Mx: A programming language for scientific computation

Tiantian Zhou [1] Hanhua Feng [2] Yong Man Ra Chang Woo Lee [3]
{tz2002, hf2048, yr86, cl588}@columbia.edu

May 13, 2003

[1]Tiantian Zhou is a second-year Ph.D student of the Department of Computer Science.
[2]Hanhua Feng is a first-year Ph.D student of the Department of Computer Science.
[3]Yong Man Ra and Chang Woo Lee are senior undergraduate students of the Fu Foundation School of Engineering and Applied Science.

Contents

Chapter 1

Introduction

In this project, we have designed and implemented a programming language, Mx, designated to scientific computations.

1.1 Background

For the past fifty years, the demands for high-performance scientific computations have been pushing various computer technologies. Among them, programming languages that are more efficient, more convenient, and more natural to scientific computations have also been developed. Unfortunately, their progress is more or less lagged behind other computer technologies.

Fortran is one of the first-generation programming languages, which has created many irreplaceable scientific packages. Although there were many other generations of Fortran standards, the backward compatibility constantly hindered its development. Fortran can hardly keep up with the development of modern programming language and software engineering techniques, modern computer hardware architecture, or even the needs for mathematical efficiency and conveniences. Meanwhile, the progress of computer science in recent years is essentially dominated by the industry of information technologies, where the market of scientific computations is too small to be seen. Universities no longer implement Fortran so that the population of Fortran programmer becomes less and less.

Scientific investigators have to attempt other possible alternatives. When they choose a programming language for their projects, they would take these issues into account: (1) suitable and efficient for scientific computation, (2) easy to learn (better be the language they have learned in class), and (3) popularity, so that their programs would not easily get out-dated, particularly for lack of maintainers. Unfortunately, these requisites can be hardly satisfied simultaneously. Some people choose to use those popular languages that are not designated for scientific computations, and have to do lots of unnecessary and tedious work in order to implement relatively simple mathematical formulas, while other people use non-standardized languages, which are included in some popular commercial software packages. The languages in Matlab[1] and IDL[2] are historically satisfactory, but the down-side is that the programs written by these languages would be affected by the economical status of their vendors, which is in turn doomed probably by the market size of these software packages.

Under this circumstance we present our programming language, Mx, as a small trial in design and implementation of programming languages for scientific computations.

1.2 Related works

Matlab by Mathworks is probably the most popular commercial language (or rather to say, software) of this kind nowadays. Another choice is IDL, which is developed by Research Systems Inc., a Kodak company. There are some other related softwares such as Maple[3], Mathcad[4], and Mathematica[5]. Some of them can do (or focus on) symbolic computations.

Invented by Kenneth E. Iverson, who received the Turing Award of 1980 for this reason, APL is a language trying to imitate standard mathematical notations, using hundreds of different "primitives" which are hard to be typed by an ordinary keyboard.

Many open-sourced numerical tools are also available. GNU Octave[6] is a general numerical interactive language, originally developed for solving chemical reactor design problems. Scilab[7] is another free numerical tool similar to Matlab.

There are a lot of libraries written in C++, trying to make use of object-oriented and operator overloading techniques to implement direct matrix/vector operations by arithmetic operators [8]. Some of them go even further: by using Blitz++[9] the C++ compiler reportedly generates much better binary code than a Fortran compiler can. Another project is POOMA[10], which is focused on parallel computing.

For Java programmers working with matrix operations and mathematics, there are two packages publicly available on the Internet. The first is java.vecmath[11], a supportive part of Java 2D graphics package. The second is Jama[12], developed by NIST with the help of Mathworks. The development of Jama has been ceased for years. Both of them have limited functions. As we can see in the later chapters, along with this project, we developed another Java Matrix Class (Jamaica), which provides a lot of functions that is not present in those previously mentioned Java matrix packages.

Java does not support operator overloading, so it is not possible to do matrix/vector operations by directly using arithmetic operators with these matrix classes. However, investigators are trying to add operator overloading into Java as well as some other features, in order to make the language more suitable to high-performance scientific computing. Titanium[13] is such a project led by computer scientists in UC Berkeley.

Besides those basic matrix/vector operations, there are many advanced frequently-used matrix algorithm in linear algebra and matrix theory. Unfortunately there are little choices of implementations: even Matlab and IDL cannot build their own. Matlab currently uses LAPACK[14] and IDL uses algorithms in Numerical Recipe (NR)[15]. Both LAPACK (and its relatives) and NR were originally written in Fortran. LAPACK can be translated to C by f2c[16], and Numerical Recipe in C was probably translated from Fortran by hand. Algorithms in NR is said to be weak in efficiency and precision [17] in part because those algorithms were originally implemented with single-precision float numbers.

1.3 Goal

The main goal of our language is to implement and achieve a simple and ease-of-use programming language for numerical computations. The programmer of our language can do matrix operations directly by arithmetic operators such as '+', '−' and '∗', as well as other frequently-used symbols. Moreover, this language also provided many internal functions for data manipulations, especially plotting and painting functions.

1.3.1 Portability

Our language will run on various platforms. For portability we will use ANTLR[18] as the syntax recognizer, which is running on a Java virtual machine(VM). The current version of our language will be interpreted and executed, by a pure java interpreter. With the help of the portability of Java programming language, Mx can be virtually run on any machine (including PDA's, hopefully).

1.3.2 Efficiency

Efficiency is always one of the most important evaluations of a programming language for scientific computing. There are two aspects of efficiency: the execution efficiency, and the programming efficiency. Because Java itself is not so efficient as we might expect in scientific computation, and also because our programming language is currently interpretive, the execution efficiency is not very applausable. To reduce interpretive inefficiency, the user of our programming language can use matrix operations instead of entry-wise operations.

However, Mx greatly improves the programming efficiency. For many scientific computational needs, especially those in which many matrix and vector operations are involved, the program written by Mx is very short, as we can see in examples given in the next chapter.

1.3.3 Ease-of-use

The language will be simple and quick-to-start. Only basic and necessary language features would be included in the language specification. Meanwhile, most of the needs in numerical computations would be fulfilled.

1.4 Main language features

In this section we describe some of main features in our language.

1.4.1 Data types

Besides some basic data types *integer*, *float* and *string*, we support four additional data types: *matrix*, *function*, *range*, and matrix mask. Matrix mask, or simply *matmask*, is used for selectively setting matrix contents.

1.4.2 Basic matrix operations

Basic matrix and vector operations are supported, e.g., addition, subtraction, multiplication, transpose and inversion. In addition, mathematic functions such as *sin* and *cos* can be applied to every entry of a matrix. Other imaging processing functions, such as image flipping and mirroring, are also provided.

1.4.3 Matrix slicing and masking

In Mx, matrix can be not only operated as a whole or entry-wise, it can be operated through its vectors, sub-matrices or selected elements as well. In the latter case, other non-selected entries of the original matrix are never touched. The operation of getting a vector or sub-matrix of a matrix for in-place operations is called *matrix slicing*, and modifying only the selected matrix entries is called *matrix masking*.

Neither java.vecmath nor Jama supports matrix slicing and masking. Upon the writing of this report, we found another Java matrix package, Jampack[19], is also publicly available. Its underlying data structure sounds supportive to matrix slicing. This package, however, was implemented differently and is in its early stage: "The code has only been lightly tested and certainly has bugs", as the author mentioned in his homepage.

Failed to find a good and suitable Java matrix package for Mx, we developed our own underlying matrix class, Jamaica. As a standalone package, separated from the Mx interpreter programs, and moderately tested, Jamaica will be soon publicly available.

1.4.4 Program flow control

The statements for program flow control were implemented, including *if-then-else, while, break, continue*, and *for*. In addition, programs in our Mx language can define and invoke subroutines, include other files, do recursive calls, and even pass functions as parameters.

1.4.5 Internal functions

The minimum but necessary set of internal functions such as *print, input, load, save, plot, paint*, and mathematical functions are included.

Chapter 2

Tutorial

2.1 Warming up

Before getting start, you should know how do enter and leave the system. To execute the Mx line-by-line interpreter, type:

```
OS% java MxMain
```

or you want to execute your program:

```
OS% java MxMain myprog.mx
```

Once you entered the Mx line-by-line mode, type (without ending ";"):

```
Mx> exit
```

to return to the operating system. Note that `exit` is not permitted in your program, please use `return` to end your program.

Another useful function is `what()`. You can use it to check all or specified variables in the system, or specified variables.

```
Mx> a=2;
Mx> what();
<double>  3.141592653589793
<int>  a = 2
<double>  2.718281828459045
Mx> what(a);
<int>  a = 2
```

We can see π and e are pre-assigned. You can use `PI` and `E` to access them (note that Mx is case-sensitive):

```
Mx> print(PI);
3.141592653589793
```

Suppose you want to save a matrix

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

to a variable, namely A, type:

```
Mx> A = [ 1, 2; 3, 4 ];
```

To print the matrix A, type:

```
Mx> print( A );
```

Then do some computations:

```
Mx> print( A * A );
```

It shows

```
A =
      7.0000 10.0000
     15.0000 22.0000
```

Let us define another matrix, say B:

```
Mx> B = [ 1, 2; 0, 1 ];
```

Then print the product AB and BA (note that they are different):

```
Mx> print( A * B );
      1.0000  4.0000
      3.0000 10.0000
Mx> print( B * A );
      7.0000 10.0000
      3.0000  4.0000
```

We can add two matrices together:

```
Mx> print( A + B );
      2.0000  4.0000
      3.0000  5.0000
```

The result can be saved to another variable:

```
Mx> S = A + B;
```

To invert a matrix, type

```
Mx> print( inv(S) );
     -2.5000  2.0000
      1.5000 -1.0000
```

or

```
Mx> print( 1/S );
     -2.5000  2.0000
      1.5000 -1.0000
```

To check its correctness, just compute the product:

```
Mx> print( S * inv(S) );
      1.0000  0.0000
      0.0000  1.0000
```

The result is an identity matrix, which can be generated by function `eye(n)` (n is the dimension):

```
Mx> print( eye(3) );
      1.0000  0.0000  0.0000
      0.0000  1.0000  0.0000
      0.0000  0.0000  1.0000
```

There are two kinds of division of two matrices, say A and B. The right-division, A/B, means $A * inv(B)$, and the left-division, $A/'B$, means $inv(B) * A$. $A/'B$ is also means to solve equation $BX = A$. For example,

```
Mx> QR = A/B;
Mx> QL = A/'B;
Mx> print( QR );
QR =
      1.0000 -0.0000
      3.0000 -2.0000
Mx> print( QR * B );
      1.0000  2.0000
      3.0000  4.0000
Mx> print( QL );
QL =
     -5.0000 -6.0000
      3.0000  4.0000
Mx> print( B * QL );
      1.0000  2.0000
      3.0000  4.0000
```

We can see that the results agree. You can also compute the determinant of a matrix:

```
Mx> print( det(A) );
-2.0
```

2.2 Matrix operations

If you want to generate a large matrix, using ways like `[1,2;3,4]` is too cumbersome. Function `zeros(m,n)` generates an $m \times n$ matrix with all entries zero:

```
Mx> print( zeros(2,3) );
      0.0000  0.0000  0.0000
      0.0000  0.0000  0.0000
```

Of course, you can use function `eye(n)` mentioned above. Sometimes you may want to generate a matrix with random numbers:

```
Mx> print( random(2,3) );
      0.2151  0.5153  0.0138
      0.0875  0.1132  0.8355
```

All random numbers are uniformly distributed between 0 and 1 (inclusive). Another very useful matrix generator is `indgen(m,n,s,dsm,dsn)`, which means *index generating*. This function generates an $m \times n$ matrix with linear increment, whose top-left entry $(0,0)$ is s, and $(1,0)$ is $s + dsm$, and $(2,0)$ is $s + dsm * 2$, and so forth. Similarly, $(0,1)$ is $s + dsn$, and $(0,2)$ is $s + dsn * 2$, and so forth. For example:

```
Mx> print( indgen(3,4,1,1,10) );
      1.0000 11.0000 21.0000 31.0000
      2.0000 12.0000 22.0000 32.0000
      3.0000 13.0000 23.0000 33.0000
```

You can use `indgen` to generate a matrix that contains a same value other than zero:

```
Mx> print( indgen(2,3,5.7,0,0) );
      5.7000  5.7000  5.7000
      5.7000  5.7000  5.7000
```

Next important thing is to get an individual entry from a matrix. In Mx, the first row is indexed as 0, and the second row is 1, and so do columns. To get the second row and the fourth element from a matrix, type:

```
Mx> C = [1,2,3,4,5;6,7,8,9,10;11,12,13,14,15];
Mx> print( C );
C =
      1.0000  2.0000  3.0000  4.0000  5.0000
      6.0000  7.0000  8.0000  9.0000 10.0000
     11.0000 12.0000 13.0000 14.0000 15.0000
Mx> print( C[1,3] );
      9.0000
```

If you want to get matrix entries with some consecutive rows and consecutive columns, you can use *range*, two integers separated by :,

```
Mx> print( C[0:1,2:3]);
      3.0000  4.0000
      8.0000  9.0000
```

You can get a single row or column by

```
Mx> print( C[:,3] );
      4.0000
      9.0000
     14.0000
Mx> print( C[2,:] );
     11.0000 12.0000 13.0000 14.0000 15.0000
```

These operation are called *matrix slicing*. Also, it is not necessary to use only rectangular region: one can select matrix entries by the results of comparing values of this matrix or other matrices with the same size. This operation is called *matrix masking*. You can save to a variable of *matmask* type the information about which elements are selected. Figure 2.1 illustrates the operations of slicing and masking.

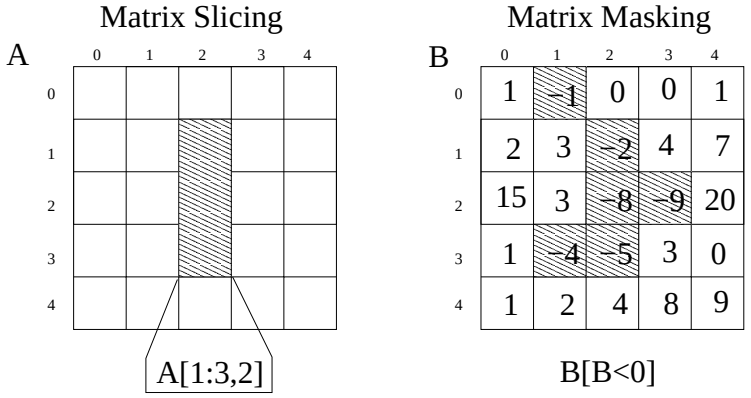

Figure 2.1: matrix slicing and masking

We can set a matmask variable indicating those entries that are greater than 2.5 and less than 7.5:

```
Mx> mask = C>2.5 and C<7.5;
```

and set these entries to zeros:

```
Mx> C[mask] = 0;
Mx> print( C );
C =
      1.0000  2.0000  0.0000  0.0000  0.0000
      0.0000  0.0000  8.0000  9.0000 10.0000
     11.0000 12.0000 13.0000 14.0000 15.0000
```

2.3 Display your results

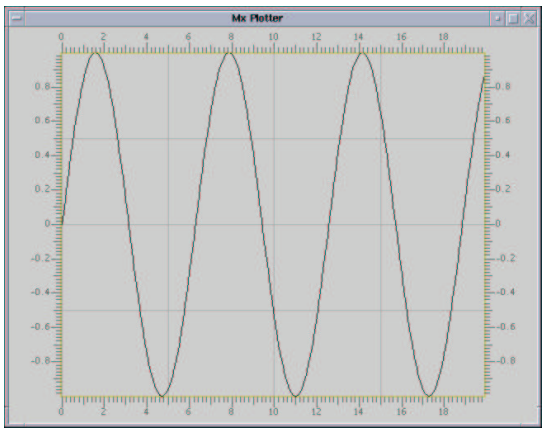

Figure 2.2: plot of $\sin(x)$

Let us begin to draw some graphs. First example is to draw a sinusoidal curve:

```
Mx> A = indgen(200,2,0,0.2,0);
Mx> A[:,1] = sin(A[:,0]);
Mx> plot(A);
```

The `plot(A);` will open a new window and draw the curve as in Figure 2.2. A more complicated example is to draw a 3-dimensional Lissajous trajectory:

```
Mx> B = zeros(200,3);
Mx> B[:,0] = sin(A[:,0]);
Mx> B[:,1] = sin(A[:,0]*1.2);
Mx> B[:,2] = sin(A[:,0]*1.4);
Mx> colormap(2);
Mx> plot(B);
```

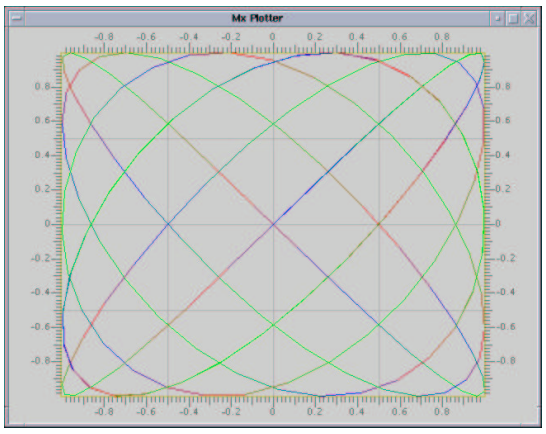

Figure 2.3: 3-d Lissajous trajectory. The z-coordinate is represented by changing colors.

