MatPix: A Matrix Based Scientific Language with GPU Computation

David Burkat dsb2117@columbia.edu
Oliver Cossairt ollie@cs.columbia.edu
Robert Hawkins rh2305@columbia.edu
Ben London bal2123@columbia.edu

December 18, 2007

Contents

Chapter 1

Introduction

1.1 White Paper

Modern GPUs are designed to perform extremely fast, highly parallelized matrix processing; but their power is often under-utilized by most common computing tasks. By offloading certain specialized processes to the GPU, one can expect massive performance gains. MatPix is a programming language for matrix arithmetic – similar to the popular software package, Matlab, but with the performance gains that are realizable by utilizing the GPU. Essentially, the MatPix compiler will translate MatPix source code into C code with embedded OpenGL calls for optimized matrix operations.

1.1.1 Goal

Historically, Matlab has been the reigning champion of matrix arithmetic programming. To be fair, MatPix poses no threat to this title. MatPix's primary contribution to programming languages is purely theoretical. We see great potential in GPU processing and propose this language as a stepping stone to a richer, fully-realized scientific computing language for the GPU. Similar endeavors are already under way; mainly by the graphics card manufacturers themselves. Nvidia recently released a library add-on for Matlab that, provided your computer has a Nvidia GPU, will process certain operations exclusively on the GPU, yielding dramatic performance improvements. Since we are not afforded the time and resources of Nvidia, we propose much more modest goals. Ultimately, we aim to implement basic arithmetic functions — simple unary and binary operations — that can be used to perform essential matrix calculations on the GPU.

1.1.2 High-level Description

Like Matlab, the language of MatPix is designed to be clear and intuitive to anyone familiar with linear algebra and a nominal amount of structured programming. In keeping with the mathematical paradigm, MatPix is not an explicitly typed language. Every variable is implicitly of type "matrix." Furthermore, no special declarations are needed. Variables can be created at any point in a program, simply by assigning them a value. Dynamic allocation of matrices at runtime is stable and completely invisible to the user. Familiar and intuitive control flow structures add basic, yet crucial, branching and looping functionality. User-defined functions enable efficiency of code. In short, MatPix is intended to facilitate scientific computing tasks with minimal programming.

1.1.3 Main Language Features

In this section, we discuss some of the main features of MatPix.

Data Types

Fundamentally, every named variable is an arbitrarily sized, 2-dimensional matrix. There is no need to explicitly declare a variable's type, as all variables are implicitly of type "matrix." All scalar values have 32-bit floating-point precision, thus eliminating most conversion and precision concerns. (Note that all indexing expressions are converted to integer values before indexing.)

Matrix Access

Matrices can be accessed in two ways:

- They can be accessed as a whole, meaning the entire contents will be retrieved or modified.
- They can be sliced along rows, columns, individual elements or contiguous blocks, thereby affecting only specific regions.

Matrix Constants

MatPix offers a simple, intuitive syntax for declaring matrix constants.

Operators

MatPix supports basic unary and binary operators that can be applied to both variables and constants. These operators include: addition; subtraction, multiplication, division, modulo, dot-product, increment, decrement, transpose, logical negation, AND, OR and relational operators. All operators (except for dot-product and transpose) perform element-wise computations on matrices.

Control Flow

Program control flow is accomplished via familiar syntactical and structural paradigms. MatPix supports if()/else() structures for branching and for() and while() loops for iteration.

User-defined Functions

MatPix supports user-defined functions to promote reusable code. Furthermore, MatPix supports recursion, thus enabling efficient, streamlined function implementation.

Output

MatPix offers a built-in print function, for simple, yet crucial, access to standard output.

Invisible Memory Management

Memory management is safe, efficient and completely invisible to the user. Memory leaks and null pointer checks are a non-issue in MatPix.

Chapter 2

Tutorial

2.1 Introduction

Now we will give an introduction to using MatPix by demonstrating some of the more salient language features. Here we feature all of the code necessary to implement a simple linear regression algorithm. Understanding all of the language features demonstrated here is crucial to building more complex programs in MatPix. We will give detailed step-by-step instructions below.

We also note here that many MatPix constructs are taken from the widely popular Matlab language. Matlab syntax is available in more detail at [1].

2.1.1 Regression Details

A multidimensional linear regression gives us the MatPix language specific details necessary for complex program construction.

We begin by giving a brief overview of the mathematics behind a regression. Let X be an $N \times (D+1)$ dimensional matrix of our input points. N is the number of points and D is the dimensionality of each point (the zero dimension is constant). Next, our output Y is an $N \times 1$ dimensional matrix. The idea behind a regression, is that we wish to find a hyperplane in D dimensions that best relates X to Y.

Obtaining our empirical risk equation and minimizing it by setting to gradient to zero, we obtain

$$\theta^* = \left(X^T X\right)^{-1} X^T Y \tag{2.1}$$

where θ^* is the optimal set of parameters for our hyperplane. We have that θ^* is a $(D+1) \times 1$ matrix representing the hyperplane

$$y = \begin{pmatrix} 1 & x_1 & ... & x_D \end{pmatrix} \begin{pmatrix} \theta_0^* \\ \theta_1^* \\ \vdots \\ \theta_D^* \end{pmatrix}$$

The general idea here is that we take as input a few D dimensional points (represented by X) and an equal number of 1 dimensional outputs (represented by Y) and return θ^*, where θ^* relates future D dimensional points to hypothesized outputs.

A linear regression is useful for real data analysis when we wish to find empirical trends. The regression gives us a model relating input data to output data. Given some input data, this enables us to make output predictions.

2.2 Implementation

2.2.1 Matrix Inversion

A crucial component to a matrix regression is being able to invert a matrix. In general, matrix inversion is one of the most important algorithms for any mathematical computation.

Here we present a simple Gauss-Jordan elimination technique for matrix inversion. For conciseness, we leave out error checking for singularities (determinants equals zero).

Implementing the Inversion

The Gauss-Jordan technique involves augmenting the input matrix with the identity, and then eliminating rows until the inverse is obtained where the identity previously resided.

Below is pseudo-code for the familiar algorithm

```
GaussJordan (X)
1:      A  <-- A horizonatally concatenated with the identity of size X.size
2:      for i <-- each row of of A
3:              divide row i by the value at position i of row i
4:              while the i-th position of row i has a zero
5:                      swap the i-th row of A with a row below below it
6:              for j <-- each row of A below i
7:                      row j <-- row j - position i of row j * row i

8:      Repeat 2-7 for A starting at the last row and eliminating upwards
9:      Return all rows of A for columns X.size+1 to the end
```

The Code

Now we have the following MatPix code for the above algorithm:

```
function eye(m)
{
    matrix Z[m,m];
    for(i=0:m-1)
    {
        Z[i,i] = 1;
    }
    return Z;
};

function gauss_jordan(V,n) //expand to use size function
{
    matrix Q[n,n*2];
    matrix E[n,n];
    E=eye(n);

    //Set Q to V augmented with nxn identity
    Q[:, 0:n-1] = V;
    Q[:, n:n*2-1] = E;
    r = 0;
    temp=0;//swap storage
    for (i=0:n-1)  //for each row
```

```
    {
        r = i;
        //Check for zero in eliminator row
        //If zero, then find row to swap with
        while((Q[i,i] ==0) && (r !=n-1))
        {
            r = r+1;
            temp = Q[i,:];
            Q[i,:] = Q[r,:];
            Q[r,:] = temp;
        }
        //Normalize the eliminator row
        if (Q[i,i]!=0)
        {
                Q[i,:] = Q[i,:]./Q[i,i];
        }

                //Use Eliminator row to eliminate
                //the column of all rows beneath
        for (k=i+1:n-1)
        {
            Q[k,:] = Q[k,:] - Q[i,:] .* Q[k,i];
        }
    }
    for (i=n-1:-1:0)
    {
        for (k = i-1:-1:0)
        {
            Q[k,:] = Q[k,:] - Q[i,:] .* Q[k,i];
        }
    }
    return Q[:,n:n*2-1];
};

d = 16;
matrix A[d, d];
A = eye(d);
A = A*5;
A[8,9]=4;
A[14,1] = 5;
A[15,1] = 15;


print("A:");
print(A);
print("inv(A):");
inverted = gauss_jordan(A,d);
print("Inverted matrix:");
print(inverted);
```

We will describe this code piece by piece.

First, we comment on how these functions have been written. We first note that Matpix functions are all pass-by-reference. Thus, we cannot write a function that modifies a matrix in the upper scope.

We now have the function

```
eye(m)
```

, which inputs a dimension, m, and returns an identity matrix of size m. First, we create an $m \times m$ matrix which is automatically initialized to the zero matrix. Next, we iterate through all rows and set each element in a diagonal position to 1.

Here we notice two important features of MatPix. First, matrices start at position 0 and end at position $m-1$. Second, we have a special Matlab-like loop construct. This constructs allows us to create an iterator, and designate its start and finish position with a very simple syntax. Such a construct is very convenient for mathematical algorithms which make heavy use of matrix iteration.

Next, we have the first portion of the main inversion function

```
function gauss_jordan(V,n) //expand to use size function
{
    matrix Q[n,n*2];
    matrix E[n,n];
    E=eye(E, n);

    //Set Q to V augmented with nxn identity
    Q[:, 0:n-1] = V;
    Q[:, n:n*2-1] = E;
```

We first notice that there are two function arguments. The first, V, is the square matrix that we wish to invert. Next, we have n, which is the dimensionality of V. Note that in MatPix, there is no way to obtain the dimensionality of a matrix. Thus, we must explicitly pass the size when required. Explicity passing the size may help the user better keep track of the dimensionality of intermediate matrices in complex algorithms.

Next, we declare and assign our augmented matrix (A in the pseudocode). Here we notice another key feature of MatPix derived from Matlab. We can assign submatrices to slices of a matrix. This flexibility allows us to construct the augmented matrix in two simple steps.

The first step is assigning V to the left side of Q and then the second is to assign E to the right side of Q. This slice assignment allows us the flexibility to manipulate matrices for many common applications such as this one.

Next, we have

```
    r = 0;
    temp=0;//swap storage
    for (i=0:n-2)  //for each row
    {
        r = i;
        //Check for zero in eliminator row
        //If zero, then find row to swap with
        while((Q[i,i] ==0) && (r !=n-1))
        {
            r = r+1;
            temp = Q[i,:];
            Q[i,:] = Q[r,:];
            Q[r,:] = temp;
        }
```

Here, i iterates through each row except the last. We then use the standard while-loop construct to check if we have a zero in the leading column of row i. We then swap row i for a row beneath it until we have

exhausted all rows below. If we find a row with a non-zero leading element, then that row becomes our new eliminating row. If we find no such row, then the entire leading column is zero.

Notice that the array slice assignment operation is very useful for swapping rows.

Next,

```
       //Normalize the eliminator row
      if (Q[i,i]!=0)
      {
              Q[i,:] = Q[i,:]./Q[i,i];
      }
              //Use Eliminator row to eliminate
              //the column of all rows beneath
      for (k=i+1:n-1)
      {
          Q[k,:] = Q[k,:] - Q[k,i] .* Q[i,:];
      }
  }
```

The first "if" clause checks to make sure that column i is not all zeros. This condition is false if the previous while loop iterates through all rows without finding a non-zero leading element row to swap with.

We then normalize the row by its leading element. We accomplish this by dividing the entire row by element i. This allow us to use row i to eliminate all rows below it. Row elimination occurs through a sequence of elementary row operations. We multiply row i by element i in row j. Then, we subtract the result from row j so that the new row j has a 0 in the leading position. We continue this process for each row beneath i. Note that each pass of the outer loop eliminates a single column of the matrix. This is the column that we refer to as the leading column.

The last section of the code,

```
for (i=n-1:1)
    {
        for (k = i-1:-1:0)
        {
            Q[k,:] = Q[k,:] - Q[k,i] .* Q[i,:];
        }
    }
ret = Q[:,n:n*2];
print("inverted result");
print(ret);
```

, simply repeats this entire process in reverse. This ensures that the upper triangle of Q is eliminated as well. Here we notice two convenient "for loop" constructs. First, the outer loop allows us to iterate from some number, in this case $n-1$, down to 1. For extra clarity, we can also iterate backwards using the more familiar Matlab style (the inner loop).

After this entire process, the left section of matrix Q is the identity matrix and the right section is the inverse of V. Thus, we use the matrix slicing operator to return the right half of Q.

2.3 Putting it all together

Simply renaming the function "gauss_jordan" to "inv", we have the following sample code for a multidimensional linear regression.

```
n = 16; d = 2;
matrix X[n,d];
X[:, 0] = 1;
for(i=0:n-1){
        X[i, 1:d-1] = i;
}

print("X input value");
print(X);

matrix theta[d,1];
for(i=0:d-1){
        theta[i, 0] = 5.2*i+.3;
}
print("the theta regression values");
print(theta);

Y = X*theta;

print("the actual Y values");
print(Y);

theta_solve = inv(X'*X, d)*X'*Y;
print("regression solution for theta:");
print(theta_solve);

err = theta-theta_solve;
print("solution error:");
print(err);
```

We test the linear regression as follows. We create our inputs, X as well as our regression parameters θ. We then calculate Y as $X^T\theta$. Thus, Y is simply the output of applying θ to X. We then use the regression formula to calculate θ^* from X, Y. We compare θ^* to θ to make sure the regression yielded accurate results. The regression was a success if $\theta^* = \theta$.

The first 6 lines construct our input matrix, X. The output of the first two print statements is thus

```
X input value
1, 0
1, 1
1, 2
1, 3
1, 4
1, 5
1, 6
1, 7
1, 8
1, 9
1, 10
1, 11
1, 12
1, 13
1, 14
```

```
1, 15
```

Next, we assign our θ values. This vector represents the parameters for our hyperplane. The output of the next print statement is

```
the theta regression values
0.3
5.5
```

Using our θ values, we next derive our Y values as a linear transformation on X. The next print statements output

```
the actual Y values
0.3
5.8
11.3
16.8
22.3
27.8
33.3
38.8
44.3
49.8
55.3
60.8
66.3
71.8
77.3
82.8
```

Finally, the crucial part of this code,

```
theta_solve = inv(X'*X, d)*X'*Y;
```

solves for θ^* using the equation for the linear regression that we discussed earlier in 2.1. The output of the final print statements is

```
regression solution for theta:
0.3
5.5

solution error:
0
0
```

We note here that our regressed θ^* is equal to θ. Thus, we solved and verified the linear regression. We note here that the GPU computations yield numerical inaccuracies when using GPU emulation software. However, as we can see, the hardware GPU calculations are indeed accurate.

2.4 Notes

We have presented a detailed description of the features of MatPix using the example of a Linear Regression. We see that the language features of MatPix are very well suited to the very general class of mathematical problems involving matrix manipulations. The unique feature of MatPix is that each of these matrix manipulations is computed on the GPU.

The MatPix language featues allow us to compute a multidimensional linear regression in very few lines. The algorithm would be many more lines of code in commonly used high level languages such as Java/C/C++.

The power of MatPix is that a relatively small set of language features allow this succinct algorithmic description. First, all MatPix object are matrices. Having matrices as first class objects is crucial when dealing with sets of data. Next, the key feature of matrix slicing allows very complex assignment and indexing in a single command. We can both read from and write to a Matrix using simple syntax Finally, the "for"-loop index constructor allows us to effortlessly implement the commonly used paradigm of iteration.

From this tutorial, we can see that it would not be difficult to implement a comprehensive matrix library in MatPix. While the inverse is a key function, we could just as easily implement the determinant, eigenvalue decomposition, and other such functions with similar ease.

Chapter 3

Language Reference Manual

3.1 Introduction

This language reference manual contains detailed descriptions of each of our language constructs. We present here each language construct and its MatPix semantics.

3.1.1 How to read this document

All character values are contained within double quotes, i.e. "A", "/*", etc. It is implied that, when using the language, the double-quotes should be omitted. Code examples will be indented and in Courier New font. Grammatical definitions follow ANTLR syntax, and will be indented and italicized.

3.2 Lexical Conventions

A token can be a non-empty string of non-whitespace characters. There are five categories of tokens: identifiers, keywords, constants, operators and separators. During parsing, whitespace (space, tab, newlines) and comments are ignored, except as they serve to separate tokens; at least one of these non-tokens is required to separate certain tokens.

3.2.1 Comments

The language supports both single line and multi-line (i.e. traditional "C-style") comments. Single line comments begin with two forward-slash characters "//" and terminate at the end of the line in which they appear. Multi-line comments begin with the characters "/*" and terminate after the first instance of the characters "*/" is encountered.

3.2.2 Identifiers

An identifier is a non-empty string over upper and lower case letters, digits, and the underscore "_" character. The first character must be a letter. Identifiers are case-sensitive.

The following are examples of valid identifiers:

```
myid   my_id   myId123   a
```

The following are examples of invalid identifiers:

```
123myId   $myId   my#Id   _123MyId
```

The grammar for identifiers is as follows:

```
identifier
    : ( 'a'..'z' | 'A'..'Z' ) ( ( 'a'..'z' | 'A'..'Z' ) | ("0".."9") | "_" )* ;
```

3.2.3 Keywords

The following identifiers are reserved for use as keywords, and may not be used otherwise:

```
print
if
else
for
while
return
function
matrix
```

3.3 Expressions

3.3.1 Constants

Scalar Constants

All scalar constants are single precision floating point values, following closely the well-known C syntax. In its full form, a scalar constant consists of an integer part, followed by a decimal point, followed by a fractional part, followed by an optionally signed integer exponent. Only the fractional part may be omitted. If the fractional part is omitted, the period must also be omitted. The exponent is in all cases optional.

Both the integer and fractional parts are non-empty strings over digits '0' ... '9'. The exponent must be preceded by an 'e' (case-insensitive), optionally followed by a sign, followed by a non-empty string over digits '0' ... '9'.

The following are examples of valid scalar constants:

```
123   123.45   0.45   123.45e-6   123E4
```

The grammar for scalar constants is as follows:

```
DIGIT
    : ("0".."9");
DECIMAL
    : DIGIT+ ( "." DIGIT+ )? ;
SCALAR
    : DECIMAL ( 'e' (MINUS)? DIGIT+ )? ;
```

Matrix Constants

Matrix constants begin with a left bracket '[', followed by a series of rows, separated by the row separator '|' and optional whitespace, and closed with a right bracket ']'. Each row consists of a series of scalar constants, separated by commas and optional whitespace. All rows must contain the same number of scalars, except for the last row, which may optionally be empty.

The following are examples of valid matrix constants:

```
[ 1, 2, 3 ]

[ 1, 2 | 3, 4 ]
```

```
[ 1, 2 | ]
```

The following are examples of invalid matrix constants:

```
[ 1 2 3 ]
```

```
[ 1, 2, 3 | 4, 5 ]
```

The grammar for matrix constants is as follows:

```
cols
    : SCALAR ("," SCALAR)* ;
row
    : cols ("|")? ;
matrix
    : "[" (row)+ "]" ;
```

String Constants

String constants consist of a sequence of characters enclosed in double quotes. A double-quote character may be included in the string by writing two double quote characters side by side. An example of a string containing two double-quote characters is:

```
"A bee flew ""over"" the house"
```

Strings are only to be used inside print statements. Everything within the quotes will be printed to the output stream. This is literal transcription, with the only exception being the escape double quotes.

3.3.2 Expressions

Expressions in MatPix are combinations of unary and binary operations, matrix slicing, function calls, variables (including slices of matrix variables), and constants. Binary operations are either mathematical or logical operations. Assignment is handled as a binary operator. Function calls and variables can either be used as expressions on their own or in nested expressions. All operators except assignment can accept expressions as their argument(s). Assignment can only accept expressions as a right hand side argument, not at a left hand side argument.

All unary and binary operations are valid for scalars as well as for matrices. All unary operations proceed element-wise on matrices. The binary addition and subtraction can be used to add two matrices as well as two scalars. The binary multiplication will perform a matrix multiply if the matrix arguments are of correct dimensions and will perform a scalar multiply if the arguments are both scalars. If one argument is a scalar and the other a matrix, then the multiply will perform an element wise multiplication on the matrix. The binary modulo performs a modulo operation on two scalars, element-wise modulo on two matrices, and element-wise modulo on a scalar with a matrix. The division, on the other hand,

Additionally,

Unary Operations

Unary operations can be performed on any expression. The Unary operators are “-”, “!”, “++”, “–”. Respectively, these operators perform sign negation, logical negation, increment by one and decrement by one. Logical negation simply sets the variable value(s) to one if it is zero, and sets it to zero otherwise. Examples of expressions including unary operators are:

```
c = b++;
a[1:10]++;
x = -z;
b = !a[1:10];
```

The grammar for unary operations is as follows:

```
signedExpression
    : ("-")? incrementExpression;
incrementExpr
    : variable ("++" | "--")?;
```

Mathematical Binary Operations

Binary math operations take two arguments that can be any expression. The operators are "%", "*", "/", "+", and "-". Respectively, these operators perform modulo, multiplication, addition, and subtraction. Examples are:

```
x = a % b;
3 * 4 + 5 * 6;
a = 3 * (4 - 5) / 6;
z = 2 + 3*4;
```

The grammar for mathematical binary operations is as follows:

```
addExpression
    : multiplyExpression ( ("+" |"-") addExpression )?;
multiplyExpression
    : signedExpression ( ("*" | "/" | "%") multiplyExpression )? ;
```

Logical Operations

Logical operations take two arguments that can be any expression and return floating point values of zero or one. The supported logical operators are "<", ">", "≤", "≥", "==", "!=", "&&", "||". Like the other binary operators, the logical operators perform element-wise comparisons. The "<", ">", "≤", "≥", "==", "!=" respectively perform greater than, less than, greater than or equal, less than or equal, equal, and not equal comparisons to their arguments. The "&&" and "||" operators first convert the left and right input arguments to boolean representation by setting them to zero if they have a value of one, and one otherwise. After converting to boolean, the logical operators then perform the familiar operations on the input values (logical "and" and logical "or"). Examples of logical operations are:

```
a > b;
x = a < b;
x = a == b;
x = a != b;

if (a > b)
{
    a = b;
}
```

The grammar for logical binary operations is as follows:

```
logicalExpression
    : predExpr (("&&"|"||") logicalExpression)?;
predicateExpression
    : (("!")? addExpression) (("<"|">"|"<="|">=") predicateExpression)?;
```

Function Calls

Function calls accept arguments that can be any expression and return values that can be used as temporary variables in nested expressions. MatPix grammar allows a function to either explicitly return a value or not. If a value is not returned explicitly, semantic analysis detects this and implicitly returns a scalar with value zero. Thus, it is valid in MatPix to make an assignment to a function that does not declare a return value (unless the assignee has a different size, which is not allowed for any binary operation). The grammar allows an arbitrary amount of function parameters. The size of the variable returned by a function call will depend on the function definition. If no return statement is issued in the function definition, then the function call will return zero.

Examples of function call usage are:

```
A[:] = foo(a);
foo(a, b);
a = foo(a, b);
```

The grammar for function calls is as follows:

```
functionCall
    : identifier argumentList;
argumentList
    : "(" (expression ("," expression)* )? ")" ;
```

Variables

Variables in MatPix can either be temporary (as in a constant or function call return value) or peristent (as in an identifier which refers to a variable in stored in memory, or the result of a matrix index which refers to a region of memory). Only persistent variables can be assigned to, but all variables can be used in nested expressions. Since there is no typing in MatPix, all variables are implemented as matrices. Single value floating point variables are implemented as 1x1 matrices. For convenience, a syntax that is familiar for a single value variable is provided. The syntax is enforced by automatically converting scalar values to 1x1 matrix representation, so that assignment is valid. Thus, the syntax allows a 1x1 matrix to be assigned to a scalar without indexing. The following examples demonstrate the use of variables.

```
a;
a = 1.2;
a = [1.2];
a[1:2] = foo();
z = [1,2,3];
a = (foo() + b) * c[1:2] + [1, 2];
```

The grammar for variables is as follows:

```
variable
    : (identifier | matrixIndex | constant | functionCall );
```

Matrix Indexing

MatPix allows flexible matrix indexing using customizable ranges, similar to Matlab. Matrix indexing allows us to slice arrays using integer increments.

A range is a scalar followed by an optional semicolon and scalar followed by another optional semicolon and scalar. All scalars that are used in a range expression are converted to integer representation during semantic analysis by flooring. Thus, non-integer ranges are allowed, but automatically converted to integer representation. Semantic analysis ensures that the calculated indices are within the range of valid indices of the matrix being sliced.

The grammar for matrix indexing is as follows:

```
matrixIndex

    : identifier arraySlice;

arraySlice:

    "[" rangelist "]";

rangeList:

    range ("," range )?;

range:

    (SCALAR (":" SCALAR (":" SCALAR)?)?) | ":";
```

Examples of matrix indexing are:

```
A[1:5] = [1, 2, 3, 4, 5];
A[1:2:10] = [1, 2, 3, 4, 5];
A[1:2, 1:2] = [1, 2| 3, 4];
A[1:2:4, 1:2:4] = [1, 2| 3, 4];
A[1:2:4, :] = [1, 2];
B[:, :] = [1, 2| 3, 4];
```

Note: A variable without a subscript is shorthand for a slice consisting of every element in the matrix. Thus for a 4x4 matrix C:

```
C;
```

and

```
C[:, :]
```

both return a 4x4 matrix of values. Thus the following examples are also valid syntax (assuming the matrices on both sides of the assignment have the same size and dimension)

```
C = [1,2 | 3,4];

C[1:2, 1:2] = D;

A = C;
```

Assignment

Assignment is the only binary operator does not accept all expressions as arguments. The left side of an assignment is restricted to a persistent variable (which includes a slice of a matrix), while the right side of the matrix is allowed to be any expression. The grammar does not enforce this rule and allows for any expression to be used as either argument. Semantic analysis enforces this rule by reporting an error when a constant, function call, or any nested expression is detected on the left side of an assignment. Examples of assignments are :

```
a[:,:] = [ 1, 2 | 3, 4  ];
a = foo(x,y);
a[1:2, 1:2 ] = b[1:3, 2:5 ];
foo = 3 + 4;
a = b[2:4, 5:7 ];
```

The grammar for assignment is as follows:

```
expression
    : assignment ;
assignment
    : logicalExpression ("=" logicalExpression)? ;
```

3.3.3 Operator Precendence

The primary-expression operator:

```
()
```

has highest priority and groups left-to-right. Binary and unary operators all group left-to-right, and have priority decreasing as indicated:

```
++ ??
- (unary)
^
* / %
+ ? (binary)
! < > <= >=
== !=
&& ||
```

Assignment operators all have the same (lowest) priority, and all group right-to-left.

3.4 Statements

A statement is an expression, function definition, "return", "break", or "continue" followed by a semicolon, a control statement, or just a semicolon.

The grammar for statements is as follows:

```
statement
    : expression ";"
    | functionDefinition ";"
    | controlStatement ";"
    | "return" expr ";"
    | "break"  ";"
    | "continue"  ";"
    | ";" ;
```

3.4.1 Function Definition

A function definition consists of the string literal "function" followed by an identifier and an argument list. The function body consists of a left curly bracket followed by zero or more statements followed by a right curly bracket.

The grammar for function definitions is as follows:

```
functionDefinition
    : "function" identifier definitionArgumentList "{" (statement)* "}" ;

definitionArgumentList:
    : "(" identifier ("," identifier)? ")" ;
```

3.4.2 Control Flow Statements

A Control Flow Statement consists of either an if statement, a while statement, or a for statement.

An if statement consists of the word "if" followed by a logical expression enclosed in parenthesis, followed by one or more statements enclosed in curly brackets, followed by an optional else clause. The else clause consists of the word "else" followed by one or more statements enclosed in curly brackets.

A while statement consists of the word "while" followed by a logical expression enclosed in parenthesis, followed by one or more statements enclosed in curly brackets.

A for statement consists of the word "for" followed by an identifier, followed by an equals sign, followed by a range, followed by one or more statements enclosed in curly brackets.

The grammar for control flow statements is as follows:

```
controlflow
    : ifStatement | whileStatement | forStatment;

ifStatement
    : "if" "(" logicalExpression ")" "{" (statement)+ "}" ( else "{" (statement)+ "}" )?;

whileStatement
    : "while" "(" logicalExpression ")" "{" (statement)+ "}";

forStatement
    : "for" identifier "=" range "{" (statement)+ "}";
```

Chapter 4

Project Plan

4.1 Project Planning

In general, our project planning process involved regular group meetings and email communication.

Our first goal, after group formation, was to create a viable project proposal. After 3-4 group meetings and much individual research, we decided upon MatPix. We wanted to make sure that our proposed project was simple enough to be viable given the time frame, as well as complex enough to provide unique functionality. With this goal in mind, we decided upon a Matlab-like language with added GPU computation. Initially, our goal was to divorce the matlab component from the GPU component, so that if the GPU computation proved too complex, then we would still be able to achieve a matlab-like language. However, while much work needs to be done to utilize the efficiency of GPU computation, we were able to achieve both matlab and GPU components of our project.

Our next step was to decide upon the syntax and semantics of our new language. This also involved group meetings as well as some investigation into Matlab syntax/semantics. After establishing a relatively complete conceptual picture of MatPix, we began work on the LRM and Lexer/Parser.

We leveraged googlecode and subversion version control to work on our project. Version control along with group programming allowed us to complete the LRM and Lexer/Parser in a group coding format.

The next phase involved working on the GPU backend as well as the tree walker. The GPU backend was written by Ollie, while the entire group worked on the tree walker. The symbol table and scoping issues were mostly handled by Rob.

Throughout the entire process, we developed unit tests that together functioned as a regression test for future modifications. We used these tests to make sure that new functionality did not break old functionality. These tests will be described in more detail in the Testing Plan section.

Finally, we finished writing our comprehensive tests and debugging. In the testing phase, Rob was responsible for maintaining the unit/regression tests, Ollie/Ben debugged the GPU backend, and David developed the regression algorithm and organized the final project report.

4.2 Individual Responsibilities

As a summary, general group responsibilities can be roughly defined below.

David Burkat	algorithm testing, documentation
Oliver Cossairt	GPU backend, backend testing
Robert Hawkins	parse table, unit/regression testing
Ben London	backend testing, unit/regression testing

However, the lexer/parser/tree walker was implemented largely as a group effort.

4.3 Programming Style

4.3.1 ANTLR

It is easy for ANTLR grammar syntax to become too complicated. This makes debugging the grammar very complicated. For this reason, we prefer many simple rules as opposed to few complex rules. It is crucial to intuitively name each tree node so that there is no confusion.

Parser rules should not be excessively long. In this case, It is best to break the rules into a set of simpler rules.

4.3.2 Java

Most of our Java code was in the tree-walker. This code prints out the proper C++ code. As is standard practice, all Java variables should have intuitive names. Each indentation should be 4 spaces or a tab character. For code clarity, each curly brace should occupy its own line.

In general, it is preferable to have more spacing between lines. Thus, each "else" should occupy its own line.

We try to catch all possible errors using try catch blocks in the Java Code (except run-time errors). Run time errors are caught in the C++ code.

4.3.3 C++

The C++ coding standards are very similar to the Java Standards with a few differences.

First, due to the interaction with the GPU libraries and matrix manipulation, it is difficult to have simple code. Often times, functions are complicated and have many arguments. Thus, it is important to have as much clarity as possible in the variable names. Intuitive variable naming facilitates debugging. Since many of the matrix/vector transformations involve keeping track of rows, columns, and indices, it is crucial that the C++ code is easy to understand.

4.3.4 MatPix

The MatPix coding conventions are very similar to Matlab conventions. We note that MatPix employs curly braces to mark code blocks. We felt that this code feature would make it easier for C++/Java users to adapt to MatPix.

