JME Language Proposal

Daniel Gordon COMS W4115

Introduction and Goals

JME (pronounced *jay+me*) is a lightweight language that allows programmers to easily perform statistic computations on tabular data as part of data analysis. The language is designed to perform calculations and operations on primitive data structures like a vector or matrix, as well as a higher-level structure like a map. The language syntax of JME somewhat resembles javascript, however the semantics are that of a functional language. JME is designed as a scripting tool for single time analysis of a data set. JME contains native capability to scan data from an external file and make the data available as map objects. However the external file must conform to a strict style definition. JME has several commonly used statistical measures built in (such as mean, median, mode…) but will allow the user to express any number of statistical analysis algorithms through the creation of functions.

A goal of JME is to analyze data sets in which each row represents a single entity and the columns of data are individual attributes of that entity. For example below is a table of NFL Quarterbacks and their stats for the 2013 season:

Name	Comp	Att	Yds	TD	Int
Peyton Manning	450	659	5477	55	10
Drew Brees	446	650	5162	39	12
Andy Dalton	363	586	4293	33	20
Phillip Rivers	378	544	4478	32	11
Tony Romo	342	535	3828	31	10

A goal of a JME script might be to sort the players (rows) based on the number of TDs (column attribute). Another, more complex goal of a JME script might be to calculate the Passer Rating of each player (a numeric result based on an algorithm computation using data attributes Comp, Att, TD, and Int) and sort the players based on this previously undefined statistic.

Primitive Data Types

For simplicity JME will support a small number of primitive data types:

- int, float, boolean, and String

Data Structures: Vectors and Matrices

JME primarily operates on data structures. The simplest structure is a numeric vector consisting of a single list of numbers. To initialize a vector of 5 numbers in JME we use the protected word "***struct***" and might look something like this:

```
var x = struct(1, 2, 3, 4, 5);
```

In this case the vector is initialized with integers 1-5 and assigned to the variable x. The statement is ended with the semicolon character.

We can perform simple arithmetic operations on the vector, in which each element of the vector is transformed:

```
var y = x * 2;
```

The variable y now represents a vector containing the numbers 2, 4, 6, 8, 10. We can also compute the mean of vector y by passing it to the statistical function:

```
mean(y) == 6 == sum(y)/len(y)
```

The mean function is equivalent to the sum of vector y divided by the length of vector y.

Finally we can access individual elements of a vector using subscript:

```
y [0];
> returns 2
```

Matrices are defined similarly to vectors but use the curly brace character to define each row in the matrix:

```
var z = struct( {1,2,3},
                {4,5,6} ,
                {7,8,9});
```

Accessing individual elements of a matrix is also similar using subscript:

```
Z[1][2];
>returns 6
```

Commonly accepted vector and matrix operations, such as multiplication by a scalar or matrix addition and subtraction (assuming same dimensions) can be expressed in JME using standard arithmetic statements:

var x = struct(1,2,3);
x * 2;
>returns a new vector: (2, 4, 6)

Other operations like transpose, dot or cross product can be achieved through the creation of functions.

Data Structures: Map

JME also supports a higher-level data structure of a map consisting of multiple key-value pairs. The key must always be a string but the value can be any of the primitive data types such as an int, or a String, or even a data structure like a vector. The syntax of initializing a map with a row of data from the QB example above might look like this:

var manning = map("comp"=450, "att"=659, "yds"=5477, "td"=55, int="10", "name"="Peyton Maning";

One of the advantages of using a map is that you can use the key attributes to access individual elements of the map:
var touchdowns = manning@"td";
>returns 55

Unlike vectors and matrices, map elements cannot be transformed using basic arithmetic expressions such as multiplication by a scalar:
var z = Manning * 8;
throws an invalid expression error

Functions

JME has a few built in functions for common statistical algorithms and allows users to define their own functions. Functions can be defined with zero or more parameters but must always return something. Functions are defined with the restricted word "*function*" and curly braces to mark the beginning and end. A function implementing the Passer Rating statistic in the above example would look something like this where the parameter *player* is a map:

```
function passerRating (player) {
        var a = (player@comp/player@att - .3) * 5;
        var b = (player@yds / player@att -3) * .25;
        var c = (player@td / player@att ) * 20;
        var d = 2.375 – (player@int/player@att * 25);
        return ((a + b + c + d)/6) * 100;
}
```

passerRating(manning);
> returns 115.11

In the above example the manning map is passed as a parameter to the passer rating function.

Logical Controls

JME will support logical control with if/else statements. Else statements will be optional and elseif will also be supported. The a syntax would look like the following:

```
if(<boolean expression>) {
        <statement>
} else if (<boolean expression>) {
        <statement>
} else {
        <statement>
}
```

Looping

JME supports both ***for*** and ***while*** looping structures. ***While*** loops will operate on an expression with the following syntax:

```
while(<boolean expression>) {
        <statement>
}
```

For loops are provided as a convenience and can only be used against a data structure. The simplest example is iterating over the individual elements of a vector:

```
var x = struct(1, 2, 3,4,5);
var total = 0;
for(element in x) {
        total += element;
}
print total;
> outputs 15
```

Caveats and Limitations

The focus of the language will be creating a solid foundation for performing operations on vector and matrix structures. Supporting the map data structure as outlined may be a little ambitious and might be de-scoped from the language. Likewise supporting the read-in of an external file might also be too ambitious for this project and may be de-scoped.