The plot window is shown in Figure 2.3. The z-coordinate is represented by changing colors. Finally, let us plot something

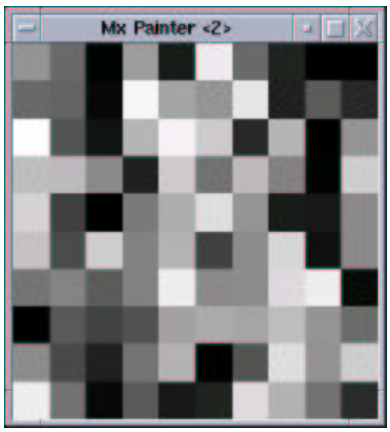

Figure 2.4: painting a random matrix

```
Mx> C = random(10,10);
Mx> colormap(1);
Mx> paint(C);
```

A 10 by 10 image is very small, but you can resize it as in Figure 2.4.

2.4 Using program files

When things cannot be done in several lines, you can edit a program file. Use any kind of text editor to create a program file `myprog.mx` contains (don't type in line numbers):

```
1:  func grid( x, y ) {
2:      c = (x/25+y/25)%2;
3:      return c;
4:  }
5:
6:  func distance( a, b ) = sqrt(
7:      (a-100)*(a-100)+(b-100)*(b-100));
8:
9:  A = zeros(200,200);
10:
11: for ( i = 0:199, j=0:199 ) {
12:     A[i,j] = grid( i, j );
13:     if ( distance(i,j) >= 60 )
14:         A[i,j] = 1-A[i,j];
15: }
16:
17: colormap(1);
18: paint(A);
```

This program draws a circle in a chess board. (Figure 2.5)

Lines 1-4 define a function that determines if point (x, y) is in a black or white square. You can define functions at the beginning of the program. A function definition always starts with the keyword `func`. (Several keywords are used in this language, explained in the next chapter, Language Reference Manual.) Function identifier follows the func keyword. Identifier can consist of one or more alphanumeric characters. Following the identifier is the variable or argument list enclosed by parenthesis. It can consist of zero or more

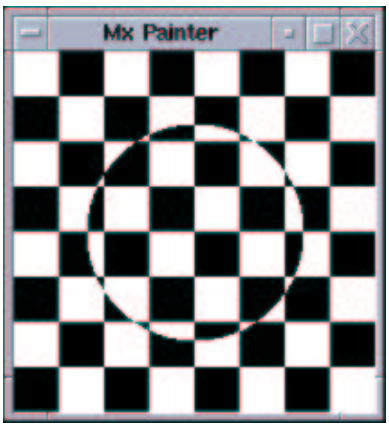

Figure 2.5: circle in a chess board

identifiers. The function body can contain an expression led by an assignment (=), or a group of statements enclosed in { }.

Lines 6-7 define a function that calculate the distance between (a, b) and the center of the board, $(100, 100)$.

Line 9 initializes a 100 by 100 matrix. Then in line 11-15, we use a for loop to do the same operations for all points. This is similar to the *for/do* loop in many other programming languages. The difference is that Mx have a range type to specify the range of iteration.

Line 12 is an example of calling a function. The return value of a function can be assigned to a variable, or a matrix entry.

Line 13 uses the *if* conditional statement to check whether a point (a, b) is inside the circle. There are two types of conditional statements, *if* and *if-else*. The uses of these are exactly the same as C and Java. Keyword `if` is followed by a logical or relational expression. The relational expression compares two numbers by operators $<$, $>$, $==$, $!=$, $<=$, or $>=$. A logical expression can be a boolean expression of two or more relational expressions, using keywords `and`, `or`, and `not` as operators.

The last two lines select the black-white colormap and draw the matrix as an image. The picture is shown in Figure 2.5

2.5 Example: Lorenz equations

We now begin to show some complicated examples. The first example is the Lorenz equations. The Lorenz equations were discovered in 1963 by Edward Lorenz, a meteorologist at MIT, when he and other mathematicians did research in the modeling of convection that takes place in the atmosphere. They presented a model with three equations, which is now the famous Lorenz equations:

$$\begin{aligned} \frac{dy_0}{dt} &= \alpha(y_1 - y_0) \\ \frac{dy_1}{dt} &= y_0(r - y_2) - y_1 \\ \frac{dy_2}{dt} &= y_0 y_1 - b y_2 \end{aligned}$$

Given an initial condition, they found the system state (three variables) will not go to infinity, and it is not convergent, either. A small fluctuation will cause large difference in the future, which might be the reason that the weather is not predictable in a long term. This phenomenon is called chaos.

To simulate such an system, we need a good algorithm to solve differential equations numerically. The most popular, good and simple algorithm is the fourth-order Runge-Kutta algorithm. Given the parameters

and initial conditions that $\alpha = 10$, $r = 28$, $b = 8/3$, and $x(0) = 10$, $y(0) = 0$, $z(0) = 10$, now we code both Runge-Kutta algorithm and Lorenz equations in Mx:

```
/* Simulation of lorenz equation */

a = 10;
b = 8/3.0;
r = 28;

func Lorenz ( y, t ) = [ a*(y[1]-y[0]);
                         -y[0]*y[2] + r*y[0] - y[1];
                         y[0]*y[1] - b*y[2] ];

func RungeKutta( f, y, t, h )
{
    k1 = h * f( y, t );
    k2 = h * f( y+0.5*k1, t+0.5*h );
    k3 = h * f( y+0.5*k2, t+0.5*h );
    k4 = h * f( y+k3, t+h );
    return y + (k1+k4)/6.0 + (k2+k3)/3.0;
}

N = 20000;
p = zeros(N+1,3);
t = 0.0;
h = 0.001;
x = [ 10; 0; 10 ];
p[0,:] = x';

for ( i=1:N )
{
    x = RungeKutta( Lorenz, x, t, h );
    p[i,:] = x';
    t += h;
}

colormap(3);
plot(p);
return 0;
```

The important features in Mx used in this program include vector operations, function variables, and plotting. Figure 2.6 shows what the program generates.

2.6 Example: fractals

Let us draw some pictures of fractals. This example draws the Sierpinski triangle, which is generated by removing the lower-right quarter from the square and recurring on rest quarters with the same action:

```
/* A program that plots the
 * Sierpinski Triangle */

func sier(A)
{
    n = width(A);
```

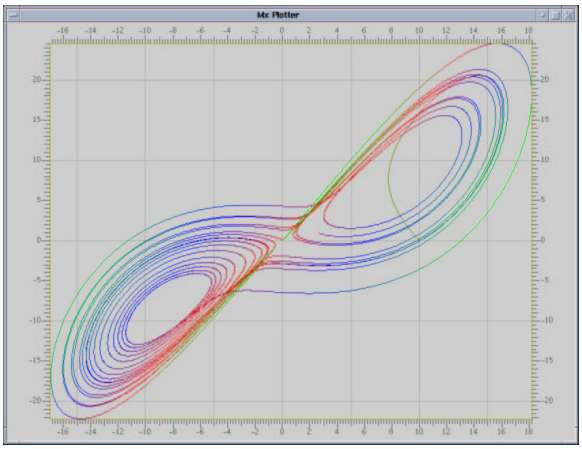

Figure 2.6: plot of a solution of Lorenz equations. Parameters are $\alpha = 10$, $r = 28$, $b = 8/3$, and the initial condition is $(10, 0, 10)$. The z coordinates are represented in color.

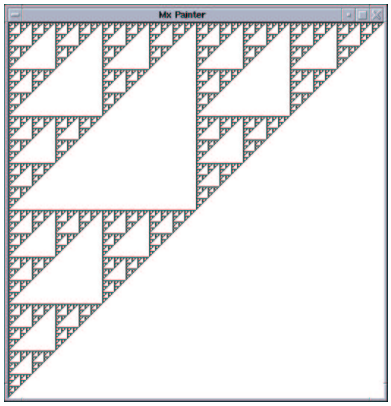

Figure 2.7: the Sierpinski triangle

```
    if ( n <= 1 )
    {
        A[0,0] = -1.0;
        return;
    }

    sier(A[0::n/2,0::n/2]);
    sier(A[0::n/2,n/2:]);
    sier(A[n/2:,0::n/2]);
}

A=zeros(512,512);
sier(A);
paint(A);
return 0;
```

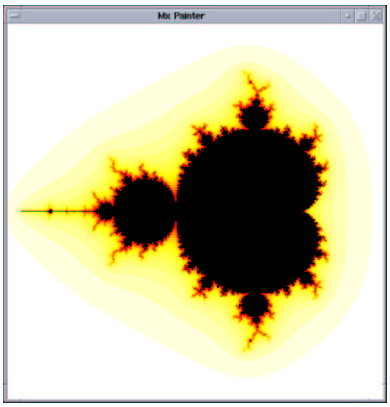

Figure 2.8: the Mandelbrot set. The range of x-axis is $[-2.1, 0.9]$ and the range of y-axis is $[-1.5, 1.5]$.

The important features used in the program include matrix slicing, recursion and painting. Figure 2.7 shows the result.

The second example is to draw another important and famous fractal image: the Mandelbrot set. The value of each point is generated by the speed of divergence of Z by this recursive formula

$$Z^{(i+1)} = (Z^{(i)})^2 + Z^{(0)},$$

where $Z^{(0)}$ is the coordinates of the point, represented by a complex number.

The important features of Mx programming language used here are internal functions, matrix masking, and painting over an existing image. Figure 2.8 shows what the program draws.

```
/* generate Mandelbrot sets */

MAX = 1.5;
border = 100.0;
N = 512;
delta = MAX*2.0/N;

CX = indgen(N,N,-MAX-0.6,0,delta);
CY = indgen(N,N,-MAX,delta,0);
X = CX;
Y = CY;
R = zeros( N, N );

mask = R < -1;
colormap(4);
paint( R, -100, 0 );

for ( i=0:40 )
{
    T = mul(X,X) - mul(Y,Y);
    Y = 2*mul(X,Y)+CY;
    X = T+CX;
    msk = mul(X,X) + mul(Y,Y)
          >= border*border;
    R[msk and not mask] = -sqrt(i+1);
    mask = mask or msk;
```

```
    opaint( R, mask, 0, 0, -5, -2 );
}

colormap();
color(0,0,0);
opaint( R, not mask );

return 0;
```

2.7 Example: image processing

These examples are going to show that Mx programming language can be used to do some basic image processing.

The first example (Figure 2.9) shows the eight orientations of Mr. Potato Head (geometric transformation):

```
/* simple geometric transformation */

A = load( "potato.dat", "byte", 128,128 );
colormap(1);
paint( [ A, flip(A), mirror(A), flip(mirror(A));
     A', flip(A'), mirror(A'), flip(mirror(A'))] );
return 0;
```

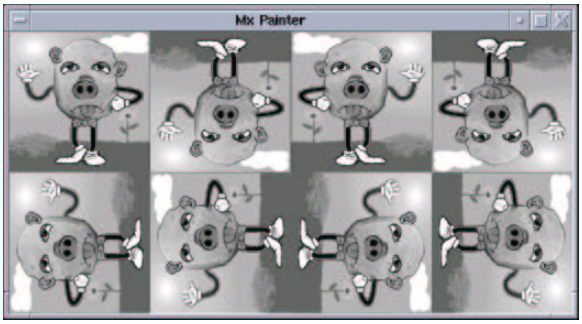

Figure 2.9: global geometric image transformation

The second example (Figure 2.10) shows effects of various local geometric operations:

```
/* An image processing example */

blur   = [ 0,0,0,1,1,1,0,0,0;
           0,0,1,2,3,2,1,0,0;
           0,1,2,3,4,3,2,1,0;
           1,2,3,4,4,4,3,2,1;
           1,3,4,4,5,4,4,3,1;
           1,2,3,4,4,4,3,2,1;
```

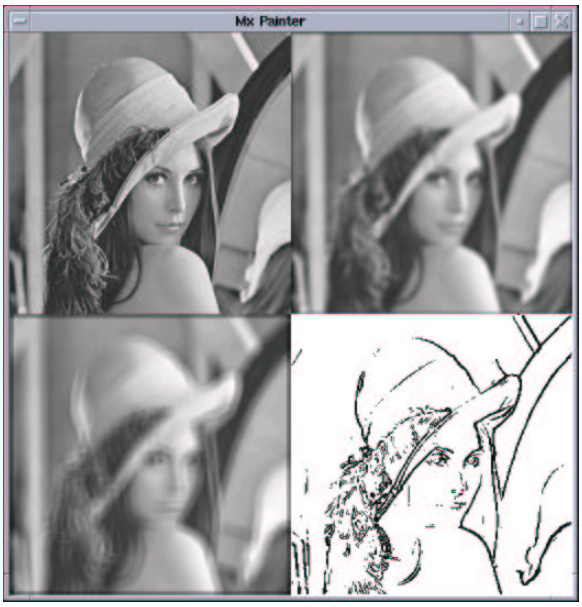

Figure 2.10: local geometric operations

```
            0,1,2,3,4,3,2,1,0;
            0,0,1,2,3,2,1,0,0;
            0,0,0,1,1,1,0,0,0 ];

drunk  = [ 1,1,1,1,1,1,1,1,1;
            1,0,0,0,0,0,0,0,1;
            1,0,0,0,0,0,0,0,1;
            1,0,0,0,0,0,0,0,1;
            1,0,0,0,0,0,0,0,1;
            1,0,0,0,0,0,0,0,1;
            1,0,0,0,0,0,0,0,1;
            1,0,0,0,0,0,0,0,1;
            1,1,1,1,1,1,1,1,1 ];

A = load( "lenna.dat", "byte", 256, 0 );

m1 = height(blur);
n1 = width(blur);
B = zeros( height(A)+m1, width(A)+n1);
for ( i=0::height(A), j=0::width(A) )
    B[i::m1,j::n1] += blur * A[i,j];

m2 = height(drunk);
n2 = width(drunk);
C = zeros( height(A)+m2, width(A)+n2);
for ( i=0::height(A), j=0::width(A) )
    C[i::m2,j::n2] += drunk * A[i,j];

/* Fast Sobel algorithm */
D = A;
```