As usual, we encourage the use of descriptive commenting in MatPix. Often times, Matrix Mathematics can result in very dense and complicated code. It is important to use comments to describe otherwise opaque manipulations.

While we would like to encourage descriptive variable naming in MatPix, our language has a similar problem to other mathematical languages. In a complicated algorithm, it is often difficult to assign intuitive names to variables which store intermediate computations. When such a problem occurs, it is often best to adopt canonical mathematical symbol names (A, B, X, Y, alpha, gamma, etc) when such exist.

4.4 Tools and Languages

The Lexer, Parser, and Tree walker were written in ANTLR. The ANTLR compiler generated java code. ANTLR was particularly well suited for a concise compiler. Our compiler backend, however, was written and compiled with GNU C++. That is, our generated Java Code had as its ouptut C++ code. C++ was used specifically for access to the GPU library calls. Python and Makefile were used for scripting the unit/regression tests. Python allowed us to easily conduct comprehensive unit and regression tests. This testing suite allowed us to make incremental progress while being assured that no new features broke old functionality.

We used GoogleCode for our project repository. This involved SVN for version control. All project related files were stored in GoogleCode under the MIT open source license. Googlecode provided the server space for our project, while SVN facilitated coordinated development.

4.5 Timeline

Below is the project timeline. These dates are anticipated completion deadlines for each section of our project. They were only preliminary and subject to change given unanticipated scheduling conflicts.

September 25	Project Proposal
October 5	LRM Outline
October 18	LRM, Parser, and Lexer
October 30	Preliminary Tree Walker
November 16	GPU Backend,
November 23	Grammar File, Scoping Complete
November 24-December 18	Testing & Debugging
December 14-18	Project Report
December 18	Project Due

We note here that the bulk of the project work was in the debugging and testing phase. For this reason, we dedicated a significant amount of time to fixing errors and developing tests to verify the integrity of the language.

Debugging involved fixing errors and then verifying the regression tests. The culmination of the testing phase was the completion of the matrix inverse algorithm and the accompanying regression algorithm. We discuss the layout of our regression and unit tests in the Testing Plan section.

4.6 Project Log

Below is the project log. The project log documents the dates of important events in the timeline of the MatPix project.

9-04-2007	PLT Group Formed
9-06-2007	MatPix Proposed
9-10-2007	VC & IDE Established
9-18-2007	GPU Proof of Concept
9-24-2007	Project Proposal Written
10-5-2007	LRM First Draft
10-17-2007	Parser, Lexer First Draft
10-30-2007	Tree Walker Draft
11-16-2007	GPU Backend Completed
11-22-2007	Tree Walker Complete
11-23-2007	Symbol Table Complete
11-24-2007	Testing & Debugging Commences
12-10-2007	Code Finalized (Walker, Backend, etc)
12-13-2007	Report Started
12-14-2007	Inversion & Regression Work!
12-17-2007	Report Finalized, LRM updated

Chapter 5

Architecture Design

5.1 Introduction

The Matpix language provides an execution model that facilitates shared computation between a CPU and a GPU. GPUs provide highly accelerated arithmetic computations with limited control flow features for imperative programming. The GPU computation paradigm is similar to functional programming in the sense that data is essentially immutable during computation. Matpix, however, is not a functional language, so executing imperative computations in Matpix requires shared processing between the GPU and CPU. At the core of the Matpix compiler is a facility to divide program execution into control flow instructions that execute on a CPU, and arithmetic instructions that execute on a GPU.

5.2 GPGPU Computation Model

Modern GPU architecture evolved from the highly parallel graphics computations that are needed for raster based graphics rendering. Current GPUs have multiple processing units that execute a reduced instruction set in a highly parallel manner. In particular, GPUs have very poor support for branching operations. They are optimized for vector arithmetic computations where the data and computation is assumed to be completely independent. These optimizations provide highly accelerated arithmetic and memory operations with limited computational functionality.

5.2.1 Mapping Computations to the GPU

Not all computations can be mapped to the GPU. Computations that are most suitable for the GPU are "kernel" based operations that perform the same operation on a large array of data. These kernel operations are performed once per element of the array. Data dependent branching on the GPU is extremely expensive, and is typically implemented by following each execution branch simultaneously.

5.2.2 Computation

The last major update to the OpenGL standard included an interface to create and run programs that get executed directly by the processing units on a GPU. The interface includes a standard C-like language called "Opengl Shading Language" (GLSL) as well as standards to compile and link programs written in GLSL to instructions that execute on the GPU. GLSL programs come in to flavors, fragment and vertex. For reasons I do not go into here, all Matpix GPU computations are implemented as fragment programs.

The GPU computation can be thought of as a program that loops through every element in an array and applies a kernel to that element. Probably the biggest limitation in the GPU computation model lays in the fact that there is very limited control given to the programmer over the outer loop in this computation.

For instance, there is no way to tell the GPU to stop executing the loop once it has begun execution. In addition, the extremely expensive final stage of a GPU computation where output pixel values are written to the frame buffer cannot be cancelled once this main loop is entered. This means that if great care is not taken, much of the computational load of the GPU will be entirely wasted. So although GLSL does include common control flow statements such as "for" loops and "if-then" statements, these control flow statements are not as useful as they might first appear to be. In an attempt to maximize the computation load on the GPU, Matpix implements all control flow operations on the CPU.

5.2.3 Memory

Modern GPUs provide extremely fast memory in quantities typically a factor of 2-4 less than CPUs. With close to 1GB VRAM available on current GPUs, memory is plentiful enough for most applications. However, inherent to the GPU optimization model is limited memory operations. GPUs do not provide random-access memory like most CPUs do. Instead, GPUs provide Read-Only memory operations during execution, and Write-only operations after the computation is completed.

Read-only Memory : Textures

In graphics, textures are used to add a high level of visual complexity to models with relatively simple geometry. Textures are loaded onto the GPU by transferring 2D image data from CPU RAM to the GPU, where they are then stored in local texture memory. Texture read operations are allowed during GPU fragment program execution, where the value of an output pixel is being calculated. For GPGPU calculations, textures are treated as arbitrary memory regions and texture read operations are treated as the memory read operations that allow access to this region of memory.

Write-only Memory : Frame Buffer

After a fragment program is executed, the calculated value of the output pixel written to the a region of texture memory known as the "Frame Buffer". During typical graphics applications, after fragment program execution is finished, contents of the frame buffer are sent directly to the graphics port which converts the pixel data to a signal that can be accepted by a computer monitor such as an LCD screen. For GPGU computations, the contents of the frame buffer contain the output of the fragment program execution, and the results must be sent back to the CPU for output or further processing. It is also possible to copy the contents of the frame buffer into a region of texture memory (so that it can be read during fragment program execution).

5.2.4 OpenGL Requirements

There are two important GPU features that are required for GPGPU computation. These features are provided on modern OpenGL driver implementations for all dedicated graphics cards that were produced in the last 3-5 years. The first is the "shader program object" extension which provides the interface for compiling, linking and executing fragment programs. The second, slightly less ubiquitous feature is the "texture float" extension which provides floating point textures. Typically, textures are just 8-bits per color, and are treated in an analogous manner to unsigned characters in 'C'. A special OpenGL extension must be enabled to allow floating point texture data to be stored on the GPU. Currently, all floating point GPU data is 32-bit. There is no double precision floating point data representation or computation on even the most state of the art GPUs.

5.2.5 Software Mode

GPUs typically come in two flavors: "dedicated" graphics cards, and "integrated" graphics chips. Dedicated graphics cards are connected to the motherboard on slots that allow the GPU to communicate with the

CPU via an external BUS such as PCI, AGP, or PCI-Express. Integrated graphics chips are typically found on consumer level laptops, and often have a dramatically reduced feature set in comparison to their dedicated counterparts. For this reason, integrated graphics cards typically do not provide floating point textures, which means they cannot be used to perform GPGPU computations. However, OpenGL drivers on most operating systems provide a mechanism to turn off all GPU hardware acceleration and emulate GPU computations in software. These software OpenGL implementations typically do provide floating point textures and therefore can be used for GPGPU computations. The Matpix compiler has the facility to create executable code that can either run in "hardware accelerated" mode or "software" mode. This was necessary to implement because several members of the group needed to be able to work on their laptops with non-GPGPU compliant integrated graphics chips.

5.3 CPU Computation

As already mentioned, Matpix redirects all control flow operations to be performed on the CPU. Matpix source files are compiled into C++ source files that are then compiled with a C++ compiler and linked against the Matpix "standard" library. All control flow statements in Matpix source files are implemented as analogous statements in C++ syntax when they are compiled.

5.4 Communication Between the GPU and CPU

Because the GPU sits on a dedicated BUS with a limited bottleneck, data transfer between the CPU and GPU can be a huge limitation in GPU based computation. Indeed, this is often "the" limiting factor for GPU computations. Matpix attempts to resolve this limitation by minimizing data transfers as much as possible. Data is transferred from the CPU to GPU only when scalar constants are assigned to array elements. Data is transferred from the GPU to the CPU only when the output is required for print out (by the "print" function). The result of all computations are kept on the GPU by copying the contents of the frame buffer to texture memory so that the result of fragment program computation are available for the next operation.

5.5 The Complete System

The Matpix compiler consists of a compilation environment and a run-time environment. The compiler consist of a front-end which processes Matpix source files, and a back end which processes compiled Matpix source files into an executable. The compiler front-end consists of the familiar lexer, parser, and tree walker. Together, these components perform syntactic and static semantic error checking on Matpix source programs, and produces a C++ source file output for error-free programs. The back-end consists of the Matpix C++ standard library, which manages GPU/CPU communication via OpenGL, and performs dynamic semantic analysis on matrix computations. The Matpix C++ standard library also produces fragment program source files which are sent to the GPU for compilation and linking. The runtime environment consist of the familiar C++ run-time, and the OpenGL run-time, which communicates to the GPU via the AGP/PCI-Express BUS. The OpenGL run-time is responsible for initializing and activating fragment programs, and delegating data transfer between the CPU and GPU.

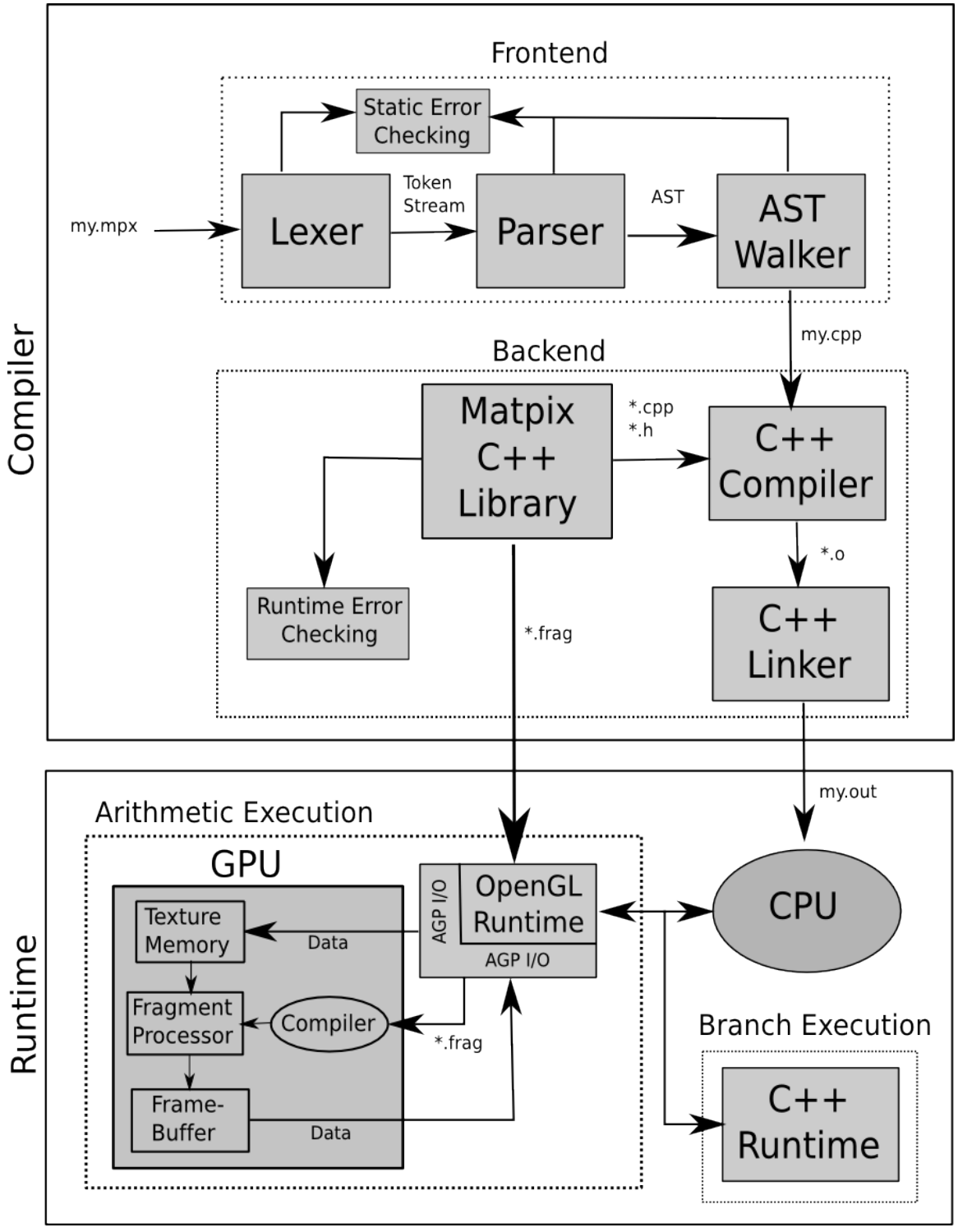

Figure 5.1: Matpix compiler block diagram

Chapter 6

Testing Plan

6.1 Test Plan

All tests can be executed with the command `make test`. The testing framework includes a Makefile, a python script, and a set of MatPix source files (extension .mpx). The two main scripts are:

File	Description	Authors
Makefile	compile MatPix and C++ code, execute run.py	Ollie, Rob
run.py	compare error logs, execute binaries, compare results	Ollie, Rob

6.1.1 Test Suite

The set of tests we used to test our language checks functionality and errors. Errors could occur in the lexer, parser, tree walker, or at runtime. We will consider lexer, parser, and tree walker errors to be compile time errors.

Successful Tests

The tests for functionality are:

File	Description	Authors
arithmetic.mpx	basic arithmetic	Ben
constmatrix.mpx	create a constant matrix	Ollie
ctrlFlow.mpx	types of control flow - if, else, while, for	Ben
dotprod.mpx	dot product matrix operations	Ollie
functiondef.mpx	function definitions	Rob
gauss_jordan.mpx	gauss jordan algorithm	David
logictest.mpx	boolean operations	Ben
recursive.mpx	recursive factorial function	Rob
regression.mpx	regression	David
sliceAssign.mpx	assign to a slice of a matrix	Ben
sum.mpx	sum of vector using dot product	David
transpose.mpx	transpose a matrix	David

These tests represent all of the functionality of MatPix. We test scalar and matrix basic mathematical operations, function definitions and calls, all types of control flow, and assignment to a slice or portion of

a matrix. We also demonstrate that one can write advanced algorithms that require linear algebra such as linear regression in MatPix.

Compiler Error Tests

The following tests cause the MatPix-ANTLR compiler to fail, rightfully:

File	Description	Authors
lvalue_fail_m.mpx	invalid string to the left of an assignment	Rob
scope_if_fail_m.mpx	identifier is out of scope in an if statement	Ben
scope_func1_fail_m.mpx	Out of scope ID used in function	Rob
scope_func2_fail_m.mpx	ID defined in function is used in main	Rob
funcCall_notDef_fail_m.mpx	Undefined function is called	Rob
funcDef_dup_fail_m.mpx	Two functions defined with same signature	Rob
syntax_fail_m.mpx	Some syntax errors	Rob
matIndex_fail_m.mpx	Indexes a matrix before it is created	David

The above tests were chosen to test all the places the compiler could fail. We allow all expressions to appear on the left hand side of an assignment in the parser, so we catch invalid left-values as an error in the tree walker. Scope functionality is handled uniformly for all new scopes (function definitions, control flows, stand alone blocks of code); thus, it is sufficient to test that scope operates as expected in any one of these types of blocks. Syntax errors are caught by the parser and/or lexer and are reported. If any error is found during MatPix compilation, the c++ file is not generated.

Runtime Error Tests

The following tests fail at runtime:

File	Description	Authors
badDim_add_fail_r.mpx	dimensions do not match in addition operation	Rob
badDim_dotprod_fail_r.mpx	Inner dimensions do not match in dot product operation	David
range_inv_row_fail_r.mpx	Invalid range from high to low in row	Rob
range_inv_col_fail_r.mpx	Invalid range from high to low in column	Rob
range_oob_col_fail_r.mpx	Range out of bounds in column slicing	Ben
range_oob_row_fail_r.mpx	Range out of bounds in row slicing	Ben
range_neg_fail_r.mpx	Range is negative	Ben

The above tests were chosen to test anywhere we expect a runtime error might occur. Since we allow matrices to be redefined with different sizes, we must verify matrix dimensions at runtime. We need to check matrix dimensions in mathematical operations and when indexing a matrix. When indexing a matrix, the ranges must be within range and in increasing order. That is, the range 1:3 is valid but the range 3:1 is not.

6.1.2 Test Automation

Testing is completely automated and is handled by `Makefile` and `run.py`. `Makefile` follows:

Makefile

```
# the c++ compilation rules
SHELL=/bin/bash
```

```
CC = g++
INCLUDES = -I../
CFLAGS = -O -c
LDFLAGS = -framework OpenGL -framework GLUT -lglew

# the MPIX compilation paths for java execution
CLASSPATH = /Applications/Eclipse/plugins/org.antlr_2.7.6/antlr.jar:../bin

# c++ matpix library files
MPX_SRC = ../MatPix/util.cpp ../MatPix/PBuffer.cpp
MPX_OBJS = ${MPX_SRC:%.cpp=%.o}

# Successful and runtime-error tests go here
TESTS = badDim_dotprod_fail_r.mpx range_inv_row_fail_r.mpx range_inv_col_fail_r.mpx range_oob_col_fail_

# mpx failures (scoping, etc) go here
TESTS_MPX_FAIL = constmatrix_notScalar_fail_m.mpx matIndex_fail_m.mpx syntax_fail_m.mpx constmatrix_fai
TEST_MPX_FAIL_CPP = ${TESTS_MPX_FAIL:%.mpx=cpp/%.cpp}

# the intermediate files follow from substitution rules
TEST_CPP = ${TESTS:%.mpx=cpp/%.cpp}
TEST_OBJS = ${TESTS:%.mpx=cpp/%.o}
TEST_EXEC = ${TESTS:%.mpx=%}
TEST_EXEC2 = ${TESTS:%.mpx=bin/%}
TEST_FILES = ${TEST_CPP} ${TEST_OBJS} ${TEST_EXEC2}

# the log files for each compilation step and running the tests
LOG_TEST_CPP = ${TESTS:%.mpx=cpp/log/%.cpp.log}
LOG_TEST_OBJS = ${TESTS:%.mpx=cpp/log/%.o.log}
LOG_TEST_EXEC = ${TESTS:%.mpx=bin/log/%.log}
LOG_TEST_FILES = ${LOG_TEST_CPP} ${LOG_TEST_OBJS} ${LOG_TEST_EXEC}

# this line says not to delete the cpp files
# which are "intermediate" files
.SECONDARY:  ${TEST_CPP}

# the main make rule - i.e. what happens if you just type "make"
all: ${TEST_EXEC} test_fail

# the rule to create .cpp files from .mpx files
# note - the rule assumes the input files are in the current dir
# and the output files go in the cpp/ directory
cpp/%.cpp: mpx/%.mpx
        @echo "trying to MPIX compile " $@ " from " $<
        java -cp ${CLASSPATH} Main $< $@ $(HAS_GPU) 2> $<.log
        mv $<.log mpx/log/

# the rule to create .o files from .cpp files
# note - the rule assumes the input and output files are in the same dir
%.o: %.cpp
```

```
        @echo "trying to create object file " $@ " from " $<
        g++ ${INCLUDES} ${CFLAGS} -o $@ $< 2> $<.log
        mv $<.log cpp/log/

# the rule to create the executables
# note - the rule assumes the input files are in the cpp/ dir
# and the output files go in the bin/ directory
${TEST_EXEC} : ${TEST_OBJS} ${MPX_OBJS}
        @echo "trying to create executable " $@ " from " $@.o
        g++ -o bin/$@ ${LDFLAGS} cpp/$@.o ${MPX_OBJS}

clean:
        rm -f ${TEST_FILES} ${LOG_TEST_FILES} ${MPX_OBJS}
        @echo "all cleaned up!"

# this will run all the tests
# print out the log info
test_fail:  ${TEST_MPX_FAIL_CPP}
test : ${TEST_EXEC} test_fail
#       @echo "running executable " $@
#       bin/$@ > bin/log/$@.log
#       ${foreach tests, ${TEST_EXEC}, ${shell echo "bin/${tests} > bin/log/${tests}.log" } }
#       ${foreach tests, ${TEST_EXEC}, ${shell bin/${tests} > bin/log/${tests}.log } }
        bin/run.py
```

The `Makefile` compiles all the code and stores error logs aside for later comparison. Executing `make test` causes all the code to be compiled and executed via `run.py`. The python script tests the result of each test listed above: successful tests, MatPix-ANTLR compiler errors, and runtime errors. Each test result's error and standard output is compared with expected error and standard output for that test. The expected output for each test is stored in `bin/pass/`. If a test has the suffix `_fail_m.mpx` then its "pass log" will be the standard error reported by the MatPix-ANTLR compiler. If a test has the suffix `_fail_r.mpx` then its pass log is the standard error reported upon execution. At the end of each test, if the pass log and the expected output are not identical, an error is reported for that test.

run.py

```
#!/usr/bin/python

import os, difflib, sys, re

def isExec(name):
        if (name == "run.py" or name == "log" or name == ".svn" or name == "pass"):
                return False
        return True

print os.getcwd()
#filenames = os.listdir(os.getcwd()+"/bin")
stdin, stdout, stderr = os.popen3('cat Makefile | egrep "^TESTS" | cut -d "=" -f2 | tr " " "\n" | sed "s
files = stdout.read()
filenames = files.splitlines()

stdin, stdout, stderr = os.popen3('cat Makefile | egrep "^TESTS_MPX_FAIL" | cut -d "=" -f2 | tr " " "\n
```

```
fail_mpx_files = stdout.read()
fail_mpx_filenames = fail_mpx_files.splitlines()

fail_mpx_filenames.extend(filenames)

for filename in filenames:

        if isExec(filename):
                is_fail_mpxTest = re.search("_fail_m$", filename)
                is_fail_runTest = re.search("_fail_r$", filename)

                passfile = None
                pass_string = None
                passfile_lines = None
                try:
                        passfile = open("bin/pass/"+filename+".log")
                        pass_string = passfile.read()
                        passfile.close();
                        passfile = open("bin/pass/"+filename+".log")
                        passfile_lines = passfile.readlines()
                except IOError:
                        print "no passing to compare with file for ", filename
                        continue

                desired_mpx_err = None
                desired_cpp_err = None
                desired_run_err = None

                if is_fail_mpxTest:
                        desired_mpx_err = pass_string
                elif is_fail_runTest:
                        desired_mpx_err = ""
                        desired_cpp_err = ""
                        desired_run_err = pass_string
                else:
                        desired_mpx_err = ""
                        desired_cpp_err = ""
                        desired_run_err = ""


                # first see if the test compiled ok
                mpxLog = open("mpx/log/"+filename+".mpx.log")
                log = mpxLog.read()
                mpxLog.close()
                if not log == desired_mpx_err:
                        print "MATPIX DID NOT COMPILE AS EXPECTED (ERROR): ", filename
                        lines =  log.splitlines()
                        for line in lines:
                                print line
                        break
                else:
```

```
        print "MATPIX COMPILED AS EXPECTED (SUCCESSFUL): ", filename

if is_fail_mpxTest:
        continue

# first see if we matpix compiled ok
cppLog = open("cpp/log/"+filename+".cpp.log")
log = cppLog.read()
cppLog.close()
if not log == desired_cpp_err:
        print "CPP DID NOT COMPILE AS EXPECTED (ERROR): ", filename
        lines =  log.splitlines()
        for line in lines:
                print line
        break
else:
        print "CPP COMPILED AS EXPECTED (SUCCESSFUL): " , filename

print "running ", filename
stdin, stdout, stderr = os.popen3("bin/"+filename)

err = stderr.read()
if not err == desired_run_err:
        print "RUNTIME STDERR NOT AS EXPECTED (ERROR): ", filename
        lines =  err.splitlines()
        for line in lines:
                print line
else:
        print "RUNTIME STDERR AS EXPECTED (SUCCESSFUL): ", filename

if is_fail_runTest:
        continue

logfile = open("bin/log/"+filename+".log", "w")
loglines = stdout.readlines()

if passfile:
        d = difflib.Differ()
        ## do a diff with the passing log files
        result = list(d.compare(passfile_lines, loglines))
        ## find lines in the compare that start with +/-
        match = re.search('^(-|\+)', "\n".join(result), re.M)
        if match:
                print "ERROR IN COMPUTATION FOR: ", filename
                print "#####################################################"
                print "############  DIFF RESULT   #########################"
                print "#####################################################"
                sys.stdout.writelines(result)
                print "#####################################################"
                print "#####################################################"
                logfile.writelines(result)
```

```
                else:
                        print "RUNNNING ", filename, " PASSED!!"
                        print "#####################################################"
                        logfile.writelines(loglines)
        else:
                logfile.writelines(loglines)
```

6.1.3 Example tests

Three tests that demonstrate a good amount of our language are Sum, Gauss Jordan, and Slice Assign.

Sum: tests/mpx/sum.mpx

The MatPix code for Sum follows.

```
// summing test

print("A = [1, 2 , 3, 4, 5]");
A = [1, 2 , 3, 4, 5];

print("matrix B[5, 1] = 1");
matrix B[5, 1] = 1;

// sum over A
print("sum over A using A * B");
sum = A * B;
print("sum = ", sum);

// sum using for loop
print("sum using for loop 1..5");
sum = 0;
for ( i=1:5 ) {
        sum = sum + i;
}
print("sum = ", sum);
```

Sum: tests/cpp/sum.cpp

The generated C++ code for Sum follows.

```
#include "MatPix/matpix.hpp"

int main(){

   bool forceSoftwareRender=true;;
   initMatPix(forceSoftwareRender);

   cout << "A = [1, 2 , 3, 4, 5]" << endl;
   vector<float> vec_1;
   float array_1[] = {1,2,3,4,5};
   copy(array_1, array_1+5, back_inserter(vec_1));
   Matrix MP_A(0);
   MP_A.assign(vec_1, 1, 5);
   cout << "matrix B[5, 1] = 1" << endl;
   Matrix MP_B(5, 1);
   MP_B.assign(1);
   cout << "sum over A using A * B" << endl;
   Matrix MP_sum(0);
   MP_sum.assign(MP_A.dot(MP_B));
   cout << "sum = " << MP_sum;
   cout << "sum using for loop 1..5" << endl;
   MP_sum.assign(0);
   for (Iter MP_i=1;MP_i<=5;++MP_i)  {
      MP_sum.assign(MP_sum.add(MP_i));
   }

   cout << "sum = " << MP_sum;

   cleanupMatPix();
```

```
   return 0;
}
```

Sum: tests/bin/pass/sum.log

The output of Sum follows.

```
A = [1, 2 , 3, 4, 5]
matrix B[5, 1] = 1
sum over A using A * B
sum = 15
sum using for loop 1..5
sum = 15
```

Gauss Jordan: tests/mpx/gauss_jordan.mpx

The MatPix code for Gauss Jordan follows.

```
function eye(m)
{
    matrix Z[m,m];
    for(i=0:m-1)
    {
        Z[i,i] = 1;
    }
    return Z;
};

function gauss_jordan(V,n) //expand to use size function
{
    matrix Q[n,n*2];
    matrix E[n,n];
    E=eye(n);

    //Set Q to V augmented with nxn identity
    Q[:, 0:n-1] = V;
    Q[:, n:n*2-1] = E;
    r = 0;
    temp=0;//swap storage
    for (i=0:n-1)  //for each row
    {
        r = i;
        //Check for zero in eliminator row
        //If zero, then find row to swap with
        while((Q[i,i] ==0) && (r !=n-1))
        {
            r = r+1;
            temp = Q[i,:];
            Q[i,:] = Q[r,:];
            Q[r,:] = temp;
        }
        //Normalize the eliminator row
        if (Q[i,i]!=0)
        {
                Q[i,:] = Q[i,:]./Q[i,i];
        }

                //Use Eliminator row to eliminate
                //the column of all rows beneath
        for (k=i+1:n-1)
        {
            Q[k,:] = Q[k,:] - Q[i,:] .* Q[k,i];
        }
    }
    for (i=n-1:-1:0)
    {
        for (k = i-1:-1:0)
        {
            Q[k,:] = Q[k,:] - Q[i,:] .* Q[k,i];
        }
    }
    return Q[:,n:n*2-1];
};

d = 16;
matrix A[d, d];
A = eye(d);
```

```
A = A*5;
A[8,9]=4;
A[14,1] = 5;
A[15,1] = 15;


print("A:");
print(A);
print("inv(A):");
inverted = gauss_jordan(A,d);
print("Inverted matrix:");
print(inverted);
```

