PLT Project Proposal

Group Members
Siddhi Mittal
Sahil Yakhmi
Damien Fenske-Corbiere
Dan Aprahamian

Description
We plan to implement a MATLAB-like language for numerical computation. This language will allow functions to be entered as literals. Our language will have the following types - Scalar, String, nD-Array, Matrix (derived from nD-Array) and Mathematical Function(mfunc). It will contain loops and conditionals. It will not have a boolean type - 0 equals false and everything else is true. Integration will be limited to single variable functions only. The derivative with respect to a particular variable of a multi-variable function can also be calculated at some value of that variable.

Problem our language can solve/ how it can be used
This language would have the capability to -

- Evaluate mathematical expressions.
- Manipulate mathematical functions (add, subtract, multiply, divide, integrate, differentiate).
- Graph mathematical functions.
- Matrix operations on numbers.
- Solve linear systems of multi variable equations.

Some examples of its syntax and an explanation of what it does

- Every statement in our language ends with a semi-colon.
  nX = 2;

- You cannot declare a scalar without simultaneously assigning it. To declare and assign a scalar variable, place the variable name on the left side of a single equals sign, and a literal on the right side.
  nX = 2;

- You declare a function literal with the following mapping notation. This syntax allows us to easily distinguish between scalar and function parameters or array elements.
  fFunc1 = (x) -> x + 1;

Interesting and representative program in our language.

/* Basic Assignment and computation on scalars and mfuncs */

```
nX = 2;                          /* Creates scalar nX and assigns it the value 2 */
nY = 3;                          /* Creates scalar nY and assigns it the value 3 */
fFunc1 = (x) -> x + 1;           /* Creates function fFunc1. fFunc 1 takes one
                                    variable x, and returns the scalar value = x + 1 */
fFunc2 = (x) -> 2x - 3;          /* Assigns function (x) -> 2x - 3 to fFunc2*/
fSum = (x, y) -> x + y;          /* Assigns function f(x, y)= x + y to new variable
                                    fSum*/

nZ = nX + nY;                    /*  nZ now equals 5*/
nZ = fSum(nX, nY);               /*  nZ now equals 5*/
nZ = fFunc1(nX);                 /*  nZ now equals 3*/
nZ = fFunc2(nX + nY);            /*  nZ now equals 7*/
fTwoFunc= fFunc1 + fFunc2;       /* fTwoFunc now equals (x) -> (x + 1) + (2x -
                                    3)*/
```

/* Examples of Boolean Logic and Control Flow*/

```
bX = 1;                          /* scalar bX is now equal to 1 (effective to true)
*/
bY = 0;                          /* scalar bY is now equal to 0 (effective to false)
*/
bZ = 1;                          /* scalar bZ is now equal to 1 (effective to true)
*/
bResult;                         /* empty variable bResult */
fCond1 = (x, y, z) -> (x + y) * z;  /* Assigns function f(x, y, z) = (x + y) * z to new
                                    variable fCond1. This can be used both for
                                    regular numeric computation and for boolean
                                     algebra*/
fCond2 = (x, y) -> x > y;        /* Assigns function f(x, y) =x > y to new
                                    variable fCond1. Returns 1 if true, 0 if false. This
                                    is to be used for boolean algebra */

if fCond1(bX, bY, bZ):           /* if bZ AND (bX or bY)*/
   bResult = 1;
else:
   bResult = 0;

if fCond2(nX, nY):               /* if nX > nY/
   bResult = 1;
else:
   bResult = 0;
```

```
while fCond2(nX, nY):          /* while nX > nY */
  nY++;                        /* increment nY by 1 */
```

/* Examples of arrays and matrices */

```
aArr1 = [5];                   /* Creates array of length 5. All values are
                                  initially set to 0*/
aArr2 = {1, 2, 3, 4};          /* Creates length 4 array with values 1, 2, 3, 4*/
aaArrX = [5, 4];               /* Creates two dimensional 5x4 array. This
                                  qualifies as a matrix*/
aaArrXX = [5, 4];
aaArrY = [4, 2];               /* Creates two dimensional 4x2 array. This
                                  qualifies as a matrix*/
aaArrX[0][1] = 2;              /* Assigns a value to one of the array elements /*
aaArrX[0] = aArr2;             /* Assigns an entire row of values */
aaArrX[-][1] = aArr1;          /* Assigns an entire column of values */

/* Assign other values here*/
aaArrZ = aaArrX + aaArrXX;     /* Performs element-wise addition on the
                                  two matrices and returns new matrix*/
aaArrZ = aaArrX * aaArrXX;     /* Performs element-wise multiplication on the
                                  two matrices and returns new matrix*/
aaArrZ = aaArrX # aaArrY;      /* Does matrix multiplication and returns new
                                  matrix*/
```