Programming Language and Translator

Project Proposal

Language For Linear Algebra (LFLA)

Author:	*UNI:*
Chenzhe Qian	cq2185
Guitian Lan	gl2510
Jin Liang	jl4598
Zhiyuan Guo	zg2201

September 30, 2015

Motivation

Linear algebra is one of the most important subject for undergraduate STEM students. Currently, the undergraduate linear algebra courses are matrix oriented, as opposed to theory oriented. However, due to the abstractness of linear algebra, many students still find it too difficult to catch its essence. We believe that computer science should play a crucial role in math education - especially helping students to learn tough subjects like linear algebra. Yet, unfortunately, the common available math software involving linear algebra like MATLAB and R only deal with matrix. This only leave students with the impression that linear algebra is just matrix. Moreover, in MATLAB the vectors are treated as the (1 x m) or (n x 1) matrix, and this is not conceptually correct. It often leads to confusion and problems in coding and debugging. Therefore, these math oriented language cannot truly help the students understand the fundamentals and the beauty of the linear algebra. Given this situation, we would like to create a domain language to help students and teachers in learning and teaching linear algebra. The Language for Linear Algebra(LFLA) is mainly designed for educational purpose.

Features

- In LFLA we have several primitive types direct corresponding to the concept in linear algebra. Beside the matrix, we also have vector, vector space, affine spaces and inner product space. It will tell the students that linear algebra is more than just matrix.
- Besides the normal matrix calculation, our language emphasizes the relation between vector space, vectors and matrix considered as a map. We believe that understanding their relation is important in learning the linear algebra and students can get insight by using the language to exploring the linear algebra world.
- We take the best to make the language syntax similar to the math notation, so that learning the language will not be a large overhead.

Built-in Types

Types

```
var      // An integer or a float as scalar
vector   // An element of the real coordinate space $R^n$.
         // An n-vector is a list of n numbers.
vecspace // Vector space. It's a collection of vectors.
matrix   // A rectangular array of numbers, arranged in
         // rows and columns.
inspace  // Inner product space. It's a vector space
         // with an inner product.
affspace // Affine space.
```

Type Declaration and Initializaition

VECTOR

```
// Declare a vector v
vector v;

// Declare and initialize a vector v
vector v = [$a_1$,$a_2$, ...,$a_N$]; //Separate by comma

// Examples
vector v;
vector v = [1,2,3];
```

MATRIX

```
// Declare a matrix m
matrix m;

// Declare and initialize a matrix m
matrix m = [
                $a_{11}$,$a_{12}$,...,$a_{1M}$;
                $a_{21}$,$a_{22}$,...,$a_{2M}$;
```

```
                              ......
                      $a_{N1}$,$a_{N2}$,...,$a_{NM}$
                  ];
    // Each element in vector is separated by comma
    // Each vector is separated by semicolon

//Examples
matrix m;
matrix m = [1,2; 3,4];

// Note matrix cannot interchange with vector
vector v1 = [1,3];
vector v2 = [3,4];
matrix m = [v1; v2]; // This is not allowed
```

VECSPACE

```
//A vector space is defined as a function of
// multiple vectors
vecspace vs = L($v_1$, $v_2$,..., $v_N$);

//Examples
vector v1 = [1,0];
vector v2 = [0,1];
vecspace vs = L($v_1$, $v_2$);
```

Operators

LFLA contains all operators in common languages like Java and C++, except for bit manipulation operators. However, there are subtile differences between scalar-scaler, scalar-object and object-object operations. Here objects are vectors, matrix, vector space, affine space and inner product space.

Scalar-Scalar operations

Assignment	=	Assign a value to a var
Addition	+	Add two var
Subtract	−	Subtract one var from another
Multiply	∗	Multiply two var
Division	/	Divide one var from another
Exponential	ˆ	Exponential computation on var
Modular	%	Modular computation on var
Increment	++	Add 1 to the value of var
Decrement	−−	Subtract 1 from the value of var
Compare	!=, ==, <, >, <=, >=	Normal comparison operations on var

Scalar-Object operations

Addition	+.	Add a var to each element in vector, matrix
Subtract	−.	Subtract a var from each element in vector, matrix
Multiply	∗.	Multiply a var to each element in vector, matrix
Division	/.	Each element in vector, matrix is divided by a var
Exponential	ˆ.	Exponential computation on each element in vector, matrix
Modular	%.	Modular computation on each element in vector, matrix

Object-Object operations

Assignment	=	Assign an object, like vector, matrix, vecspace
Addition	+	Add corresponding elements in two objects
Subtract	−	Subtract between corresponding elements in two objects
Multiply	∗	Multiply corresponding elements in two objects
Transpose	´	Transpose of a matrix
Inner Product	<,>	Inner product
Lie Bracket	[,]	Lie Bracket
Compare	<, >, ==, @	Some specific comparisons

Library Functions

In LFLA language, it provides several library functions.

- sqrt(var)
  Take square root of a scalar

- ceil(var)
  Take ceiling of a scalar
- floor(var)
  Take floor of a scalar
- dim(vector v)
  Give the dimension of a vector
- dim(matrix m)
  Give the dimensions of a matrix
- basis(vecspace vs)
  Give out one a array of basis (vector) of a vector space.
- rank(matrix m)
  Give the rank of a matrix
- trace(matrix m)
  Give the trace of a square matrix
- eigenValue(matrix m)
  Give the eigenvalues of a matrix
- image(matrix m)
  Give the image of a matrix
- orthoBasis(inspace i)
  Give the orthogonal basis of a inner product space

Control Flows and Function

```
// if else statement
if expr
{
    statements;
}
```

```
    else if
    {
        statements;
    }
    else
    {
        statements;
    }
    // for loop
    for expr
    {
        statements;
        break;
    }
    // while loop
    while expr
    {
        statements;
        continue;
    }
    // define a function
    function name(args)
    {
        stataments;
        return [value1, value2, ..];
    }
```

Sample Program

```
// Problem 1: Given an array of vectors of the same
// dimension, judge wether they are linear independent.
// Solution:
function linearIndep(vector[] vectors)
{
    if vectors.length == 0
        return true;
    if vectors.length > dim(vectors[0])
        return false;

    var i;
    vecspace vs;
    for i=0:vectors.length
    {
        if vectors[i]@vs
            return false;

        vs = vs + L(vectors[i]);
    }
    return true;
}

// Problem 2: Given a inner product space W = (V,a),
// please orthonormalising a basis of vector space of V
// by the Gram-Schmidt method
// Solution:
othonomalising(vector[] basis)
{
    var i, j;
    for i=0:basis.length
    {
        vector v = basis[i];
        for j=0:i
        {
            w = w-a<basis[i], basis[j]>*.basis[j];
        }
        basis[i] = w/.sqrt(a<w, w>);
    }
}
```