Gauss Jordan: tests/cpp/gauss_jordan.cpp

The generated C++ code for Gauss Jordan follows.

```
#include "MatPix/matpix.hpp"

Matrix eye (Matrix MP_m)  {
   Matrix MP_Z(MP_m, MP_m);
   for (Iter MP_i=0;MP_i<=MP_m.sub(1).getFloatValue();++MP_i)  {
      MP_Z.sliceAssign(RangeList(MP_i.getFloatValue()),RangeList(MP_i.getFloatValue()),1);
   }

   return MP_Z;
}

Matrix gauss_jordan (Matrix MP_V,Matrix MP_n)  {
   Matrix MP_Q(MP_n, MP_n.mul(2));
   Matrix MP_E(MP_n, MP_n);
   MP_E.assign(eye(MP_n));
   MP_Q.sliceAssign(RangeList(),RangeList(0,MP_n.sub(1).getFloatValue()),MP_V);
   MP_Q.sliceAssign(RangeList(),RangeList(MP_n.getFloatValue(),MP_n.mul(2).sub(1).getFloatValue()),MP_E);
   Matrix MP_r(0);
   MP_r.assign(0);
   Matrix MP_temp(0);
   MP_temp.assign(0);
   for (Iter MP_i=0;MP_i<=MP_n.sub(1).getFloatValue();++MP_i)  {
      MP_r.assign(MP_i);
      while ((MP_Q.slice(RangeList(MP_i.getFloatValue()),RangeList(MP_i.getFloatValue()))==0&&MP_r!=MP_n.sub(1)).getFloatValue())  {
         MP_r.assign(MP_r.add(1));
         MP_temp.assign(MP_Q.slice(RangeList(MP_i.getFloatValue()),RangeList()));
         MP_Q.sliceAssign(RangeList(MP_i.getFloatValue()),RangeList(),MP_Q.slice(RangeList(MP_r.getFloatValue()),RangeList()));
         MP_Q.sliceAssign(RangeList(MP_r.getFloatValue()),RangeList(),MP_temp);
      }

      if ((MP_Q.slice(RangeList(MP_i.getFloatValue()),RangeList(MP_i.getFloatValue()))!=0).getFloatValue())  {
         MP_Q.sliceAssign(RangeList(MP_i.getFloatValue()),RangeList(),MP_Q.slice(RangeList(MP_i.getFloatValue()),RangeList()).div(MP_Q.slice(Range
      }

      for (Iter MP_k=MP_i.add(1).getFloatValue();MP_k<=MP_n.sub(1).getFloatValue();++MP_k)  {
         MP_Q.sliceAssign(RangeList(MP_k.getFloatValue()),RangeList(),MP_Q.slice(RangeList(MP_k.getFloatValue()),RangeList()).sub(MP_Q.slice(Range
      }

   }

   for (Iter MP_i=MP_n.sub(1).getFloatValue();MP_i>=0;MP_i+=Matrix(1).negate().getFloatValue())  {
      for (Iter MP_k=MP_i.sub(1).getFloatValue();MP_k>=0;MP_k+=Matrix(1).negate().getFloatValue())  {
         MP_Q.sliceAssign(RangeList(MP_k.getFloatValue()),RangeList(),MP_Q.slice(RangeList(MP_k.getFloatValue()),RangeList()).sub(MP_Q.slice(Range
      }

   }

   return MP_Q.slice(RangeList(),RangeList(MP_n.getFloatValue(),MP_n.mul(2).sub(1).getFloatValue()));
}


int main(){

   bool forceSoftwareRender=true;;
   initMatPix(forceSoftwareRender);

   Matrix MP_d(0);
   MP_d.assign(16);
   Matrix MP_A(MP_d, MP_d);
   MP_A.assign(eye(MP_d));
```

```
   MP_A.assign(MP_A.mul(5));
   MP_A.sliceAssign(RangeList(8),RangeList(9),4);
   MP_A.sliceAssign(RangeList(14),RangeList(1),5);
   MP_A.sliceAssign(RangeList(15),RangeList(1),15);
   cout << "A:" << endl;
   cout << MP_A;
   cout << "inv(A):" << endl;
   Matrix MP_inverted(0);
   MP_inverted.assign(gauss_jordan(MP_A,MP_d));
   cout << "Inverted matrix:" << endl;
   cout << MP_inverted;

   cleanupMatPix();
   return 0;
}
```

Gauss Jordan: tests/bin/pass/gauss_jordan.log

The output of Gauss Jordan follows.

```
A:
5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 5, 4, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0
0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0
0, 15, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5
inv(A):
Inverted matrix:
0.2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0.2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0.2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0.2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0.2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0.2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0.2, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0.2, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0.2, -0.16, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.2, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.2, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.2, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.2, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.2, 0, 0
0, -0.2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.2, 0
0, -0.6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.2
```

Slice Assign: tests/mpx/sliceAssign.mpx

The MatPix code for Slice Assign follows.

```
// control flow tests

matrix A[16, 16];
matrix B[16, 16];
matrix C[16, 16];
matrix D[16, 16];
matrix E[16, 16];
matrix F[16, 16];
matrix H[16, 16];

// various slice assignments from constants
A[1, 2] = -1.1;
B[:, :] = 5.1;
C[0, :] = 0.5;
D[:, 0] = 0.75;
E[0, 0:5] = 1.5;
F[0:5, 0] = 1.75;
```

```
H[0:5, 0:5] = 2.5;

// shouldn't compile
//F[1:10] = 3;

print("A[1, 2] = -1.1");
print(A);
print("B[:, :] = 5.1");
print(B);
print("C[0, :] = 0.5");
print(C);
print("D[:, 0] = 0.75");
print(D);
print("E[0, 0:5] = 1.5");
print(E);
print("F[0:5, 0] = 1.75");
print(F);
print("H[0:5, 0:5] = 2.5");
print(H);

// now some slice to slice assignments
H[1, 2] = A[1, 2];
print("H[1, 2] = A[1, 2]");
print(H);

H[0, :] = C[0, :];
print("H[0, :] = C[0, :]");
print(H);

H[:, 0] = D[:, 0];
print("H[:, 0] = D[:, 0]");
print(H);

H[0, 0:5] = E[0, 0:5];
print("H[0, 0:5] = E[0, 0:5]");
print(H);

H[6:11, 1] = E[0, 0:5]';
print("H[6:11, 1] = E[0, 0:5]'");
print(H);

H[0:5, 0] = F[0:5, 0];
print("H[0:5, 0] = F[0:5, 0]");
print(H);

H[0:5, 0:5] = B[0:5, 0:5];
print("H[0:5, 0:5] = B[0:5, 0:5]");
print(H);

H[:, :] = B[:, :];
print("H[:, :] = B[:, :]");
print(H);
```

Slice Assign: tests/cpp/sliceAssign.cpp

The generated C++ code for Slice Assign follows.

```
#include "MatPix/matpix.hpp"

int main(){

   bool forceSoftwareRender=true;;
   initMatPix(forceSoftwareRender);

   Matrix MP_A(16, 16);
   Matrix MP_B(16, 16);
   Matrix MP_C(16, 16);
   Matrix MP_D(16, 16);
   Matrix MP_E(16, 16);
   Matrix MP_F(16, 16);
   Matrix MP_H(16, 16);
   MP_A.sliceAssign(RangeList(1),RangeList(2),Matrix(1.1).negate());
   MP_B.sliceAssign(RangeList(),RangeList(),5.1);
   MP_C.sliceAssign(RangeList(0),RangeList(),0.5);
   MP_D.sliceAssign(RangeList(),RangeList(0),0.75);
   MP_E.sliceAssign(RangeList(0),RangeList(0,5),1.5);
   MP_F.sliceAssign(RangeList(0,5),RangeList(0),1.75);
```

```
   MP_H.sliceAssign(RangeList(0,5),RangeList(0,5),2.5);
   cout << "A[1, 2] = -1.1" << endl;
   cout << MP_A;
   cout << "B[:, :] = 5.1" << endl;
   cout << MP_B;
   cout << "C[0, :] = 0.5" << endl;
   cout << MP_C;
   cout << "D[:, 0] = 0.75" << endl;
   cout << MP_D;
   cout << "E[0, 0:5] = 1.5" << endl;
   cout << MP_E;
   cout << "F[0:5, 0] = 1.75" << endl;
   cout << MP_F;
   cout << "H[0:5, 0:5] = 2.5" << endl;
   cout << MP_H;
   MP_H.sliceAssign(RangeList(1),RangeList(2),MP_A.slice(RangeList(1),RangeList(2)));
   cout << "H[1, 2] = A[1, 2]" << endl;
   cout << MP_H;
   MP_H.sliceAssign(RangeList(0),RangeList(),MP_C.slice(RangeList(0),RangeList()));
   cout << "H[0, :] = C[0, :]" << endl;
   cout << MP_H;
   MP_H.sliceAssign(RangeList(),RangeList(0),MP_D.slice(RangeList(),RangeList(0)));
   cout << "H[:, 0] = D[:, 0]" << endl;
   cout << MP_H;
   MP_H.sliceAssign(RangeList(0),RangeList(0,5),MP_E.slice(RangeList(0),RangeList(0,5)));
   cout << "H[0, 0:5] = E[0, 0:5]" << endl;
   cout << MP_H;
   MP_H.sliceAssign(RangeList(6,11),RangeList(1),MP_E.slice(RangeList(0),RangeList(0,5)).transpose());
   cout << "H[6:11, 1] = E[0, 0:5]'" << endl;
   cout << MP_H;
   MP_H.sliceAssign(RangeList(0,5),RangeList(0),MP_F.slice(RangeList(0,5),RangeList(0)));
   cout << "H[0:5, 0] = F[0:5, 0]" << endl;
   cout << MP_H;
   MP_H.sliceAssign(RangeList(0,5),RangeList(0,5),MP_B.slice(RangeList(0,5),RangeList(0,5)));
   cout << "H[0:5, 0:5] = B[0:5, 0:5]" << endl;
   cout << MP_H;
   MP_H.sliceAssign(RangeList(),RangeList(),MP_B.slice(RangeList(),RangeList()));
   cout << "H[:, :] = B[:, :]" << endl;
   cout << MP_H;

   cleanupMatPix();
   return 0;
}
```

Slice Assign: tests/bin/pass/sliceAssign.log

The output of Slice Assign follows.

```
A[1, 2] = -1.1
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, -1.1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
B[:, :] = 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
```

```
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
C[0, :] = 0.5
0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
D[:, 0] = 0.75
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
E[0, 0:5] = 1.5
1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
F[0:5, 0] = 1.75
1.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
1.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
1.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
1.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
1.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
1.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
H[0:5, 0:5] = 2.5
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```

```
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
H[1, 2] = A[1, 2]
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
2.5, 2.5, -1.1, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
H[0, :] = C[0, :]
0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5
2.5, 2.5, -1.1, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
H[:, 0] = D[:, 0]
0.75, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5
0.75, 2.5, -1.1, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
H[0, 0:5] = E[0, 0:5]
1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5
0.75, 2.5, -1.1, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```

```
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
H[6:11, 1] = E[0, 0:5]'
1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5
0.75, 2.5, -1.1, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
H[0:5, 0] = F[0:5, 0]
1.75, 1.5, 1.5, 1.5, 1.5, 1.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5
1.75, 2.5, -1.1, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
1.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
1.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
1.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
1.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
H[0:5, 0:5] = B[0:5, 0:5]
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
H[:, :] = B[:, :]
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
```

Chapter 7

Lessons Learned

7.1 Lessons Learned

During the course of our project, we came to appreciate the following lessons, which we enumerate for the benefit of future classes.

- Avoiding nondeterminism is not a trivial affair. This became especially problematic when dealing with assignment. In particular, our language dictates that assignment can only happen to variables, which are entailed by expressions, but not to expressions in general. At first, we attempted to enforce this rule in the parser, but kept running into the problem of nondeterminism. After hours of failed solutions, we decided that this problem was better left to static semantic analysis, which worked as desired. So perhaps the lesson learned in this case would be that not all nondeterminism issues can be solved in the parser; some are better left to semantic analysis.

- A primary concern of any matrix-based language should be that of static versus dynamic sizing of vectors and matrices. Our original intention was to enforce static array sizes, so as to allow for static semantic analysis of matrix indexing. This would mean that all bounds checking would occur at compile time. Unfortunately, when we attempted to write our matrix inversion function, we soon realized that even simple tasks would be impossible without dynamically sized arrays. Thus, all of our static semantic analysis was thrown out the window, and we implemented runtime bounds checking.

- When dealing with proprietary architectures, one should be especially mindful of semantics. Very late into our project, we realized that the indexing format used by OpenGL was the opposite of the traditional indexing format used in mathematics, which is $row \times column$. This incongruity required many hours of work to reconcile it.

- We also found it useful to set meeting times and dates in order to facilitate regular meetings. Although we did not know what we needed to accomplish at first, it was useful to meet, brainstorm, and get everyone's mind on the project.

Chapter 8

Future Work

To test the performance of Matpix, a set of identical programs were written both in Matpix and python and computation speed was compared. Each program consists of an initialization of two matrices, an assignment to all their elements, and a loop performing a binary operation where the result is assigned to one of the matrices. Both the size of the two matrices, and the number of iterations the operations were varied in the test suite. The tests were run on a dual core iMac with a ATI x800 graphics card. As is evident in the add, subtract, multiply and divide plots below, Matpix shows strong performance gains for atomic arithmetic operations. For these operations, computation speed decreases exponentially with greater number of iterations, and the performance of Matpix is more than a factor of two greater than python for sufficiently large matrices.

Much optimization is still required before Matpix can utilize the full computational power of the GPU. As an example, consider the performance of an example single purpose GPGPU program the authors tested on their home machine [**?**]. The program consisted of 100 iterations of an atomic MAD operation performed on two 1,000,000 element arrays, and was 50 times faster when run on the GPU than when using a standard C program. In contrast, a Matpix program performing 128 iterations of a multiply operation on two 1024x1024 matrices is only 2.5 faster than a python program. From this comparison it is evident that significant optimization is required before Matpix can achieve the performance of fixed function GPGPU programs written directly in C using OpenGL.

Once atomic operations are optimized, load balancing data transferring between the CPU and GPU can also be introduced to further achieve optimal execution speeds. The authors considered several possible methods for achieving this that were not implemented. When data transfer to and from the GPU does not outweigh the increase in computational efficiency, data transfer should be omitted, and computations should be performed on the CPU. In addition, asymmetry between GPU memory read and write speeds dictates that as many computations as possible should be performed inside each fragment program to minimize the number of write operations. Currently Matpix only performs one operation per fragment program for all operations but the dot product. The authors expected that this operation in particular would demonstrate significant speed improvements since it specifically avoids extraneous write operations. The figures below show that despite this expectation the Matpix dot product operator still performs well below that of python. The authors speculate that a performance increase was not demonstrated because the GPU code that is currently used in the Matpix backend is does not use the optimal OpenGL feature set which allows the computation to be shared by all of the fragment processors on the GPU.

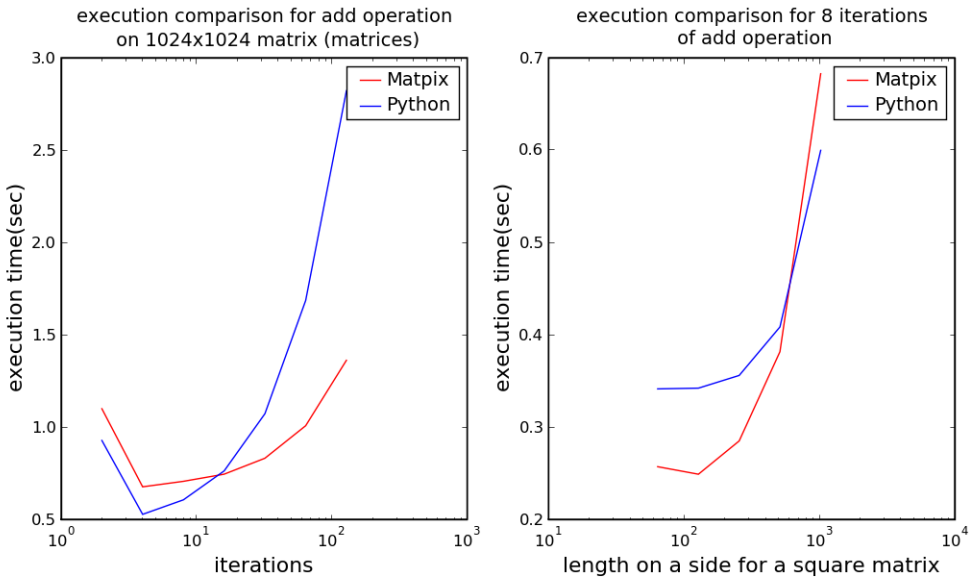

Figure 8.1: Performance of atomic add operation

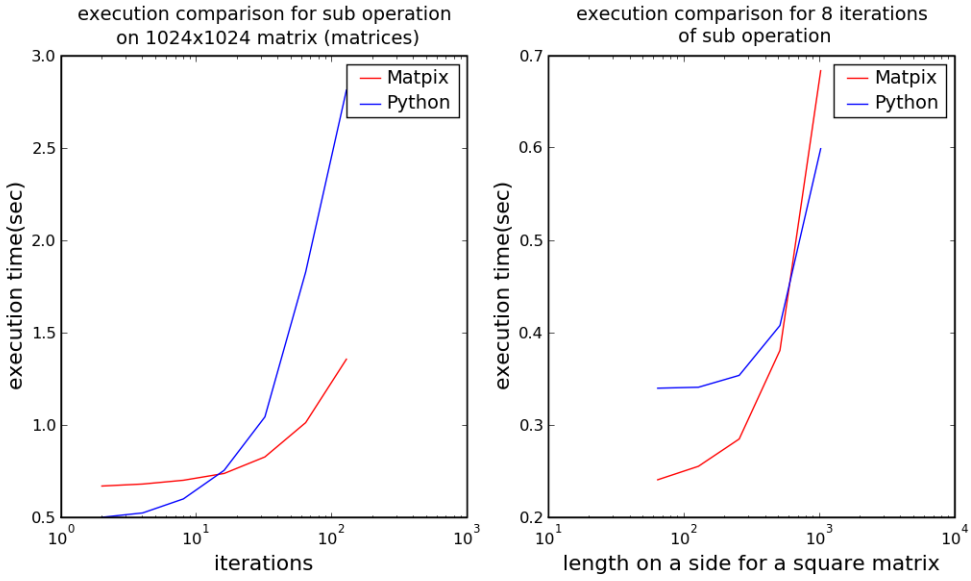

Figure 8.2: Performance of atomic subtract operation

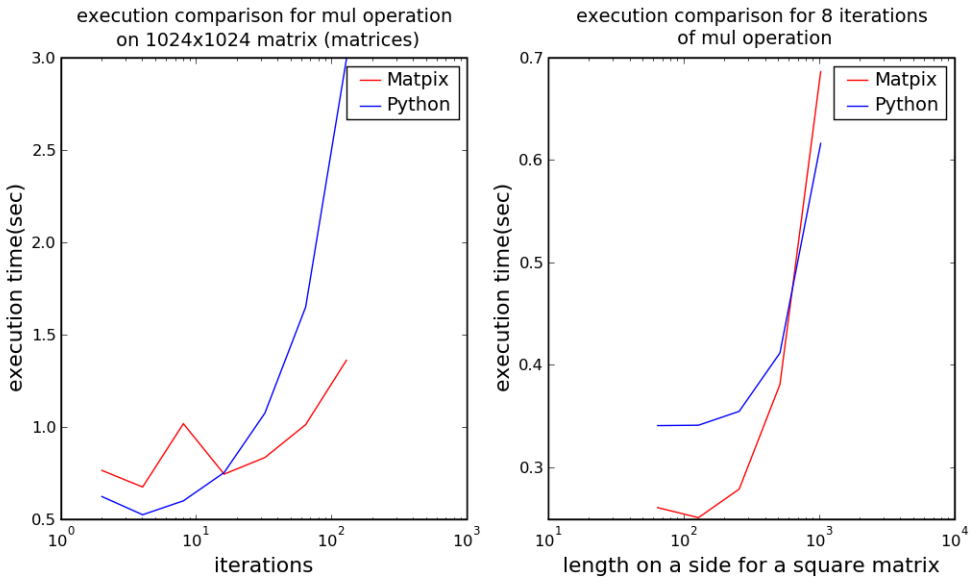

Figure 8.3: Performance of atomic multiply operation

Figure 8.4: Performance of atomic divide operation

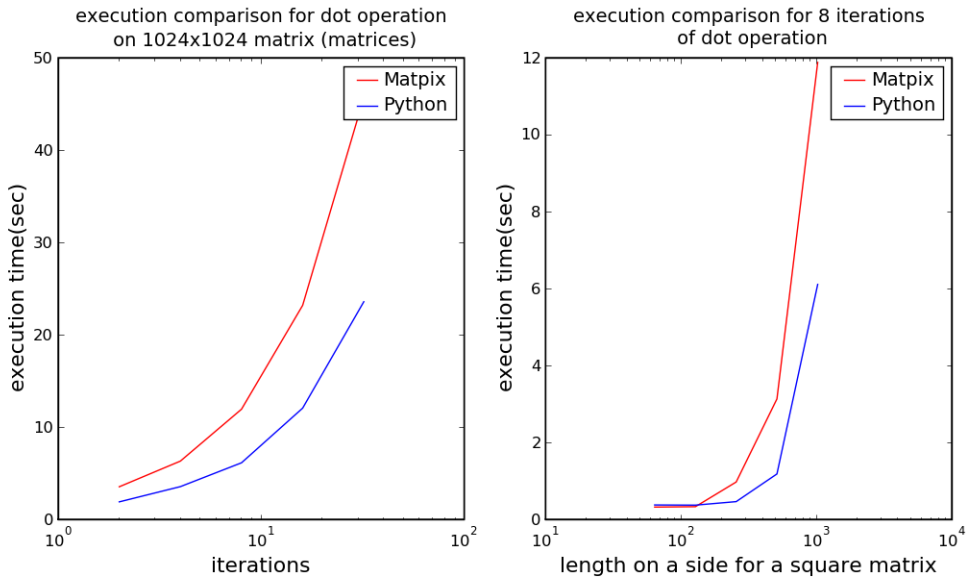

Figure 8.5: Performance of atomic dot product operation

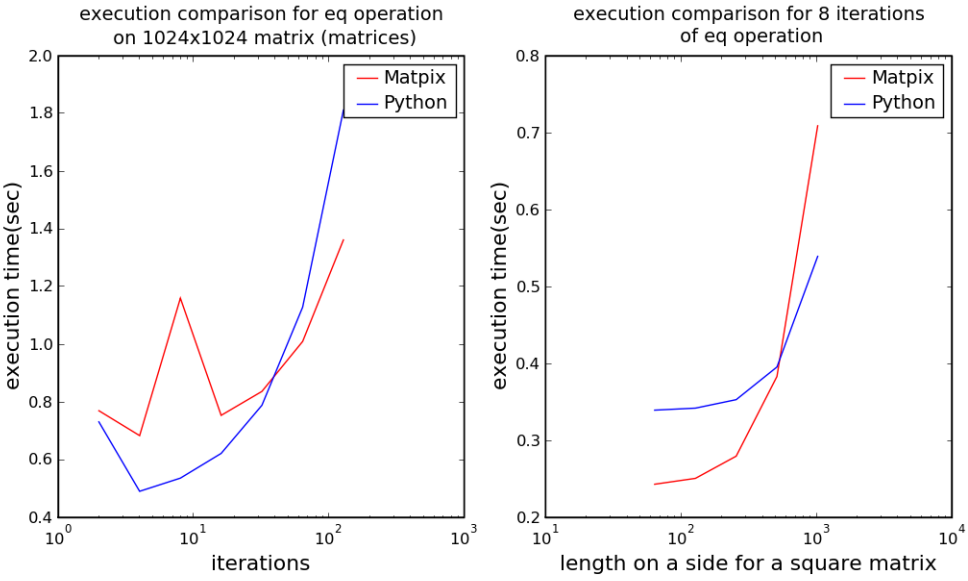

Figure 8.6: Performance of atomic "is equal" test

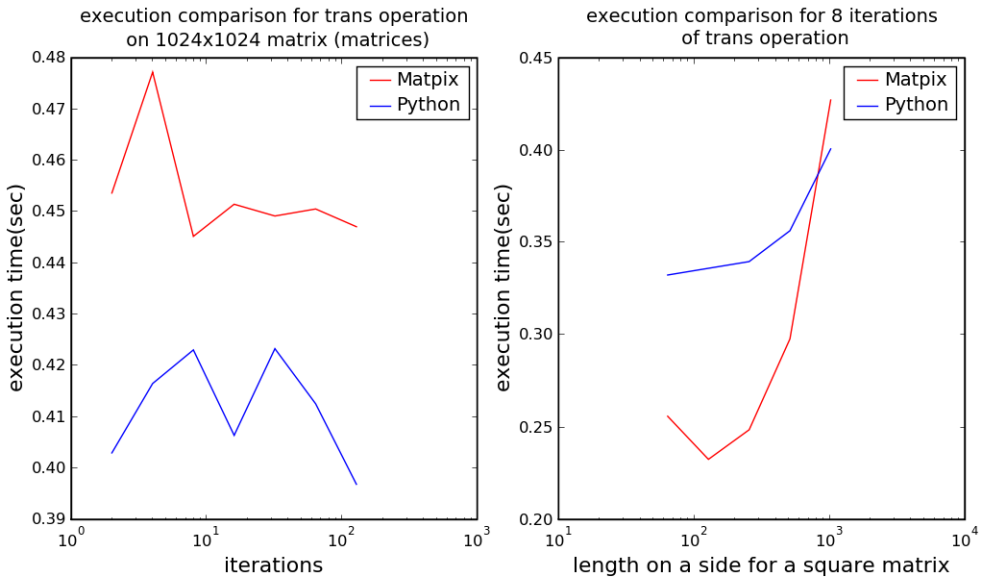

Figure 8.7: Performance of atomic transpose operation

Bibliography

[1] Matlab, http://www.mathworks.com/

Appendix A

Syntax Overview

Here we give a succinct overview of the syntax of MatPix for user reference.

A.1 Lexer Grammar

```
PLUS   : ‘+’ ;
MINUS  : ‘-’ ;
MUL    : ".*";
DIV    : "./";
MOD        : ‘%’ ;
ASSIGN : ‘=’ ;
SEMI   : ‘;’ ;
DOTPROD: ‘*’ ;
COMMA  : ’,’ ;
COLON  : ‘:’ ;
POUND  : ‘#’ ;

LBRACKET : ‘[’ ;
RBRACKET : ‘]’ ;
LSCOPE : ‘{’ ;
RSCOPE : ‘}’ ;
ROWSEP : ‘|’ ;

NOT : ‘!’;
AND : "&&";
OR : "||";
LOGGT : ‘>’;
LOGLT : ‘<’;
LOGGTE : ">=";
LOGLTE : "<=";
LOGEQ : "==";
LOGNEQ : "!=";
LPAREN : "(" ;
RPAREN : ")" ;
INC : "++";
DEC : "--";
TRANS : "’";
```

```
LETTER : ( 'a'..'z' | 'A'..'Z' ) ;
DIG : '0'..'9' ;
INT : (DIG)+ ;
NUMBER : INT ('.' INT)? | '.' INT;

EXPON : ('e' | 'E') (MINUS)? INT ;
FLOAT : NUMBER (EXPON)?                          ;

SL_COMMENT : "//" (~'\n')*;
ML_COMMENT : "/*" .* "*/";

ID : LETTER (LETTER | DIG | '_')* ;

SCALAR : (MINUS FLOAT) | FLOAT ;

STRING :  '"' ( '"' '"' | ~('"'))*  '"';

WS  :   ( ' ' | '\t' | '\n' | '\r' );
```

A.2 Parser Grammar

```
program : (stmt)+ EOF;
constant : SCALAR       | constmatrix | STRING ;
arraySlice : LBRACKET rangeList RBRACKET;
rangeList : range COMMA range;
range : (expr (COLON expr (COLON  expr )?)?) | COLON ;
constmatrix : LBRACKET (row)+ RBRACKET;
row : cols (ROWSEP)?;
cols : SCALAR (COMMA SCALAR)*;
matrixdecl : "matrix" ID (LBRACKET expr COMMA expr RBRACKET)?;
stmt : expr SEMI | functionDefinition SEMI | controlflow | "return" expr SEMI |
        SEMI | LSCOPE (stmt)+ RSCOPE;
expr : matrixdecl (ASSIGN logicExpr)? | logicExpr (ASSIGN logicExpr)?;
logicExpr : predExpr ((AND | OR) logicExpr)?;
predExpr : ((NOT)? addExpr) ((LOGGT|LOGLT|LOGGTE|LOGLTE|LOGEQ|LOGNEQ) predExpr)?;
addExpr : mulExpr ( (PLUS | MINUS) mulExpr )*;
mulExpr : signExpr ( (MUL | DIV | MOD | DOTPROD) signExpr )*;
signExpr : (MINUS)? incExpr ;
incExpr : atom (INC | DEC | TRANS)?;
atom : identifier ( arraySlice | argList)? | constant | LPAREN expr RPAREN;
identifier : ID ;
functionDefinition :  "function" ID defArgList LSCOPE newScope RSCOPE;
newScope : (stmt)*;
defArgList : LPAREN (ID (COMMA ID)* )? RPAREN;
argList : LPAREN (expr (COMMA expr)* )? RPAREN;
controlflow : (ifstmt | whilestmt | forstmt);
predicate: logicExpr;
ifstmt : "if" LPAREN predicate RPAREN LSCOPE newScope RSCOPE
        ("else" LSCOPE newScope RSCOPE )?;
whilestmt : "while" LPAREN predicate RPAREN LSCOPE newScope  RSCOPE;
```

```
forstmt: "for" LPAREN ID ASSIGN range RPAREN LSCOPE newScope RSCOPE;
```

Appendix B

Code Repository

Here we present a complete listing of all source code and testing code for our language.