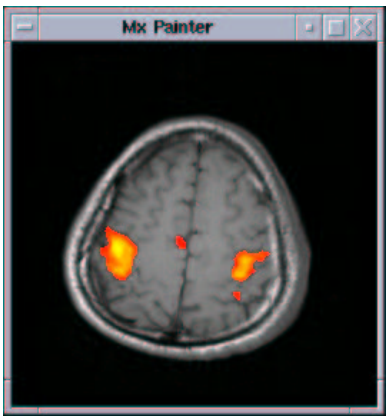

Figure 2.11: Brain activation map of finger tapping, overlaid on a anatomical image. Data was sampled by functional MRI (fMRI) techniques and pre-processed with cross-correlation (CC) algorithm (pre-processed elsewhere).

```
for ( i=1::height(A)-2, j=1::width(A)-2 )
    D[i,j]=abs(A[i-1,j+1]+2*A[i,j+1]+A[i+1,j+1]
              -A[i-1,j-1]-2*A[i,j-1]-A[i+1,j-1])
          +abs(A[i-1,j-1]+2*A[i-1,j]+A[i-1,j+1]
              -A[i+1,j-1]-2*A[i+1,j]-A[i+1,j+1]);

D = D/max(D);
mask = D > 0.2;
D[mask] = 0;
D[not mask] = 1;

colormap(1);
paint( [A'/max(A), B[m1/2::256,n1/2::256]'/max(B);
        C[m2/2::256,n2/2::256]'/max(C), D'],
     0,1);
return 0;
```

The third example (Figure 2.11) shows the brain activation of one of our authors: his corresponding brain regions of sequential finger tapping movement. The data was acquired on a magnetic resonance imaging (MRI) scanner.

```
/* show activation regions of a
 * human brain from the fMRI data */

A = load( "mri-anat.sdt", "short", 256, 256 );
B = load( "mri-func.fdt", "float", 256, 256 );

A = flip(A');
B = flip(B');
paint( A );
colormap(4);
opaint( B, B>0.7, 0, 0, 0.7, 1.0 );
return 0;
```

Chapter 3

Language Reference Manual

3.1 Lexical conventions

3.1.1 Comments

The characters "/∗" start a comment terminated by "∗/", while the characters "//" start a single-line comment.

3.1.2 Identifiers

An identifier consists of letters, digits and underscore "_". The first character of an identifier should be a letter or underscore. Upper and lower case letters are considered different.

3.1.3 Keywords

These identifiers are reserved as keywords:

```
for     if       else    loop
break   continue return  exit
include func     and     or
not     true     false
```

3.1.4 Numbers

A number consists of digits, optional decimal point "." and optional "e" followed by signed integer exponent. Integers and floating numbers will be distinguished.

3.1.5 Strings

A string is a sequence of characters enclosed by double quotes "″". A double quote inside the string is represented by two consecutive double quotes.

3.1.6 Other tokens

Some symbolic characters or sequences of symbolic characters are used in the language:

```
{   }   (   )   [   ]   ,   ;
+   -   *   /   %   /'
=   +=  -=  *=  /=  /'=
>=  <=  >   <   ==  !=
'   :   ::
```

3.2 Types

In this language we do not have any explicit type specifications. As a result, the language is not statically-typed, though this language is possibly compiled to Java source code. Data types are distinguished at run time. Each variable is bounded by a type tag, which could be checked at run time.

Implemented run-time types:

- `bool` boolean values being either **true** or **false**
- `int` : 32-bit integers
- `double` : 64-bit IEEE floating format
- `matrix` : M-by-n matrix containing entries of the double type
- `string` : string of characters
- `function` : user-defined functions
- `matmask` : indicating what elements are selected in a matrix
- `range` : Only for temporary values (cannot be assigned to a variable), used as matrix indexes

3.3 Expression

3.3.1 Primary expressions

Primary expressions include identifiers, constants, function calls, access to matrix/vectors elements or sub-matrix/sub-vectors, transpose of a matrix, anonymous matrices or vectors, and any other expressions surrounded by "(" and ")".

3.3.2 Identifier

An identifier itself is a left-value expression. It will be evaluated to some values bounded to this identifier.

3.3.3 Constant

A constant is a right-value expression, which will be evaluated to the constant itself.

Function call

A function call consists of a function identifier, followed by a list of arguments enclosed by "(" and ")". The list of arguments contains zero or more arguments separated by ",". Each argument is an expression.

Function call is a right-value expression.

Access to array elements or sub-arrays

This primary expression consists of an array(matrix/vector) identifier or other left-value expressions, followed by a list of indexes enclosed by "[" and "]". The list of indexes contains one or two range specifiers.

This expression is itself left-value.

Matrix transpose

The right-value transpose expression consists of an array identifier or other left-value expressions, following by transpose operator "′".

Anonymous arrays

An anonymous array is a way to build up a matrix or vector via individual elements. Each element of an array could be an expression, separated by "," or ";", where ";" separates columns of an array. The whole anonymous array is surrounded by "[" and "]".

(expression)

A parenthesized expression is a primary expression, which returns the value of enclosed expression. The presence of parentheses is to boost the precedence.

3.3.4 Arithmetic expressions

Arithmetic expressions take primary expressions as operands.

Unary arithmetic operators

Unary operators "+" and "−" can be prefixed to a expression. "+" operator returns the expression itself whereas the "−" operator returns the negative of the primary expression.

They are applicable to int, double, and matrix values.

Multiplicative operators

Binary operators "∗", "/", "%" and "/′" indicate multiplication, division, modulo and prefixed division, [1] respectively. They are grouped left to right.

They are applicable to int, double, and matrix values.

Additive operators

Binary operators "+", "−" indicate addition and subtraction, respectively. They are grouped left to right.

They are applicable to int, double, and matrix values. Operator "+" is also applicable to string values.

3.3.5 Relational expression

Binary relational operators ">=", "<=", "==", "! =", ">" and "<" indicate whether the first operand is greater than or equal to, less than or equal to, equal to, not equal to, greater than, or less than the second operand, respectively. A bool value is returned in case that both operands are numbers (int and double), and exactly two operands are needed.

These operators are also applicable to matrix. If two matrices are compared, a matmask value is returned. It can be assigned to a variable, or directly used as the index of a matrix.

3.3.6 Logical expression

Logical operators take relational expressions as operands. Among these operators, operator **not** has the highest precedence, then operator **and**, and operator **or** is the lowest.

These operators are applicable to either bool values or matmask values.

Not operator

A logical expression consisting of a **not** operator followed by a relational expression returns the logical negation of this relational expression.

[1] Modulo is effective for both integers and floating numbers, but not for matrices. For integers and floating numbers, division and prefixed division do exactly the same operation, while for matrices, A/B means AB^{-1} and $A/'B$ means $B^{-1}A$.

And operator

Operator **and** indicates the logical and of two relational expressions. It is short-circuited for bool values, but not for matmask values.

Or operator

Operator **or** indicates the logical or of two relational expressions. It is short-circuited for bool values, but not for matmask values.

3.4 Matrix indexes

Matrix/vector indexes can have only one component, or two components separated by comma ",". If two components are supplied, each index component indicates the index on the corresponding dimension. Both index components can be a single arithmetic expression with an integer-compatible value, or two arithmetic expressions separated by ":" or "::" to form a *range*. If only one component is supplied, this component can be integer-valued expression, range, or matmask.

If two arithmetic expressions of a range are separated by ":", the first expression indicates the lower bound of the range, and the second expression indicates the upper bound of the range. The upper bound is the largest possible index if it is omitted. If two bounds are both omitted, a single ":" means all elements on this dimension are included.

If two arithmetic expressions of a range are separated by "::", the first expression (non-negative value) indicates one bound of the range, and the second expression (positive value) indicates how many elements are included. None of two expressions can be omitted in this case.

If range or matmask is used, the language is basically operating on the selected elements of the original matrix. The expression $A[expr]$ and $A[range, range]$ are left-values, and the original elements of matrix A will be operated.

A 1-by-1 sub-matrix can be handled as a double.

3.5 Statements

Statements are basically elements of a program. A sequence of statements will be executed sequentially, unless the flow-control statements indicate otherwise.

Note that in the following specifications, elements enclosed in < and > will be replaced by some corresponding component.

3.5.1 statements in "{" and "}"

A group of zero or more statements can be surrounded by "{" and "}", in which case they altogether are treated as a single statement.

3.5.2 Assignments

An assignment is in this form:

```
<left-value expr> = <right-value expr>;
```

where ";" is the terminator of this assignment statement.

3.5.3 Conditional statements

A conditional statement is in this form:

```
if ( <logical expression> )
    <statement>
```

or

```
if ( <logical expression> )
    <statement>
else
    <statement>
```

If the logical expression returns true, the first statement is executed, otherwise the optional second statement is executed.

3.5.4 Iterative statements

Iterative statements are basically loops. There are two kinds of loops, **for** and **loop**.

For statement

The **for** statement is usually used for indexes of array elements. It has this form: (note that id means identifier)

```
for ( <id> = <range> )
    <statement>
```

or

```
for ( <id1> = <range>, <id2> = <range>  )
    <statement>
```

More `<id> = <range>` can be accepted. The `range` field is the same as matrix indexes. `<id1>` also acts as the label of the for statement.

Loop statement

The **Loop** statement is the general iterative statement. It is in this form:

```
loop <statement>
```

or

```
loop (<label>) <statement>
```

An infinite loop will be introduced if none of break and return is used in the loop statement. The `label` is an identifier that can be used by break or continue statements.

Break statement

The **break** statement will break the inner-most or labeled iterative statement. This statement consists of keyword **break**, optionally followed by a label, then followed by a ";".

Continue statement

The **continue** statement will end the current iteration of the inner-most or labeled iterative statement and proceed to its next iteration. This statement consists of keyword **continue**, optionally followed by a label, then followed by a ";".

3.5.5 Function invocation and return

Function call

Different from other expressions, a function call followed by a ";" can be a single statement.

Return statement

The return statement is used inside the function definition body, in order to return from the function at that point. It can be followed by an optional expression, for the return value, and a ";".

3.5.6 Inclusion of other programs

Keyword **include** is used to include other program files. The format of this statement is:

```
include <string of the path> ;
```

3.6 Function definition

A function definition is in this form:

```
func <id> ( <var-list> ) = <expression> ;
```

or

```
func <id> ( <var-list> ) { <body> }
```

Field `id` indicates the name of this function. Field `var-list` is a list of identifier of (formal) arguments, separated by commas ",". The list of arguments can be empty. Field body is zero, one or multiple statements. In the first form, the function returns the evaluated value of field expression, whereas the in the second form, the return statement should be used in the body in order to return a value.

3.7 Internal functions

3.7.1 Console output functions

Print

Function print() takes one or more arguments and print them to the standard output one by one. This function returns the first argument.

What

Function what() prints the information of variables, for debugging purposes. It takes zero or more arguments. If it takes no argument, it will print all variables in the current scope.

3.7.2 Picture drawing functions

Paint

Function paint() draws a matrix in a new window as an image. It takes one or three arguments. The first argument should be a matrix, and the rest optional arguments indicate the minimum and maximum values of this matrix, which can be different from the real minimum and maximum values for display. This function returns the first argument: the matrix to be drawn.

Opaint

Function opaint() draws a matrix in the current window, over the existing image. The format of opaint() is one of

```
opaint( matrix )
opaint( matrix, mask )
opaint( matrix, x, y )
opaint( matrix, mask, x, y )
opaint( matrix, x, y, min, max )
opaint( matrix, mask, x, y, min, max)
```

where argument `mask` is a matmask of the matrix to be drawn. Arguments `x` and `y` are the upper-left coordinates to draw the matrix (default to be (0,0)). Arguments `min` and `max` are similar to those in paint().

Plot

Function plot() takes one or two arguments. The first argument should be a matrix with 2 or 3 columns, indicating x-, y-, and optionally z-coordinates of a data point. Each row of the matrix is a data point to be drawn. The second argument is optional, and have two rows and same number of columns as the first argument, indicating the minimum (the first row) and maximum (the second row) of the first argument.

3.7.3 Color

Function color() sets the current color. It needs three double arguments for red, green and blue values, in this order. These values should be no less than 0 and no more than 1.

3.7.4 Colormap

Function colormap() sets a colormap for painting images and plotting with changing colors. It takes an argument of matrix type. This matrix should have three rows: the first rows for blue values, the second for green values and the third for red values. None of the entries can be less than 0 or more than 1. Alternatively, the internal color maps can be used, in which case the user use an integer as the only argument.

argument	color map
0	black/white
1	grayscaled
2	all colors (red→orange→yellow→green→blue→violet)
3	dark colors (same as all colors but darker)
4	hot colors (red→yellow→white)
5	warm colors (green→blue)
6	cold colors (blue→violet)

3.7.5 File I/O functions

Load

Function load() loads binary data from a data file. It takes four or five arguments, in one of the following formats:

```
load( file, type, m, n )
load( file, type, m, n, skip )
```

where argument `file` is a string indicating the file name, argument `type` is a string indicating the data type in the file, and `m` and `n` are two dimensions. Optional argument `skip` tells the function how many bytes should be skipped before reading the data from the file.

Either `m` or `n`, but not both, can be zero, in which case load() will compute this dimension according to the file size.

The argument `type` should have one of the following values: "byte", "short", "int", "float", or "double".

Save

Function save() saves a binary data to a data file. It takes three or four arguments, and has one of the following formats:

```
save( matrix, file, type )
save( matrix, file, type, skip )
```

where *matrix* is the matrix to be saved, and other arguments are described in load().

3.7.6 Matrix dimensions

Functions width() and height() each take one matrix argument and return the sizes of the horizontal and vertical dimensions, respectively.

3.7.7 Matrix generators

Let `m` and `n` be two dimensions, following functions will generate a new matrix:

```
zeros( m, n )
random( m, n )
eye( n )
indgen( m, n, s, dsm, dsn )
```

Function zeros() returns a new matrix containing only zeros. Function random() returns a new matrix of uniformly distributed random values within range [0,1]. Function eye() returns the identity matrix, i.e., a square matrix whose diagonal entries are ones and other entries are zeros. The last function, indgen(), generates a matrix, in which the difference between two vertically adjacent entries are all equal to `dsm` and the difference between two horizontally adjacent entries are all equal to `dsn`. The first entry of the new matrix is `s`. If `dsm` and `dsn` are both zero, indgen() will generate a matrix having all entries equal to `s`.

3.7.8 Matrix operations

```
inv( matrix )
det( matrix )
flip( matrix )
mirror( matrix )
```

Function inv() inverts a matrix (just like 1/`matrix` does). Function det() returns the determinant of a matrix (as double). Function flip() returns a vertically flipped matrix. Function mirror() returns a horizontally mirrored matrix.

```
mul( A, B )
div( A, B )
```

Functions mul() and div() compute the product and the quotient, respectively, of corresponding entries in two matrix A and B having the same size. They both return a matrix with the same size.

3.7.9 Order statistics

Function min() returns the minimum value of a matrix. Function max() returns the maximum value of a matrix. They both take a matrix as the argument.

