defined a similar manner to the preceding statements:

```
function function_name(identifier_list);
        statement_list;
        return expression;
end;
```

and can be called at will.

2.9 Examples

Let us start with a simple Hello World program:

```
% Hello World
print('Hello World'); % Prints 'Hello World'
```

```
% Assignment
tensor_b = 42.0; % Create 1-dimensional tensor with a single
                 % nonzero index
tensor_c = 'string'; % Create a 1-dimensional tensor with
                     % 6 nonzero indices
tensor_d = [[45,42],[39,36]]; % Create a 2-dimensional tensor
```

```
% Arithmetic
print(1+1); % Displays 2
print([5,4]-1); % Displays [4,3]
print(2*2); % Displays 4
print(5/7); % Displays some float
```

```
% Logic Operators
print(42<54); % Displays 0.0
print([15,17]==15); % Displays [1.0,0.0]
```

```
% Loops:
% For Loop
for i=[1,2,4,5,8];
        disp(i);
end;

```

```
% While Loop
i=0;
while i==0;
        print('This will not end.');
end;
```

```
% Using Tensors
A([[1,2],[2,3]]) = A([[1,2],[2,3]]) + [2,3];
                                      % Add a value of 2 to
                                      % coordinate (1,2)
                                      % and 3 to (2,3)
% Tensor Products
Z ={1} A{2}.B{1}; % Dot product over A's second dimension and
                  % B's first
Z = A{2},{-1};    % Shift (or roll) A's second dimension by -1
```