B.1 Parser

B.1.1 MPLexParseWalk.g

```
/*
 * Lexer, Parser, Treewalker for Matpix
 *
 * Oliver Cossairt, Ben London, David Burkat, Robert Hawkins
 * Fall 2007
 */
////////////////////////////////////////////////////////////////////

class MPLexer extends Lexer;
options {
    testLiterals = false; // By default, don't check tokens against keywords
    k = 2;                // Need to decide when strings literals end
    charVocabulary = '\3'..'\377'; // Accept all eight-bit ASCII characters
}

{
    int error_count = 0;
    public void reportError(String s) {
        super.reportError(s);
        error_count++;
    }
    public void reportError(RecognitionException e) {
        super.reportError(e);
        error_count++;
    }
}


PLUS   : '+' ;
MINUS  : '-' ;
MUL    : ".*";
DIV    : "./";
MOD        : '%' ;
ASSIGN : '=' ;
SEMI   : ';' ;
DOTPROD: '*' ;
COMMA  : ',' ;
COLON  : ':' ;
POUND  : '#' ;

LBRACKET : '[' ;
RBRACKET : ']' ;
LSCOPE : '{' ;
RSCOPE : '}' ;
ROWSEP : '|' ;
```

```
NOT : '!';
AND : "&&";
OR : "||";
LOGGT : '>';
LOGLT : '<';
LOGGTE : ">=";
LOGLTE : "<=";
LOGEQ : "==";
LOGNEQ : "!=";
LPAREN : "(" ;
RPAREN : ")" ;
INC : "++";
DEC : "--";
TRANS : "'";

// A little unorthodox: most punctuation characters get their own rule,
// but since we're using "(" and ")" in the parser, we need parenthesis
// to match as keywords, Thus, we set testLiterals true for this rule.

protected LETTER : ( 'a'..'z' | 'A'..'Z' ) ;
protected DIG : '0'..'9' ;
protected INT : (DIG)+ ;
protected NUMBER : INT ('.' INT)?
                                | '.' INT
                                ;

protected EXPON : ('e' | 'E') (MINUS)? INT ;
protected FLOAT : NUMBER (EXPON)?
                               ;

SL_COMMENT : "//" (~'\n')* { $setType(Token.SKIP); };
ML_COMMENT : "/*" (options { greedy = false; } : .)* "*/" { $setType(Token.SKIP); };

ID
options {
    testLiterals = true;
}
    : LETTER (LETTER | DIG | '_')* ;

SCALAR : (MINUS FLOAT) | FLOAT ;

// Strings are "like this ""double quotes"" doubled to include them"
// Note that testLiterals are false so we don't have to worry about
// strings such as "if"
STRING :  '"'! ( '"' '"'! | ~('"'))*  '"'!;

WS  :   ( ' '
        | '\t'
        | '\n' { newline(); }
        | '\r'
        ) { $setType(Token.SKIP); }
    ;

//////////////////////////////////////////////////////////////////////
class MPParser extends Parser;
options {
    buildAST = true; // Enable AST building
    k = 2;           // Need to distinguish between ID by itself and ID ASSIGN
}

tokens {
  ARGS;
  ARRAY_SLICE;
  CONST_MATRIX;
  CONST_SCALAR;
  CONST_STRING;
  ELSE;
  EXPR;
  FOR;
  FOR_ITER;
  FUNC_CALL;
  FUNC_DEF;
  IF;
  IF_ELSE;
  PRED;
  PROGRAM;
  MATRIX_ROW;
```

```
  MATRIX_COL;
  MATRIX_DECLR;
  NEW_SCOPE;
  RANGE_ALL;
  RANGE_ONE;
  RANGE_TWO;
  RANGE_THREE;
  RANGE_LIST;
  RETURN;
  SIGN_MINUS;
  STATEMENT;
  VARIABLE;
  WHILE;
}


{
    int error_count = 0;
    public void reportError(String s) {
        super.reportError(s);
        error_count++;
    }
    public void reportError(RecognitionException e) {
        super.reportError(e);
        error_count++;
    }
}


program
        : (stmt)+ EOF!
        { #program = #([PROGRAM, "PROGRAM"], program); }
    ;

// a matrix consist of one or more rows
constant : SCALAR
                        {#constant = #([CONST_SCALAR, "CONST_SCALAR"], #constant);}
                 | constmatrix
                 | STRING
                        {#constant = #([CONST_STRING, "CONST_STRING"], #constant);}
                 ;

arraySlice : LBRACKET! rangeList RBRACKET!
                   ;

// max 2D matrix
rangeList : range COMMA! range
                {#rangeList = #([RANGE_LIST, "RANGE_LIST"], rangeList);}
                  ;

range! : (
                        r1:expr {#range = #([RANGE_ONE, "RANGE_ONE"], r1);}
                        (COLON! r2:expr {#range = #([RANGE_TWO, "RANGE_TWO"], r1, r2);}
                                (COLON!  r3:expr {#range = #([RANGE_THREE, "RANGE_THREE"], r1, r3);})?
                        )?
                )
                | COLON!
                        {#range = #([RANGE_ALL, "RANGE_ALL"]);}
          ;

constmatrix
                : LBRACKET! (row)+ RBRACKET!
                        {#constmatrix = #([CONST_MATRIX, "CONST_MATRIX"], constmatrix);}
                ;
// a row consists of one or more comma separated expressions
row : cols
        {#row = #([MATRIX_ROW, "MATRIX_ROW"], row);}
        (ROWSEP!)?
        ;

cols : SCALAR (COMMA! SCALAR)*
                {#cols = #([MATRIX_COL, "MATRIX_COL"], cols);}
                ;

matrixdecl : "matrix"! ID (LBRACKET! expr COMMA! expr RBRACKET!)?
                {#matrixdecl = #([MATRIX_DECLR, "MATRIX_DECLR"], matrixdecl);}
           ;
```

```
stmt : expr SEMI!
        { #stmt = #([STATEMENT, "STATEMENT"],stmt); }
         | functionDefinition SEMI!
         | controlflow
         | "return"! e:expr SEMI!
                {#stmt = #([RETURN, "RETURN"], e);}
         | SEMI!
     | LSCOPE! (stmt)+ RSCOPE!
        {#stmt = #([NEW_SCOPE, "NEW_SCOPE"], stmt);}
         ;

expr :  matrixdecl (ASSIGN^ logicExpr)?
        |       logicExpr (ASSIGN^ logicExpr)?
         ;

logicExpr : predExpr ((AND^|OR^) logicExpr)?
                  ;

predExpr : ((NOT^)? addExpr) ((LOGGT^|LOGLT^|LOGGTE^|LOGLTE^|LOGEQ^|LOGNEQ^) predExpr)?
                 ;

addExpr : mulExpr ( (PLUS^ | MINUS^) mulExpr )*
                ;

mulExpr : signExpr ( (MUL^ | DIV^ | MOD^ | DOTPROD^) signExpr )*
                ;

signExpr : (m:MINUS^ {#m.setType(SIGN_MINUS);})? incExpr ;

incExpr : atom (INC^ | DEC^ | TRANS^)?
                ;

atom
        : name:identifier {#atom = #([VARIABLE, "VARIABLE"], name);}
                (
                        slice:arraySlice! {#atom = #([ARRAY_SLICE, "ARRAY_SLICE"], atom, slice);}
                        | argList {#atom = #([FUNC_CALL, "FUNC_CALL"], name);}
                )?
        | constant
    | LPAREN! expr RPAREN!
    ;

identifier : ID ;

functionDefinition :
    "function"! ID defArgList LSCOPE! newScope RSCOPE!
        {#functionDefinition = #([FUNC_DEF, "FUNC_DEF"], #functionDefinition);}
    ;

newScope : (stmt)*
        {#newScope = #([NEW_SCOPE, "NEW_SCOPE"], #newScope);}
    ;

defArgList : LPAREN! (ID (COMMA! ID)* )? RPAREN!
        {#defArgList = #([ARGS, "ARGS"], #defArgList);}
                  ;

argList : LPAREN! (expr (COMMA! expr)* )? RPAREN!
        {#argList = #([ARGS, "ARGS"], #argList);}
        ;

controlflow : (ifstmt | whilestmt | forstmt)
                        ;

predicate: logicExpr
        {#predicate = #([PRED, "PRED"], #predicate);}
        ;

ifstmt! :
        "if"! LPAREN! pred:predicate RPAREN!
        LSCOPE! body:newScope RSCOPE!
                {#ifstmt = #([IF, "IF"], pred, body);}
        ("else"! LSCOPE! elseBody:newScope RSCOPE!
                {#ifstmt = #([IF_ELSE, "IF_ELSE"], pred, elseBody);} )?
        ;
```

```
whilestmt :
        "while"! LPAREN! predicate RPAREN!
        LSCOPE! newScope  RSCOPE!
        {#whilestmt = #([WHILE, "WHILE"], #whilestmt);}
        ;

forstmt! :
        "for"^ LPAREN! iter:ID ASSIGN^ iterRange:range RPAREN!
        LSCOPE! body:newScope RSCOPE!
                {#forstmt = #([FOR, "FOR"],
                        #([FOR_ITER, "FOR_ITER"],
                                #(ASSIGN, iter, iterRange)),
                        body);}
        ;

class MPWalker extends TreeParser;
{
        MPSymbolTable top = new MPSymbolTable(null);
        int constMatNum = 0;
}
program returns [String r] {r = ""; MPExpression e;String a;} :
        #(PROGRAM
                (
                        a=newScope {r += a + "\n";}
                        | a=stmt {r += top.getTabs()+a + "\n";}
                        | FUNC_DEF
                        | ~(FUNC_DEF | STATEMENT | NEW_SCOPE | FOR | IF | IF_ELSE | RETURN | WHILE)
                        {
                                MPException.error("Syntax error.");
                        }
                )+
        )
        ;

stmt returns [String r]
        {
                r = "";
                MPExpression e;
                String ctrl, a;
        }
        : #(STATEMENT (e=expr
        {
                if(!(e.pre.equals("")))
                        r += e.pre + ";\n"+top.getTabs();
                r += e.s + ";";
        }
        | e=matDecl {r = e.s + ";";}
        ))
        | ctrl=controlFlow { r = ctrl; }
        | e=return_stmt { r = "return "+ e.s + ";"; }
        ;

return_stmt returns [MPExpression r]
        {
                r = new MPExpression();
        }
        : #(RETURN r=expr)
        ;


controlFlow returns [String r]
        {
                r = "";
                String a;
        }
        : r=if_stmt
        ;

if_stmt returns [String r]
        {
                r = "";
                MPExpression var1;
                String var2, var3;
                String[] rlist;
        }
        : #(IF { r = "if ("; }
                #(PRED var1=logicExpr {r += "("+var1.s+").getFloatValue()";})
```

```
        { r += ") "; }
        var2=newScope {r += var2;}
   )
| #(IF_ELSE { r = "if (; }
        #(PRED var1=logicExpr {r += "("+var1.s+").getFloatValue()";})
        { r += ") "; }
        var2=newScope {r += var2;}
        { r += top.getTabs()+"else "; }
        var2=newScope {r += var2;}
   )
| #(WHILE { r = "while (; }
        #(PRED var1=logicExpr {r += "("+var1.s+").getFloatValue()";})
        { r += ") "; }
        var2=newScope {r += var2;}
   )
| #(FOR { r = "for (; }
        #(FOR_ITER
                #(ASSIGN iter:ID rlist=forRange)
                        {
                                top.putV(iter.getText());
                                String mpname = MPExpression.MPName(iter.getText());
                                if ( rlist.length == 2 ) {
                                        String a,c;
                                        try
                                        {
                                                Float.parseFloat(rlist[0]);
                                                a = rlist[0];
                                        }
                                        catch(NumberFormatException e)
                                        {
                                                a = rlist[0] + ".getFloatValue()";
                                        }
                                        try
                                        {
                                                Float.parseFloat(rlist[1]);
                                                c = rlist[1];
                                        }
                                        catch(NumberFormatException e)
                                        {
                                                c = rlist[1] + ".getFloatValue()";
                                        }
                                        r += "Iter "+mpname+"="+a+";"+mpname+"<="+c+";++"+mpname;
                                }
                                else if ( rlist.length == 3 ) {
                                        String a,b,c;
                                        try
                                        {
                                                Float.parseFloat(rlist[0]);
                                                a = rlist[0];
                                        }
                                        catch(NumberFormatException e)
                                        {
                                                a = rlist[0] + ".getFloatValue()";
                                        }
                                        try
                                        {
                                                Float.parseFloat(rlist[1]);
                                                b = rlist[1];
                                        }
                                        catch(NumberFormatException e)
                                        {
                                                b = rlist[1] + ".getFloatValue()";
                                        }
                                        try
                                        {
                                                Float.parseFloat(rlist[2]);
                                                c = rlist[2];
                                        }
                                        catch(NumberFormatException e)
                                        {
                                                c = rlist[2] + ".getFloatValue()";
                                        }
                                        r += "Iter "+mpname+"="+a+";"+mpname+">="+c+";"+mpname+"+="+b;
                                }
                        }
         )
        { r += ") "; }
```

```
                var2=newScope {r += var2;}
           )
        ;

forRange returns [String[] r]
        {
                r = new String[3];
                String[] r2;
                String[] r3;
        }
        : r2=forRangeTwo { r = r2; }
        | r3=forRangeThree { r = r3; }
        ;

forRangeTwo returns [String[] r] { r = new String[2]; MPExpression a,b;}
        : #(RANGE_TWO a=expr b=expr)
                {
                        try
                        {
                                r[0] = a.s;
                                r[1] = b.s;
                        }
                        catch(NumberFormatException e)
                        {
                                MPException.error("cannot parse float ");
                        }
                }
        ;
forRangeThree returns [String[] r] { r = new String[3]; MPExpression a,b,c;}
        : #(RANGE_THREE a=expr b=expr c=expr)
                {
                        r[0] = a.s;
                        r[1] = b.s;
                        r[2] = c.s;
                }
        ;

logicExpr returns [MPExpression r]
        {
                r = new MPExpression();
                MPExpression expr1, expr2;
        }
        : #(LOGEQ expr1=expr expr2=expr) { r.s = expr1.s + "==" + expr2.s; }
        | #(LOGNEQ expr1=expr expr2=expr) { r.s = expr1.s + "!=" + expr2.s; }
        | #(LOGGT expr1=expr expr2=expr) { r.s = expr1.s + ">" + expr2.s; }
        | #(LOGLT expr1=expr expr2=expr) { r.s = expr1.s + "<" + expr2.s; }
        | #(LOGGTE expr1=expr expr2=expr) { r.s = expr1.s + ">=" + expr2.s; }
        | #(LOGLTE expr1=expr expr2=expr) { r.s = expr1.s + "<=" + expr2.s; }
        | #(AND expr1=expr expr2=expr) { r.s = expr1.s + "&&" + expr2.s; }
        | #(OR expr1=expr expr2=expr) { r.s = expr1.s + "||" + expr2.s; }
        ;

// NOTE: LVAR PRINTS UNCLOSED PARENTHESIS, ASSIGN CLOSES PARENTHESIS AFTER
// INSERTING ASSIGNEE AS ARGUMENT
lvar returns [MPExpression r]
        {
                r = new MPExpression();
                String slice, var;
                MPExpression a;
        }
        : #(VARIABLE id:ID )
        {
                String name = id.getText();
                r.name = name;
                if (!top.containsV(name)) {
                        top.putV(name);
                        r.s += "Matrix "+r.MPName()+"(0);\n"+top.getTabs();
                }
                r.s += r.MPName();
                r.s += ".assign(";
        }
        | a=matDecl
                {
                        top.putV(a.name);
                        r.s = a.s+";\n"+top.getTabs()+a.MPName()+".assign(";
                }
        | #(ARRAY_SLICE #(VARIABLE id1:ID) slice=lRangeList)
```

```
        {
                r.name = id1.getText();
                r.s = r.MPName();
                r.s += slice;
                if (!top.containsV(r.name)) {
                        String message = "Cannot index matrix " + r.name + " before it is initialized.";
                        MPException.error(message);
                }
        }
        |
                node:~(VARIABLE | MATRIX_DECLR | ARRAY_SLICE)
                {
                        MPException.error(node.getText()+" cannot appear to the left of an assignment.");
                }
        ;

lRangeList returns [String r]
        {
                r = "";
                String var1, var2;
        }
        : #(RANGE_LIST
                var1=lrange { r = ".sliceAssign(RangeList("+var1+"),"; }
                (var2=lrange { r += "RangeList("+var2+"),"; })?
           )
        ;

rRangeList returns [String r]
        {
                r = "";
                String var1, var2;
        }
        : #(RANGE_LIST
                var1=lrange { r = ".slice(RangeList("+var1+")"; }
                (var2=lrange { r += ",RangeList("+var2+")"; })?
           )
                { r += ")"; }
        ;

lrange returns [String r]
        {
                r = "";
                String rAll,r1,r2,r3;
        }
        : RANGE_ALL
        | r1=rangeOne { r = r1; }
        | r2=rangeTwo { r = r2; }
        | r3=rangeThree { r = r3; }
        ;

rangeOne returns [String r] {r=""; MPExpression a;}
        : #(RANGE_ONE a=expr)
        {
                String as=a.s;
                try
                {
                        Float.parseFloat(as);
                }
                catch(NumberFormatException e)
                {
                        as += ".getFloatValue()";
                }
                r = as;
        }
        ;
rangeTwo returns [String r] {r=""; MPExpression a,b;}
        : #(RANGE_TWO a=expr b=expr)
        {
                String as=a.s;
                String bs=b.s;
                try
                {
                        Float.parseFloat(as);
                }
                catch(NumberFormatException e)
                {
                        as += ".getFloatValue()";
```

```
                }
                try
                {
                        Float.parseFloat(bs);
                }
                catch(NumberFormatException e)
                {
                        bs += ".getFloatValue()";
                }
                r = as + "," + bs;
        }
        ;
rangeThree returns [String r] {r=""; MPExpression a,b,c;}
        : #(RANGE_THREE a=expr b=expr c=expr)
        {
                String as=a.s;
                String bs=b.s;
                String cs=c.s;
                try
                {
                        Float.parseFloat(as);
                }
                catch(NumberFormatException e)
                {
                        as += ".getFloatValue()";
                }
                try
                {
                        Float.parseFloat(bs);
                }
                catch(NumberFormatException e)
                {
                        bs += ".getFloatValue()";
                }
                try
                {
                        Float.parseFloat(cs);
                }
                catch(NumberFormatException e)
                {
                        cs += ".getFloatValue()";
                }
                r= as + "," + bs + "," + cs;
        }
        ;

// NOTE: LVAR PRINTS UNCLOSED PARENTHESIS, ASSIGN CLOSES PARENTHESIS AFTER
// INSERTING ASSIGNEE AS ARGUMENT
assignment returns [MPExpression r]
        {
                r = new MPExpression();
                MPExpression a, b;
        }
        :
        #(ASSIGN  (a=lvar b=expr)
                {
                r.s += a.s+b.s+")";
                r.pre = b.pre;
                }
        )
        ;
binop returns [MPExpression r]
        {
                r = new MPExpression();
                MPExpression a,b;
        }
        : #(DOTPROD  a=expr b=expr)
        {
                float aFloat, bFloat;
                try { // is left float?
                        aFloat = Float.parseFloat(a.s);
                        try { // is right float?
                                bFloat = Float.parseFloat(b.s);
                                // both floats
                                // normal scalar arithmetic
                                r.s = aFloat + "*" + bFloat;
                        }
```

```
                catch(NumberFormatException e1) {
                        // left is float and right is variable
                        // swap left and right
                        r.s = b.s+".mul("+a.s+")";
                }
        }
        catch(NumberFormatException e1) {
                try { // is right float?
                        bFloat = Float.parseFloat(b.s);
                        // left is variable and right is float
                        r.s = a.s+".mul("+b.s+")";
                }
                catch(NumberFormatException e2) {
                        // both are variables
                        // execute dotprod
                        r.s = a.s+".dot("+b.s+")";
                }
        }
}
|
(
          #(MUL  a=expr b=expr) {
                float aFloat, bFloat;
                try { // is left var float?
                        aFloat = Float.parseFloat(a.s);
                        try { // is right var float?
                                bFloat = Float.parseFloat(b.s);
                                // both floats
                                // normal scalar arithmetic
                                r.s = aFloat + "*" + bFloat;
                        }
                        catch(NumberFormatException e) {
                                // left is float and right is variable
                                // swap left and right
                                r.s = b.s+".mul("+a.s+")";
                        }
                }
                catch(NumberFormatException e) {
                        // left is variable
                        r.s = a.s+".mul("+b.s+")";
                }
          }
        | #(DIV  a=expr b=expr) {
                float aFloat, bFloat;
                try { // is left var float?
                        aFloat = Float.parseFloat(a.s);
                        try { // is right var float?
                                bFloat = Float.parseFloat(b.s);
                                // both floats
                                // normal scalar arithmetic
                                r.s = aFloat + "/" + bFloat;
                        }
                        catch(NumberFormatException e) {
                                // left is float and right is variable
                                // convert left to matrix of same size
                                // and execute op
                                r.s = "Matrix("+a.s+","+b.s+".w_,"+b.s+".h_).div("+b.s+")";
                        }
                }
                catch(NumberFormatException e) {
                        // left is variable
                        r.s = a.s+".div("+b.s+")";
                }
          }
        | #(MOD  a=expr b=expr) {
                float aFloat, bFloat;
                try { // is left var float?
                        aFloat = Float.parseFloat(a.s);
                        try { // is right var float?
                                bFloat = Float.parseFloat(b.s);
                                // both floats
                                // normal scalar arithmetic
                                r.s = aFloat + "%" + bFloat;
                        }
                        catch(NumberFormatException e) {
                                // left is float and right is variable
                                // convert left to matrix of same size
```

```
                                        // and execute op
                                        r.s = "Matrix("+a.s+","+b.s+".w_,"+b.s+".h_).mod("+b.s+")";
                                }
                        }
                        catch(NumberFormatException e) {
                                // left is variable
                                r.s = a.s+".mod("+b.s+")";
                        }
                  }
                | #(PLUS  a=expr b=expr) {
                        float aFloat, bFloat;
                        try { // is left var float?
                                aFloat = Float.parseFloat(a.s);
                                try { // is right var float?
                                        bFloat = Float.parseFloat(b.s);
                                        // both floats
                                        // normal scalar arithmetic
                                        r.s = aFloat + "+" + bFloat;
                                }
                                catch(NumberFormatException e) {
                                        // left is float and right is variable
                                        // swap left and right
                                        r.s = b.s+".add("+a.s+")";
                                }
                        }
                        catch(NumberFormatException e) {
                                // left is variable
                                r.s = a.s+".add("+b.s+")";
                        }
                  }
                | #(MINUS  a=expr b=expr) {
                        float aFloat, bFloat;
                        try { // is left var float?
                                aFloat = Float.parseFloat(a.s);
                                try { // is right var float?
                                        bFloat = Float.parseFloat(b.s);
                                        // both floats
                                        // normal scalar arithmetic
                                        r.s = aFloat + "-" + bFloat;
                                }
                                catch(NumberFormatException e) {
                                        // left is float and right is variable
                                        // convert left to matrix of same size
                                        // and execute op
                                        r.s = "Matrix("+a.s+","+b.s+".w_,"+b.s+".h_).sub("+b.s+")";
                                }
                        }
                        catch(NumberFormatException e) {
                                // left is variable
                                r.s = a.s+".sub("+b.s+")";
                        }
                  }
        )
        ;

variable returns [String r]
        {
                r = "";
        }
        :
                #(VARIABLE id:ID)
                {
                        String var=id.getText();
                        if (!top.containsV(var)) {
                                MPException.error("Variable <"+var+"> does not exist.");
                        }
                        r = MPExpression.MPName(var);
                }
        ;


expr returns [MPExpression r]
        {
                r = new MPExpression();
                MPExpression a,b, constmat;
                String rlist, var;
        }
```

```
: a=assignment { r = a; }
| a=logicExpr { r = a; }
| a=binop {r = a;}
| #(SIGN_MINUS a=expr)
{
        r.s = "Matrix("+a.s+").negate()";
}
| #(TRANS  a=expr)
{
        r.s = a.s + ".transpose()";
}
| #(ARRAY_SLICE var=variable rlist=rRangeList)
        {
                r.s = var + rlist;
        }
| #(INC var=variable)
        {
                r.s = var + "++";
        }
| #(DEC var=variable)
        {
                r.s = var + "--";
        }
| #(NOT var=variable)
        {
                r.s = var + ".notOp()";
        }
| var=variable
        {
                r.s = var;
        }
| #(CONST_SCALAR s:SCALAR)
        {
                r.s = s.getText();
        }
| #(CONST_STRING str:STRING)
        {
                String myStr = str.getText().replaceAll("\"", "\\\\\"");
                r.s = "\"" + myStr + "\"";
        }
| #(CONST_MATRIX constmat=constmatrix) { r.s =  constmat.s; r.pre = constmat.pre;}
| #(FUNC_CALL funcName:ID
{
        java.util.ArrayList<String> args = new java.util.ArrayList<String>();
}
        #(ARGS  (a=expr {
                                args.add(a.s);
                                }
                )*
        )
        )
        {
                String fname = funcName.getText();
                if ( fname.equals("print")) {
                        if (args.size() == 0) {
                                r.s += "cout << endl";
                        }
                        else {
                                r.s += "cout << ";
                        }
                        for (int i = 0; i < args.size(); i++) {
                                if (i==args.size()-1) {
                                        if (args.get(i).substring(0,1).equals("\""))
                                                r.s += args.get(i) + " << endl";
                                        else
                                                r.s += args.get(i);
                                }
                                else
                                        r.s += args.get(i) + " << ";
                        }
                }
                else {
                        if (top.containsF(fname, args.size())) {
                                r.s = fname + "(";
                                for (int i = 0; i < args.size(); i++) {
                                        r.s += args.get(i) + ",";
                                }
```

```
                                        if (args.size() > 0) {
                                                r.s = r.s.substring(0, r.s.length()-1);
                                        }
                                        r.s += ")";
                                }
                                else {
                                        MPException.error("Function <"+fname+"> with "+args.size()+" arguments has not been defined.");
                                }
                        }
                }
        ;

newScope returns [String r]
        {
                MPSymbolTable savedScope = top;
                top = new MPSymbolTable(savedScope);
                r=" {\n";
                String a="";
        }
        :
                #(NEW_SCOPE
                        (a=stmt {r += top.getTabs()+a+"\n";})*
                )
                {
                        top = savedScope;
                        r += top.getTabs()+"}\n";
                }
                ;


//Matches multiple function definitions
funcDef returns [String r]
        {
                r="";
                String args = "";
                String arg_names = "";
                String a, b;
                String fname;
                MPSymbolTable savedScope = null;
        }:
        #(PROGRAM
                (#(FUNC_DEF name:ID
                        {
                                fname = name.getText();
                                r += "Matrix "+fname+" (";
                                arg_names = "";
                                args = "";
                        }
                        #(ARGS
                                (id:ID
                                        {
                                                args+="Matrix " + MPExpression.MPName(id.getText())+",";
                                                arg_names += id.getText() + " ";
                                        }
                            )*
                         )
                    {
                                if (args.length() > 0)
                                args=args.substring(0, args.length()-1);
                                r += args+") ";
                    }
                    #(fb:NEW_SCOPE
                    {
                        String[] arglist;
                                if (arg_names.equals("")) {
                                        arglist = new String[0];
                                }
                                else {
                                        arglist = arg_names.split(" ");
                                }
                        if (!top.containsF(fname, arglist.length)) {
                                        top.putF(name.getText(), arglist.length);
                                        savedScope = top;
                                        top = new MPSymbolTable(savedScope, 0);
                                        for (int i = 0; i < arglist.length; i++) {
                                                top.putV(arglist[i]);
                                        }
```

```
                        }
                        else {
                                MPException.error("A function named <"+fname+"> with "+arglist.length+" arguments already exists.");
                        }
                                b = newScope(fb);
                                r += b+"\n";
                                top = savedScope;
                    }
                        )
                )
                {arg_names = "";}
                | ~(FUNC_DEF)
                )*
        )
        ;
matDecl returns [MPExpression r]
        {
                r=new MPExpression();
                MPExpression xdim, ydim;
        }
        :
        #(MATRIX_DECLR name:ID
                (xdim=expr ydim=expr
                        {
                                String token = name.getText();
                                top.putV(token);
                                r.name = token;
                                r.s = "Matrix " + MPExpression.MPName(token) + "(" + xdim.s + ", " + ydim.s + ")";
                        }
                |
                {
                        String token = name.getText();
                        top.putV(token);
                        r.name = token;
                        r.s = "Matrix " + MPExpression.MPName(token) + "(0)";
                }
                )
        )

        ;

constmatrix returns [MPExpression r]
        {
                constMatNum++;
                int w =0;
                int h = -1;
                r = new MPExpression();
                String arrayName = "array_" + Integer.toString(constMatNum);
                String vecName = "vec_" + Integer.toString(constMatNum);
                r.pre="vector<float> " + vecName + ";\n";
                r.pre+=top.getTabs()+"float " + arrayName + "[] = {";
                java.util.ArrayList<String> a;
        }
                :
                (a=row
                {
                        if(!(h == a.size() || h == -1))
                                MPException.error("size of columns must be the same for constant matrix declaration");
                        w = w+1;
                        h = a.size();
                        for(int i=0; i < h; i++)
                        {
                                        String val = a.get(i);
                                        r.pre+=val + ",";
                        }
                }
                )+
                {
                        r.pre = r.pre.substring(0, r.pre.length()-1);
                        r.pre+="};\n";
                        r.pre+=top.getTabs()+"copy(" + arrayName + ", " + arrayName + "+" + Integer.toString(w*h) + ", back_inserter(" + vecName +
                        r.s = vecName + ", " + Integer.toString(w) + ", " + Integer.toString(h);
                }
                ;

// a row consists of one or more comma separated expressions
row returns [java.util.ArrayList<String> r]
```

```
        {
                r = new java.util.ArrayList<String>();
                java.util.ArrayList<String> a;
        }
        : #(MATRIX_ROW a=cols)
        {
                r=a;
        }
        ;

cols returns [java.util.ArrayList<String> r]
        {
                r = new java.util.ArrayList<String>();
        }
        : #(MATRIX_COL
                (
                        a:SCALAR
                                {
                                        r.add(a.getText());
                                }
                )+
          )
        ;
```

B.1.2 Main.java

```
/*
 * MatPix front-end for its ANTLR lexer/parser.
 */

import java.io.*;
import antlr.CommonAST;
import antlr.debug.misc.ASTFrame;

public class Main {
  public static void main(String args[]) {
    try {
      //DataInputStream input = new DataInputStream(System.in);
        FileInputStream input = new FileInputStream(args[0]);

      // Create the lexer and parser and feed them the input
      MPLexer lexer = new MPLexer(input);
      MPParser parser = new MPParser(lexer);

      MPWalker treeParser = new MPWalker();
      parser.program(); // "file" is the main rule in the parser

      // Get the AST from the parser
      CommonAST parseTree = (CommonAST)parser.getAST();
      String funcDefStr = treeParser.funcDef(parseTree);
      String programStr = treeParser.program(parseTree);

      if (parser.error_count > 0) {
          MPException.error("Error found in the parser.");
      }
      if (lexer.error_count > 0) {
          MPException.error("Error found in the lexer.");
      }
      // print the program string to file
          String forceSoftware="true;";
      FileWriter fw;
      if (args.length > 1)
      {
          String outfile = args[1];
          fw = new FileWriter (outfile);
          if (args.length > 2)
          {
                  forceSoftware="false;";
          }
      }
      else
      {
          String infile = args[0];
          String[] split = infile.split("\\.");
          fw = new FileWriter ("cpp/" + split[0] + ".cpp");
```

```
      }
      PrintWriter pw = new PrintWriter (fw);
      pw.println("#include \"MatPix/matpix.hpp\"");
      pw.println("");
      if (!funcDefStr.equals(""))
          pw.println(funcDefStr);
      pw.println("int main(){");
      pw.println("");
      pw.println("   "+"bool forceSoftwareRender=" + forceSoftware + ";");
      pw.println("   "+"initMatPix(forceSoftwareRender);");
      pw.println("");
      pw.print(programStr);
      pw.println("");
      pw.println("   "+"cleanupMatPix();");
      pw.println("   "+"return 0;");
      pw.println("}");
      pw.close();

      // Print the AST in a human-readable format
      //System.out.println(parseTree.toStringList());


      // Open a window in which the AST is displayed graphically
      if (args.length > 3)
      {
          System.out.println(funcDefStr);
          System.out.println(programStr);
          ASTFrame frame = new ASTFrame("AST from the MatPix parser", parseTree);
          frame.setVisible(true);
      }

    } catch(Exception e) { System.err.println("Exception: "+e); }
  }
}
```

B.2 Compiler

B.2.1 MPSymbolTable.java

```
import java.util.Hashtable;
import java.util.Enumeration;

//TODO: Add function definitions to symbol table, differentiate between matrix and func types

public class MPSymbolTable {
	private Hashtable<String,String> tableV;
	private Hashtable<String,String> tableF;
	protected MPSymbolTable outer;
	private int depth;

	public MPSymbolTable(MPSymbolTable st) {
		tableV = new Hashtable<String,String>();
		tableF = new Hashtable<String,String>();
		outer = st;
		if (st == null) {
			depth = 1;
		}
		else {
			depth = st.getDepth()+1;
		}
	}
	public MPSymbolTable(MPSymbolTable st, int d) {
		tableV = new Hashtable<String,String>();
		tableF = new Hashtable<String,String>();
		outer = st;
		depth = d;
	}
	public int getDepth() {
		return depth;
	}
	public String getTabs() {
		String str = "";
		for(int i = 0; i < depth; i++) {
			str += "   ";
		}
		return str;
	}
	public void putV(String token) {
		tableV.put(token, "");
	}
	public boolean containsV(String token) {
		for (MPSymbolTable tab = this; tab != null; tab = tab.outer) {
			if (tab.tableV.containsKey(token))
				return true;
		}
		return false;
	}
	public void putF(String token, int numArgs) {
		token = token+numArgs;
		tableF.put(token, "");
	}
	public boolean containsF(String token, int numArgs) {
		token = token+numArgs;
		for (MPSymbolTable tab = this; tab != null; tab = tab.outer) {
			if (tab.tableF.containsKey(token))
				return true;
		}
		return false;
	}
	public void printInScopeVars() {
		System.err.println("In scope identifiers:");
		for (MPSymbolTable tab = this; tab != null; tab = tab.outer) {
			Enumeration<String> keys = tab.tableV.keys();
			while (keys.hasMoreElements()) {
				System.err.println(keys.nextElement());
			}
			System.err.println("------");
		}
	}
	public void printInScopeFuncs() {
		System.err.println("In scope functions:");
```

```
                for (MPSymbolTable tab = this; tab != null; tab = tab.outer) {
                        Enumeration<String> keys = tab.tableF.keys();
                        while (keys.hasMoreElements()) {
                                System.err.println(keys.nextElement());
                        }
                        System.err.println("------");
                }
        }

}
```