3.7.10 Mathematic functions

The following mathematic functions are supported:

round(x)	round to the nearest integer
ceil(x)	smallest integer greater than
floor(x)	largest integer less than
abs(x)	absolute value
exp(x)	e^x
log(x)	$\ln x$
sin(x)	$\sin x$
cos(x)	$\cos x$
tan(x)	$\tan x$
asin(x)	$\arcsin x$
acos(x)	$\arccos x$
atan(x)	$\arctan x$
pow(x, y)	x^y

Note that x can be either a double or a matrix. In the latter case, this function is called for all entries of this matrix, and a result matrix with the same dimensions is returned. Argument y cannot be a matrix.

3.7.11 Other functions

Count

Function count() counts the number of entries selected by a matmask. It accepts a matmask as the only argument.

Chapter 4

Project Plan

4.1 Team Responsibilities

For the past two months, our team has held regular meetings for an hour to discuss and update our goals. Besides the meetings, however, we have set a goal for each member of the group to achieve some type of accomplishment by setting goals. Here are the fundamental tasks for each member:

TianTian Zhou	Team leader; parser and front-end
Hanhua Feng	Architecture and back-end
Yong Man Ra	Testing and documentation
Chang Woo Lee	Testing and documentation

4.2 Programming style (coding conventions)

4.2.1 Antlr coding style

If an Antlr rule is short and contains only one choice without any action, then it will be written in one line. The colon ":" always starts at the ninth column unless the string in front of ":" is too long. In the one line mode, ";" is at the end of this line.

If an Antlr rule has multiple choice or actions, the ";" is placed in a separated line at the ninth column. "|" is also at ninth column. Short actions are in the same line of its rule and at the right half of the screen. Long action lines start at the thirteenth column of the next line. Code in actions follows the Java coding style.

Lexem (token) names are in upper cases and syntactic items (non-terminals) are in lower cases, which may contain the underscore "_".

4.2.2 Java coding style

As the programmers using multiple programming languages, we use a coding style that is a little different to the standard Java coding style.

Indentation and spacing:

- Indentation of each level is 4 spaces.
- The left brace "{" occupies a full line and is at the same column as the first character of the previous text.
- The right brace "}" occupies a full line and at the same column of its corresponding "{".
- One space between arguments and their parentheses "(" ")", but no space between function name and "(".

- Add spaces between operators and operands for outer expressions.
- Use spaces, do not use tabs.
- The then-part and else-part of an "if" statement must not be at the same line of "if" or "else". Similar for "for" statement.

Names:

- Class names start with "Mx" followed by English words whose first character is capital.
- Variables are in lower cases, and words are separated by underscore "_".
- Do not use long names for local variables. More concise, the better.
- Method names are English words whose first characters are capital except for the first word.

Follow Java javadoc standard for comments.

4.3 Project Timeline

Here are the deadlines for each specific task that was due.

Feb 18	Language white paper
First week of March	Language grammar
March 27	Language reference manual
Second week of April	Supporting libraries
Third week of April	Front-end
Last week of April	Back-end
First week of May	Error recovery
May 13th	Entire project

4.4 Software project environment

Most of the programs are written in Java. The lexer and parser are written in Antlr, and will be translated to java code. We also wrote a class in Java with macros, which will be explained later.

4.4.1 Operating systems

Since this project is developed in pure Java, it can be used anywhere that a Java VM is running. However, we have used other tools, such as TCL and GNU m4. TCL is available on many platforms including Unix systems, Windows systems, and MAC. GNU m4 is a Unix application. However, GNU m4 is not necessary to be installed everywhere, unless the template java code must be regenerated.

Our team members use Windows and Linux simultaneously. Also thanks to the Cygwin project, we can easily run TCL, SSH and GNU m4 on a Windows box.

4.4.2 Java 1.4

Java is a "simple, object-oriented, network-savvy, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, dynamic language".

Unfortunately, Java is not designed for scientific computing. Writing our Mx interpreter in Java does not prove that Java is feasible for scientific computing. It just proves that scientific computing *can* be done in Java. If someone hates programming languages other than Java, please try our Java matrix class, Jamaica.

4.4.3 Antlr

The language parser is written in Antlr, "a language tool that provides a framework for constructing recognizers, compilers, and translators from grammatical descriptions containing C++ or Java actions."

4.4.4 CVS

CVS is a well-known concurrent version system for developers. We kept our CVS repository on a Linux machine, and use SSH to check in/out our programs with secured network connections.

4.4.5 TCL

TCL is a shell-like scripting language, and is more efficient and elegant than shell programs. The testing executor program is written in TCL. Please refer to the testing chapter for more details.

4.4.6 GNU m4

Inspecting our code, one can find many pieces of similar code. In our opinion, this may happen in many interpretive programming language applications, as long as there are many types. Java does not support macros, template and even function parameters (One can use derived classes and late-binding to implement these, but it is not a good solutions, because if so, instead of many similar code segments, he will get many similar classes, saving in many similar program files!). Getting tired of typing similar code, we resort to using an external macro-replacing tool, GNU m4, which has a moderate user population. It is used only for internal functions, because there are too many similar functions: sin, cos, ...

4.5 Project log

Here are the dates of significant project milestones. Most of them come from the CVS logs.

Feb 10	CVS repository initialized
Feb 10	The first version of white paper
Feb 15	Finalized the white paper
Feb 25	The first version of syntax
March 6	Tiantian made significant modifications on syntax
March 21	The language reference manual is added to CVS
March 25	Front-end started
April 1	The first version of grammar
April 1	The back-end started
April 9	The first version of Jamaica
April 17	The first version of the back-end
April 25	Early testing stage
May 2	Painter and plotter ready
May 2	Test executor program ready
May 6	Started final report
May 7	The third testing stage: examples
May 9	Most tests are done
May 10	The first version of final report

Chapter 5

Architecture Design

The Mx interpreter consists of these components: a lexer that reads the user input or program files to tokens, a parser that analyzes the syntactic structure of the program and converts it to an abstract syntax tree, a tree walker that travels the abstract syntax tree and calls corresponding functions, an executor that looks up symbol tables and diverts operations to type systems, a supporting matrix library, and a simple error and exception processing mechanism. Figure 5.1 shows its block diagram.

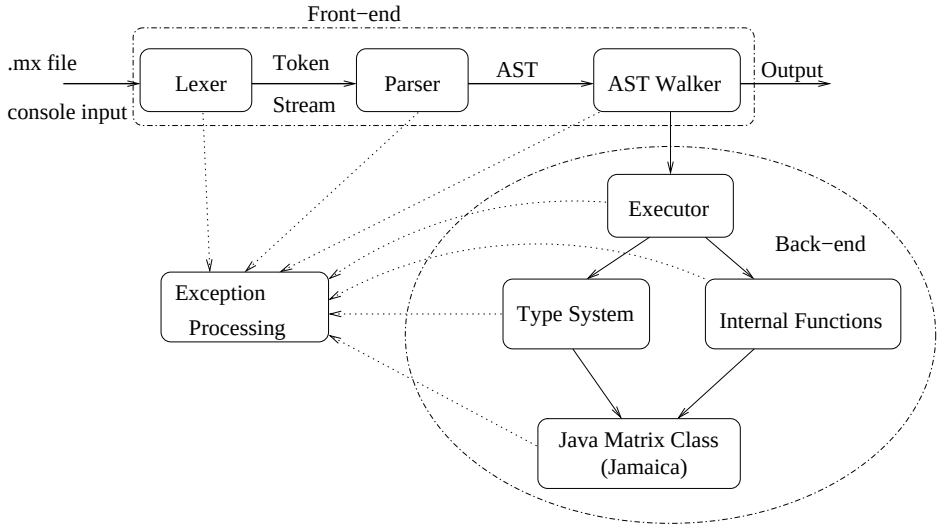

Figure 5.1: the system diagram of the Mx interpreter

The lexer, parser and tree walker are implemented by Antlr[18]. The lexer and parser are implemented in Antlr file *grammar.g*, and the tree walker is implemented in Antlr file *walker.g*. Source file *walker.g* contains short actions calling the methods in the class MxInterpreter.

The main() method is in the class MxMain. It first checks whether the input is coming from the console or a file, then initializes the lexer, the parser and the tree walker in differentiated ways. The class MxMain also handles exceptions and errors occurs in other classes, and prints corresponding messages.

The back end consists of a complicated type systems. All data are encapsulated in classes, which are derived from a single class *MxDataType*. This class does the common jobs of all types, and produce error messages, if a method is not specialized (overrided) by the derived class. The specialized function of simple types such as *bool, int, double*, and *string* are in classes *MxBool*, *MxInt*, *MxDouble* and *MxString*, respectively. These classes are simply wrapper classes of Java elemental types. Other data types, such as *range, matmask, function*, and *matrix* are in classes *MxRange*, *MxBitArray*, *MxFunction*, and *MxMatrix*. These

classes are wrapper classes of our Java matrix classes *jamaica.Range*, *jamaica.BitArray* and *jamaica.Matrix*. By using late-binding methods we implemented differentiated functions for various types. There is also an *MxException* class, for delivering error messages. The class *MxInterpreter* also manipulates multi-level symbol tables, implemented by *MxSymbolTable*. Figure 5.2 illustrates the type system in the Mx interpreter.

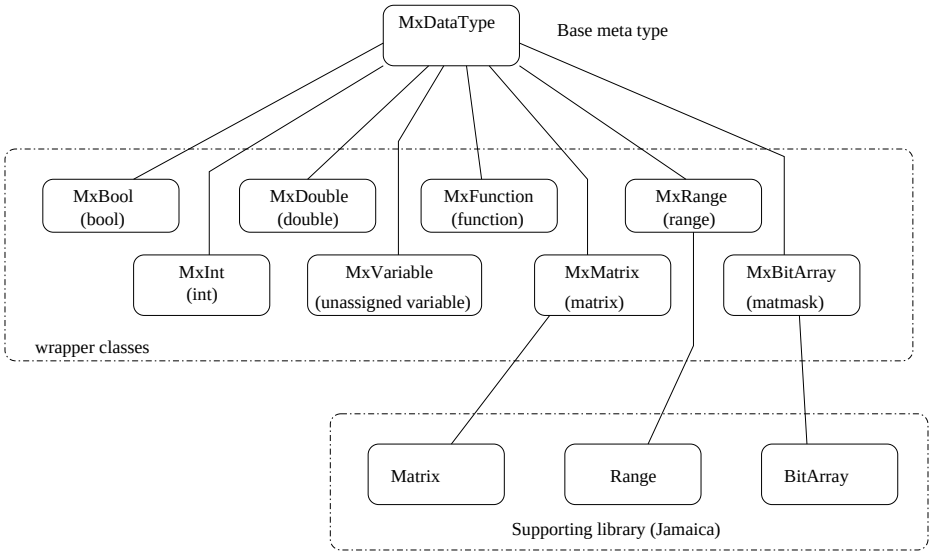

Figure 5.2: the type system of the Mx interpreter

In the supporting library, Jamaica, besides the classes mentioned earlier, there are Plotter and Painter classes. These two classes will only use basic functions such as windows management and drawing functions in the java.swing package. We have heavily documented the supporting library, with the javadoc standard, so one can easily know how to use the matrix classes. On the other hand, the interpretive classes are just slightly in-line documented, like many non-library open-source programs we have seen.

We have also designed a translative system, as an alternative, which may accept the Mx program files and translate to Java program files. This expected system includes a run time environment for operations of *range*, *matmask* and *matrix* types (Operations of other data types can be directly executed without using other libraries). For this reason, we have separated the supporting library from the interpreter and type systems. Given limited time (the Mx programming language is quite large by now), we cannot implement two different systems, but we have some discussions on this translative system in the chapter of related issues.

Chapter 6

Testing Plan

6.1 Goal

Since our compiler is to conduct mathematic operation, testing is essential and must cover every possible input (especially special values), otherwise the results we get would be dramatically different.

Testing is evolved throughout the compiler development, not only at the final stage, but also from the very beginning when the grammar was implemented. Our approach, probably like many others, is to test each piece of our compiler and sew them together to conduct the final test.

6.2 The first stage: unit testing

Initially, we noticed that there are certain Java packages supporting matrix operations. However, as we are using them, it turned that these packages did not support some of the functionalities that we need. Thus, we ended up writing our own matrix operation library. To ensure that all operations do what we expect, we use Matlab as our guidance for programming and unit testing. This library has its own unit testing program. This testing program saved us a lot of time in the later testing stages, albeit neither exhaustive nor complete. For the other part of this project, we also did a little testing by hand, in order to avoid fundamental errors in the system architecture.

6.3 The second Stage: regression test

Since many things can go wrong, and bugs could be hidden anywhere, regression test is important in the project. What we want to guarantee is what changes we make right now do not break the code we have had before. Thus, only testing the changes we made currently is not enough. On the other hand, to run through all the previous tests again and again is time consuming and cumbersome. As a result, we decide to build a script to automate this procedure. The script, written in tcl, takes a database, which contains one or many small *.mx programs and their expected returning values, and evaluates whether these *.mx programs are doing what they are expected to do.

This script greatly simplified our regression test job. At any time after the code modification, the batched test will immediately tell us whether we have broken some previously good code. Also, hundreds of small testing files are avoided. (With hundreds of small files, maintenance of the project files will be much more cumbersome: it would be very hard to track revisions of these files in the CVS repository.) To be clear, we write up different databases for different testing targets, such as boolean test, loop test, function call test, and range test. This approach eventually makes our testing systematic and assorted.

6.3.1 The testing database: a small awk-like script

The testing database file contains one or more entries of the following:

```
<expected-result> { <a-small-Mx-program> }
```

Both `expected-result` and `Mx-program` can span multiple lines. The `expected-result` part contains one or more items separated by spaces or newlines. Each of the items can have the following format:

`!`	switch between stdout and stderr (stdout by default)
`...`	ignore everything that follows
`/<regexp>/`	match by regular expression
`[<num>,<num>]`	match a number in a range
`<other>`	exactly match. number comparison for numbers

We can add comments in the testing database. Character `;` starts a line of comments, string `;;` starts a line of comments that will be printed when a new test is going to be made, and string `;;;` starts a line of comments that will be immediately printed when the executor program is scanning the database.

6.3.2 The testing database executor: a TCL program

We developed a TCL program to parse this testing database. This program scans the database, subtracts pieces of code surrounded by { and }, executes our Mx interpreter written in Java, and then compares the results printed by the Mx interpreter with the expected results given in the database, item by item.

6.3.3 A sample of the testing database

```
;;; samples.tdb: a sample of testing
;;; Hanhua Feng
;;; database for the Mx

;; assign
1 2 3 4 { a=[1,2;3,4]; return a; }
;; transpose
1 3 2 4 { a=[1,2;3,4]'; return a; }
;; transpose transpose
1 2 3 4 { a=[1,2;3,4]''; return a; }
;; triple transposes
1 2 4 { return [1;2;4]'''; }
;; vector element
3 { a=[1,2,3]; return a[2]; }
;; matrix element
3 { a = [1,2;3,4;5,6]; return a[1,0];}
;; matrix add
1 2 3 4 { return [1,1;1,1]+[0,1;2,3]; }
;; matrix sub
1 1 1 1 { return [2,3;4,5]-[1,2;3,4]; }
;; matrix +=
1 2 3 4 { a=[4,3;2,1];
          a+=[-3,-1;1,3];
          return a; }
;; matrix -=
1 2 3 4 { a=[2,2;4,6];
          a-=[1,0;1,2];
          return a; }
;; matrix unary -
3 -4 5 -6 { return -[-3,4,-5,6]; }
;; matrix element
4 { a = [1,2,3]; return a[2]+1; }
;; matrix element
```

```
4 { a = [1;2;3]; return 1+a[2]; }
;; matrix element
4 { a = [1;2;3]; return a[0]+a[2]; }
;; Error
! Error: ... { return true+3; }
;; Error
! Error: ... { return [1,2]*[1,2]; }
```

6.4 The third stage: advanced testing examples

Since some inter-connected errors might not be found by those small code segments, larger testing programs with scientific computing needs are extremely necessary. All programs in the chapter of tutorial, especially those complete examples, were acting as testing codes. These examples not only helped us find some bugs, but also greatly boosted our confidence of our Mx programming language. Please refer to the chapter of tutorial for these testing examples.