B.2.2 MPExpression.java

```
public class MPExpression {
        public String s, pre;
        public String name;
        private static String PFX = "MP_";

        MPExpression (String s, String name) {
                this.s = s;
                this.pre = "";
                this.name = name;
        }
        MPExpression (String s) {
                this.s = s;
                this.pre = "";
                this.name = "";
        }
        MPExpression () {
                s = "";
                pre = "";
                name = "";
        }
        public String MPName() {
                if (name.length() > 0)
                        return PFX+name;
                else
                        MPException.error("String <"+s+"> does not have a name");
                return "";
        }
        public static String MPName(String var) {
                return PFX+var;
        }
}
```

B.2.3 MPException.java

```
class MPException extends RuntimeException {
        static final long serialVersionUID = 1;
        MPException(String m) {
                System.err.println("ERROR: "+m);
        }
        public static void error(String m) {
                throw new MPException("Illegal operation: "+m);
        }
}
```

B.3 GPU Backend

B.3.1 matpix.hpp

```
#ifndef __MATPIX__HPP
#define __MATPIX__HPP

#include "matrix.hpp"
#include "util.hpp"

#endif //__MATPIX__HPP
```

B.3.2 matrix.hpp

```
#ifndef __MATRIX__HPP
#define __MATRIX__HPP

//#define DEBUG__

#include <iostream>
#include <string>
#include <vector>
#include <sstream>
#include "util.hpp"

#include <boost/shared_ptr.hpp>

using namespace std;
using namespace boost;

// declare matrix class for subsequent use
class Matrix;

class Iter
{
public:
        Iter(float f) : f_(f) {};
        Iter(const Iter& i) : f_(i.f_) {};
        Iter operator=(const Iter& i) {f_ = i.f_;};
        Iter operator++() {++f_; return *this;};
        Iter operator+=(const Iter& i) {f_ += i.f_; return *this;};
        bool operator==(const Iter& i) {return f_ == i.f_;}
        bool operator!=(const Iter& i) {return f_ != i.f_;}
        bool operator<(const Iter& i) {return f_ < i.f_;}
        bool operator>(const Iter& i) {return f_ > i.f_;}
        bool operator<=(const Iter& i) {return f_ <= i.f_;}
        bool operator>=(const Iter& i) {return f_ >= i.f_;}
        bool operator&&(const Iter& i) {return f_ && i.f_;}
        bool operator||(const Iter& i) {return f_ || i.f_;}
        Iter add(const Iter& i) {Iter ret(f_ + i.f_); return ret;};
        Iter sub(const Iter& i) {Iter ret(f_ - i.f_); return ret;};
        Iter mul(const Iter& i) {Iter ret(f_ * i.f_); return ret;};
        Iter div(const Iter& i) {Iter ret(f_ / i.f_); return ret;};
//      Iter mod(const Iter& i) {Iter ret(f_ % i.f_); return ret;};
//      Matrix add(const Matrix& m) {return m.add(Matrix(f_));};
//      Matrix sub(const Matrix& m) {return m.sub(Matrix(f_));};
//      Matrix mul(const Matrix& m) {return m.mul(Matrix(f_));};
//      Matrix div(const Matrix& m) {return m.div(Matrix(f_));};
//      Matrix div(const Matrix& m) {return m.mod(Matrix(f_));};
//      Matrix operator==(const Matrix& m) {return f_ == m.getFloatValue();}
//      Matrix operator!=(const Matrix& m) {return f_ != m.getFloatValue();}
//      Matrix operator<(const Matrix& m) {return f_ < m.getFloatValue();}
//      Matrix operator>(const Matrix& m) {return f_ > m.getFloatValue();}
//      Matrix operator<=(const Matrix& m) {return f_ <= m.getFloatValue();}
//      Matrix operator>=(const Matrix& m) {return f_ >= m.getFloatValue();}
//      Matrix operator&&(const Matrix& m) {return f_ && m.getFloatValue();}
//      Matrix operator||(const Matrix& m) {return f_ || m.getFloatValue();}
        float getFloatValue() {return f_;};
        float f_;
};
class RangeList
{
public:
        RangeList() : _rs(0) {};
        RangeList(float r1) : _rs(1), _r1(r1) {};
```

```
        RangeList(float r1, float r2): _rs(2),  _r1(r1), _r2(r2) {};
        float _r1, _r2;
        int _rs;
};

class TextureID
{
public:
        TextureID()
        {
                glGenTextures (1, &id_);
        };
        ~TextureID()
        {
                glDeleteTextures (1, &id_);
        }
        GLuint id_;
};
typedef shared_ptr<TextureID> TextureIDPtr;


class Matrix
{
public:
        typedef shared_ptr<Matrix> MatrixPtr;
        Matrix(const Matrix& m) : w_(m.w_), h_(m.h_), texID_(m.texID_)
        {
        }
        Matrix(const Matrix &h, const Matrix &w) : texID_(new TextureID())
        {
                w_ = w.getFloatValue();
                h_ = h.getFloatValue();
                setupTexture();
                assign(0.0);
        };
        Matrix(int h, int w, int , int) : w_(w), h_(h), texID_(new TextureID())
        {
                setupTexture();
        };
        Matrix(int h, int w) : w_(w), h_(h), texID_(new TextureID())
        {
                setupTexture();
                assign(0.0);
        };
        Matrix(float f) : w_(1), h_(1), texID_(new TextureID())
        {
                setupTexture();
                assign(f);
        };
        Matrix(float f, int h, int w) : w_(w), h_(h), texID_(new TextureID())
        {
                setupTexture();
                assign(f);
        };
        Matrix(float f, const Matrix &h, const Matrix &w) : texID_(new TextureID())
        {
                w_ = w.getFloatValue();
                h_ = h.getFloatValue();
                setupTexture();
                assign(f);
        };
        Matrix(const Iter& i) : w_(1), h_(1), texID_(new TextureID())
        {
                setupTexture();
                assign(i.f_);
        }

        Matrix& operator=(const Matrix &m)
        {
                w_ = m.w_; h_ = m.h_; texID_ = m.texID_;
        }
        string print() const
        {
                vector<float> data(w_*h_);
                getTexture(data);
                return printVector(data);
        };
```

```
        Matrix slice(RangeList range1, RangeList range2)
        {
                int x, y, w, h;
                getRange(range1, range2, x, y, w, h);
                Matrix ret(h, w);
                compute(ret.texID_->id_, texID_->id_, 0, ret.w_, ret.h_, "slice", 0, x, y);
                return ret;
        };
        Matrix& assign(const vector<float>& data, int h, int w)
        {
                w_ = w;
                h_ = h;
                setupTexture();
                setTexture(data);
        }
        Matrix& assign(float val)
        {
                vector<float> data(w_*h_, val);
                setTexture(data);
        }
        Matrix& assign(const Matrix& b)
        {
                w_ = b.w_;
                h_ = b.h_;
                texID_ = b.texID_;
                return *this;
        };
        void sliceAssign(const RangeList& range1, const RangeList& range2, float val)
        {
                int x, y, w, h;
                getRange(range1, range2, x, y, w, h);
                //printf ("%d, %d, %d, %d\n", x, w, y, h);
                vector<float> data(w*h, val);
                setTextureSub(data, x, y, w, h);
        };
        void sliceAssign(const RangeList& range1, const RangeList& range2, const Matrix& b)
        {
                if (usingFBO) {
                        int x, y, w, h;
                        getRange(range1, range2, x, y, w, h);
                        bindTexToFBO(b.texID_->id_);
                        glDrawBuffer(GL_COLOR_ATTACHMENT0_EXT);
                        glBindTexture(textureParameters.texTarget, texID_->id_);
                        glCopyTexSubImage2D(textureParameters.texTarget, 0, x, y, 0, 0, w, h);
                        checkGLErrors("copytexsubimage");
                }
                else
                {
//                      int x, y, w, h;
//                      getRange(range1, range2, x, y, w, h);
//                      //printf ("%d, %d, %d, %d\n", x, w, y, h);
//                      compute(texID_->id_, b.texID_->id_, 0, w, h, "slice", 0, 0, 0, x, y);
                        if(b.w_==1 && b.h_==1)
                        {
                                sliceAssign(range1, range2, b.getFloatValue());
                        }
                        else
                        {
                                int x, y, w, h;
                                getRange(range1, range2, x, y, w, h);
                                //printf ("%d, %d, %d, %d\n", x, w, y, h);
                                compute(texID_->id_, b.texID_->id_, 0, w, h, "slice", 0, 0, 0, x, y);
                        }
                }
        };
        void getRange(const RangeList& range1, const RangeList& range2, int& x, int& y, int& w, int& h)
        {
                // rows
                if (range1._rs ==0)
                {
                        y = 0;
                        h = h_;
                }
                else if (range1._rs ==1)
                {
                        if ( range1._r1 < 0 ) {
                                cerr << "Error: No negative row indices. Exiting program.\n";
```

```
                        cleanupMatPix();
                        exit(1);
                }
                if (range1._r1 >= h_) {
                        cerr << "Error: Out-of-bounds row value: " << range1._r1 << ". Exiting program.\n";
                        cleanupMatPix();
                        exit(1);
                }
                y = range1._r1;
                h = 1;
        }
        else if (range1._rs ==2)
        {
                if ( range1._r1 < 0 ) {
                        cerr << "Error: No negative row indices. Exiting program.\n";
                        cleanupMatPix();
                        exit(1);
                }
                if ( range1._r1 > range1._r2 ) {
                        cerr << "Error: Invalid row range. Exiting program.\n";
                        cleanupMatPix();
                        exit(1);
                }
                if ( range1._r2 >= h_ ) {
                        cerr << "Error: Out-of-bounds row index: " << range1._r2 << ". Exiting program.\n";
                        cleanupMatPix();
                        exit(1);
                }
                y = range1._r1;
                h = range1._r2-range1._r1+1;
        }
        // columns
        if (range2._rs ==0)
        {
                x = 0;
                w = w_;
        }
        else if (range2._rs ==1)
        {
                if ( range2._r1 < 0 ) {
                        cerr << "Error: No negative column indices. Exiting program.\n";
                        cleanupMatPix();
                        exit(1);
                }
                if (range2._r1 >= w_) {
                        cerr << "Error: Out-of-bounds column value: " << range2._r1 << ". Exiting program.\n";
                        cleanupMatPix();
                        exit(1);
                }
                x = range2._r1;
                w = 1;
        }
        else if (range2._rs ==2)
        {
                if ( range2._r1 < 0 ) {
                        cerr << "Error: No negative column indices. Exiting program.\n";
                        cleanupMatPix();
                        exit(1);
                }
                if ( range2._r1 > range2._r2 ) {
                        cerr << "Error: Invalid column range. Exiting program.\n";
                        cleanupMatPix();
                        exit(1);
                }
                if ( range2._r2 >= w_ ) {
                        cerr << "Error: Out-of-bounds column index: " << range2._r2 << ". Exiting program.\n";
                        cleanupMatPix();
                        exit(1);
                }
                x = range2._r1;
                w = range2._r2-range2._r1+1;
        }
//              cout << "x = " << x << "; y = " << y << endl;
//              cout << "w = " << w << "; h = " << h << endl;
    }
    Matrix transpose()
    {
```

```
                Matrix ret(w_, h_);
                compute(ret.texID_->id_, texID_->id_, 0, ret.w_, ret.h_, "trans");
                return ret;
        };

        Matrix unaryOp(const char* op)
        {
                Matrix ret(w_, h_);
                compute(ret.texID_->id_, texID_->id_, 0, w_, h_, op);
                assign(ret);
                return ret;
        };
        Matrix binOp(const Matrix& b, const char* op)
        {
                Matrix ret(h_, w_, 0, 0);
                Matrix b_new(b);
                // runtime semantic analysis
                if ( (w_ != b.w_) || (h_ != b.h_) ) {
                        if ( b.h_==1 && b.w_==1 ) {
                                b_new.assign(Matrix(b.getFloatValue(),h_,w_));
                        }
                        else if ( h_==1 && w_==1 ) {
                                ret = Matrix(b.h_, b.w_, 0, 0);
                                Matrix self_new(getFloatValue(),b.h_,b.w_);
                                compute(ret.texID_->id_, self_new.texID_->id_, b_new.texID_->id_, ret.w_, ret.h_, op);
                                return ret;
                        }
                        else {
                                cerr << "Error: Matrix dimensions do not agree. Exiting program.\n";
                                cerr << "Left Matrix size: (" << h_ << ", " << w_ << ")\n";
                                cerr << "Right Matrix size: (" << b.h_ << ", " << b.w_ << ")\n";
                                cleanupMatPix();
                                exit(1);
                        }
                }
//              cout << "Left Matrix size: (" << h_ << ", " << w_ << ")\n";
//              cout << "Right Matrix size: (" << b_new.h_ << ", " << b_new.w_ << ")\n";
                compute(ret.texID_->id_, texID_->id_, b_new.texID_->id_, w_, h_, op);
                return ret;
        };
        Matrix mul(const Matrix& b)
        {
                return binOp(b, "*");
        };
        Matrix add(const Matrix& b)
        {
                return binOp(b, "+");
        };
        Matrix sub(const Matrix& b)
        {
                return binOp(b, "-");
        };
        Matrix div(const Matrix& b)
        {
                return binOp(b, "/");
        };
        Matrix mod(const Matrix& b)
        {
                return binOp(b, "%");
        };
        Matrix operator==(const Matrix &b)  {
                return binOp(b, "==");
        };
        Matrix operator!=(const Matrix &b)  {
                return binOp(b, "!=");
        };
        Matrix operator<(const Matrix &b)  {
                return binOp(b, "<");
        };
        Matrix operator>(const Matrix &b)  {
                return binOp(b, ">");
        };
        Matrix operator<=(const Matrix &b)  {
                return binOp(b, "<=");
        };
        Matrix operator>=(const Matrix &b)  {
                return binOp(b, ">=");
```

```
        };
        Matrix operator&&(const Matrix &b)  {
                return binOp(b, "&&");
        };
        Matrix operator||(const Matrix &b)  {
                return binOp(b, "||");
        };
        Matrix operator++(int x) {
                return unaryOp("++");
        };
        Matrix operator--(int x) {
                return unaryOp("--");
        };
        Matrix negate() {
                return unaryOp("negate");
        };
        Matrix notOp() {
                return unaryOp("!");
        }
        float getFloatValue() const {
                if(!(w_ == 1 && h_ == 1))
                        printf("Converting to Float: Not 1x1 Matrix!\n");
                vector<float> data(1);
                getTexture(data);
                return data[0];
        }
        Matrix dot(const Matrix& b)
        {
                if(!(w_ == b.h_))
                {
                        if ( b.h_==1 && b.w_==1 ) {
                                Matrix ret(h_, w_);
                                Matrix b_new(b.getFloatValue(),h_,w_);
                                compute(ret.texID_->id_, texID_->id_, b_new.texID_->id_, ret.w_, ret.h_, "*");
                                return ret;
                        }
                        else if ( h_==1 && w_==1 ) {
                                Matrix ret(b.h_, b.w_);
                                Matrix self_new(getFloatValue(),b.h_,b.w_);
                                compute(ret.texID_->id_, self_new.texID_->id_, b.texID_->id_, ret.w_, ret.h_, "*");
                                return ret;
                        }
                        else {
                                cerr << "Matrix Dimensions for Dot Product Don't agree\n";
                                cerr << "Left Matrix size: (" << h_ << ", " << w_ << ")\n";
                                cerr << "Right Matrix size: (" << b.h_ << ", " << b.w_ << ")\n";
                                exit(0);
                        }
                }
//              cout << "Left Matrix size: (" << h_ << ", " << w_ << ")\n";
//              cout << "Right Matrix size: (" << b.h_ << ", " << b.w_ << ")\n";
                Matrix ret(h_, b.w_);
                compute(ret.texID_->id_, texID_->id_, b.texID_->id_, ret.w_, ret.h_, "dot", w_);
//              cout << "Return Matrix size: (" << ret.h_ << ", " << ret.w_ << ")\n";
                return ret;
        };

        int w_, h_; // texSize_;
        TextureIDPtr texID_;

private:
        string printVector(vector<float> data) const
        {
                int ind = 0;
                ostringstream ss;
                for(int i = 0; i < h_; i++)
                {
                        for(int j = 0; j < w_; j++) {
                                ss << data[ind++];
                                if (j < w_-1)
                                        ss << ", ";
                        }
                        ss << "\n";
                }
                return ss.str();
        }
        void setupTexture() {
```

```
                glBindTexture(textureParameters.texTarget, texID_->id_);
                glTexParameteri(textureParameters.texTarget, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
                glTexParameteri(textureParameters.texTarget, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
                glTexParameteri(textureParameters.texTarget, GL_TEXTURE_WRAP_S, GL_CLAMP);
                glTexParameteri(textureParameters.texTarget, GL_TEXTURE_WRAP_T, GL_CLAMP);
                glTexImage2D(textureParameters.texTarget, 0,
                                textureParameters.texInternalFormat, w_, h_, 0,
                                textureParameters.texFormat, GL_FLOAT, 0);
                // check if that worked
                if (glGetError() != GL_NO_ERROR) {
                        printf("glTexImage2D(): [FAIL]\n");
                }
        }
        void getTexture(vector<float>& data) const {
                if (usingFBO) {
//              if (false) {
                        bindTexToFBO(texID_->id_);
                        glReadBuffer(GL_COLOR_ATTACHMENT0_EXT);
                        glReadPixels(0, 0, w_, h_, textureParameters.texFormat,
                                        GL_FLOAT, &data[0]);
                } else {
                        glBindTexture(textureParameters.texTarget, texID_->id_);
                        glGetTexImage(textureParameters.texTarget, 0,
                                        textureParameters.texFormat, GL_FLOAT, &data[0]);
                }
                checkGLErrors("transfer from texture");
        }
        void setTexture(const vector<float>& data) {
                glBindTexture(textureParameters.texTarget, texID_->id_);
                glTexSubImage2D(textureParameters.texTarget, 0, 0, 0, w_, h_,
                                textureParameters.texFormat, GL_FLOAT, &data[0]);
                checkGLErrors("transfer to texture");
        }
        void setTextureSub(const vector<float>& data, int x, int y, int w, int h) {
                //cout << "(x,y,w,h)" << x << ", " << y << ", " << w << ", " << h << endl;
                glBindTexture(textureParameters.texTarget, texID_->id_);
                glTexSubImage2D(textureParameters.texTarget, 0, x, y, w, h,
                                textureParameters.texFormat, GL_FLOAT, &data[0]);
                checkGLErrors("transfer to texture");
        }
};


ostream& operator << (ostream& os, const Matrix& m)
{
  return os << m.print();
}
ostream& operator << (ostream& os, const TextureID& t)
{
  return os << t.id_;
}
ostream& operator << (ostream& os, const Iter& i)
{
  return os << i.f_ << endl;
}

#endif //__MATRIX__HPP
```

B.3.3 PBuffer.cpp

```
#include "PBuffer.h"

#include <AGL/agl.h>
#include <OpenGL/OpenGL.h>

GLuint format = GL_TEXTURE_RECTANGLE_EXT;
//GLuint format = GL_TEXTURE_2D;

static void checkCGLErrorImplementation(CGLError error, char* sourceFile,
                int sourceLine) {
        if (error) {
                const char* errStr;
                errStr = CGLErrorString(error);
                fprintf(stderr, "CGL Error: %s at %s:%d\n", errStr, sourceFile,
                                sourceLine);
                assert(false);
```

```
        }
}

#define checkCGLError(error) checkCGLErrorImplementation(error,__FILE__,__LINE__)

PetePBuffer* PBuffer_Create(CGLContextObj sharedContext, int width, int height,
                int colorDepth, int flags) {
        //  printf("width %d, height %d\n", width, height);
        PetePBuffer* pbuffer=new PetePBuffer;
        pbuffer->pbuffer=NULL;
        pbuffer->pbufferContext=NULL;
        pbuffer->previousContext=NULL;
        pbuffer->width=width;
        pbuffer->height=height;
        pbuffer->textureID=0;
        pbuffer->needsClearing=true;
        pbuffer->needsFlush=false;

        const int bitsPerPixel = (colorDepth*4);
        //const int bitsPerPixel = (colorDepth);
        const bool hasZBuffer = (flags&ePBufferFlag_ZBuffer);

        int i = 0;
        CGLPixelFormatAttribute pixelFormatAttributes[32];

        pixelFormatAttributes[i++] = kCGLPFARobust;
        pixelFormatAttributes[i++] = kCGLPFANoRecovery;
        pixelFormatAttributes[i++] = kCGLPFADoubleBuffer;
        //pixelFormatAttributes[i++] = kCGLPFAAccelerated;
        //pixelFormatAttributes[i++] = kCGLPFAWindow;
        pixelFormatAttributes[i++] = kCGLPFAColorSize;
        pixelFormatAttributes[i++] = (CGLPixelFormatAttribute)(bitsPerPixel);

        if (colorDepth>8)
                pixelFormatAttributes[i++] = kCGLPFAColorFloat;

        if (hasZBuffer) {
                pixelFormatAttributes[i++] = kCGLPFADepthSize;
                pixelFormatAttributes[i++] = (CGLPixelFormatAttribute)(16);
        }

        pixelFormatAttributes[i++] = (CGLPixelFormatAttribute)0;

        long numPixelFormats = 0;
        CGLPixelFormatObj pixelFormat = NULL;
        CGLError error = CGLChoosePixelFormat(pixelFormatAttributes, &pixelFormat,
                        &numPixelFormats);
        checkCGLError(error);

        error = CGLCreateContext(pixelFormat, sharedContext,
                        &pbuffer->pbufferContext);
        checkCGLError(error);

        CGLDestroyPixelFormat(pixelFormat);

        error = CGLCreatePBuffer(pbuffer->width, pbuffer->height, format, GL_RGBA,
                        0, &pbuffer->pbuffer);
        checkCGLError(error);

        return pbuffer;

}

void PBuffer_Destroy(PetePBuffer* pbuffer) {

        if (pbuffer!=NULL) {
                CGLDestroyPBuffer(pbuffer->pbuffer);
                CGLDestroyContext(pbuffer->pbufferContext);
        }

        delete pbuffer;

}

void PBuffer_Begin(PetePBuffer* pbuffer) {

        // Pete- check to ensure begin() hasn't already been called for this pbuffer
```

```
        assert(pbuffer->previousContext==NULL);

        pbuffer->previousContext=CGLGetCurrentContext();

        long screen;
        CGLError error=CGLGetVirtualScreen(pbuffer->previousContext, &screen);
        assert(!error);

        error=CGLSetCurrentContext(pbuffer->pbufferContext);
        checkCGLError(error);

        error
                        =CGLSetPBuffer(pbuffer->pbufferContext, pbuffer->pbuffer, 0, 0,
                                       screen);
        checkCGLError(error);

}

void PBuffer_End(PetePBuffer* pbuffer) {

        glFlush();

        pbuffer->needsFlush=true;

        assert(pbuffer->previousContext!=NULL);

        CGLError error=CGLSetCurrentContext(pbuffer->previousContext);
        checkCGLError(error);

        pbuffer->previousContext=NULL;

}

void PBuffer_SetTexture(PetePBuffer* pbuffer) {
        CGLContextObj currentContext=CGLGetCurrentContext();
        CGLError error=CGLTexImagePBuffer(currentContext, pbuffer->pbuffer,
                        GL_FRONT_LEFT);
        checkCGLError(error);
}

void PBuffer_Use(PetePBuffer* pbuffer) {

        if (pbuffer->needsFlush) {
                CGLContextObj currentContext=CGLGetCurrentContext();

                CGLError error=CGLSetCurrentContext(pbuffer->pbufferContext);
                checkCGLError(error);

                glFlush();

                error=CGLSetCurrentContext(currentContext);
                checkCGLError(error);

                pbuffer->needsFlush=false;
        }

        if (pbuffer->textureID==0) {
                CGLContextObj currentContext=CGLGetCurrentContext();

                glGenTextures(1, &pbuffer->textureID);
                glBindTexture(format, pbuffer->textureID);
                glTexParameteri(format, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
                glTexParameteri(format, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
                glTexParameteri(format, GL_TEXTURE_WRAP_S, GL_CLAMP);
                glTexParameteri(format, GL_TEXTURE_WRAP_T, GL_CLAMP);
                glTexParameteri(format, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
                glTexParameteri(format, GL_TEXTURE_MAG_FILTER, GL_NEAREST);

                CGLError error=CGLTexImagePBuffer(currentContext, pbuffer->pbuffer,
                                GL_FRONT_LEFT);
                checkCGLError(error);

        } else {
                glBindTexture(format, pbuffer->textureID);
        }

        //glEnable(GL_TEXTURE_RECTANGLE_EXT);
```

```
}
```

B.3.4 PBuffer.h

```
#ifndef INCLUDE_PBUFFER_UTILS_H
#define INCLUDE_PBUFFER_UTILS_H

#include <AGL/agl.h>
#include <OpenGL/OpenGL.h>

enum EPBufferFlags
{
        ePBufferFlag_ZBuffer=(1<<0),
};

typedef struct PetePBuffer_tag {
        CGLPBufferObj pbuffer;
        CGLContextObj pbufferContext;
        CGLContextObj previousContext;
        int width;
        int height;
        GLuint textureID;
        bool needsClearing;
        bool needsFlush;
        int createdWidth;
        int createdHeight;
        GLuint target;
        GLuint texFormat;
} PetePBuffer;

PetePBuffer* PBuffer_Create(CGLContextObj sharedContext, int nWidth,
                int nHeight, int colorDepth, int flags);
void PBuffer_Destroy(PetePBuffer* pbuffer);

// Surround rendering with these to draw into the pbuffer, they handle pushing and popping the old
// context for you
void PBuffer_Begin(PetePBuffer* pbuffer);
void PBuffer_End(PetePBuffer* pbuffer);

// Binds the pbuffer as a texture
void PBuffer_SetTexture(PetePBuffer* pbuffer);
void PBuffer_Use(PetePBuffer* pbuffer);

#endif // INCLUDE_PBUFFER_UTILS_H
```

B.3.5 util.cpp

```
#include "util.hpp"
/**
 * Sets up GLUT, creates "window" (better put: valid GL context, since the window is never displayed)
 */

// GLSL vars
GLuint glslProgram;
GLuint fragmentShader;
GLuint glslPrograms[20];
GLint yParam, xParam, texWidth, texHeight, commonLength, xoff, yoff;

// FBO identifier
GLuint fb;

// struct actually being used (set from command line)
struct_textureParameters textureParameters;

int forceSoftware;
bool usingFBO;
bool attached;
PetePBuffer* pbuff;

const char* shaderSrcBeg =
        "#extension GL_ARB_texture_rectangle : enable\n"
        "uniform sampler2DRect textureY;"
        "uniform sampler2DRect textureX;"
```

```
        "uniform int texWidth;"
        "uniform int texHeight;"
        "uniform int commonLength;"
        "uniform int xoff;"
        "uniform int yoff;"
        "void main(void) { ";


void initMatPix(bool forceSofwareRenderer)
{
        forceSoftware=forceSofwareRenderer;
        // global force_software macro

        textureParameters.name = "TEXRECT - float_ARB - RGBA - 32";
        textureParameters.texTarget = GL_TEXTURE_RECTANGLE_EXT;
        textureParameters.texInternalFormat = GL_RGBA32F_ARB;
        //textureParameters.texFormat = GL_RGBA;
        textureParameters.texFormat = GL_RED;

        // init glut and glew
        usingFBO=true;
        if (forceSoftware) {
                usingFBO = false;
        }

        initGLUT();
        initGLEW();

        if (!GLEW_EXT_framebuffer_object) {
#ifdef DEBUG__
                printf("FBO not available\n");
#endif
                usingFBO = false;
        }
#ifdef DEBUG__
        printf("usingFBO = %d\n", usingFBO);
#endif
        // init offscreen framebuffer
        if (usingFBO) {
                initFBO();
        }
        initGLSL();
}
//void compute(int destID, int xTexID, int yTexID, int w, int h, const char* op, int cLength, int xo, int yo, int x, int y) {
void compute(int destID, int xTexID, int yTexID, int w, int h, const char* op, int cLength, int xo, int yo, int x, int y) {
        if (usingFBO) {
                glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, textureParameters.texTarget, destID, 0);
                if (!checkFramebufferStatus()) {
                        printf("glFramebufferTexture2DEXT():\t [FAIL]\n");
                        exit (ERROR_FBOTEXTURE);
                }
        }
        else
        {
                if(pbuff) {
                        PBuffer_End(pbuff);
                        PBuffer_Destroy(pbuff);
                }
                initPbuffer(w, h);
        }

        // enable GLSL program
        setActiveShader(op);

        glUniform1i(texWidth, w);
        glUniform1i(texHeight, h);
        glUniform1i(commonLength, cLength);
        glUniform1i(xoff, xo);
        glUniform1i(yoff, yo);

        glActiveTexture(GL_TEXTURE0);
        glBindTexture(textureParameters.texTarget, yTexID);
        glUniform1i(yParam,0);

        glActiveTexture(GL_TEXTURE1);
        glBindTexture(textureParameters.texTarget, xTexID);
        glUniform1i(xParam, 1);
```

```
        if (!usingFBO) {
                PBuffer_End(pbuff);
                PBuffer_Begin(pbuff);
        }

        if (usingFBO) {
                glDrawBuffer (GL_COLOR_ATTACHMENT0_EXT);
        }

        drawQuad(w,h);

        checkGLErrors("render()");
        if (!usingFBO) {
                glBindTexture(textureParameters.texTarget, destID);
//              glCopyTexSubImage2D(textureParameters.texTarget, 0, 0, 0, x, y, w, h);
                glCopyTexSubImage2D(textureParameters.texTarget, 0, x, y, 0, 0, w, h);
                checkGLErrors("copytexsubimage");
        }
}
void cleanupMatPix()
{
        glDetachShader(glslProgram, fragmentShader);
        glDeleteShader(fragmentShader);
        glDeleteProgram(glslProgram);
        if (usingFBO) {
                glDeleteFramebuffersEXT(1,&fb);
        }
        else {
                if(pbuff)
                {
                        PBuffer_End(pbuff);
                        PBuffer_Destroy(pbuff);
                }
        }
}

void initGLUT() {
        int argc = 0; char** argv;
        glutInit ( &argc, argv );
        glutCreateWindow("SAXPY TESTS");
          if(!usingFBO) {
#ifdef DEBUG__
              printf("initializing CGL\n");
#endif
              initCGL();
          }
  }

void initPbuffer(int w, int h)
{
        CGLContextObj cctx = CGLGetCurrentContext();
#ifdef DEBUG__
    printf("pbuffer texsize = %d, %d\n", w,h);
#endif
    pbuff = PBuffer_Create(cctx, w, h, 32, 0);
    pbuff->target =  textureParameters.texTarget;
    PBuffer_Begin(pbuff);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, w, 0.0, h);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glViewport(0, 0, w, h);
}
/**
 * Sets up GLEW to initialise OpenGL extensions
 */
void initGLEW() {
        int err = glewInit();
        // sanity check
        if (GLEW_OK != err) {
                printf((char*)glewGetErrorString(err));
                exit(ERROR_GLEW);
        }
}
```

```
void CGLCheckErr(int err, const char* text) {
        if (err) {
                printf("%s\n", text);
                printf("error : %d\n", err);
        }
}
void initCGL() {
        CGLContextObj cctx = CGLGetCurrentContext();
        if (!cctx) {
                printf("couldn't get current context\n");
        }
        int err;
        CGLPixelFormatAttribute attribs[] = { kCGLPFARobust, kCGLPFANoRecovery,
                        kCGLPFADoubleBuffer, kCGLPFAColorSize,
                        (CGLPixelFormatAttribute) 128, (CGLPixelFormatAttribute)0 };
        CGLPixelFormatObj fmt;
        long screens;
        err = CGLChoosePixelFormat(attribs, &fmt, &screens);
        CGLCheckErr(err, "create pixel fomat");
        if (!fmt) {
                printf("screens = %d\n", screens);
        }
        if (fmt) {
                long accel, colorbits;
                err = CGLDescribePixelFormat(fmt, 0, kCGLPFAAccelerated, &accel);
                err = CGLDescribePixelFormat(fmt, 0, kCGLPFAColorSize, &colorbits);
                CGLCheckErr(err, "read pixel format");
                if (!err) {
#ifdef DEBUG__
                        printf("Accellerated = %d\n", accel);
                        printf("Color Bits = %d\n", colorbits);
#endif
                }
                CGLContextObj ctx;
                err = CGLCreateContext(fmt, NULL, &ctx);
                CGLCheckErr(err, "create context");
                CGLDestroyPixelFormat(fmt);
                CGLCheckErr(err, "destroy pixel format");
                if (ctx) {
                        err = CGLSetCurrentContext(ctx);
                        CGLCheckErr(err, "Set Context");
                }
        }
}

void initFBO(void) {
        // create FBO (off-screen framebuffer)
        glGenFramebuffersEXT(1, &fb);
        glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fb);
        checkGLErrors("created FBO\n");
#ifdef DEBUG__
        printf("framebuffer created: %d\n", fb);
#endif
}
void bindTexToFBO(int tex) {
        glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, textureParameters.texTarget, tex, 0);
}
void setActiveShader(const char* op)
{
        const char* ops[] = {"+", "-", "*", "/", "==", "!=", "<", ">", "<=", ">=", "&&", "||", "++", "--", "dot", "slice", "trans", "!", "%", "neg
        for(int i = 0; i < 20; i++)
        {
                if(strcmp(op, ops[i]) == 0)
                {
                        glUseProgram(glslPrograms[i]);
                        yParam = glGetUniformLocation(glslPrograms[i], "textureY");
                        xParam = glGetUniformLocation(glslPrograms[i], "textureX");
                        texWidth = glGetUniformLocation(glslPrograms[i], "texWidth");
                        texHeight = glGetUniformLocation(glslPrograms[i], "texHeight");
                        commonLength = glGetUniformLocation(glslPrograms[i], "commonLength");
                        xoff = glGetUniformLocation(glslPrograms[i], "xoff");
                        yoff = glGetUniformLocation(glslPrograms[i], "yoff");
                        checkGLErrors("setActiveShader");
                        break;
                }
        }
}
```

```
void initGLSL(void) {
        glslProgram = glCreateProgram();
        fragmentShader = glCreateShader(GL_FRAGMENT_SHADER_ARB);
        const char* ops[] = {"+", "-", "*", "/", "==", "!=", "<", ">", "<=", ">=", "&&", "||", "++", "--", "dot", "slice", "trans", "!", "%", "neg
        for(int i = 0; i < 20; i++)
        {
                glslPrograms[i] = glCreateProgram();
                setShader(glslPrograms[i], ops[i]);
        }
}
bool isBinOp(const char* op)
{
        const char* ops[] = {"+", "-", "*", "/", "==", "!=", "<", ">", "<=", ">="};
        for(int i = 0; i < 10; i++)
        {
                if(strcmp(op, ops[i]) == 0)
                        return true;
        }
        return false;
}
bool isLogicOp(const char* op)
{
        const char* ops[] = {"&&", "||"};
        for(int i = 0; i < 2; i++)
        {
                if(strcmp(op, ops[i]) == 0)
                        return true;
        }
        return false;
}
bool isUnaryOp(const char* op)
{
        const char* ops[] = {"++", "--"};
        for(int i = 0; i < 2; i++)
        {
                if(strcmp(op, ops[i]) == 0)
                        return true;
        }
        return false;
}
void setShaderSrc(const char* op, char* source)
{
        if(strcmp(op, "dot") == 0)
        {
                sprintf(source,"%s"
                                        "  vec4 outFrag = vec4(0);"
                                        "  vec2 pix = gl_TexCoord[0].st;"
                                        "  for(float i = .5; i < float(commonLength); i=i+1.0) {"
                                        "               vec4 x = texture2DRect(textureX, vec2(i, pix.y));"
                                        "               vec4 y = texture2DRect(textureY, vec2(pix.x, i));"
                                        "               outFrag += x*y;"
                                        "  }"
                                        "  gl_FragColor = outFrag;}", shaderSrcBeg);
        }
        else if(strcmp(op, "slice") == 0)
        {
                sprintf(source,"%s"
                                "  vec2 off = vec2(xoff, yoff);"
                                "  vec4 x = texture2DRect(textureX, gl_TexCoord[0].st + off);"
                                "  gl_FragColor = x;}", shaderSrcBeg);
        }
        else if(strcmp(op, "trans") == 0)
        {
                sprintf(source,"%s"
                                "  vec4 x = texture2DRect(textureX, gl_TexCoord[0].ts);"
                                "  gl_FragColor = x;}", shaderSrcBeg);
        }
        else if(isBinOp(op))
        {
                sprintf(source,"%s"
                                "  vec4 y = texture2DRect(textureY, gl_TexCoord[0].st);"
                                "  vec4 x = texture2DRect(textureX, gl_TexCoord[0].st);"
                                "  gl_FragColor = vec4(x.r %s y.r);}", shaderSrcBeg, op);
        }
        else if(isLogicOp(op))
        {
                sprintf(source,"%s"
```

```
                                "  vec4 y = texture2DRect(textureY, gl_TexCoord[0].st);"
                                "  vec4 x = texture2DRect(textureX, gl_TexCoord[0].st);"
                                "  gl_FragColor = vec4(bool(x.r) %s bool(y.r));}", shaderSrcBeg, op);
        }
        else if(isUnaryOp(op))
        {
                sprintf(source,"%s"
                                "  vec4 x = texture2DRect(textureX, gl_TexCoord[0].st);"
                                "  gl_FragColor = vec4(%sx.r);}", shaderSrcBeg, op);
        }
        else if(strcmp(op, "!") == 0)
        {
                sprintf(source,"%s"
                                "  vec4 x = texture2DRect(textureX, gl_TexCoord[0].st);"
                                "  gl_FragColor = vec4(!bool(x.r));}", shaderSrcBeg);
        }
        else if(strcmp(op, "negate") == 0)
        {
                sprintf(source,"%s"
                                "  vec4 x = texture2DRect(textureX, gl_TexCoord[0].st);"
                                "  gl_FragColor = vec4(-x.r);}", shaderSrcBeg);
        }
        else if(strcmp(op, "%") == 0)
        {
                sprintf(source,"%s"
                                "  vec4 y = texture2DRect(textureY, gl_TexCoord[0].st);"
                                "  vec4 x = texture2DRect(textureX, gl_TexCoord[0].st);"
                                "  gl_FragColor = vec4(mod(x.r, y.r));}", shaderSrcBeg, op);
        }
}
void setShader(GLint program, const char* op)
{
        char source[1024];
        setShaderSrc(op, source);
        const char* sptr = source;
#ifdef DEBUG__
        printf("making shader for: %s\n", op);
#endif

        checkGLErrors("setShader: before loading source");
        glShaderSource(fragmentShader, 1, &sptr, NULL);
        checkGLErrors("setShader: after loading source");
        glCompileShader(fragmentShader);
        checkGLErrors("setShader: after compiling");
        printShaderInfoLog(fragmentShader);
        checkGLErrors("setShader: before attaching");
        glAttachShader (program, fragmentShader);
        checkGLErrors("setShader: before linking");
        glLinkProgram(program);
        printProgramInfoLog(program);
        checkGLErrors("setShader");
}

/**
 * Checks for OpenGL errors.
 * Extremely useful debugging function: When developing,
 * make sure to call this after almost every GL call.
 */
void checkGLErrors(const char *label) {
        GLenum errCode;
        const GLubyte *errStr;

        if ((errCode = glGetError()) != GL_NO_ERROR) {
                errStr = gluErrorString(errCode);
                printf("OpenGL ERROR: ");
                printf((char*)errStr);
                printf("(Label: ");
                printf(label);
                printf(")\n.");
        }
}

void drawQuad(int w, int h)
{
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0.0, w, 0.0, h);
```

```
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glViewport(0, 0, w, h);

        // and render the quad
        checkGLErrors("before render()");
        glBegin(GL_QUADS);
        glTexCoord2f(0.0, 0.0);
        glVertex2f(0.0, 0.0);
        glTexCoord2f(w, 0.0);
        glVertex2f(w, 0.0);
        glTexCoord2f(w, h);
        glVertex2f(w, h);
        glTexCoord2f(0.0, h);
        glVertex2f(0.0, h);
        glEnd();
}

/**
 * Checks framebuffer status.
 * Copied directly out of the spec, modified to deliver a return value.
 */
bool checkFramebufferStatus() {
        GLenum status;
        status = (GLenum) glCheckFramebufferStatusEXT(GL_FRAMEBUFFER_EXT);
        switch (status) {
        case GL_FRAMEBUFFER_COMPLETE_EXT:
                return true;
        case GL_FRAMEBUFFER_INCOMPLETE_ATTACHMENT_EXT:
                printf("Framebuffer incomplete, incomplete attachment\n");
                return false;
        case GL_FRAMEBUFFER_UNSUPPORTED_EXT:
                printf("Unsupported framebuffer format\n");
                return false;
        case GL_FRAMEBUFFER_INCOMPLETE_MISSING_ATTACHMENT_EXT:
                printf("Framebuffer incomplete, missing attachment\n");
                return false;
        case GL_FRAMEBUFFER_INCOMPLETE_DIMENSIONS_EXT:
                printf("Framebuffer incomplete, attached images must have same dimensions\n");
                return false;
        case GL_FRAMEBUFFER_INCOMPLETE_FORMATS_EXT:
                printf("Framebuffer incomplete, attached images must have same format\n");
                return false;
        case GL_FRAMEBUFFER_INCOMPLETE_DRAW_BUFFER_EXT:
                printf("Framebuffer incomplete, missing draw buffer\n");
                return false;
        case GL_FRAMEBUFFER_INCOMPLETE_READ_BUFFER_EXT:
                printf("Framebuffer incomplete, missing read buffer\n");
                return false;
        }
        return false;
}

/**
 * error checking for GLSL
 */
void printProgramInfoLog(GLuint obj) {
        GLint infologLength = 0;
        GLint charsWritten = 0;
        char *infoLog;
        glGetProgramiv(obj, GL_INFO_LOG_LENGTH, &infologLength);
        if (infologLength > 1) {
                infoLog = (char *)malloc(infologLength);
                glGetProgramInfoLog(obj, infologLength, &charsWritten, infoLog);
                printf(infoLog);
                printf("\n");
                free(infoLog);
        }
}
void printShaderInfoLog(GLuint obj) {
        GLint infologLength = 0;
        GLint charsWritten = 0;
        char *infoLog;
        glGetShaderiv(obj, GL_INFO_LOG_LENGTH, &infologLength);
        if (infologLength > 1) {
                infoLog = (char *)malloc(infologLength);
                glGetShaderInfoLog(obj, infologLength, &charsWritten, infoLog);
```

```
                printf(infoLog);
                printf("\n");
                free(infoLog);
        }
}

/**
 * Prints out given vector for debugging purposes.
 */
void printVector(const float *p, const int N) {
        for (int i=0; i<N; i++)
                printf("%f\n", p[i]);
}
```

B.3.6 util.hpp

```
#ifndef __UTIL__HPP
#define __UTIL__HPP

#include <time.h>
#include <GL/glew.h>
#include <GLUT/glut.h>
#include <OpenGL/OpenGL.h>
#include <AGL/agl.h>
#include "PBuffer.h"

// error codes
#define  ERROR_GLSL        -1
#define  ERROR_GLEW        -2
#define  ERROR_TEXTURE     -3
#define  ERROR_BINDFBO     -4
#define  ERROR_FBOTEXTURE  -5
#define  ERROR_PARAMS      -6

// struct for variable parts of GL calls (texture format, float format etc)
struct struct_textureParameters {
        char* name;
        GLenum texTarget;
        GLenum texInternalFormat;
        GLenum texFormat;
        char* shader_source;
}; // texture rectangles, texture_float_ARB, RGBA, 32 bits

void initMatPix(bool forceSofwareRenderer);
void compute(int destID, int xTexID, int yTexID, int w, int h, const char* op, int commonLength=0, int xoff=0, int yoff=0, int x=0, int y=0);
void bindTexToFBO(int tex);
void cleanupMatPix();

void initFBO();
void initGLUT();
void initGLEW();
void initCGL();
void initGLSL();
void initPbuffer(int w, int h);

void setActiveShader(const char* op);
void setShader(GLint shader, const char* op);
void setShaderSrc(const char* op, char* source);
void drawQuad(int w, int h);
void setShader(const char* op);
void CGLCheckErr(int err, const char* text);
void checkGLErrors(const char *label);
bool checkFramebufferStatus();
void printProgramInfoLog(GLuint obj);
void printShaderInfoLog(GLuint obj);
void printVector(const float *p, const int N);

extern struct_textureParameters textureParameters;
extern bool usingFBO;

#endif //__UTIL__HPP
```

B.4 Testing Configuration

B.4.1 Makefile

```
# the c++ compilation rules
SHELL=/bin/bash

CC = g++
INCLUDES = -I../
CFLAGS = -O -c
LDFLAGS = -framework OpenGL -framework GLUT -lglew

# the MPIX compilation paths for java execution
CLASSPATH = /Applications/Eclipse/plugins/org.antlr_2.7.6/antlr.jar:../bin

# c++ matpix library files
MPX_SRC = ../MatPix/util.cpp ../MatPix/PBuffer.cpp
MPX_OBJS = ${MPX_SRC:%.cpp=%.o}

# Successful and runtime-error tests go here
TESTS = badDim_dotprod_fail_r.mpx range_inv_row_fail_r.mpx range_inv_col_fail_r.mpx range_oob_col_fail_r.mpx range_oob_row_fail_r.mpx range_neg_fa

# mpx failures (scoping, etc) go here
TESTS_MPX_FAIL = constmatrix_notScalar_fail_m.mpx matIndex_fail_m.mpx syntax_fail_m.mpx constmatrix_fail_m.mpx funcDef_dup_fail_m.mpx funcCall_not
TEST_MPX_FAIL_CPP = ${TESTS_MPX_FAIL:%.mpx=cpp/%.cpp}

# the intermediate files follow from substitution rules
TEST_CPP = ${TESTS:%.mpx=cpp/%.cpp}
TEST_OBJS = ${TESTS:%.mpx=cpp/%.o}
TEST_EXEC = ${TESTS:%.mpx=%}
TEST_EXEC2 = ${TESTS:%.mpx=bin/%}
TEST_FILES = ${TEST_CPP} ${TEST_OBJS} ${TEST_EXEC2}

# the log files for each compilation step and running the tests
LOG_TEST_CPP = ${TESTS:%.mpx=cpp/log/%.cpp.log}
LOG_TEST_OBJS = ${TESTS:%.mpx=cpp/log/%.o.log}
LOG_TEST_EXEC = ${TESTS:%.mpx=bin/log/%.log}
LOG_TEST_FILES = ${LOG_TEST_CPP} ${LOG_TEST_OBJS} ${LOG_TEST_EXEC}

# this line says not to delete the cpp files
# which are "intermediate" files
.SECONDARY:  ${TEST_CPP}

# the main make rule - i.e. what happens if you just type "make"
all: ${TEST_EXEC} test_fail

# the rule to create .cpp files from .mpx files
# note - the rule assumes the input files are in the current dir
# and the output files go in the cpp/ directory
cpp/%.cpp: mpx/%.mpx
        @echo "trying to MPIX compile " $@ " from " $<
        java -cp ${CLASSPATH} Main $< $@ $(HAS_GPU) 2> $<.log
        mv $<.log mpx/log/

# the rule to create .o files from .cpp files
# note - the rule assumes the input and output files are in the same dir
%.o: %.cpp
        @echo "trying to create object file " $@ " from " $<
        g++ ${INCLUDES} ${CFLAGS} -o $@ $< 2> $<.log
        mv $<.log cpp/log/

# the rule to create the executables
# note - the rule assumes the input files are in the cpp/ dir
# and the output files go in the bin/ directory
${TEST_EXEC} : ${TEST_OBJS} ${MPX_OBJS}
        @echo "trying to create executable " $@ " from " $@.o
        g++ -o bin/$@ ${LDFLAGS} cpp/$@.o ${MPX_OBJS}

clean:
        rm -f ${TEST_FILES} ${LOG_TEST_FILES} ${MPX_OBJS}
        @echo "all cleaned up!"

# this will run all the tests
# print out the log info
test_fail:  ${TEST_MPX_FAIL_CPP}
test : ${TEST_EXEC} test_fail
```

```
#       @echo "running executable " $@
#       bin/$@ > bin/log/$@.log
#       ${foreach tests, ${TEST_EXEC}, ${shell echo "bin/${tests} > bin/log/${tests}.log" } }
#       ${foreach tests, ${TEST_EXEC}, ${shell bin/${tests} > bin/log/${tests}.log } }
        bin/run.py
```

B.4.2 run.py

```
#!/usr/bin/python

import os, difflib, sys, re

def isExec(name):
        if (name == "run.py" or name == "log" or name == ".svn" or name == "pass"):
                return False
        return True

print os.getcwd()
#filenames = os.listdir(os.getcwd()+"/bin")
stdin, stdout, stderr = os.popen3('cat Makefile | egrep "^TESTS" | cut -d "=" -f2 | tr " " "\n" | sed "s/\.mpx//g" | grep .')
files = stdout.read()
filenames = files.splitlines()

stdin, stdout, stderr = os.popen3('cat Makefile | egrep "^TESTS_MPX_FAIL" | cut -d "=" -f2 | tr " " "\n" | sed "s/\.mpx//g" | grep .')
fail_mpx_files = stdout.read()
fail_mpx_filenames = fail_mpx_files.splitlines()

fail_mpx_filenames.extend(filenames)

for filename in filenames:

        if isExec(filename):
                is_fail_mpxTest = re.search("_fail_m$", filename)
                is_fail_runTest = re.search("_fail_r$", filename)

                passfile = None
                pass_string = None
                passfile_lines = None
                try:
                        passfile = open("bin/pass/"+filename+".log")
                        pass_string = passfile.read()
                        passfile.close();
                        passfile = open("bin/pass/"+filename+".log")
                        passfile_lines = passfile.readlines()
                except IOError:
                        print "no passing to compare with file for ", filename
                        continue

                desired_mpx_err = None
                desired_cpp_err = None
                desired_run_err = None

                if is_fail_mpxTest:
                        desired_mpx_err = pass_string
                elif is_fail_runTest:
                        desired_mpx_err = ""
                        desired_cpp_err = ""
                        desired_run_err = pass_string
                else:
                        desired_mpx_err = ""
                        desired_cpp_err = ""
                        desired_run_err = ""


                # first see if the test compiled ok
                mpxLog = open("mpx/log/"+filename+".mpx.log")
                log = mpxLog.read()
                mpxLog.close()
                if not log == desired_mpx_err:
                        print "MATPIX DID NOT COMPILE AS EXPECTED (ERROR): ", filename
                        lines =  log.splitlines()
                        for line in lines:
                                print line
                        break
                else:
                        print "MATPIX COMPILED AS EXPECTED (SUCCESSFUL): ", filename
```

```
if is_fail_mpxTest:
        continue

# first see if we matpix compiled ok
cppLog = open("cpp/log/"+filename+".cpp.log")
log = cppLog.read()
cppLog.close()
if not log == desired_cpp_err:
        print "CPP DID NOT COMPILE AS EXPECTED (ERROR): ", filename
        lines =  log.splitlines()
        for line in lines:
                print line
        break
else:
        print "CPP COMPILED AS EXPECTED (SUCCESSFUL): " , filename

print "running ", filename
stdin, stdout, stderr = os.popen3("bin/"+filename)

err = stderr.read()
if not err == desired_run_err:
        print "RUNTIME STDERR NOT AS EXPECTED (ERROR): ", filename
        lines =  err.splitlines()
        for line in lines:
                print line
else:
        print "RUNTIME STDERR AS EXPECTED (SUCCESSFUL): ", filename

if is_fail_runTest:
        continue

logfile = open("bin/log/"+filename+".log", "w")
loglines = stdout.readlines()

if passfile:
        d = difflib.Differ()
        ## do a diff with the passing log files
        result = list(d.compare(passfile_lines, loglines))
        ## find lines in the compare that start with +/-
        match = re.search('^(-|\+)', "\n".join(result), re.M)
        if match:
                print "ERROR IN COMPUTATION FOR: ", filename
                print "#####################################################"
                print "############  DIFF RESULT   #########################"
                print "#####################################################"
                sys.stdout.writelines(result)
                print "#####################################################"
                print "#####################################################"
                logfile.writelines(result)
        else:
                print "RUNNNING ", filename, " PASSED!!"
                print "#####################################################"
                logfile.writelines(loglines)
else:
        logfile.writelines(loglines)
```

B.5 Testing Scripts

B.5.1 Successful Tests

B.5.2 arithmetic.mpx

```
// tests arithmetic

// scalar

print("a = 1 + 2");
a = 1 + 2;
print("a = ",a);

print("a = 4 - 2");
a = 4 - 2;
print("a = ",a);

print("a = 1 * 2");
a = 1 * 2;
print("a = ",a);

print("a = 4 ./ 2");
a = 4 ./ 2;
print("a = ",a);

print("a = a + 2");
a = a + 2;
print("a = ",a);

print("a = 2 + a");
a = 2 + a;
print("a = ",a);

print("a = a - 2");
a = a - 2;
print("a = ",a);

print("a = 2 - a");
a = 2 - a;
print("a = ",a);

print("a = a * 2");
a = a .* 2;
print("a = ",a);

print("a = 2 * a");
a = 2 .* a;
print("a = ",a);

print("a = a / 2");
a = a ./ 2;
print("a = ",a);

print("a = 2 / a");
a = 2 ./ a;
print("a = ",a);

print("a = a % 2");
a = a % 2;
print("a = ",a);

print("a = 2 % a");
a = 2 % a;
print("a = ",a);

print("a++");
a++;
print("a = ",a);

print("a--");
a--;
print("a = ",a);
```

```
// matrix

print("creating 2x2 matrix b");
matrix b[2,2];
b[:,:] = 2;
print("b = ");
print(b);

print("b = b + 2");
b[:,:] = b + 2;
print("b = ");
print(b);

print("b = 2 + b");
b = 2 + b;
print("b = ");
print(b);

print("b = b - 2");
b = b - 2;
print("b = ");
print(b);

print("b = 2 - b");
b = 2 - b;
print("b = ");
print(b);

print("b = b * 2");
b = b * 2;
print("b = ");
print(b);

print("b = 2 * b");
b = 2 * b;
print("b = ");
print(b);

print("b = b / 2");
b = b ./ 2;
print("b = ");
print(b);

print("b = 2 / b");
b = 2 ./ b;
print("b = ");
print(b);

print("b = b % 2");
b = b % 2;
print("b = ");
print(b);

print("b = 2 % b");
b = 2 % b;
print("b = ");
print(b);

print("b++");
b++;
print("b = ");
print(b);

print("b--");
b--;
print("b = ");
print(b);


// matrix - 1x1 matrix operations

print("creating 1x1 matrix c");
c=.25;
print(c);

print("b[:,:] = 4");
```

```
b[:,:] = 4;
print(b);

print("b = b + c");
b = b + c;
print("b = ");
print(b);

print("b = c + b");
b = c + b;
print("b = ");
print(b);

print("b = b - c");
b = b - c;
print("b = ");
print(b);

print("b = c - b");
b = c - b;
print("b = ");
print(b);

print("b = b * c");
b = b * c;
print("b = ");
print(b);

print("b = c * b");
b = c * b;
print("b = ");
print(b);

print("b = b .* c");
b = b .* c;
print("b = ");
print(b);

print("b = c .* b");
b = c .* b;
print("b = ");
print(b);

print("b = b ./ c");
b = b ./ c;
print("b = ");
print(b);

print("b = c ./ b");
b = c ./ b;
print("b = ");
print(b);

// negation tests
print("b = -c");
b = -c;
print("b = ");
print(b);

print("b = -1");
b = -1;
print("b = ");
print(b);
```

B.5.3 arithmetic.cpp

```
#include "MatPix/matpix.hpp"

int main(){

   bool forceSoftwareRender=true;;
   initMatPix(forceSoftwareRender);

   cout << "a = 1 + 2" << endl;
   Matrix MP_a(0);
   MP_a.assign(1.0+2.0);
```

```
cout << "a = " << MP_a;
cout << "a = 4 - 2" << endl;
MP_a.assign(4.0-2.0);
cout << "a = " << MP_a;
cout << "a = 1 * 2" << endl;
MP_a.assign(1.0*2.0);
cout << "a = " << MP_a;
cout << "a = 4 ./ 2" << endl;
MP_a.assign(4.0/2.0);
cout << "a = " << MP_a;
cout << "a = a + 2" << endl;
MP_a.assign(MP_a.add(2));
cout << "a = " << MP_a;
cout << "a = 2 + a" << endl;
MP_a.assign(MP_a.add(2));
cout << "a = " << MP_a;
cout << "a = a - 2" << endl;
MP_a.assign(MP_a.sub(2));
cout << "a = " << MP_a;
cout << "a = 2 - a" << endl;
MP_a.assign(Matrix(2,MP_a.w_,MP_a.h_).sub(MP_a));
cout << "a = " << MP_a;
cout << "a = a * 2" << endl;
MP_a.assign(MP_a.mul(2));
cout << "a = " << MP_a;
cout << "a = 2 * a" << endl;
MP_a.assign(MP_a.mul(2));
cout << "a = " << MP_a;
cout << "a = a / 2" << endl;
MP_a.assign(MP_a.div(2));
cout << "a = " << MP_a;
cout << "a = 2 / a" << endl;
MP_a.assign(Matrix(2,MP_a.w_,MP_a.h_).div(MP_a));
cout << "a = " << MP_a;
cout << "a = a % 2" << endl;
MP_a.assign(MP_a.mod(2));
cout << "a = " << MP_a;
cout << "a = 2 % a" << endl;
MP_a.assign(Matrix(2,MP_a.w_,MP_a.h_).mod(MP_a));
cout << "a = " << MP_a;
cout << "a++" << endl;
MP_a++;
cout << "a = " << MP_a;
cout << "a--" << endl;
MP_a--;
cout << "a = " << MP_a;
cout << "creating 2x2 matrix b" << endl;
Matrix MP_b(2, 2);
MP_b.sliceAssign(RangeList(),RangeList(),2);
cout << "b = " << endl;
cout << MP_b;
cout << "b = b + 2" << endl;
MP_b.sliceAssign(RangeList(),RangeList(),MP_b.add(2));
cout << "b = " << endl;
cout << MP_b;
cout << "b = 2 + b" << endl;
MP_b.assign(MP_b.add(2));
cout << "b = " << endl;
cout << MP_b;
cout << "b = b - 2" << endl;
MP_b.assign(MP_b.sub(2));
cout << "b = " << endl;
cout << MP_b;
cout << "b = 2 - b" << endl;
MP_b.assign(Matrix(2,MP_b.w_,MP_b.h_).sub(MP_b));
cout << "b = " << endl;
cout << MP_b;
cout << "b = b * 2" << endl;
MP_b.assign(MP_b.mul(2));
cout << "b = " << endl;
cout << MP_b;
cout << "b = 2 * b" << endl;
MP_b.assign(MP_b.mul(2));
cout << "b = " << endl;
cout << MP_b;
cout << "b = b / 2" << endl;
MP_b.assign(MP_b.div(2));
```

```
cout << "b = " << endl;
cout << MP_b;
cout << "b = 2 / b" << endl;
MP_b.assign(Matrix(2,MP_b.w_,MP_b.h_).div(MP_b));
cout << "b = " << endl;
cout << MP_b;
cout << "b = b % 2" << endl;
MP_b.assign(MP_b.mod(2));
cout << "b = " << endl;
cout << MP_b;
cout << "b = 2 % b" << endl;
MP_b.assign(Matrix(2,MP_b.w_,MP_b.h_).mod(MP_b));
cout << "b = " << endl;
cout << MP_b;
cout << "b++" << endl;
MP_b++;
cout << "b = " << endl;
cout << MP_b;
cout << "b--" << endl;
MP_b--;
cout << "b = " << endl;
cout << MP_b;
cout << "creating 1x1 matrix c" << endl;
Matrix MP_c(0);
MP_c.assign(.25);
cout << MP_c;
cout << "b[:,:] = 4" << endl;
MP_b.sliceAssign(RangeList(),RangeList(),4);
cout << MP_b;
cout << "b = b + c" << endl;
MP_b.assign(MP_b.add(MP_c));
cout << "b = " << endl;
cout << MP_b;
cout << "b = c + b" << endl;
MP_b.assign(MP_c.add(MP_b));
cout << "b = " << endl;
cout << MP_b;
cout << "b = b - c" << endl;
MP_b.assign(MP_b.sub(MP_c));
cout << "b = " << endl;
cout << MP_b;
cout << "b = c - b" << endl;
MP_b.assign(MP_c.sub(MP_b));
cout << "b = " << endl;
cout << MP_b;
cout << "b = b * c" << endl;
MP_b.assign(MP_b.dot(MP_c));
cout << "b = " << endl;
cout << MP_b;
cout << "b = c * b" << endl;
MP_b.assign(MP_c.dot(MP_b));
cout << "b = " << endl;
cout << MP_b;
cout << "b = b .* c" << endl;
MP_b.assign(MP_b.mul(MP_c));
cout << "b = " << endl;
cout << MP_b;
cout << "b = c .* b" << endl;
MP_b.assign(MP_c.mul(MP_b));
cout << "b = " << endl;
cout << MP_b;
cout << "b = b ./ c" << endl;
MP_b.assign(MP_b.div(MP_c));
cout << "b = " << endl;
cout << MP_b;
cout << "b = c ./ b" << endl;
MP_b.assign(MP_c.div(MP_b));
cout << "b = " << endl;
cout << MP_b;
cout << "b = -c" << endl;
MP_b.assign(Matrix(MP_c).negate());
cout << "b = " << endl;
cout << MP_b;
cout << "b = -1" << endl;
MP_b.assign(Matrix(1).negate());
cout << "b = " << endl;
cout << MP_b;
```

```
   cleanupMatPix();
   return 0;
}
```

B.5.4 arithmetic.log

```
a = 1 + 2
a = 3
a = 4 - 2
a = 2
a = 1 * 2
a = 2
a = 4 ./ 2
a = 2
a = a + 2
a = 4
a = 2 + a
a = 6
a = a - 2
a = 4
a = 2 - a
a = -2
a = a * 2
a = -4
a = 2 * a
a = -8
a = a / 2
a = -3.99902
a = 2 / a
a = -0.500122
a = a % 2
a = 1.49988
a = 2 % a
a = 0.500122
a++
a = 1.50012
a--
a = 0.500122
creating 2x2 matrix b
b =
2, 2
2, 2
b = b + 2
b =
4, 4
4, 4
b = 2 + b
b =
6, 6
6, 6
b = b - 2
b =
4, 4
4, 4
b = 2 - b
b =
-2, -2
-2, -2
b = b * 2
b =
-4, -4
-4, -4
b = 2 * b
b =
-8, -8
-8, -8
b = b / 2
b =
-3.99902, -3.99902
-3.99902, -3.99902
b = 2 / b
b =
-0.500122, -0.500122
-0.500122, -0.500122
b = b % 2
```

```
b =
1.49988, 1.49988
1.49988, 1.49988
b = 2 % b
b =
0.500122, 0.500122
0.500122, 0.500122
b++
b =
1.50012, 1.50012
1.50012, 1.50012
b--
b =
0.500122, 0.500122
0.500122, 0.500122
creating 1x1 matrix c
0.25
b[:,:] = 4
4, 4
4, 4
b = b + c
b =
4.25, 4.25
4.25, 4.25
b = c + b
b =
4.5, 4.5
4.5, 4.5
b = b - c
b =
4.25, 4.25
4.25, 4.25
b = c - b
b =
-4, -4
-4, -4
b = b * c
b =
-1, -1
-1, -1
b = c * b
b =
-0.25, -0.25
-0.25, -0.25
b = b .* c
b =
-0.0625, -0.0625
-0.0625, -0.0625
b = c .* b
b =
-0.015625, -0.015625
-0.015625, -0.015625
b = b ./ c
b =
-0.0624847, -0.0624847
-0.0624847, -0.0624847
b = c ./ b
b =
-4.00098, -4.00098
-4.00098, -4.00098
b = -c
b =
-0.25
b = -1
b =
-1
```