Chapter 7

Lesson Learned

During the period of design and implementation of our Mx programming language, we have learned some lessons. These lessons are important for projects in the future.

- Before designing a system, one should investigate the feasibility of supporting tools. At the beginning, we think those previously existing matrix classes must be useful in our project. Later, when we begin to design the details of our programming language, we found those matrix classes have so many deficiencies that it will not be a better choice to add patches to it than to write a completely new one. The detailed discussion will be in the next chapter.

- The main frame of a project is its most important part. When we begin to consider how to design a system, a small change of mind might create a very different system in the future. Although there are many ways in which we can achieve our destination, but which one is the shortest is not clear, even until you have finished a project. So getting advise from who has experience would make your life much easier.

- If you find a much easier way to implement a family of functions, then use it immediately and do not fear of making some of your code useless, because it will save you a lot of time in the future.

  As an example, at the beginning, in those functions that need a double-compatible type as an argument, we check the data type individually and call the error processor in each function; but later we found, an easier way to implement this, is to call a method doubleValue() which converts a data type to double, or raise an exception if the argument is not double-compatible. The exception is a runtime exception that can spread through the stack, so those functions will be much more concise. We implemented doubleValue() during the development and initially did not modify the already coded functions, but later we found code using doubleValue() has less bugs, so we finally changed all functions that had coded before doubleValue() was implemented.

Chapter 8

Related issues

8.1 Jamaica: Why a new matrix class?

The pre-existing Java matrix classes, such as Jama and java.vecmath, lack some fundamental functions which is necessary to our programming language. The most important ones are sub-matrix sharing the same data of the super-matrix, *matrix slicing*, and assigning selected elements of a matrix, *matrix masking*. Also, it is not very possible to add these functions by a wrapper. In Jama, the fundamental data structure does not support matrix slicing, and we can not get the source code of java.vecmath.

8.1.1 Underlying data structures

It is well known that there are two access methods for matrices or multiple dimensional arrays. The first one is using a continuous one dimensional array, the actual index of a matrix entry is computed by adding the second index to the product of the second dimension and the first index, (or reverse, whatever) i.e., if A is an $m \times n$ matrix,

$$A[i, j] = A[n * i + j] \quad (\text{or, } A[i + m * j]).$$

The second implementation is using a level of indirection,

$$A[i, j] = *(*(A + i) + j),$$

where $*$ is to get the content of a pointer. Many systems use the second approach, such as numerical recipe and Jama. The C programming language uses the first approach to allocate a fix-sized multiple dimensional array. People might think the first approach is inefficient because of multiplication; however, there is one more memory access in the second approach: the memory access is at the third location, which might have bad cache performance. Moreover, since all arrays in Java are objects, and Java will check index bounds for arrays, so in Java a level of indirection might be much more inefficient.

Jamaica uses the first approach. The underlying data structure contains the following fields:

- the one-dimensional array
- two dimensions of the matrix, m and n
- the start index of the upper-left entry of the matrix, s
- the difference of the indexes between two vertical neighboring entries, ms
- the difference of the indexes between two horizontal neighboring entries, ns

This data structure will make in-place transpose, flip, mirror, and slicing much easier: just create a new object and share the same array. For example, to create a transposed matrix, just swap m and n, and also swap ms and ns. To get a sub-matrix, just re-compute the start index s. We can even get an in-place diagonal vector of a matrix.

8.1.2 Comparison of functions between Jama and Jamaica

This is a brief comparison between Jama and our Java matrix class, Jamaica.

	Jamaica	Jama
status	developing	discontinued
matrix arithmetic operations	yes	yes
matrix generators (random,zeros,...)	yes	yes
index matrix generator (indgen)	yes	no
matrix inverse	yes	yes
sub-matrix	yes	yes
in-place sub-matrix	yes	no
transpose	yes	yes
flip/mirror	yes	no
matrix masking assignment	yes	no
matrix comparison	yes	no
solve linear equations	yes	yes
LU decomposition	yes	yes
QR decomposition	no	yes
singular value decomposition	no	yes
eigenvalues/Eigenvectors	no	yes
Cholesky decomposition	no	yes
printing	yes	yes
plotting and painting	yes	no
file I/O	yes	no

8.2 Mx: Implemented as a translative language

Conceptually, our Mx interpreter receives input program files, processes them, and generates desired outputs, so it can also be considered as a compiler or translator. However, practically, we define program processors that directly generate the result of the program as interpreters, and those that generate an intermediate program in other programming language as translators, and those that generates code which can be directly executed on hardware as a compilers.

Under this definition, initially, we want to create a translative system. After a few discussions, we conclude that it will be too hard to implement a translative system like Mx in just one semester, although by now I personally think it is possibly not that hard. So here we just present some ideas in translating the Mx programing language or something-alike, such as Matlab and IDL, to an imperative, statical-scoped, strong-typed programming language such as C, and hope that it will be helpful for people who want to build such a system. These ideas might contain some fundamental errors, or might have been in some formal literatures which we have never got a chance to read.

8.2.1 Interpretive versus translative: which is easier?

In our opinion, if the source language and the destination language are very similar, the translative job will be easy, even easier than developing an interpretive one. Our Mx programming language is not very similar to most of imperative programming languages such as C, C++ and Java, so the translative job would be a little less easy. However, we think for flow control statements, the translative procedure might be simpler than to actually execute it, because most of the programming languages have the similar flow-control statements. For those operators, we can translate it to function calls. For example,

```
B = random(8,8);
A = zeros(4,4);
A[C>3]=B[4:7,5:8];
```

can be translated to Java code

```
B = Matrix.random( 8, 8 );
A = new Matrix( 4, 4 );
A.assign( C.gt(3), B.slicing( new Range(4,7), new Range(5:8) ) );
```

using the Jamaica package.

Another difficulty of translator over interpreter is that the translator is harder to be debugged and tested. Besides, the supporting library evolves along with the project; so without a fixed interface, the code in the translator will not be stable.

8.2.2 Matrix operation: in-line expansion

Someone may think that it is not necessary to call the matrix class methods in generated Java code; we can generate in-line loop code like the following:

```
B = new double[64];
for ( int i=0; i=8; i++ )
    for ( int j=0; j<8; j++ )
        B[i*8+j] = Math.random();
A = new double[16];
for ( int i=0; i=4; i++ )
    for ( int j=0; j<4; j++ )
        A[i*4+j] = 0;
for ( int i=0; i=4; i++ )
    for ( int j=0; j=4; j++ )
        if ( C[i*4+j] > 3 )
            A[i*4+j] = B[(4+i)*8+(5+j)];
```

To improve performance, the assignments of B and A can be both reduced to one-level loops, and the multiplications can also be saved. However the two levels of loops of the last assignment cannot be reduced. Moreover, performance of these lines of code can be largely affected if one swaps two levels of loops. For example, in a C program,

```
int i, j, A[N][N];
for ( i=0; i<N; i++ )    /* [1] */
    for ( j=0; j<N; j++ )
        ... A[i][j] ...;
for ( j=0; j<N; j++ )    /* [2] */
    for ( i=0; i<N; i++ )
        ... A[i][j] ...;
```

The execution time of [1] might be much shorter than that of [2], because caching technology is used everywhere in modern computer architectures, so sequential access will be much faster than random access.

However, we think it would turn out to be a very hard project, which is impossible to finish in just one semester as a course project.

8.2.3 Variables without declaration

A common property of interpretive programming language is to use a variable without declaration. Our Mx programming language is not an exception. People might think that, without declaration, the type of a variable will be hard to be determined in compiling time. However, there are ways to get round of it:

1. The type of left-hand side of an assignment can be determined by the operator and operands, and the type of an operand can be determined recursively. In each assignment, we mark the data types of the right-hand-sides.

2. Check whether the right-hand sides of all assignments of this variable are the same. If so, the type of this variable is determined. Functional programming language does not need declarations, because there are no assignments.

3. If not so, check whether these types can be all promoted to some type, which is also compatible at all access points of this variable. For example, here x can be promoted to double in the second line of the following:

```
if ( ... ) {
   x = 3;
} else {
   x = 3.5;
}
z = x + 1.5;
```

4. Checking the return type of a function might be a hard job, and it is necessary for determination of types of variables. This can be done by checking type in the return statements. If there is a recursive call, it can still be done by introducing some algorithms about logic.

5. If all above efforts failed, we can promote it to the most general type (or say a meta type), which is exactly the base class in the type system of our interpreter, class *MxDataType*. (Of course, classes of our interpreter must be included in the run-time environment.) As an alternative, the translator might print messages to ask whether the programmer can change it.

6. All global variables accessible by programs but not examined by the translator can be handled as variables of the meta type.

8.2.4 Dual scoping

We didn't mention the scoping problem in the language reference manual. However, we have considered the scoping problem in our implementation. Translative languages prefer static scoping, so by default our Mx programming language is also static scoping. However, we have implemented the *dual scoping* mechanism in the interpreter because during the design stage, we do not know what is feasible, static or dynamic. In the current implementation, each time we create a sub symbol table, it has two parent symbol tables, one is its static parent, the other is its dynamic parent. We can add a qualifier before using a variable to force the interpreter to search this variable in the dynamic ancestors of the current symbol table.

In the current implementation, if we use static scoping, the deep binding mechanism is used naturally, whereas, if we use dynamic scoping, the shallow binding mechanism is used.

Bibliography

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

[2] IDL, http://www.rsinc.com/idl/index.asp

[3] Maple, http://www.maplesoft.com/

[4] Mathcad, http://www.mathsoft.com/products/mathcad/

[5] Mathematica, http://www.wolfram.com/products/mathematica/

[6] GNU Octave, http://www.octave.org/

[7] Scilab, http://www.scilab.org/

[8] Object-Oriented Numerics Page, http://www.oonumerics.org/oon/

[9] Blitz++, http://www.oonumerics.org/blitz/

[10] POOMA, http://www.codesourcery.com/pooma/pooma

[11] Java.vecmath, http://java.sun.com/products/java-media/3D/collateral/j3d_api/j3d_api_3.html

[12] Jama, http://math.nist.gov/javanumerics/jama/

[13] Titanium, http://www.cs.berkeley.edu/Research/Projects/titanium/

[14] http://www.netlib.org/lapack/

[15] William Press, Saul Teukolsky, William Vetterling and Brain Flannery, Numerical Recipes in Fortran 77, 2nd Edition, Cambridge University Press, 1992, http://www.nr.com

[16] f2c, http://www.netlib.org/f2c/

[17] Various authors, http://www.starlink.rl.ac.uk/star/docs/sc13.htx/N-a2b3c1.html , http://www.lysator.liu.se/c/num-recipes-in-c.html , and http://www.nr.com/bug-rebutt.html

[18] Antlr, http://www.antlr.org/

[19] Jampack, ftp://thales.cs.umd.edu/pub/Jampack/Jampack/AboutJampack.html

Appendix A

Language Syntax

A.1 Lexical rules

alpha → 'a'..'z' | 'A'..'Z' | '_'
digit → '0'..'9'
id → **alpha** (**alpha**|**digit**)*
number → ((digit)+ ('.' (digit)*)? | '.' (digit)+) ((E|e) (+|-)? (digit)+)?
string → '"' (∼('"') | ('"' '"'))* '"'

Skipped tokens

linebreak → '$\backslash n$'
whitespace → ' ' | '$\backslash t$' | '$\backslash r$' | **linebreak**
comment → ('/*' (∼'*/')* '*/') | '//' (∼**linebreak**)* **linebreak**

A.2 Syntactic rules

program → (statement | func-def)*
statement → for-stmt | if-stmt | loop-stmt | break-stmt | return-stmt
| load-stmt | assignment | func-call-stmt | '{' (statement)* '}'
for-stmt → 'for' '(' **id** '=' range (',' **id** '=' range)* ')' statement
if-stmt → 'if' '(' expression ')' statement ('else' statement)?
loop-stmt → 'loop' ('(' **id** ')')? statement
break-stmt → 'break' (**id**)? ';'
cont-stmt → 'continue' (**id**)? ';'
return-stmt → 'return' expression ';'
load-stmt → 'include' **string** ';'
assignment → l-value ('=' | '+=' | '−=' | '*=' | '/=' | '%=' | '/′=') expression ';'
func-call-stmt → func-call ';'
func-call → **id** '(' expr-list ')'
expr-list → expression (',' expression)* | ϵ
func-def → 'func' **id** '(' var-list ')' func-body
var-list → **id** (',' **id**)* | ϵ
func-body → '=' expression ';' | '{' (statement)* '}'
expression → logic-term ('or' logic-term)*
logic-term → logic-factor ('and' logic-factor)*
logic-factor → ('not')? relat-expr
relat-expr → arith-expr (('>=' | '<=' | '>' | '<' | '==' | '! =') arith-expr)?
arith-expr → arith-term (('+'|'−') arith-term)*

arith-term → arith-factor (('∗'|'/'|'%'|'/′') arith-factor)*
arith-factor → ('+'|'−') r-value | r-value '′'
r-value → l-value | func-call | constant | array
l-value → **id** ('[' index ']')*
index → range (',' range)?
range → expression (':' (expression)? | '::' expression)? | ':'
constant → **number**
array → '[' expression ((','|';') expression)* ']'

Appendix B

Code listing

These are source files, testing script and databases, and examples, as of May 13. (Generated files are not included.)

```
./src
-rw-r--r--    1 hanhua   phd          6770 May 12 17:42 grammar.g
-rw-r--r--    1 hanhua   phd          6155 May 12 17:42 walker.g
-rw-r--r--    1 hanhua   phd          4628 May 12 19:44 MxMain.java
-rw-r--r--    1 hanhua   phd         11544 May 12 17:42 MxInterpreter.java
-rw-r--r--    1 hanhua   phd          1116 May 12 19:44 MxBitArray.java
-rw-r--r--    1 hanhua   phd          1555 May 12 19:44 MxBool.java
-rw-r--r--    1 hanhua   phd          3806 May 12 19:44 MxDataType.java
-rw-r--r--    1 hanhua   phd          3771 May 12 19:44 MxDouble.java
-rw-r--r--    1 hanhua   phd           347 May 12 19:44 MxException.java
-rw-r--r--    1 hanhua   phd          1963 May 12 19:44 MxFunction.java
-rw-r--r--    1 hanhua   phd          3979 May 12 19:44 MxInt.java
-rw-r--r--    1 hanhua   phd         10306 May 12 19:44 MxMatrix.java
-rw-r--r--    1 hanhua   phd          1852 May 12 19:44 MxRange.java
-rw-r--r--    1 hanhua   phd           999 May 12 19:44 MxString.java
-rw-r--r--    1 hanhua   phd          2914 May 12 19:44 MxSymbolTable.java
-rw-r--r--    1 hanhua   phd           597 May 12 19:44 MxVariable.java
-rw-r--r--    1 hanhua   phd         15394 May 12 17:16 MxInternalFunction.m4
-rw-r--r--    1 hanhua   phd          1429 May 11 12:14 Makefile

./src/jamaica
-rw-r--r--    1 hanhua   phd         42670 May 12 17:13 Matrix.java
-rw-r--r--    1 hanhua   phd          5645 May 12 20:13 BitArray.java
-rw-r--r--    1 hanhua   phd          4569 May 12 20:13 Range.java
-rw-r--r--    1 hanhua   phd         11397 May 12 17:13 Painter.java
-rw-r--r--    1 hanhua   phd         14551 May 12 17:13 Plotter.java
-rw-r--r--    1 hanhua   phd         10061 May 12 17:13 MatrixTest.java

./test
-rwxr-xr-x    1 hanhua   phd          6847 May  9 10:50 mxtest.tcl
-rw-r--r--    1 hanhua   phd          1515 May  7 10:24 bool.tdb
-rw-r--r--    1 hanhua   phd          1133 May  7 10:39 double.tdb
-rw-r--r--    1 hanhua   phd          1181 May  9 18:41 infunc.tdb
-rw-r--r--    1 hanhua   phd           943 May  7 10:36 int.tdb
-rw-r--r--    1 hanhua   phd           863 May  9 13:55 samples.tdb
-rw-r--r--    1 hanhua   phd           175 May 13 12:44 tzfor.tdb
```

```
-rw-r--r--    1 hanhua   phd           407 May 13 12:41 tzfunc.tdb
-rw-r--r--    1 hanhua   phd           209 May 13 12:49 tzloop.tdb
-rw-r--r--    1 hanhua   phd          2086 May 13 12:37 tzmisc.tdb
-rw-r--r--    1 hanhua   phd           684 May 13 12:41 tzmx.tdb
-rw-r--r--    1 hanhua   phd           633 May 13 12:40 tzrange.tdb
-rw-r--r--    1 hanhua   phd           170 May 13 12:40 tzstring.tdb
-rw-r--r--    1 hanhua   phd           264 May  9 13:59 fmri.mx
-rw-r--r--    1 hanhua   phd          1394 May 11 23:12 lenna.mx
-rw-r--r--    1 hanhua   phd           609 May 11 23:12 lorenz.mx
-rw-r--r--    1 hanhua   phd           548 May  9 13:58 mandelbrot.mx
-rw-r--r--    1 hanhua   phd           209 May 11 23:12 potato.mx
-rw-r--r--    1 hanhua   phd           291 May  9 13:57 sierpinski.mx
-rwxr-xr-x    1 hanhua   phd           114 May 13 13:44 runall.sh
```

File listing begins at the next page. $*$.mx files are not included because they are already in the chapter of tutorial.