B.5.5 constmatrix.mpx

```
print("A = [1, 2, 3 | 4, 5, 6 | 7, 8, 9 | 10, 11, 12 ]");
A = [1, 2, 3 | 4, 5, 6 | 7, 8, 9 | 10, 11, 12 ];
print(A);

print("B = [1, 2, 3 , 4]");
B = [1, 2, 3 , 4];
print(B);
```

```
print("C = [1 | 2 | 3 | 4]");
C = [1 | 2 | 3 | 4];
print(C);
```

B.5.6 constmatrix.cpp

```
#include "MatPix/matpix.hpp"

int main(){

   bool forceSoftwareRender=true;;
   initMatPix(forceSoftwareRender);

   cout << "A = [1, 2, 3 | 4, 5, 6 | 7, 8, 9 | 10, 11, 12 ]" << endl;
   vector<float> vec_1;
   float array_1[] = {1,2,3,4,5,6,7,8,9,10,11,12};
   copy(array_1, array_1+12, back_inserter(vec_1));
   Matrix MP_A(0);
   MP_A.assign(vec_1, 4, 3);
   cout << MP_A;
   cout << "B = [1, 2, 3 , 4]" << endl;
   vector<float> vec_2;
   float array_2[] = {1,2,3,4};
   copy(array_2, array_2+4, back_inserter(vec_2));
   Matrix MP_B(0);
   MP_B.assign(vec_2, 1, 4);
   cout << MP_B;
   cout << "C = [1 | 2 | 3 | 4]" << endl;
   vector<float> vec_3;
   float array_3[] = {1,2,3,4};
   copy(array_3, array_3+4, back_inserter(vec_3));
   Matrix MP_C(0);
   MP_C.assign(vec_3, 4, 1);
   cout << MP_C;

   cleanupMatPix();
   return 0;
}
```

B.5.7 constmatrix.log

```
A = [1, 2, 3 | 4, 5, 6 | 7, 8, 9 | 10, 11, 12 ]
1, 2, 3
4, 5, 6
7, 8, 9
10, 11, 12
B = [1, 2, 3 , 4]
1, 2, 3, 4
C = [1 | 2 | 3 | 4]
1
2
3
4
```

B.5.8 ctrlFlow.mpx

```
// control flow tests

a = 3;
print("a = ", a);
if ( a == 3 ) {
        print("a == 3");
}
else {
        print("a != 3");
}
if ( a != 5 ) {
        print("a != 5");
}
else {
        print("a == 5");
}
```

```
if ( a > 1 ) {
        print("a > 1");
}
else {
        print("a <= 1");
}
if ( a < 4 ) {
        print("a < 4");
}
else {
        print("a >= 4");
}
if ( a <= 3 ) {
        print("a <= 3");
}
else {
        print("a > 3");
}
if ( a >= 3 ) {
        print("a >= 3");
}
else {
        print("a < 3");
}

print("while loop: 0..9");
i = 0;
while (i <= 9) {
        print(i);
        i = i + 1;
}

print("for loop: 0..9");
i = 1;
for (i=0:9) {
        print(i);
}

print("initializing 10x1 matrix b = 0..9");
matrix b[10,1] = [0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9];
print("for loop to index b[i,:]");
for (i=0:9) {
        print(b[i,:]);
}
print("while loop to index b[i,:]");
i = 0;
while (i <= 9) {
        print(b[i,:]);
        i = i + 1;
}
/*
print("iterator operator tests");
print("a = ",a);
for ( i=0:5 ) {
        print("i = ",i);
        if ( i == a ) {
                print("i == a");
        }
        if ( i != a ) {
                print("i != a");
        }
        if ( i < a ) {
                print("i < a");
        }
        if ( i <= a ) {
                print("i <= a");
        }
        if ( i > a ) {
                print("i > a");
        }
        if ( i >= a ) {
                print("i >= a");
        }
        if ( i==2 && a==3 ) {
                print("i==2 && a==3");
        }
        if ( i==2 || a==3 ) {
```

```
                print("i==2 && a==3");
        }
}
*/
```

B.5.9 ctrlFlow.cpp

```
#include "MatPix/matpix.hpp"

int main(){

   bool forceSoftwareRender=true;;
   initMatPix(forceSoftwareRender);

   Matrix MP_a(0);
   MP_a.assign(3);
   cout << "a = " << MP_a;
   if ((MP_a==3).getFloatValue())  {
      cout << "a == 3" << endl;
   }
   else  {
      cout << "a != 3" << endl;
   }

   if ((MP_a!=5).getFloatValue())  {
      cout << "a != 5" << endl;
   }
   else  {
      cout << "a == 5" << endl;
   }

   if ((MP_a>1).getFloatValue())  {
      cout << "a > 1" << endl;
   }
   else  {
      cout << "a <= 1" << endl;
   }

   if ((MP_a<4).getFloatValue())  {
      cout << "a < 4" << endl;
   }
   else  {
      cout << "a >= 4" << endl;
   }

   if ((MP_a<=3).getFloatValue())  {
      cout << "a <= 3" << endl;
   }
   else  {
      cout << "a > 3" << endl;
   }

   if ((MP_a>=3).getFloatValue())  {
      cout << "a >= 3" << endl;
   }
   else  {
      cout << "a < 3" << endl;
   }

   cout << "while loop: 0..9" << endl;
   Matrix MP_i(0);
   MP_i.assign(0);
   while ((MP_i<=9).getFloatValue())  {
      cout << MP_i;
      MP_i.assign(MP_i.add(1));
   }

   cout << "for loop: 0..9" << endl;
   MP_i.assign(1);
   for (Iter MP_i=0;MP_i<=9;++MP_i)  {
      cout << MP_i;
   }
```

```
   cout << "initializing 10x1 matrix b = 0..9" << endl;
   vector<float> vec_1;
   float array_1[] = {0,1,2,3,4,5,6,7,8,9};
   copy(array_1, array_1+10, back_inserter(vec_1));
   Matrix MP_b(10, 1);
   MP_b.assign(vec_1, 10, 1);
   cout << "for loop to index b[i,:]" << endl;
   for (Iter MP_i=0;MP_i<=9;++MP_i)  {
      cout << MP_b.slice(RangeList(MP_i.getFloatValue()),RangeList());
   }

   cout << "while loop to index b[i,:]" << endl;
   MP_i.assign(0);
   while ((MP_i<=9).getFloatValue())  {
      cout << MP_b.slice(RangeList(MP_i.getFloatValue()),RangeList());
      MP_i.assign(MP_i.add(1));
   }


   cleanupMatPix();
   return 0;
}
```

B.5.10 ctrlFlow.log

```
a = 3
a == 3
a != 5
a > 1
a < 4
a <= 3
a >= 3
while loop: 0..9
0
1
2
3
4
5
6
7
8
9
for loop: 0..9
0
1
2
3
4
5
6
7
8
9
initializing 10x1 matrix b = 0..9
for loop to index b[i,:]
0
1
2
3
4
5
6
7
8
9
while loop to index b[i,:]
0
1
2
3
4
5
6
7
8
```

```
9
```

B.5.11 dotprod.mpx

```
// declarations
print("matrix A[9, 1]");
print("matrix B[1, 9]");
matrix A[9, 1];
matrix B[1, 9];

A[:, :] = 3;
B[:, :] = 2;

print("A before sum");
print(A);

print("B before sum");
print(B);

// calculations
outer = A*B;

print("outer product");
print(outer);

// calculations
inner = B*A;

print("inner product");
print(inner);

print("C = [ 2, 3 | 1, 4 | 5, 1 ]");
C = [ 2, 3 | 1, 4 | 5, 1 ];
print(C);
print("D = [ 1, 2 | 4, 3 ]");
D = [ 1, 2 | 4, 3 ];
print(D);

print("E = C * D");
E = C * D;
print(E);
```

B.5.12 dotprod.cpp

```
#include "MatPix/matpix.hpp"

int main(){

   bool forceSoftwareRender=true;;
   initMatPix(forceSoftwareRender);

   cout << "matrix A[9, 1]" << endl;
   cout << "matrix B[1, 9]" << endl;
   Matrix MP_A(9, 1);
   Matrix MP_B(1, 9);
   MP_A.sliceAssign(RangeList(),RangeList(),3);
   MP_B.sliceAssign(RangeList(),RangeList(),2);
   cout << "A before sum" << endl;
   cout << MP_A;
   cout << "B before sum" << endl;
   cout << MP_B;
   Matrix MP_outer(0);
   MP_outer.assign(MP_A.dot(MP_B));
   cout << "outer product" << endl;
   cout << MP_outer;
   Matrix MP_inner(0);
   MP_inner.assign(MP_B.dot(MP_A));
   cout << "inner product" << endl;
   cout << MP_inner;
   cout << "C = [ 2, 3 | 1, 4 | 5, 1 ]" << endl;
   vector<float> vec_1;
   float array_1[] = {2,3,1,4,5,1};
```

```
   copy(array_1, array_1+6, back_inserter(vec_1));
   Matrix MP_C(0);
   MP_C.assign(vec_1, 3, 2);
   cout << MP_C;
   cout << "D = [ 1, 2 | 4, 3 ]" << endl;
   vector<float> vec_2;
   float array_2[] = {1,2,4,3};
   copy(array_2, array_2+4, back_inserter(vec_2));
   Matrix MP_D(0);
   MP_D.assign(vec_2, 2, 2);
   cout << MP_D;
   cout << "E = C * D" << endl;
   Matrix MP_E(0);
   MP_E.assign(MP_C.dot(MP_D));
   cout << MP_E;

   cleanupMatPix();
   return 0;
}
```

B.5.13 dotprod.log

```
matrix A[9, 1]
matrix B[1, 9]
A before sum
3
3
3
3
3
3
3
3
3
B before sum
2, 2, 2, 2, 2, 2, 2, 2
outer product
6, 6, 6, 6, 6, 6, 6, 6
6, 6, 6, 6, 6, 6, 6, 6
6, 6, 6, 6, 6, 6, 6, 6
6, 6, 6, 6, 6, 6, 6, 6
6, 6, 6, 6, 6, 6, 6, 6
6, 6, 6, 6, 6, 6, 6, 6
6, 6, 6, 6, 6, 6, 6, 6
6, 6, 6, 6, 6, 6, 6, 6
inner product
54
C = [ 2, 3 | 1, 4 | 5, 1 ]
2, 3
1, 4
5, 1
D = [ 1, 2 | 4, 3 ]
1, 2
4, 3
E = C * D
14, 13
17, 14
9, 13
```

B.5.14 functiondef.mpx

```
function add(a,b)
{
        return a+b;
};

matrix a = [1, 2 | 3, 4];
matrix b = [5, 6 | 7, 8];
print("a: \n",a);
print("b: \n",b);
print();
print("This is a+b");
print(add(a,b));
```

```
print();
print("This is the dot product of a and b");
print(dotprod(a, b));

function dotprod(a, b)
{
        return a*b;
};
```

B.5.15 functiondef.cpp

```
#include "MatPix/matpix.hpp"

Matrix add (Matrix MP_a,Matrix MP_b)  {
   return MP_a.add(MP_b);
}

Matrix dotprod (Matrix MP_a,Matrix MP_b)  {
   return MP_a.dot(MP_b);
}


int main(){

   bool forceSoftwareRender=true;;
   initMatPix(forceSoftwareRender);

   vector<float> vec_1;
   float array_1[] = {1,2,3,4};
   copy(array_1, array_1+4, back_inserter(vec_1));
   Matrix MP_a(0);
   MP_a.assign(vec_1, 2, 2);
   vector<float> vec_2;
   float array_2[] = {5,6,7,8};
   copy(array_2, array_2+4, back_inserter(vec_2));
   Matrix MP_b(0);
   MP_b.assign(vec_2, 2, 2);
   cout << "a: \n" << MP_a;
   cout << "b: \n" << MP_b;
   cout << endl;
   cout << "This is a+b" << endl;
   cout << add(MP_a,MP_b);
   cout << endl;
   cout << "This is the dot product of a and b" << endl;
   cout << dotprod(MP_a,MP_b);

   cleanupMatPix();
   return 0;
}
```

B.5.16 functiondef.log

```
a:
1, 2
3, 4
b:
5, 6
7, 8

This is a+b
6, 8
10, 12

This is the dot product of a and b
19, 22
43, 50
```

B.5.17 gauss_jordan.mpx

```
function eye(m)
{
    matrix Z[m,m];
    for(i=0:m-1)
```

```
    {
        Z[i,i] = 1;
    }
    return Z;
};

function gauss_jordan(V,n) //expand to use size function
{
    matrix Q[n,n*2];
    matrix E[n,n];
    E=eye(n);

    //Set Q to V augmented with nxn identity
    Q[:, 0:n-1] = V;
    Q[:, n:n*2-1] = E;
    r = 0;
    temp=0;//swap storage
    for (i=0:n-1)  //for each row
    {
        r = i;
        //Check for zero in eliminator row
        //If zero, then find row to swap with
        while((Q[i,i] ==0) && (r !=n-1))
        {
            r = r+1;
            temp = Q[i,:];
            Q[i,:] = Q[r,:];
            Q[r,:] = temp;
        }
        //Normalize the eliminator row
        if (Q[i,i]!=0)
        {
                Q[i,:] = Q[i,:]./Q[i,i];
        }

                //Use Eliminator row to eliminate
                //the column of all rows beneath
        for (k=i+1:n-1)
        {
            Q[k,:] = Q[k,:] - Q[i,:] .* Q[k,i];
        }
    }
    for (i=n-1:-1:0)
    {
        for (k = i-1:-1:0)
        {
            Q[k,:] = Q[k,:] - Q[i,:] .* Q[k,i];
        }
    }
    return Q[:,n:n*2-1];
};

d = 16;
matrix A[d, d];
A = eye(d);
A = A*5;
A[8,9]=4;
A[14,1] = 5;
A[15,1] = 15;


print("A:");
print(A);
print("inv(A):");
inverted = gauss_jordan(A,d);
print("Inverted matrix:");
print(inverted);
```

B.5.18 gauss_jordan.cpp

```
#include "MatPix/matpix.hpp"

Matrix eye (Matrix MP_m)  {
   Matrix MP_Z(MP_m, MP_m);
   for (Iter MP_i=0;MP_i<=MP_m.sub(1).getFloatValue();++MP_i)  {
      MP_Z.sliceAssign(RangeList(MP_i.getFloatValue()),RangeList(MP_i.getFloatValue()),1);
```

```
   }

   return MP_Z;
}

Matrix gauss_jordan (Matrix MP_V,Matrix MP_n)  {
   Matrix MP_Q(MP_n, MP_n.mul(2));
   Matrix MP_E(MP_n, MP_n);
   MP_E.assign(eye(MP_n));
   MP_Q.sliceAssign(RangeList(),RangeList(0,MP_n.sub(1).getFloatValue()),MP_V);
   MP_Q.sliceAssign(RangeList(),RangeList(MP_n.getFloatValue(),MP_n.mul(2).sub(1).getFloatValue()),MP_E);
   Matrix MP_r(0);
   MP_r.assign(0);
   Matrix MP_temp(0);
   MP_temp.assign(0);
   for (Iter MP_i=0;MP_i<=MP_n.sub(1).getFloatValue();++MP_i)  {
      MP_r.assign(MP_i);
      while ((MP_Q.slice(RangeList(MP_i.getFloatValue()),RangeList(MP_i.getFloatValue()))==0&&MP_r!=MP_n.sub(1)).getFloatValue())  {
         MP_r.assign(MP_r.add(1));
         MP_temp.assign(MP_Q.slice(RangeList(MP_i.getFloatValue()),RangeList()));
         MP_Q.sliceAssign(RangeList(MP_i.getFloatValue()),RangeList(),MP_Q.slice(RangeList(MP_r.getFloatValue()),RangeList()));
         MP_Q.sliceAssign(RangeList(MP_r.getFloatValue()),RangeList(),MP_temp);
      }

      if ((MP_Q.slice(RangeList(MP_i.getFloatValue()),RangeList(MP_i.getFloatValue()))!=0).getFloatValue())  {
         MP_Q.sliceAssign(RangeList(MP_i.getFloatValue()),RangeList(),MP_Q.slice(RangeList(MP_i.getFloatValue()),RangeList()).div(MP_Q.slice(Range
      }

      for (Iter MP_k=MP_i.add(1).getFloatValue();MP_k<=MP_n.sub(1).getFloatValue();++MP_k)  {
         MP_Q.sliceAssign(RangeList(MP_k.getFloatValue()),RangeList(),MP_Q.slice(RangeList(MP_k.getFloatValue()),RangeList()).sub(MP_Q.slice(Range
      }

   }

   for (Iter MP_i=MP_n.sub(1).getFloatValue();MP_i>=0;MP_i+=Matrix(1).negate().getFloatValue())  {
      for (Iter MP_k=MP_i.sub(1).getFloatValue();MP_k>=0;MP_k+=Matrix(1).negate().getFloatValue())  {
         MP_Q.sliceAssign(RangeList(MP_k.getFloatValue()),RangeList(),MP_Q.slice(RangeList(MP_k.getFloatValue()),RangeList()).sub(MP_Q.slice(Range
      }

   }

   return MP_Q.slice(RangeList(),RangeList(MP_n.getFloatValue(),MP_n.mul(2).sub(1).getFloatValue()));
}


int main(){

   bool forceSoftwareRender=true;;
   initMatPix(forceSoftwareRender);

   Matrix MP_d(0);
   MP_d.assign(16);
   Matrix MP_A(MP_d, MP_d);
   MP_A.assign(eye(MP_d));
   MP_A.assign(MP_A.mul(5));
   MP_A.sliceAssign(RangeList(8),RangeList(9),4);
   MP_A.sliceAssign(RangeList(14),RangeList(1),5);
   MP_A.sliceAssign(RangeList(15),RangeList(1),15);
   cout << "A:" << endl;
   cout << MP_A;
   cout << "inv(A):" << endl;
   Matrix MP_inverted(0);
   MP_inverted.assign(gauss_jordan(MP_A,MP_d));
   cout << "Inverted matrix:" << endl;
   cout << MP_inverted;

   cleanupMatPix();
   return 0;
}
```

B.5.19 gauss_jordan.log

```
A:
5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```

```
0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 5, 4, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0
0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0
0, 15, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5
inv(A):
Inverted matrix:
0.199951, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0.199951, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0.199951, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0.199951, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0.199951, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0.199951, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0.199951, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0.199951, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0.199951, -0.159922, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0.199951, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.199951, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.199951, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.199951, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.199951, 0
0, -0.199902, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.199951, 0
0, -0.599707, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.199951
```

B.5.20 logictest.mpx

```
// declarations
matrix A[4, 4];
matrix B[4, 4];
matrix C[4, 4];

A[:, :] = 3;
B[:, :] = 2;
C[:, :] = 0;

print("A before test");
print(A);

print("B before test");
print(B);

print("C before test");
print(C);

// calculations
C = A==B;

print("C = A==B");
print(C);

// calculations
C = A!=B;

print("C = A!=B");
print(C);

// calculations
C = A<B;

print("C = A<B");
print(C);

// calculations
C = A>B;

print("C = A>B");
print(C);
```

```
// calculations
C = A&&B;

print("C = A&&B");
print(C);

// calculations
C = A||B;

print("C = A||B");
print(C);

// calculations
C = !A;

print("C = !A");
print(C);
```

B.5.21 logictest.cpp

```
#include "MatPix/matpix.hpp"

int main(){

   bool forceSoftwareRender=true;;
   initMatPix(forceSoftwareRender);

   Matrix MP_A(4, 4);
   Matrix MP_B(4, 4);
   Matrix MP_C(4, 4);
   MP_A.sliceAssign(RangeList(),RangeList(),3);
   MP_B.sliceAssign(RangeList(),RangeList(),2);
   MP_C.sliceAssign(RangeList(),RangeList(),0);
   cout << "A before test" << endl;
   cout << MP_A;
   cout << "B before test" << endl;
   cout << MP_B;
   cout << "C before test" << endl;
   cout << MP_C;
   MP_C.assign(MP_A==MP_B);
   cout << "C = A==B" << endl;
   cout << MP_C;
   MP_C.assign(MP_A!=MP_B);
   cout << "C = A!=B" << endl;
   cout << MP_C;
   MP_C.assign(MP_A<MP_B);
   cout << "C = A<B" << endl;
   cout << MP_C;
   MP_C.assign(MP_A>MP_B);
   cout << "C = A>B" << endl;
   cout << MP_C;
   MP_C.assign(MP_A&&MP_B);
   cout << "C = A&&B" << endl;
   cout << MP_C;
   MP_C.assign(MP_A||MP_B);
   cout << "C = A||B" << endl;
   cout << MP_C;
   MP_C.assign(MP_A.notOp());
   cout << "C = !A" << endl;
   cout << MP_C;

   cleanupMatPix();
   return 0;
}
```

B.5.22 logictest.log

```
A before test
3, 3, 3, 3
3, 3, 3, 3
3, 3, 3, 3
3, 3, 3, 3
```

```
B before test
2, 2, 2, 2
2, 2, 2, 2
2, 2, 2, 2
2, 2, 2, 2
C before test
0, 0, 0, 0
0, 0, 0, 0
0, 0, 0, 0
0, 0, 0, 0
C = A==B
0, 0, 0, 0
0, 0, 0, 0
0, 0, 0, 0
0, 0, 0, 0
C = A!=B
1, 1, 1, 1
1, 1, 1, 1
1, 1, 1, 1
1, 1, 1, 1
C = A<B
0, 0, 0, 0
0, 0, 0, 0
0, 0, 0, 0
0, 0, 0, 0
C = A>B
1, 1, 1, 1
1, 1, 1, 1
1, 1, 1, 1
1, 1, 1, 1
C = A&&B
1, 1, 1, 1
1, 1, 1, 1
1, 1, 1, 1
1, 1, 1, 1
C = A||B
1, 1, 1, 1
1, 1, 1, 1
1, 1, 1, 1
1, 1, 1, 1
C = !A
0, 0, 0, 0
0, 0, 0, 0
0, 0, 0, 0
0, 0, 0, 0
```

B.5.23 recursive.mpx

```
function fact(n)
{
        if (n==0) {
                return 1;
        }
        return n*fact(n-1);
};

print("a = 5");
a=5;
print("b=fact(a)");
b=fact(a);
print("b = ", b);
```

B.5.24 recursive.cpp

```
#include "MatPix/matpix.hpp"

Matrix fact (Matrix MP_n)  {
   if ((MP_n==0).getFloatValue())  {
      return 1;
   }

   return MP_n.dot(fact(MP_n.sub(1)));
}
```

```
int main(){

   bool forceSoftwareRender=true;;
   initMatPix(forceSoftwareRender);

   cout << "a = 5" << endl;
   Matrix MP_a(0);
   MP_a.assign(5);
   cout << "b=fact(a)" << endl;
   Matrix MP_b(0);
   MP_b.assign(fact(MP_a));
   cout << "b = " << MP_b;

   cleanupMatPix();
   return 0;
}
```

B.5.25 recursive.log

```
a = 5
b=fact(a)
b = 120
```

B.5.26 regression.mpx

```
// inversion
function eye(m)
{
        matrix Z[m,m];
    for(i=0:m-1)
    {
        Z[i,i] = 1;
    }
    return Z;
};

function inv(V,n) //expand to use size function
{
    matrix Q[n,n*2];
    matrix E[n,n];
    E=eye(n);

    //Set Q to V augmented with nxn identity
    Q[:, 0:n-1] = V;
    Q[:, n:n*2-1] = E;
    r = 0;
    temp=0;//swap storage
    for (i=0:n-1)  //for each row
    {
        r = i;
        //Check for zero in eliminator row
        //If zero, then find row to swap with
        while((Q[i,i] ==0) && (r !=n-1))
        {
            r = r+1;
            temp = Q[i,:];
            Q[i,:] = Q[r,:];
            Q[r,:] = temp;
        }
        //Normalize the eliminator row
        if (Q[i,i]!=0)
        {
                Q[i,:] = Q[i,:]./Q[i,i];
        }

                //Use Eliminator row to eliminate
                //the column of all rows beneath
        for (k=i+1:n-1)
        {
            Q[k,:] = Q[k,:] - Q[i,:] .* Q[k,i];
        }
    }
    for (i=n-1:-1:0)
```

```
    {
        for (k = i-1:-1:0)
        {
            Q[k,:] = Q[k,:] - Q[i,:] .* Q[k,i];
        }
    }
    return Q[:,n:n*2-1];
};

// the linear regression test - woohoo!


n = 16; d = 2;
matrix X[n,d];
X[:, 0] = 1;
for(i=0:n-1){
        X[i, 1:d-1] = i;
}

print("X input value");
print(X);

matrix theta[d,1];
for(i=0:d-1){
        theta[i, 0] = 5.2*i+.3;
}
print("the theta regression values");
print(theta);

Y = X*theta;

print("the actual Y values");
print(Y);

theta_solve = inv(X'*X, d)*X'*Y;
print("regression solution for theta:");
print(theta_solve);

err = theta-theta_solve;
print("solution error:");
print(err);
```

B.5.27 regression.cpp

```
#include "MatPix/matpix.hpp"

Matrix eye (Matrix MP_m)  {
   Matrix MP_Z(MP_m, MP_m);
   for (Iter MP_i=0;MP_i<=MP_m.sub(1).getFloatValue();++MP_i)  {
      MP_Z.sliceAssign(RangeList(MP_i.getFloatValue()),RangeList(MP_i.getFloatValue()),1);
   }

   return MP_Z;
}

Matrix inv (Matrix MP_V,Matrix MP_n)  {
   Matrix MP_Q(MP_n, MP_n.mul(2));
   Matrix MP_E(MP_n, MP_n);
   MP_E.assign(eye(MP_n));
   MP_Q.sliceAssign(RangeList(),RangeList(0,MP_n.sub(1).getFloatValue()),MP_V);
   MP_Q.sliceAssign(RangeList(),RangeList(MP_n.getFloatValue(),MP_n.mul(2).sub(1).getFloatValue()),MP_E);
   Matrix MP_r(0);
   MP_r.assign(0);
   Matrix MP_temp(0);
   MP_temp.assign(0);
   for (Iter MP_i=0;MP_i<=MP_n.sub(1).getFloatValue();++MP_i)  {
      MP_r.assign(MP_i);
      while ((MP_Q.slice(RangeList(MP_i.getFloatValue()),RangeList(MP_i.getFloatValue()))==0&&MP_r!=MP_n.sub(1)).getFloatValue())  {
         MP_r.assign(MP_r.add(1));
         MP_temp.assign(MP_Q.slice(RangeList(MP_i.getFloatValue()),RangeList()));
         MP_Q.sliceAssign(RangeList(MP_i.getFloatValue()),RangeList(),MP_Q.slice(RangeList(MP_r.getFloatValue()),RangeList()));
         MP_Q.sliceAssign(RangeList(MP_r.getFloatValue()),RangeList(),MP_temp);
      }

      if ((MP_Q.slice(RangeList(MP_i.getFloatValue()),RangeList(MP_i.getFloatValue()))!=0).getFloatValue())  {
         MP_Q.sliceAssign(RangeList(MP_i.getFloatValue()),RangeList(),MP_Q.slice(RangeList(MP_i.getFloatValue()),RangeList()).div(MP_Q.slice(Range
```

```
      }

      for (Iter MP_k=MP_i.add(1).getFloatValue();MP_k<=MP_n.sub(1).getFloatValue();++MP_k)  {
         MP_Q.sliceAssign(RangeList(MP_k.getFloatValue()),RangeList(),MP_Q.slice(RangeList(MP_k.getFloatValue()),RangeList()).sub(MP_Q.slice(Range
      }

   }

   for (Iter MP_i=MP_n.sub(1).getFloatValue();MP_i>=0;MP_i+=Matrix(1).negate().getFloatValue())  {
      for (Iter MP_k=MP_i.sub(1).getFloatValue();MP_k>=0;MP_k+=Matrix(1).negate().getFloatValue())  {
         MP_Q.sliceAssign(RangeList(MP_k.getFloatValue()),RangeList(),MP_Q.slice(RangeList(MP_k.getFloatValue()),RangeList()).sub(MP_Q.slice(Range
      }

   }

   return MP_Q.slice(RangeList(),RangeList(MP_n.getFloatValue(),MP_n.mul(2).sub(1).getFloatValue()));
}


int main(){

   bool forceSoftwareRender=true;;
   initMatPix(forceSoftwareRender);

   Matrix MP_n(0);
   MP_n.assign(16);
   Matrix MP_d(0);
   MP_d.assign(2);
   Matrix MP_X(MP_n, MP_d);
   MP_X.sliceAssign(RangeList(),RangeList(0),1);
   for (Iter MP_i=0;MP_i<=MP_n.sub(1).getFloatValue();++MP_i)  {
      MP_X.sliceAssign(RangeList(MP_i.getFloatValue()),RangeList(1,MP_d.sub(1).getFloatValue()),MP_i);
   }

   cout << "X input value" << endl;
   cout << MP_X;
   Matrix MP_theta(MP_d, 1);
   for (Iter MP_i=0;MP_i<=MP_d.sub(1).getFloatValue();++MP_i)  {
      MP_theta.sliceAssign(RangeList(MP_i.getFloatValue()),RangeList(0),MP_i.mul(5.2).add(.3));
   }

   cout << "the theta regression values" << endl;
   cout << MP_theta;
   Matrix MP_Y(0);
   MP_Y.assign(MP_X.dot(MP_theta));
   cout << "the actual Y values" << endl;
   cout << MP_Y;
   Matrix MP_theta_solve(0);
   MP_theta_solve.assign(inv(MP_X.transpose().dot(MP_X),MP_d).dot(MP_X.transpose()).dot(MP_Y));
   cout << "regression solution for theta:" << endl;
   cout << MP_theta_solve;
   Matrix MP_err(0);
   MP_err.assign(MP_theta.sub(MP_theta_solve));
   cout << "solution error:" << endl;
   cout << MP_err;

   cleanupMatPix();
   return 0;
}
```

B.5.28 regression.log

```
X input value
1, 0
1, 1
1, 2
1, 3
1, 4
1, 5
1, 6
1, 7
1, 8
1, 9
1, 10
1, 11
1, 12
```

```
1, 13
1, 14
1, 15
the theta regression values
0.3
5.5
the actual Y values
0.3
5.8
11.3
16.8
22.3
27.8
33.3
38.8
44.3
49.8
55.3
60.8
66.3
71.8
77.3
82.8
regression solution for theta:
0.293782
5.50082
solution error:
0.00621778
-0.00082016
```

B.5.29 sliceAssign.mpx