B.1 Parser

B.1.1 src/grammar.g

```
/*
 * grammar.g : the lexer and the parser, in ANTLR grammar.
 *
 * @author Tiantian Zhou - tz2002@columbia.edu
 *
 * @version $Id: grammar.g,v 1.15 2003/05/13 23:25:22 hanhua Exp $
 */

class MxAntlrLexer extends Lexer;

options{
    k = 2;
    charVocabulary = '\3'..'\377';
    testLiterals = false;
    exportVocab = MxAntlr;
}

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

protected
ALPHA   : 'a'..'z' | 'A'..'Z' | '_' ;

protected
DIGIT   :   '0'..'9';

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

NL      : ('\n' | ('\r' '\n') => '\r' '\n' | '\r')
            { $setType(Token.SKIP); newline(); }
        ;

COMMENT : ( "/*" (
                    options {greedy=false;} :
                    (NL)
                    | ~( '\n' | '\r' )
                )* "*/"
            | "//" (~( '\n' | '\r' ))* (NL)
          )                       { $setType(Token.SKIP); }
        ;
```

```
LDV_LDVEQ : "/'" (
                    ('=') => '=' { $setType(LDVEQ); }
                    | { $setType(LDV); }
                 );

LPAREN  : '(';
RPAREN  : ')';
MULT    : '*';
PLUS    : '+';
MINUS   : '-';
RDV     : '/';
MOD     : '%';
SEMI    : ';';
LBRACE  : '{';
RBRACE  : '}';
LBRK    : '[';
RBRK    : ']';
ASGN    : '=';
COMMA   : ',';
PLUSEQ  : "+=";
MINUSEQ : "-=";
MULTEQ  : "*=";
RDVEQ   : "/=";
MODEQ   : "%=";
GE      : ">=";
LE      : "<=";
GT      : '>';
LT      : '<';
EQ      : "==";
NEQ     : "!=";
TRSP    : '\'';
COLON   : ':';
DCOLON  : "::";

ID  options { testLiterals = true; }
        : ALPHA (ALPHA|DIGIT)*
        ;

/* NUMBER example:
 * 1, 1e, 1., 1.e10, 1.1, 1.1e10
 */

NUMBER  : (DIGIT)+ ('.' (DIGIT)*)? (('E'|'e') ('+'|'-')? (DIGIT)+)? ;

/* Hanhua: the definition of the string might be different */
STRING  : '"'!
              (  ~('"' | '\n')
               | ('"'!'"')
              )*
          '"'!
        ;
```

```
class MxAntlrParser extends Parser;

options{
    k = 2;
    buildAST = true;
    exportVocab = MxAntlr;
}


tokens {
   // PROG;             // Hanhua: PROG is actually statements
   STATEMENT;
   ARRAY_ROW;
   ARRAY;
   VAR_LIST;
   EXPR_LIST;
   FUNC_CALL;
   // FUNC_BODY;        // Hanhua: this level is not necessary
   // FUNC_BODY_ASGN;   // Hanhua: this level is not necessary
   FOR_CON;
   LOOP;
   UPLUS;
   UMINUS;
}

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

program
        : ( statement | func_def )* EOF!
            {#program = #([STATEMENT,"PROG"], program); }
        ;

statement
        : for_stmt
        | if_stmt
        | loop_stmt
        | break_stmt
        | return_stmt
        | load_stmt
        | assignment
        | func_call_stmt
        | LBRACE! (statement)* RBRACE!
```

```
            {#statement = #([STATEMENT,"STATEMENT"], statement); }
        ;

for_stmt
        : "for"^ LPAREN! for_con RPAREN! statement
        ;

for_con
        : ID ASGN! range (COMMA! ID ASGN! range)*
            {#for_con = #([FOR_CON,"FOR_CON"], for_con); }
        ;

if_stmt
        : "if"^ LPAREN! expression RPAREN! statement
            (options {greedy = true;}: "else"! statement )?
        ;

loop_stmt!
        : "loop" ( LPAREN! id:ID RPAREN! )? stmt:statement
            {
                if ( null == #id )
                    #loop_stmt = #([LOOP,"loop"], #stmt);
                else
                    #loop_stmt = #([LOOP,"loop"], #stmt, #id);
            }
        ;

break_stmt
        : "break"^ (ID)? SEMI!
        ;

continue_stmt
        : "continue"^ (ID)? SEMI!
        ;

return_stmt
        : "return"^ (expression)? SEMI!
        ;

load_stmt
        : "include"^ STRING SEMI!
        ;

assignment
        : l_value ( ASGN^ | PLUSEQ^ | MINUSEQ^ | MULTEQ^ | LDVEQ^
                    | MODEQ^ | RDVEQ^
                  ) expression SEMI!
        ;

func_call_stmt
        : func_call SEMI!
        ;
```

```
func_call
        : ID LPAREN! expr_list RPAREN!
            {#func_call = #([FUNC_CALL,"FUNC_CALL"], func_call); }
        ;

expr_list
        : expression ( COMMA! expression )*
            {#expr_list = #([EXPR_LIST,"EXPR_LIST"], expr_list); }
        |   /*nothing: Hanhua: although empty, we should have a node */
            {#expr_list = #([EXPR_LIST,"EXPR_LIST"], expr_list); }
        ;

func_def
        : "func"^ ID LPAREN! var_list RPAREN! func_body
        ;

var_list
        : ID ( COMMA! ID )*
            {#var_list = #([VAR_LIST,"VAR_LIST"], var_list); }
        |   /* nothing. Hanhua: necessary to add an node for empty list */
            {#var_list = #([VAR_LIST,"VAR_LIST"], var_list); }
        ;

// Hanhua: an intermediate level is removed
func_body
        : ASGN! a:expression SEMI!
            {#func_body = #a; }
        | LBRACE! (statement)* RBRACE!
            {#func_body = #([STATEMENT,"FUNC_BODY"], func_body); }
        ;

expression
        : logic_term ( "or"^ logic_term )*
        ;

logic_term
        : logic_factor ( "and"^ logic_factor )*
        ;

logic_factor
        : ("not"^)? relat_expr
        ;

relat_expr
        : arith_expr ( (GE^ | LE^ | GT^ | LT^ | EQ^ | NEQ^) arith_expr )?
        ;

arith_expr
        : arith_term ( (PLUS^ | MINUS^) arith_term )*
        ;

arith_term
        : arith_factor ( (MULT^ | LDV^ | MOD^ | RDV^) arith_factor )*
```

```
        ;

arith_factor
        : PLUS! r_value
            {#arith_factor = #([UPLUS,"UPLUS"], arith_factor); }
        | MINUS! r_value
            {#arith_factor = #([UMINUS,"UMINUS"], arith_factor); }
        | r_value (TRSP^)*;


r_value
        : l_value
        | func_call
        | NUMBER
        | STRING
        | "true"
        | "false"
        | array
        | LPAREN! expression RPAREN!
        ;

l_value
        : ID^ ( LBRK! index RBRK! )*
        ;

index
        : range (COMMA! range)?
            {#index = #([EXPR_LIST,"INDEX"], index); }
        ;

range
        : expression (COLON^ (expression)? | DCOLON^ expression )? | COLON^
        ;

array_row
        : expression (COMMA! expression)*
            {#array_row = #([ARRAY_ROW,"ARRAY_ROW"], array_row); }
        ;

array
        : LBRK! array_row (SEMI! array_row)* RBRK!
            {#array = #([ARRAY,"ARRAY"], array); }
        ;

cl_statement
        : ( statement | func_def )
        | "exit"
            { System.exit(0); }
        | EOF!
            { System.exit(0); }
        ;
```

B.1.2 src/walker.g

```
/*
 * walker.g : the AST walker.
 *
 * @author Tiantian Zhou - tz2002@columbia.edu
 *
 * @version $Id: walker.g,v 1.24 2003/05/12 21:42:43 hanhua Exp $
 */

{
import java.io.*;
import java.util.*;
}

class MxAntlrWalker extends TreeParser;
options{
    importVocab = MxAntlr;
}

{
    static MxDataType null_data = new MxDataType( "<NULL>" );
    MxInterpreter ipt = new MxInterpreter();
}

expr returns [ MxDataType r ]
{
    MxDataType a, b;
    Vector v;
    MxDataType[] x;
    String s = null;
    String[] sx;
    r = null_data;
}
        : #("or" a=expr right_or:.)
            {
                if ( a instanceof MxBool )
                    r = ( ((MxBool)a).var ? a : expr(#right_or) );
                else
                    r = a.or( expr(#right_or) );
            }
        | #("and" a=expr right_and:.)
            {
                if ( a instanceof MxBool )
                    r = ( ((MxBool)a).var ? expr(#right_and) : a );
                else
                    r = a.and( expr(#right_and) );
            }
        | #("not" a=expr)           { r = a.not(); }
        | #(GE a=expr b=expr)       { r = a.ge( b ); }
        | #(LE a=expr b=expr)       { r = a.le( b ); }
        | #(GT a=expr b=expr)       { r = a.gt( b ); }
        | #(LT a=expr b=expr)       { r = a.lt( b ); }
        | #(EQ a=expr b=expr)       { r = a.eq( b ); }
```

```
| #(NEQ a=expr b=expr)       { r = a.ne( b ); }
| #(PLUS a=expr b=expr)      { r = a.plus( b ); }
| #(MINUS a=expr b=expr)     { r = a.minus( b ); }
| #(MULT a=expr b=expr)      { r = a.times( b ); }
| #(LDV a=expr b=expr)       { r = a.lfracts( b ); }
| #(RDV a=expr b=expr)       { r = a.rfracts( b ); }
| #(MOD a=expr b=expr)       { r = a.modulus( b ); }
| #(UPLUS a=expr)            { r = a; }
| #(UMINUS a=expr)           { r = a.uminus(); }
| #(PLUSEQ a=expr b=expr)    { r = a.add( b ); }
| #(MINUSEQ a=expr b=expr)   { r = a.sub( b ); }
| #(MULTEQ a=expr b=expr)    { r = a.mul( b ); }
| #(LDVEQ a=expr b=expr)     { r = a.ldiv( b ); }
| #(RDVEQ a=expr b=expr)     { r = a.rdiv( b ); }
| #(MODEQ a=expr b=expr)     { r = a.rem( b ); }
| #(COLON (c1:. (c2:.)?)?)
    {
        r = MxRange.create( (null==#c1) ? null : expr(#c1),
                            (null==#c2) ? null : expr(#c2) );
    }
| #(DCOLON a=expr b=expr)    { r = MxRange.create( a, b, 1 ); }
| #(ASGN a=expr b=expr)      { r = ipt.assign( a, b ); }
| #(FUNC_CALL a=expr x=mexpr)
    { r = ipt.funcInvoke( this, a, x ); }
| #(ARRAY                    { v = new Vector(); }
     (a=expr                 { v.add( a ); }
     )*
   )
    { r = MxMatrix.joinVert( ipt.convertExprList( v ) ); }
| #(ARRAY_ROW                { v = new Vector(); }
     (a=expr                 { v.add( a ); }
     )+
   )
    { r = MxMatrix.joinHori( ipt.convertExprList( v ) ); }
| num:NUMBER                 { r = ipt.getNumber( num.getText() ); }
| str:STRING                 { r = new MxString( str.getText() ); }
| "true"                     { r = new MxBool( true ); }
| "false"                    { r = new MxBool( false ); }
| #(id:ID                    { r = ipt.getVariable( id.getText() ); }
      ( x=mexpr              { r = ipt.subMatrix( r, x ); }
      )*
   )
| #(TRSP a=expr)             { r = a.transpose(); }
| #("for" x=mexpr forbody:.)
    {
        MxInt[] values = ipt.forInit( x );
        while ( ipt.forCanProceed( x, values ) )
        {
            r = expr( #forbody );
            ipt.forNext( x, values );
        }
        ipt.forEnd( x );
    }
```

```
        | #("if" a=expr thenp:. (elsep:.)?)
            {
                if ( !( a instanceof MxBool ) )
                    return a.error( "if: expression should be bool" );
                if ( ((MxBool)a).var )
                    r = expr( #thenp );
                else if ( null != elsep )
                    r = expr( #elsep );
            }
        | #(STATEMENT (stmt:. { if ( ipt.canProceed() ) r = expr(#stmt); } )*)
        | #(LOOP loopbody:.
                  (loopid:ID        { s = loopid.getText(); }
                  )?
           )
            {
                while ( ipt.canProceed() )
                {
                    r = expr( #loopbody );
                    ipt.loopNext( s );
                }
                ipt.loopEnd( s );
            }
        | #("break" (breakid:ID      { s = breakid.getText(); }
                    )?
           )                        { ipt.setBreak( s ); }
        | #("continue" (contid:ID    { s = contid.getText(); }
                       )?
           )                        { ipt.setContinue( s ); }
        | #("return" ( a=expr        { r = ipt.rvalue( a ); }
                     )?
           )                        { ipt.setReturn( null ); }
        | #("func" fname:ID sx=vlist fbody:.)
            { ipt.funcRegister( fname.getText(), sx, #fbody ); }
        | #("include" a=expr)        { ipt.include( a ); }
        ;

mexpr returns [ MxDataType[] rv ]
{
    MxDataType a;
    rv = null;
    Vector v;
}
        : #(EXPR_LIST         { v = new Vector(); }
                ( a=expr      { v.add( a ); }
                )*
           )                  { rv = ipt.convertExprList( v ); }
        | a=expr              { rv = new MxDataType[1]; rv[0] = a; }
        |  #(FOR_CON          { v = new Vector(); }
                ( s:ID a=expr { a.setName( s.getText() ); v.add(a); }
                )+
            )                 { rv = ipt.convertExprList( v ); }
        ;
```

```
vlist returns [ String[] sv ]
{
    Vector v;
    sv = null;
}
        : #(VAR_LIST     { v = new Vector(); }
             (s:ID       { v.add( s.getText() ); }
             )*
          )              { sv = ipt.convertVarList( v ); }
        ;
```

B.2 Interpreter

B.2.1 src/MxMain.java

```
import java.io.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.RecognitionException;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;
import jamaica.*;

/**
 * The main class
 *
 * @author Hanhua Feng - hf2048@columbia.edu
 * @version $Id: MxMain.java,v 1.15 2003/05/13 17:34:31 hanhua Exp $
 */
class MxMain {
    static boolean verbose = false;

    public static void execFile( String filename ) {
        try
        {
            InputStream input = ( null != filename ) ?
                (InputStream) new FileInputStream( filename ) :
                (InputStream) System.in;

            MxAntlrLexer lexer = new MxAntlrLexer( input );

            MxAntlrParser parser = new MxAntlrParser( lexer );

            // Parse the input program
            parser.program();

            if ( lexer.nr_error > 0 || parser.nr_error > 0 )
            {
                System.err.println( "Parsing errors found. Stop." );
                return;
            }

            CommonAST tree = (CommonAST)parser.getAST();

            if ( verbose )
            {
                // Print the resulting tree out in LISP notation
                System.out.println(
                    "================ tree structure ======================" );
                System.out.println( tree.toStringList() );
            }

            MxAntlrWalker walker = new MxAntlrWalker();
            // Traverse the tree created by the parser
```

```
        if ( verbose )
            System.out.println(
                "================ program output ======================" );

        MxDataType r = walker.expr( tree );

        if ( verbose )
            System.out.println(
                "================ program return ======================" );
        if ( null != r )
            r.print();

        if ( verbose )
        {
            System.out.println(
                "================ global variables ====================" );
            walker.ipt.symt.what();
        }

    } catch( IOException e ) {
        System.err.println( "Error: I/O: " + e );
    } catch( RecognitionException e ) {
        System.err.println( "Error: Recognition: " + e );
    } catch( TokenStreamException e ) {
        System.err.println( "Error: Token stream: " + e );
    } catch( Exception e ) {
        System.err.println( "Error: " + e );
    }

    while ( Painter.frameCount() > 0 || Plotter.frameCount() > 0 )
    {
        try
        {
            Thread.sleep( 1000 );
        } catch ( InterruptedException e ) {}
    }
}

public static void commandLine() {
    InputStream input = (InputStream) new DataInputStream( System.in );
    MxAntlrWalker walker = new MxAntlrWalker();

    for ( ;; )
    {
        try
        {
            while( input.available() > 0 )
                input.read();
        }
        catch ( IOException e ) {}

        System.out.print( "Mx> " );
```

```
            System.out.flush();

            MxAntlrLexer lexer = new MxAntlrLexer( input );
            MxAntlrParser parser = new MxAntlrParser( lexer );

            try
            {
                parser.cl_statement();
                CommonAST tree = (CommonAST)parser.getAST();
                MxDataType r = walker.expr( tree );
                if ( verbose && r != null)
                    r.print();
            } catch( RecognitionException e ) {
                System.err.println( "Recognition exception: " + e );
            } catch( TokenStreamException e ) {
                if ( e instanceof TokenStreamIOException ) {
                    System.err.println( "Token I/O exception" );
                    break;
                }
                System.err.println( "Error: Token stream: " + e );
            } catch( MxException e ) {
                System.err.println( "Error: Interpretive: " + e );
                e.printStackTrace();
            } catch( RuntimeException e ) {
                System.err.println( "Error: Runtime: " + e );
                e.printStackTrace();
            } catch( Exception e ) {
                System.err.println( "Error: " + e );
                e.printStackTrace();
            }

        }
    }

    public static void main( String[] args ) {

        verbose = args.length >= 1 && args[0].equals( "-v" );

        boolean batch = args.length >=1 && args[0].equals( "-b" );

        if ( args.length >= 1 && args[args.length-1].charAt(0) != '-' )
            execFile( args[args.length-1] );
        else if ( batch )
            execFile( null );
        else
            commandLine();

        System.exit( 0 );
    }
}
```

B.2.2 src/MxInterpreter.java

```
import java.util.*;
import java.io.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.RecognitionException;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;
import jamaica.*;

/** Interpreter routines that is called directly from the tree walker.
 *
 * @author Hanhua Feng - hf2048@columbia.edu
 * @version $Id: MxInterpreter.java,v 1.27 2003/05/12 21:42:43 hanhua Exp $
 */
class MxInterpreter {
    MxSymbolTable symt;

    final static int fc_none = 0;
    final static int fc_break = 1;
    final static int fc_continue = 2;
    final static int fc_return = 3;

    private int control = fc_none;
    private String label;

    private Random random = new Random();

    public MxInterpreter() {
        symt = new MxSymbolTable( null, null );
        registerInternal();
    }

    public MxDataType[] convertExprList( Vector v ) {
        /* Note: expr list can be empty */
        MxDataType[] x = new MxDataType[v.size()];
        for ( int i=0; i<x.length; i++ )
            x[i] = (MxDataType) v.elementAt( i );
        return x;
    }

    public static String[] convertVarList( Vector v ) {
        /* Note: var list can be empty */
        String[] sv = new String[ v.size() ];
        for ( int i=0; i<sv.length; i++ )
            sv[i] = (String) v.elementAt( i );
        return sv;
    }

    public static MxDataType getNumber( String s ) {
        if ( s.indexOf( '.' ) >= 0
             || s.indexOf( 'e' ) >= 0 || s.indexOf( 'E' ) >= 0 )
            return new MxDouble( Double.parseDouble( s ) );
```

```
    return new MxInt( Integer.parseInt( s ) );
}

public static int getDataType( String s ) {
    if ( s.equals( "double" ) )
        return Matrix.DT_DOUBLE;
    if ( s.equals( "int" ) )
        return Matrix.DT_INT;
    if ( s.equals( "short" ) )
        return Matrix.DT_SHORT;
    if ( s.equals( "byte" ) )
        return Matrix.DT_BYTE;
    if ( s.equals( "float" ) )
        return Matrix.DT_FLOAT;
    throw new MxException( "Unknown data type" );
}

public MxDataType getVariable( String s ) {
    // default static scoping
    MxDataType x = symt.getValue( s, true, 0 );
    if ( null == x )
        return new MxVariable( s );
    return x;
}


public MxDataType rvalue( MxDataType a ) {
    if ( null == a.name )
        return a;
    return a.copy();
}

public MxDataType deepRvalue( MxDataType a ) {
    if ( null == a.name )
        return a;
    if ( a instanceof MxMatrix )
        return ((MxMatrix)a).deepCopy();
    return a.copy();
}

public MxDataType assign( MxDataType a, MxDataType b ) {
    if ( null != a.name )
    {
        MxDataType x = deepRvalue( b );
        x.setName( a.name );
        symt.setValue( x.name, x, true, 0 );  // scope?
        return x;
    }

    if ( a instanceof MxMatrix )
    {
        BitArray mask = ((MxMatrix)a).mask;
        if ( null == mask )
```

```
        {
            if ( b instanceof MxMatrix )
                ((MxMatrix)a).mat.assign( ((MxMatrix)b).mat );
            else
                ((MxMatrix)a).mat.assign( MxDouble.doubleValue( b ) );
        }
        else
        {
            if ( b instanceof MxMatrix )
                ((MxMatrix)a).mat.assign( mask, ((MxMatrix)b).mat );
            else
                ((MxMatrix)a).mat.assign( mask, MxDouble.doubleValue( b ) );
        }
        return a;
    }

    return a.error( b, "=" );
}

public MxDataType subMatrix( MxDataType x, MxDataType[] mexpr ) {
    if ( x instanceof MxMatrix )
    {
        if ( 2 == mexpr.length )
        {
            Range mr, nr;

            if ( mexpr[0] instanceof MxInt
                 || mexpr[0] instanceof MxDouble )
                mr = new Range( MxInt.intValue( mexpr[0] ), 1, 1 );
            else if ( mexpr[0] instanceof MxRange )
                mr = ((MxRange)mexpr[0]).getRange(
                    ((MxMatrix)x).mat.height() );
            else
                return x.error( mexpr[0], "slicing" );

            if ( mexpr[1] instanceof MxInt
                 || mexpr[1] instanceof MxDouble )
                nr = new Range( MxInt.intValue( mexpr[1] ), 1, 1 );
            else if ( mexpr[1] instanceof MxRange )
                nr = ((MxRange)mexpr[1]).getRange(
                    ((MxMatrix)x).mat.width() );
            else
                return x.error( mexpr[1], "slicing" );

            return new MxMatrix( ((MxMatrix)x).mat.slice( mr, nr ) );
        }

        if ( 1 == mexpr.length )
        {
            if ( mexpr[0] instanceof MxBitArray )
                return new MxMatrix( ((MxMatrix)x).mat,
                                     ((MxBitArray)mexpr[0]).ba );
```

```
            Range range;

            if ( mexpr[0] instanceof MxInt
                 || mexpr[0] instanceof MxDouble )
                range = new Range( MxInt.intValue( mexpr[0] ), 1, 1 );
            else if ( mexpr[0] instanceof MxRange )
                range = ((MxRange)mexpr[0]).range;
            else
                return x.error( mexpr[0], "slicing" );

            if ( 1 == ((MxMatrix)x).mat.width() )
                return new MxMatrix(
                    ((MxMatrix)x).mat.slice( range, new Range( 0, 0 ) ) );
            else if ( 1 == ((MxMatrix)x).mat.height() )
                return new MxMatrix(
                    ((MxMatrix)x).mat.slice( new Range( 0, 0 ), range ) );
        }
    }

    return x.error( "slicing" );
}

public MxDataType funcInvoke(
    MxAntlrWalker walker,  MxDataType func,
    MxDataType[] params ) throws antlr.RecognitionException {

    // func must be an existing function
    if ( !( func instanceof MxFunction ) )
        return func.error( "not a function" );

    // Is this function an internal function?
    // Names of formal args are not necessary for internal functions.
    if ( ((MxFunction)func).isInternal() )
        return execInternal( ((MxFunction)func).getInternalId(),
                             params );

    // otherwise check numbers of actual and formal arguments
    String[] args = ((MxFunction)func).getArgs();
    if ( args.length != params.length )
        return func.error( "unmatched length of parameters" );

    // create a new symbol table
    symt = new MxSymbolTable( ((MxFunction)func).getParentSymbolTable(),
                              symt );

    // assign actual parameters to formal arguments
    for ( int i=0; i<args.length; i++ )
    {
        MxDataType d = rvalue( params[i] );
        d.setName( args[i] );
        symt.setValue( args[i], d, false, 0 );
    }
```

```
    // call the function body
    MxDataType r = walker.expr( ((MxFunction)func).getBody() );

    // no break or continue can go through the function
    if ( control == fc_break || control == fc_continue )
        throw new MxException( "nowhere to break or continue" );

    // if a return was called
    if ( control == fc_return )
        tryResetFlowControl( ((MxFunction)func).name );

    // remove this symbol table and return
    symt = symt.dynamicParent();

    return r;
}

public void funcRegister( String name, String[] args, AST body ) {
    symt.put( name, new MxFunction( name, args, body, symt ) );
}

public void setBreak( String label ) {
    this.label = label;
    control = fc_break;
}

public void setContinue( String label ) {
    this.label = label;
    control = fc_continue;
}

public void setReturn( String label ) {
    this.label = label;
    control = fc_return;
}

public void tryResetFlowControl( String loop_label ) {
    if ( null == label || label.equals( loop_label ) )
        control = fc_none;
}

public void loopNext( String loop_label ) {
    if ( control == fc_continue )
        tryResetFlowControl( loop_label );
}

public void loopEnd( String loop_label ) {
    if ( control == fc_break )
        tryResetFlowControl( loop_label );
}

public boolean canProceed() {
    return control == fc_none;
```

```
}

public MxInt[] forInit( MxDataType[] mexpr ) {
    // very much like function call
    for ( int i=0; i<mexpr.length; i++ )
        if ( ! ( mexpr[i] instanceof MxRange )  )
        {
            mexpr[i].error( "for: [range expected]" );
            return null;
        }

    // create a new symbol table
    symt = new MxSymbolTable( symt, symt );

    MxInt[] x = new MxInt[mexpr.length];
    for ( int i=0; i<mexpr.length; i++ )
    {
        x[i] = new MxInt( ((MxRange)mexpr[i]).range.first() );
        x[i].setName( mexpr[i].name );
        symt.setValue( mexpr[i].name, x[i], false, 0 );
    }

    symt.setReadOnly();
    return x;
}

public boolean forCanProceed( MxDataType[] mexpr, MxInt[] values ) {
    if ( control != fc_none )
        return false;

    for ( int i=mexpr.length-1; ; i-- )
    {
        if ( ((MxRange)mexpr[i]).range.contain( values[i].var ) )
            return true;
        if ( 0 == i )
            return false;

        values[i].var = ((MxRange)mexpr[i]).range.first();
        values[i-1].var =
            ((MxRange)mexpr[i-1]).range.next( values[i-1].var );
    }
}

public void forNext( MxDataType[] mexpr, MxInt[] values ) {

    values[ values.length-1 ].var++;
    if ( control == fc_continue )
        tryResetFlowControl( mexpr[0].name );
}

public void forEnd( MxDataType[] mexpr ) {
    if ( control == fc_break )
        tryResetFlowControl( mexpr[0].name );
```

```
        // remove this symbol table
        symt = symt.dynamicParent();
    }

    public MxDataType execInternal( int id, MxDataType[] params ) {
        return MxInternalFunction.run( symt, id, params );
    }

    public void registerInternal() {
        MxInternalFunction.register( symt );
    }

    public MxDataType include( MxDataType file ) {
        if ( file instanceof MxString )
        {
            try
            {
                InputStream input =
                    (InputStream) new FileInputStream( ((MxString)file).var );

                MxAntlrLexer lexer = new MxAntlrLexer( input );
                MxAntlrParser parser = new MxAntlrParser( lexer );

                parser.program();
                if ( lexer.nr_error > 0 || parser.nr_error > 0 )
                    throw new MxException( "parsing erros" );
                CommonAST tree = (CommonAST)parser.getAST();
                MxAntlrWalker walker = new MxAntlrWalker();
                // share the symbol table
                walker.ipt.symt = this.symt;
                return walker.expr( tree );
            }
            catch ( Exception e )
            {
                throw new MxException( "include failed" );
            }
        }

        throw new MxException( "why parser let another data type go through?" );
    }
}
```