```
// control flow tests

matrix A[16, 16];
matrix B[16, 16];
matrix C[16, 16];
matrix D[16, 16];
matrix E[16, 16];
matrix F[16, 16];
matrix H[16, 16];

// various slice assignments from constants
A[1, 2] = -1.1;
B[:, :] = 5.1;
C[0, :] = 0.5;
D[:, 0] = 0.75;
E[0, 0:5] = 1.5;
F[0:5, 0] = 1.75;
H[0:5, 0:5] = 2.5;

// shouldn't compile
//F[1:10] = 3;

print("A[1, 2] = -1.1");
print(A);
print("B[:, :] = 5.1");
print(B);
print("C[0, :] = 0.5");
print(C);
print("D[:, 0] = 0.75");
print(D);
print("E[0, 0:5] = 1.5");
print(E);
print("F[0:5, 0] = 1.75");
print(F);
print("H[0:5, 0:5] = 2.5");
print(H);

// now some slice to slice assignments
H[1, 2] = A[1, 2];
print("H[1, 2] = A[1, 2]");
print(H);

H[0, :] = C[0, :];
print("H[0, :] = C[0, :]");
```

```
print(H);

H[:, 0] = D[:, 0];
print("H[:, 0] = D[:, 0]");
print(H);

H[0, 0:5] = E[0, 0:5];
print("H[0, 0:5] = E[0, 0:5]");
print(H);

H[6:11, 1] = E[0, 0:5]';
print("H[6:11, 1] = E[0, 0:5]'");
print(H);

H[0:5, 0] = F[0:5, 0];
print("H[0:5, 0] = F[0:5, 0]");
print(H);

H[0:5, 0:5] = B[0:5, 0:5];
print("H[0:5, 0:5] = B[0:5, 0:5]");
print(H);

H[:, :] = B[:, :];
print("H[:, :] = B[:, :]");
print(H);
```

B.5.30 sliceAssign.cpp

```
#include "MatPix/matpix.hpp"

int main(){

   bool forceSoftwareRender=true;;
   initMatPix(forceSoftwareRender);

   Matrix MP_A(16, 16);
   Matrix MP_B(16, 16);
   Matrix MP_C(16, 16);
   Matrix MP_D(16, 16);
   Matrix MP_E(16, 16);
   Matrix MP_F(16, 16);
   Matrix MP_H(16, 16);
   MP_A.sliceAssign(RangeList(1),RangeList(2),Matrix(1.1).negate());
   MP_B.sliceAssign(RangeList(),RangeList(),5.1);
   MP_C.sliceAssign(RangeList(0),RangeList(),0.5);
   MP_D.sliceAssign(RangeList(),RangeList(0),0.75);
   MP_E.sliceAssign(RangeList(0),RangeList(0,5),1.5);
   MP_F.sliceAssign(RangeList(0,5),RangeList(0),1.75);
   MP_H.sliceAssign(RangeList(0,5),RangeList(0,5),2.5);
   cout << "A[1, 2] = -1.1" << endl;
   cout << MP_A;
   cout << "B[:, :] = 5.1" << endl;
   cout << MP_B;
   cout << "C[0, :] = 0.5" << endl;
   cout << MP_C;
   cout << "D[:, 0] = 0.75" << endl;
   cout << MP_D;
   cout << "E[0, 0:5] = 1.5" << endl;
   cout << MP_E;
   cout << "F[0:5, 0] = 1.75" << endl;
   cout << MP_F;
   cout << "H[0:5, 0:5] = 2.5" << endl;
   cout << MP_H;
   MP_H.sliceAssign(RangeList(1),RangeList(2),MP_A.slice(RangeList(1),RangeList(2)));
   cout << "H[1, 2] = A[1, 2]" << endl;
   cout << MP_H;
   MP_H.sliceAssign(RangeList(0),RangeList(),MP_C.slice(RangeList(0),RangeList()));
   cout << "H[0, :] = C[0, :]" << endl;
   cout << MP_H;
   MP_H.sliceAssign(RangeList(),RangeList(0),MP_D.slice(RangeList(),RangeList(0)));
   cout << "H[:, 0] = D[:, 0]" << endl;
   cout << MP_H;
   MP_H.sliceAssign(RangeList(0),RangeList(0,5),MP_E.slice(RangeList(0),RangeList(0,5)));
   cout << "H[0, 0:5] = E[0, 0:5]" << endl;
   cout << MP_H;
   MP_H.sliceAssign(RangeList(6,11),RangeList(1),MP_E.slice(RangeList(0),RangeList(0,5)).transpose());
```

```
   cout << "H[6:11, 1] = E[0, 0:5]'" << endl;
   cout << MP_H;
   MP_H.sliceAssign(RangeList(0,5),RangeList(0),MP_F.slice(RangeList(0,5),RangeList(0)));
   cout << "H[0:5, 0] = F[0:5, 0]" << endl;
   cout << MP_H;
   MP_H.sliceAssign(RangeList(0,5),RangeList(0,5),MP_B.slice(RangeList(0,5),RangeList(0,5)));
   cout << "H[0:5, 0:5] = B[0:5, 0:5]" << endl;
   cout << MP_H;
   MP_H.sliceAssign(RangeList(),RangeList(),MP_B.slice(RangeList(),RangeList()));
   cout << "H[:, :] = B[:, :]" << endl;
   cout << MP_H;

   cleanupMatPix();
   return 0;
}
```

B.5.31 sliceAssign.log

```
A[1, 2] = -1.1
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, -1.1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
B[:, :] = 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
C[0, :] = 0.5
0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
D[:, 0] = 0.75
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```

```
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
E[0, 0:5] = 1.5
1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
F[0:5, 0] = 1.75
1.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
1.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
1.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
1.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
1.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
1.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
H[0:5, 0:5] = 2.5
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
H[1, 2] = A[1, 2]
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
2.5, 2.5, -1.1, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
H[0, :] = C[0, :]
0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5
2.5, 2.5, -1.1, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
H[:, 0] = D[:, 0]
0.75, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5
0.75, 2.5, -1.1, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
H[0, 0:5] = E[0, 0:5]
1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5
0.75, 2.5, -1.1, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
H[6:11, 1] = E[0, 0:5]'
1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5
0.75, 2.5, -1.1, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
H[0:5, 0] = F[0:5, 0]
1.75, 1.5, 1.5, 1.5, 1.5, 1.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5
1.75, 2.5, -1.1, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
1.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
1.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
1.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
1.75, 2.5, 2.5, 2.5, 2.5, 2.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```

```
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
H[0:5, 0:5] = B[0:5, 0:5]
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 1.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
H[:, :] = B[:, :]
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1
```

B.5.32 sum.mpx

```
// summing test

print("A = [1, 2 , 3, 4, 5]");
A = [1, 2 , 3, 4, 5];

print("matrix B[5, 1] = 1");
matrix B[5, 1] = 1;

// sum over A
print("sum over A using A * B");
sum = A * B;
print("sum = ", sum);

// sum using for loop
print("sum using for loop 1..5");
sum = 0;
for ( i=1:5 ) {
        sum = sum + i;
}
print("sum = ", sum);
```

B.5.33 sum.cpp

```
#include "MatPix/matpix.hpp"

int main(){
```

```
   bool forceSoftwareRender=true;;
   initMatPix(forceSoftwareRender);

   cout << "A = [1, 2 , 3, 4, 5]" << endl;
   vector<float> vec_1;
   float array_1[] = {1,2,3,4,5};
   copy(array_1, array_1+5, back_inserter(vec_1));
   Matrix MP_A(0);
   MP_A.assign(vec_1, 1, 5);
   cout << "matrix B[5, 1] = 1" << endl;
   Matrix MP_B(5, 1);
   MP_B.assign(1);
   cout << "sum over A using A * B" << endl;
   Matrix MP_sum(0);
   MP_sum.assign(MP_A.dot(MP_B));
   cout << "sum = " << MP_sum;
   cout << "sum using for loop 1..5" << endl;
   MP_sum.assign(0);
   for (Iter MP_i=1;MP_i<=5;++MP_i)  {
      MP_sum.assign(MP_sum.add(MP_i));
   }

   cout << "sum = " << MP_sum;

   cleanupMatPix();
   return 0;
}
```

B.5.34 sum.log

```
A = [1, 2 , 3, 4, 5]
matrix B[5, 1] = 1
sum over A using A * B
sum = 15
sum using for loop 1..5
sum = 15
```

B.5.35 transpose.mpx

```
// control flow tests

matrix A[16, 10];
matrix B[10, 16];

// various slice assignments from constants
A[:, 0] = -1.1;
A[0, :] = 0.175;
B = A';

print("A[:, 0] = -1.1; A[0, :] = 0.175");
print(A);

print("B = A'");
print(B);
```

B.5.36 transpose.cpp

```
#include "MatPix/matpix.hpp"

int main(){

   bool forceSoftwareRender=true;;
   initMatPix(forceSoftwareRender);

   Matrix MP_A(16, 10);
   Matrix MP_B(10, 16);
   MP_A.sliceAssign(RangeList(),RangeList(0),Matrix(1.1).negate());
   MP_A.sliceAssign(RangeList(0),RangeList(),0.175);
   MP_B.assign(MP_A.transpose());
   cout << "A[:, 0] = -1.1; A[0, :] = 0.175" << endl;
   cout << MP_A;
   cout << "B = A'" << endl;
```

```
   cout << MP_B;

   cleanupMatPix();
   return 0;
}
```

B.5.37 transpose.log

```
A[:, 0] = -1.1; A[0, :] = 0.175
0.175, 0.175, 0.175, 0.175, 0.175, 0.175, 0.175, 0.175, 0.175, 0.175
-1.1, 0, 0, 0, 0, 0, 0, 0, 0, 0
-1.1, 0, 0, 0, 0, 0, 0, 0, 0, 0
-1.1, 0, 0, 0, 0, 0, 0, 0, 0, 0
-1.1, 0, 0, 0, 0, 0, 0, 0, 0, 0
-1.1, 0, 0, 0, 0, 0, 0, 0, 0, 0
-1.1, 0, 0, 0, 0, 0, 0, 0, 0, 0
-1.1, 0, 0, 0, 0, 0, 0, 0, 0, 0
-1.1, 0, 0, 0, 0, 0, 0, 0, 0, 0
-1.1, 0, 0, 0, 0, 0, 0, 0, 0, 0
-1.1, 0, 0, 0, 0, 0, 0, 0, 0, 0
-1.1, 0, 0, 0, 0, 0, 0, 0, 0, 0
-1.1, 0, 0, 0, 0, 0, 0, 0, 0, 0
-1.1, 0, 0, 0, 0, 0, 0, 0, 0, 0
-1.1, 0, 0, 0, 0, 0, 0, 0, 0, 0
-1.1, 0, 0, 0, 0, 0, 0, 0, 0, 0
B = A'
0.175, -1.1, -1.1, -1.1, -1.1, -1.1, -1.1, -1.1, -1.1, -1.1, -1.1, -1.1, -1.1, -1.1, -1.1, -1.1
0.175, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.175, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.175, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.175, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.175, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.175, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.175, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.175, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0.175, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```

B.6 Compiler Error Tests

B.6.1 lvalue_fail_m.mpx

```
1+2 = 4+4;
```

B.6.2 lvalue_fail_m.log

```
ERROR: Illegal operation: + cannot appear to the left of an assignment.
Exception: MPException
```

B.6.3 scope_if_fail_m.mpx

```
matrix A[1,1];
if (A > 1) {
        matrix B[2,2];
}
A = B;
```

B.6.4 scope_if_fail_m.log

```
ERROR: Illegal operation: Variable <B> does not exist.
Exception: MPException
```

B.6.5 scope_func1_fail_m.mpx

```
matrix A[1,1];

function abcd() {
        B=A; //This is not allowed
};
```

B.6.6 scope_func1_fail_m.log

```
ERROR: Illegal operation: Variable <A> does not exist.
Exception: MPException
```

B.6.7 scope_func2_fail_m.mpx

```
function abcd() {
        matrix B;
};

matrix C;
abcd();
C = B; // Not allowed.  B is not in scope
```

B.6.8 scope_func2_fail_m.log

```
ERROR: Illegal operation: Variable <B> does not exist.
Exception: MPException
```

B.6.9 funcCall_notDef_fail_m.mpx

```
function foo() {
        return 1;
};

matrix a = 1;
foo(a); //foo with one argument is not defined
```

B.6.10 funcCall_notDef_fail_m.log

```
ERROR: Illegal operation: Function <foo> with 1 arguments has not been defined.
Exception: MPException
```

B.6.11 funcDef_dup_fail_m.mpx

```
function foo(a, b) {

};

function foo(c, d) {

};
```

B.6.12 funcDef_dup_fail_m.log

```
ERROR: Illegal operation: A function named <foo> with 2 arguments already exists.
Exception: MPException
```

B.6.13 syntax_fail_m.mpx

```
vector z; //vector is not a valid type
matrix a //missing semicolon
```

B.6.14 syntax_fail_m.log

```
line 1:8: unexpected token: z
line 3:1: unexpected token: null
line 3:1: unexpected token: null
line 3:1: expecting SEMI, found 'null'
ERROR: Illegal operation: Variable <vector> does not exist.
Exception: MPException
```

B.6.15 matIndex_fail_m.mpx

```
a[1,2] = 3; //Not valid
```

B.6.16 matIndex_fail_m.log

```
ERROR: Illegal operation: Cannot index matrix a before it is initialized.
Exception: MPException
```

B.7 Runtime Error Tests

B.7.1 badDim_add_fail_r.mpx

```
matrix B[16, 16];
matrix C[3, 3] = 2;

Z = C + B;
```

B.7.2 badDim_add_fail_r.cpp

```
#include "MatPix/matpix.hpp"

int main(){

   bool forceSoftwareRender=true;;
   initMatPix(forceSoftwareRender);

   Matrix MP_B(16, 16);
   Matrix MP_C(3, 3);
   MP_C.assign(2);
   Matrix MP_Z(0);
   MP_Z.assign(MP_C.add(MP_B));

   cleanupMatPix();
   return 0;
}
```

B.7.3 badDim_add_fail_r.log

```
Error: Matrix dimensions do not agree. Exiting program.
Left Matrix size: (3, 3)
Right Matrix size: (16, 16)
```

B.7.4 badDim_dotprod_fail_r.mpx

```
matrix B[3, 16] = 4;
matrix C[3, 16] = 2;

Z = C * B;
```

B.7.5 badDim_dotprod_fail_r.cpp

```
#include "MatPix/matpix.hpp"

int main(){

   bool forceSoftwareRender=true;;
   initMatPix(forceSoftwareRender);

   Matrix MP_B(3, 16);
   MP_B.assign(4);
   Matrix MP_C(3, 16);
   MP_C.assign(2);
   Matrix MP_Z(0);
   MP_Z.assign(MP_C.dot(MP_B));

   cleanupMatPix();
   return 0;
}
```

B.7.6 badDim_dotprod_fail_r.log

```
Matrix Dimensions for Dot Product Don't agree
Left Matrix size: (3, 16)
Right Matrix size: (3, 16)
```

B.7.7 range_inv_row_fail_r.mpx

```
matrix A[4,2];
A[1:0,1] = 2;
```

B.7.8 range_inv_row_fail_r.cpp

```
#include "MatPix/matpix.hpp"

int main(){

   bool forceSoftwareRender=true;;
   initMatPix(forceSoftwareRender);

   Matrix MP_A(4, 2);
   MP_A.sliceAssign(RangeList(1,0),RangeList(1),2);

   cleanupMatPix();
   return 0;
}
```

B.7.9 range_inv_row_fail_r.log

```
Error: Invalid row range. Exiting program.
```

B.7.10 range_inv_col_fail_r.mpx

```
matrix A[4,2];
A[0:1,1:0] = 2;
```

B.7.11 range_inv_col_fail_r.cpp

```
#include "MatPix/matpix.hpp"

int main(){

   bool forceSoftwareRender=true;;
   initMatPix(forceSoftwareRender);

   Matrix MP_A(4, 2);
   MP_A.sliceAssign(RangeList(0,1),RangeList(1,0),2);

   cleanupMatPix();
   return 0;
}
```

B.7.12 range_inv_col_fail_r.log

```
Error: Invalid column range. Exiting program.
```

B.7.13 range_oob_col_fail_r.mpx

```
matrix A[4,2];
A[0:3,2] = 2;
```

B.7.14 range_oob_col_fail_r.cpp

```
#include "MatPix/matpix.hpp"

int main(){

   bool forceSoftwareRender=true;;
   initMatPix(forceSoftwareRender);

   Matrix MP_A(4, 2);
   MP_A.sliceAssign(RangeList(0,3),RangeList(2),2);

   cleanupMatPix();
   return 0;
}
```

B.7.15 range_oob_col_fail_r.log

```
Error: Out-of-bounds column value: 2. Exiting program.
```

B.7.16 range_oob_row_fail_r.mpx

```
matrix A[4,2];
A[1:4,:] = 2;
```

B.7.17 range_oob_row_fail_r.cpp

```
#include "MatPix/matpix.hpp"

int main(){

   bool forceSoftwareRender=true;;
   initMatPix(forceSoftwareRender);

   Matrix MP_A(4, 2);
   MP_A.sliceAssign(RangeList(1,4),RangeList(),2);

   cleanupMatPix();
   return 0;
}
```

B.7.18 range_oob_row_fail_r.log

```
Error: Out-of-bounds row index: 4. Exiting program.
```

B.7.19 range_neg_fail_r.mpx

```
matrix A[4,1] = 1;
A[-1:3,:] = 2;
```

B.7.20 range_neg_fail_r.cpp

```
#include "MatPix/matpix.hpp"

int main(){

   bool forceSoftwareRender=true;;
   initMatPix(forceSoftwareRender);

   Matrix MP_A(4, 1);
   MP_A.assign(1);
   MP_A.sliceAssign(RangeList(Matrix(1).negate().getFloatValue(),3),RangeList(),2);

   cleanupMatPix();
   return 0;
}
```

B.7.21 range_neg_fail_r.log

```
Error: No negative row indices. Exiting program.
```

B.8 Performance Tests

B.8.1 Timing Test Script

```
#!/usr/local/bin/python

import os, difflib, sys, re, time
from numpy import *
import pylab as p
import matplotlib.axes3d as p3
from timeit import Timer

CLASSPATH = "/Applications/Eclipse/plugins/org.antlr_2.7.6/antlr.jar:../../bin"
LDFLAGS = "-framework OpenGL -framework GLUT -lglew"
INCLUDES = "-I../../"
MPX_OBJS = "../../MatPix/util.o ../../MatPix/PBuffer.o"
CFLAGS = "-O -c"

#HAS_GPU=""
HAS_GPU="1"

binops = [["+", "add"], ["-", 'sub'], [".*", 'mul'], ["./", 'div'],
          ["==", 'eq'], ["!=", 'neq'], ["<", 'lessthan'], [">", "greaterthan"],
          ["<=", "leq"], [">=", "geq"], ["*", "dot"], ["%", "mod"]]
logicops = [["&&", "and"], ["||", "or"]]
post_unops = [["++", "inc"], ["--", "dec"]]
pre_unops = [["'", "trans"]]
all_ops = binops+logicops+logicops+pre_unops
#all_ops = [["+", "add"], ["-", 'sub'], [".*", 'mul'], ["./", 'div'], ["'", "eq"]]
#all_ops = [["+", "add"]]

def mpx_compile(file):
        outfile = os.getcwd()+"/cpp/" + os.path.basename(file) + ".cpp"
        cmd = "java -cp " + CLASSPATH + " Main " + file + " " + outfile + " " + HAS_GPU
        print cmd
        stdin, stdout, stderr = os.popen3(cmd)
        print stdout.read(), stderr.read()
        cpp_compile(outfile)

def cpp_compile(file):
        outfile = file+".o"
        cmd = "g++ " + INCLUDES + " " + CFLAGS + " -o "+ outfile +  " " + file
        print cmd
        stdin, stdout, stderr = os.popen3(cmd)
        print stdout.read(), stderr.read()
        obj_compile(outfile)

def obj_compile(file):
        outfile, ext = os.path.splitext(os.path.basename(file))
        outfile, ext = os.path.splitext(outfile)
        outfile, ext = os.path.splitext(outfile)
        outfile, ext = os.path.splitext(outfile)
        outfile = os.getcwd()+"/bin/" + outfile
        cmd = " g++ -o " + outfile + " " +  LDFLAGS + " " +  file + " " + MPX_OBJS
        print cmd
        stdin, stdout, stderr = os.popen3(cmd)
        print stdout.read(), stderr.read()


def make_op(filename, source):
        mpx = open(filename, 'w')
        mpx.write(source)
        mpx.close()
        os.popen("chmod 755 " + filename)

def build_base(loops, size):
        loop_sz = "_"+str(loops)+"_"+str(size)
        #loop_sz = "_"+str(size)
        mpxdir = os.getcwd()+"/mpx/"
        pydir = os.getcwd()+"/python/"
```

```
        src, npy = make_base(loops, size)
        make_op(mpxdir+"base"+loop_sz + ".txt", src)
        make_op(pydir+"base"+loop_sz + ".py", npy)

def make_base(loops, size):
        src = "matrix A["+str(size)+","+str(size)+"];\n"
        src+= "matrix B["+str(size)+","+str(size)+"];\n"
        src+= "A[:, :] = .25;\n"
        src+= "B[:, :] = .75;\n"
        src+= "print(\"glFinish()\");\n"
        npy = "#!/usr/local/bin/python\n"
        npy+= "from numpy import *\n"
        npy+= "A = zeros((" + str(size) + ", " + str(size) + "))\n"
        npy+= "B = zeros((" + str(size) + ", " + str(size) + "))\n"
        npy+= "A[:,:] = .75*ones((" + str(size) + ", " + str(size) + "))\n"
        npy+= "B[:,:] = .25*ones((" + str(size) + ", " + str(size) + "))\n"
        return src, npy

def build_tests(loops, size):
        loop_sz = "_"+str(loops)+"_"+str(size)
        #loop_sz = "_"+str(size)
        mpxdir = os.getcwd()+"/mpx/"
        pydir = os.getcwd()+"/python/"
        for op in binops:
                opname, filename = op
                src, npy = make_binop(loops, size, opname)
                make_op(mpxdir+filename+loop_sz + ".txt", src)
                make_op(pydir+filename+loop_sz + ".py", npy)
        for op in logicops:
                opname, filename = op
                src, npy = make_logicop(loops, size, opname)
                make_op(mpxdir+filename+loop_sz + ".txt", src)
                make_op(pydir+filename+loop_sz + ".py", npy)
        for op in pre_unops:
                opname, filename = op
                src, npy = make_preunop(loops, size, opname)
                make_op(mpxdir+filename+loop_sz + ".txt", src)
                make_op(pydir+filename+loop_sz + ".py", npy)
        for op in post_unops:
                opname, filename = op
                src, npy = make_postunop(loops, size, opname)
                make_op(mpxdir+filename+loop_sz + ".txt", src)
                make_op(pydir+filename+loop_sz + ".py", npy)

def make_binop(loops, size, op):
        src = "matrix A["+str(size)+","+str(size)+"];\n"
        src+= "matrix B["+str(size)+","+str(size)+"];\n"
        src+= "A[:, :] = .25;\n"
        src+= "B[:, :] = .75;\n"
        src+= "for(i=0:" + str(loops-1) +")\n"
        src+= "{\n"
        src+= "A = A " + op + " B;\n"
        src+= "}\n"
        src+= "print(\"glFinish()\");\n"

#!/usr/bin/python
        npy = "#!/usr/local/bin/python\n"
        npy+= "from numpy import *\n"
        npy+= "A = zeros((" + str(size) + ", " + str(size) + "))\n"
        npy+= "B = zeros((" + str(size) + ", " + str(size) + "))\n"
        npy+= "A[:,:] = .75*ones((" + str(size) + ", " + str(size) + "))\n"
        npy+= "B[:,:] = .25*ones((" + str(size) + ", " + str(size) + "))\n"
        npy+= "for i in range(" + str(loops) + "):\n"
        if op == ".*":
                npy+= "\tA = A*B\n"
                #npy+= "A = A*B\n"
        elif op == ".*":
                npy+= "\tA = A*B\n"
                #npy+= "A = A*B\n"
        elif op == "./":
                npy+= "\tA = A/B\n"
                #npy+= "A = A/B\n"
        elif op == "*":
                npy+= "\tA = dot(A,B)\n"
                #npy+= "A = dot(A,B)\n"
        else:
                npy+= "\tA = A " + op + "B\n"
```

```
                #npy+= "A = A " + op + "B\n"
        return src, npy

def make_logicop(loops, size, op):
        src = "matrix A["+str(size)+","+str(size)+"];\n"
        src+= "matrix B["+str(size)+","+str(size)+"];\n"
        src+= "A[:, :] = .25;\n"
        src+= "B[:, :] = .75;\n"
        src+= "for(i=0:" + str(loops-1) +")\n"
        src+= "{\n"
        src+= "A = A " + op + " B;\n"
        src+= "}\n"
        src+= "print(\"glFinish()\");\n"

        npy = "#!/usr/local/bin/python\n"
        npy+= "from numpy import *\n"
        npy+= "A = zeros((" + str(size) + ", " + str(size) + "))\n"
        npy+= "B = zeros((" + str(size) + ", " + str(size) + "))\n"
        npy+= "A[:,:] = .75*ones((" + str(size) + ", " + str(size) + "))\n"
        npy+= "B[:,:] = .25*ones((" + str(size) + ", " + str(size) + "))\n"
        npy+= "A.dtype=bool\n"
        npy+= "B.dtype=bool\n"
        npy+= "for i in range(" + str(loops) + "):\n"
        npy+= "\tA = A " + op + "B\n"
        #npy+= "C = A " + op + "B\n"
        return src, npy

def make_postunop(loops, size, op):
        src = "matrix A["+str(size)+","+str(size)+"];\n"
        src+= "A[:, :] = .25;\n"
        src+= "for(i=0:" + str(loops-1) +")\n"
        src+= "{\n"
        src+= "B = A" + op + ";\n"
        src+= "}\n"
        src+= "print(\"glFinish()\");\n"

        npy = "#!/usr/local/bin/python\n"
        npy+= "from numpy import *\n"
        npy+= "A = zeros((" + str(size) + ", " + str(size) + "))\n"
        npy+= "A[:,:] = .75*ones((" + str(size) + ", " + str(size) + "))\n"
        npy+= "for i in range(" + str(loops) + "):\n"
        npy+= "\tA = A +1\n"
#       npy+= "B = A +1\n"
        return src, npy

def make_preunop(loops, size, op):
        src = "matrix A["+str(size)+","+str(size)+"];\n"
        src+= "A[:, :] = .25;\n"
        src+= "for(i=0:" + str(loops-1) +")\n"
        src+= "{\n"
        src+= "B = A"+ op + ";\n"
        src+= "}\n"
        src+= "print(\"glFinish()\");\n"

        npy = "#!/usr/local/bin/python\n"
        npy+= "from numpy import *\n"
        npy+= "A = zeros((" + str(size) + ", " + str(size) + "))\n"
        npy+= "A[:,:] = .75*ones((" + str(size) + ", " + str(size) + "))\n"
        npy+= "for i in range(" + str(loops) + "):\n"
        npy+= "\tA = A.transpose()\n"
#       npy+= "B = A.transpose()\n"
        return src, npy

loops = 2**array(range(1, 8))
matsize = 2**array(range(6,11))
source_dir = os.getcwd()+"/mpx"
exec_dir = os.getcwd()+"/bin/"
py_dir = os.getcwd()+"/python/"
log = "%16s \t Matpix Speed(ms) \t Python speed(ms)\n" %("operation")

loop = 8
for sz in matsize:
        print "sz: %d" %(sz)
        build_tests(loop, sz)
        build_base(loop, sz)
msz = matsize[size(matsize)-1]
for loop in loops:
```

```
        print "loop: %d" %(loop)
        build_tests(loop, msz)
        build_base(loop, sz)

for op in all_ops:
        opname, filename = op
        if not filename == ".svn":
                loop = 8
                for sz in matsize:
                        name = filename + "_"+str(loop)+"_"+str(sz) + ".txt"
                        #name = filename + "_"+str(sz) + ".txt"
                        print "compiling " + filename
                        mpx_compile(source_dir + "/" + name)
                for sz in matsize:
                        name = "base" + "_"+str(loop)+"_"+str(sz) + ".txt"
                        print "compiling " + filename
                        mpx_compile(source_dir + "/" + name)

                msz = matsize[size(matsize)-1]
                for loop in loops:
                        name = filename + "_"+str(loop)+"_"+str(msz) + ".txt"
                        print "compiling " + filename
                        mpx_compile(source_dir + "/" + name)
                for loop in loops:
                        name = "base" + "_"+str(loop)+"_"+str(msz) + ".txt"
                        print "compiling " + filename
                        mpx_compile(source_dir + "/" + name)

for op in all_ops:
        opname, filename = op
        if not filename == ".svn":
#       if filename == "add":
                try:
                        msz = matsize[size(matsize)-1]
                        mpxtime = array(loops, dtype=float)
                        pytime = array(loops, dtype=float)
                        i=0
                        for loop in loops:
                                name = filename + "_"+str(loop)+"_"+str(msz)
                                base = "base" + "_"+str(loop)+"_"+str(msz)
                                #name = filename + "_"+str(msz)
                                print "executing " + name
                                t = Timer("os.popen('"+exec_dir+name+"')", "import os")
                                t2 = Timer("os.popen('"+exec_dir+base+"')", "import os").timeit(1)
                                mpxtime[i] = t.timeit(1) - t2
                                t = Timer("os.popen('"+py_dir+name+".py')", "import os")
                                pytime[i] = t.timeit(1) - t2
                                log += "%16s \t %06f \t %06f\n" %(filename, mpxtime[i], pytime[i])
                                i+=1
                        print log
                        make_op(filename+"loops.txt", log)

                        p.figure(figsize=(12,6))
                        p.subplot(121)
                        p.semilogx(loops, mpxtime, 'r', loops, pytime, 'b')
                        p.title("execution comparison for " + filename + " operation \n on %dx%d matrix (matrices)" %(msz, msz))
                        p.ylabel("execution time(sec)", fontsize=16)
                        p.xlabel("iterations", fontsize=16)
                        p.legend(("Matpix", "Python"))
                        make_op("log/"+filename+"loops.txt", log)

                        log=""
                        loop = 8
                        mpxtime = array(matsize, dtype=float)
                        pytime = array(matsize, dtype=float)
                        i=0
                        for sz in matsize:
                                #name = filename + "_"+str(sz)
                                name = filename + "_"+str(loop)+"_"+str(sz)
                                base = "base" + "_"+str(loop)+"_"+str(msz)
                                print "executing " + name
                                t = Timer("os.popen('"+exec_dir+name+"')", "import os")
                                t2 = Timer("os.popen('"+exec_dir+base+"')", "import os").timeit(1)
                                mpxtime[i] = t.timeit(1)-t2
                                t = Timer("os.popen('"+py_dir+name+".py')", "import os")
                                pytime[i] = t.timeit(1)-t2
                                log += "%16s \t %06f \t %06f\n" %(filename, mpxtime[i], pytime[i])
```

```
                i+=1
        print log
        make_op("log/"+filename+"sizes.txt", log)

        p.subplot(122)
        p.semilogx(matsize, mpxtime, 'r', matsize, pytime, 'b')
        p.title("execution comparison for 8 iterations \n of " + filename + " operation")
        p.ylabel("execution time(sec)", fontsize=16)
        p.xlabel("length on a side for a square matrix", fontsize=16)
        p.legend(("Matpix", "Python"))

        p.savefig("../../../documents/figures/" + filename + ".png")
        #p.show()
except:
        print "rouble executing " + filename
```