B.2.3 src/MxBitArray.java

```
import java.io.PrintWriter;
import jamaica.BitArray;

/**
 * the wrapper class for jamaica.BitArray
 *
 * @author Hanhua Feng - hf2048@columbia.edu
 * @version $Id: MxBitArray.java,v 1.12 2003/05/12 23:44:34 hanhua Exp $
 */
class MxBitArray extends MxDataType {
    BitArray ba;

    public MxBitArray( BitArray ba ) {
        this.ba = ba;
    }

    public String typename() {
        return "matmask";
    }

    public MxDataType copy() {
        return new MxBitArray( ba );
    }

    public void print( PrintWriter w ) {
        if ( name != null )
            w.print( name + " = " );
        w.print( ba.toString() );
        w.println();
    }

    public MxDataType and( MxDataType b ) {
        if ( b instanceof MxBitArray )
            return new MxBitArray( ba.and( ((MxBitArray)b).ba ) );
        return error( b, "and" );
    }

    public MxDataType or( MxDataType b ) {
        if ( b instanceof MxBitArray )
            return new MxBitArray( ba.or( ((MxBitArray)b).ba ) );
        return error( b, "or" );
    }

    public MxDataType not() {
        return new MxBitArray( ba.not() );
    }
}
```

B.2.4 src/MxBool.java

```
import java.io.PrintWriter;

/**
 * the wrapper class for boolean
 *
 *
 * @author Hanhua Feng - hf2048@columbia.edu
 * @version $Id: MxBool.java,v 1.9 2003/05/12 23:44:34 hanhua Exp $
 */
class MxBool extends MxDataType {
    boolean var;

    MxBool( boolean var ) {
        this.var = var;
    }

    public String typename() {
        return "bool";
    }

    public MxDataType copy() {
        return new MxBool( var );
    }

    public void print( PrintWriter w ) {
        if ( name != null )
            w.print( name + " = " );
        w.println( var ? "true" : "false" );
    }


    public MxDataType and( MxDataType b ) {
        if ( b instanceof MxBool )
            return new MxBool( var && ((MxBool)b).var );
        return error( b, "and" );
    }

    public MxDataType or( MxDataType b ) {
        if ( b instanceof MxBool )
            return new MxBool( var || ((MxBool)b).var );
        return error( b, "or" );
    }

    public MxDataType not() {
        return new MxBool( !var );
    }

    public MxDataType eq( MxDataType b ) {
        // not exclusive or
        if ( b instanceof MxBool )
            return new MxBool( ( var && ((MxBool)b).var )
                               || ( !var && !((MxBool)b).var ) );
```

```
        return error( b, "==" );
    }

    public MxDataType ne( MxDataType b ) {
        // exclusive or
        if ( b instanceof MxBool )
            return new MxBool( ( var && !((MxBool)b).var )
                               || ( !var && ((MxBool)b).var ) );
        return error( b, "!=" );
    }
}
```

B.2.5 src/MxDataType.java

```
import java.io.PrintWriter;

/**
 * The base data type class (also a meta class)
 *
 * Error messages are generated here.
 *
 * @author Hanhua Feng - hf2048@columbia.edu
 * @version $Id: MxDataType.java,v 1.17 2003/05/12 23:44:34 hanhua Exp $
 */
public class MxDataType
{
    String name;   // used in hash table

    public MxDataType() {
        name = null;
    }

    public MxDataType( String name ) {
        this.name = name;
    }

    public String typename() {
        return "unknown";
    }

    public MxDataType copy() {
        return new MxDataType();
    }

    public void setName( String name ) {
        this.name = name;
    }

    public MxDataType error( String msg ) {
        throw new MxException( "illegal operation: " + msg
                               + "( <" + typename() + "> "
                               + ( name != null ? name : "<?>" )
                               + " )" );
    }

    public MxDataType error( MxDataType b, String msg ) {
        if ( null == b )
            return error( msg );
        throw new MxException(
            "illegal operation: " + msg
            + "( <" + typename() + "> "
            + ( name != null ? name : "<?>" )
            + " and "
            + "<" + typename() + "> "
            + ( name != null ? name : "<?>" )
            + " )" );
```

```
}

public void print( PrintWriter w ) {
    if ( name != null )
        w.print( name + " = " );
    w.println( "<undefined>" );
}

public void print() {
    print( new PrintWriter( System.out, true ) );
}

public void what( PrintWriter w ) {
    w.print( "<" + typename() + ">  " );
    print( w );
}

public void what() {
    what( new PrintWriter( System.out, true ) );
}

public MxDataType assign( MxDataType b ) {
    return error( b, "=" );
}

public MxDataType transpose() {
    return error( "\'" );
}

public MxDataType uminus() {
    return error( "-" );
}

public MxDataType plus( MxDataType b ) {
    return error( b, "+" );
}

public MxDataType add( MxDataType b ) {
    return error( b, "+=" );
}

public MxDataType minus( MxDataType b ) {
    return error( b, "-" );
}

public MxDataType sub( MxDataType b ) {
    return error( b, "-=" );
}

public MxDataType times( MxDataType b ) {
    return error( b, "*" );
}
```

```
public MxDataType mul( MxDataType b ) {
    return error( b, "*=" );
}

public MxDataType lfracts( MxDataType b ) {
    return error( b, "/" );
}

public MxDataType rfracts( MxDataType b ) {
    return error( b, "/\'" );
}

public MxDataType ldiv( MxDataType b ) {
    return error( b, "/=" );
}

public MxDataType rdiv( MxDataType b ) {
    return error( b, "/\'=" );
}

public MxDataType modulus( MxDataType b ) {
    return error( b, "%" );
}

public MxDataType rem( MxDataType b ) {
    return error( b, "%=" );
}

public MxDataType gt( MxDataType b ) {
    return error( b, ">" );
}

public MxDataType ge( MxDataType b ) {
    return error( b, ">=" );
}

public MxDataType lt( MxDataType b ) {
    return error( b, "<" );
}

public MxDataType le( MxDataType b ) {
    return error( b, "<=" );
}

public MxDataType eq( MxDataType b ) {
    return error( b, "==" );
}

public MxDataType ne( MxDataType b ) {
    return error( b, "!=" );
}

public MxDataType and( MxDataType b ) {
```

```
        return error( b, "and" );
    }

    public MxDataType or( MxDataType b ) {
        return error( b, "or" );
    }

    public MxDataType not() {
        return error( "not" );
    }
}
```

B.2.6 src/MxDouble.java

```
import java.io.PrintWriter;

/**
 * The wrapper class for double
 *
 * @author Hanhua Feng - hf2048@columbia.edu
 * @version $Id: MxDouble.java,v 1.13 2003/05/12 23:44:34 hanhua Exp $
 */
class MxDouble extends MxDataType {
    double var;

    public MxDouble( double x ) {
        var = x;
    }

    public String typename() {
        return "double";
    }

    public MxDataType copy() {
        return new MxDouble( var );
    }

    public static double doubleValue( MxDataType b ) {
        if ( b instanceof MxDouble )
            return ((MxDouble)b).var;
        if ( b instanceof MxInt )
            return (double) ((MxInt)b).var;
        if ( b instanceof MxMatrix && ((MxMatrix)b).demoteable() )
            return ((MxMatrix)b).mat.get(0,0);
        b.error( "cast to double" );
        return 0;
    }

    public void print( PrintWriter w ) {
        if ( name != null )
            w.print( name + " = " );
        w.println( Double.toString( var ) );
    }

    public MxDataType uminus() {
        return new MxDouble( -var );
    }

    public MxDataType plus( MxDataType b ) {
        return new MxDouble( var + doubleValue(b) );
    }

    public MxDataType add( MxDataType b ) {
        var += doubleValue( b );
        return this;
    }
```

```
public MxDataType minus( MxDataType b ) {
    return new MxDouble( var - doubleValue(b) );
}

public MxDataType sub( MxDataType b ) {
    var -= doubleValue( b );
    return this;
}

public MxDataType times( MxDataType b ) {
    if ( b instanceof MxMatrix )
        return b.times( this );
    return new MxDouble( var * doubleValue(b) );
}

public MxDataType mul( MxDataType b ) {
    var *= doubleValue( b );
    return this;
}

public MxDataType lfracts( MxDataType b ) {
    if ( b instanceof MxMatrix && !((MxMatrix)b).demoteable() )
    {
        return new MxMatrix(
            ((MxMatrix)b).mat.invert().selfmul( 1.0 / var ) );
    }

    return new MxDouble( var / doubleValue(b) );
}

public MxDataType rfracts( MxDataType b ) {
    return lfracts( b );
}

public MxDataType ldiv( MxDataType b ) {
    var /= doubleValue(b);
    return this;
}

public MxDataType rdiv( MxDataType b ) {
    return ldiv( b );
}

public MxDataType modulus( MxDataType b ) {
    return new MxDouble( var % doubleValue(b) );
}


public MxDataType rem( MxDataType b ) {
    var %= doubleValue( b );
    return this;
}
```

```
    public MxDataType gt( MxDataType b ) {
        if ( b instanceof MxMatrix && !((MxMatrix)b).demoteable() )
            return b.gt( this );
        return new MxBool( var > doubleValue(b) );
    }

    public MxDataType ge( MxDataType b ) {
        if ( b instanceof MxMatrix && !((MxMatrix)b).demoteable() )
            return b.ge( this );
        return new MxBool( var >= doubleValue(b) );
    }

    public MxDataType lt( MxDataType b ) {
        if ( b instanceof MxMatrix && !((MxMatrix)b).demoteable() )
            return b.lt( this );
        return new MxBool( var < doubleValue(b) );
    }

    public MxDataType le( MxDataType b ) {
        if ( b instanceof MxMatrix && !((MxMatrix)b).demoteable() )
            return b.le( this );
        return new MxBool( var <= doubleValue(b) );
    }

    public MxDataType eq( MxDataType b ) {
        if ( b instanceof MxMatrix && !((MxMatrix)b).demoteable() )
            return b.eq( this );
        return new MxBool( var == doubleValue(b) );
    }

    public MxDataType ne( MxDataType b ) {
        if ( b instanceof MxMatrix && !((MxMatrix)b).demoteable() )
            return b.ne( this );
        return new MxBool( var != doubleValue(b) );
    }
}
```

B.2.7 src/MxException.java

```
/**
 * Exception class: messages are generated in various classes
 *
 * Can we do better?
 *
 * @author Hanhua Feng - hf2048@columbia.edu
 * @version $Id: MxException.java,v 1.4 2003/05/12 23:44:34 hanhua Exp $
 */
class MxException extends RuntimeException {
    MxException( String msg ) {
        System.err.println( "Error: " + msg );
    }
}
```

B.2.8 src/MxFunction.java

```
import java.io.PrintWriter;
import antlr.collections.AST;

/**
 * The function data type (including internal functions)
 *
 * @author Hanhua Feng - hf2048@columbia.edu
 * @version $Id: MxFunction.java,v 1.9 2003/05/12 23:44:34 hanhua Exp $
 */
class MxFunction extends MxDataType {
    // we need a reference to the AST for the function entry
    String[] args;
    AST body;            // body = null means an internal function.
    MxSymbolTable pst;   // the symbol table of static parent
    int id;              // for internal functions only

    public MxFunction( String name, String[] args,
                       AST body, MxSymbolTable pst) {
        super( name );
        this.args = args;
        this.body = body;
        this.pst = pst;
    }

    public MxFunction( String name, int id ) {
        super( name );
        this.args = null;
        this.id = id;
        pst = null;
        body = null;
    }

    public final boolean isInternal() {
        return body == null;
    }

    public final int getInternalId() {
        return id;
    }

    public String typename() {
        return "function";
    }

    public MxDataType copy() {
        return new MxFunction( name, args, body, pst );
    }

    public void print( PrintWriter w ) {
        if ( body == null )
        {
            w.println( name + " = <internal-function> #" + id );
```

```
        }
        else
        {
            if ( name != null )
                w.print( name + " = " );
            w.print( "<function>(" );
            for ( int i=0; ; i++ )
            {
                w.print( args[i] );
                if ( i >= args.length - 1 )
                    break;
                w.print( "," );
            }
            w.println( ")" );
        }
    }

    public String[] getArgs() {
        return args;
    }

    public MxSymbolTable getParentSymbolTable() {
        return pst;
    }

    public AST getBody() {
        return body;
    }
}
```

B.2.9 src/MxInt.java

```
import java.io.PrintWriter;

/**
 * the wrapper class of int
 *
 * @author Hanhua Feng - hf2048@columbia.edu
 * @version $Id: MxInt.java,v 1.13 2003/05/12 23:44:34 hanhua Exp $
 */
class MxInt extends MxDataType {
    int var;

    public MxInt( int x ) {
        var = x;
    }

    public String typename() {
        return "int";
    }

    public MxDataType copy() {
        return new MxInt( var );
    }

    public static int intValue( MxDataType b ) {
        if ( b instanceof MxDouble )
            return (int) ((MxDouble)b).var;
        if ( b instanceof MxInt )
            return ((MxInt)b).var;
        if ( b instanceof MxMatrix && ((MxMatrix)b).demoteable() )
            return (int) ((MxMatrix)b).mat.get(0,0);
        b.error( "cast to int" );
        return 0;
    }

    public void print( PrintWriter w ) {
        if ( name != null )
            w.print( name + " = " );
        w.println( Integer.toString( var ) );
    }

    public MxDataType uminus() {
        return new MxInt( -var );
    }

    public MxDataType plus( MxDataType b ) {
        if ( b instanceof MxInt )
            return new MxInt( var + intValue(b) );
        return new MxDouble( var + MxDouble.doubleValue(b) );
    }

    public MxDataType add( MxDataType b ) {
        var += intValue( b );
```

```
    return this;
}

public MxDataType minus( MxDataType b ) {
    if ( b instanceof MxInt )
        return new MxInt( var - intValue(b) );
    return new MxDouble( var - MxDouble.doubleValue(b) );
}

public MxDataType sub( MxDataType b ) {
    var -= intValue( b );
    return this;
}

public MxDataType times( MxDataType b ) {
    if ( b instanceof MxMatrix )
        return b.times( this );
    if ( b instanceof MxInt )
        return new MxInt( var * intValue(b) );
    return new MxDouble( var * MxDouble.doubleValue(b) );
}

public MxDataType mul( MxDataType b ) {
    var *= intValue( b );
    return this;
}

public MxDataType lfracts( MxDataType b ) {
    if ( b instanceof MxMatrix && !((MxMatrix)b).demoteable() )
    {
        return new MxMatrix(
            ((MxMatrix)b).mat.invert().selfmul( 1.0 / (double)var ) );
    }

    if ( b instanceof MxInt )
        return new MxInt( var / intValue(b) );
    return new MxDouble( var / MxDouble.doubleValue(b) );
}

public MxDataType rfracts( MxDataType b ) {
    return lfracts( b );
}

public MxDataType ldiv( MxDataType b ) {
    var /= intValue(b);
    return this;
}

public MxDataType rdiv( MxDataType b ) {
    return ldiv( b );
}

public MxDataType modulus( MxDataType b ) {
```

```
        if ( b instanceof MxInt )
            return new MxInt( var % intValue(b) );
        return new MxDouble( var % MxDouble.doubleValue(b) );
    }


    public MxDataType rem( MxDataType b ) {
        var %= intValue( b );
        return this;
    }

    public MxDataType gt( MxDataType b ) {
        if ( b instanceof MxInt )
            return new MxBool( var > intValue(b) );
        return b.lt( this );
    }

    public MxDataType ge( MxDataType b ) {
        if ( b instanceof MxInt )
            return new MxBool( var >= intValue(b) );
        return b.le( this );
    }

    public MxDataType lt( MxDataType b ) {
        if ( b instanceof MxInt )
            return new MxBool( var < intValue(b) );
        return b.gt( this );
    }

    public MxDataType le( MxDataType b ) {
        if ( b instanceof MxInt )
            return new MxBool( var <= intValue(b) );
        return b.ge( this );
    }

    public MxDataType eq( MxDataType b ) {
        if ( b instanceof MxInt )
            return new MxBool( var == intValue(b) );
        return b.eq( this );
    }

    public MxDataType ne( MxDataType b ) {
        if ( b instanceof MxInt )
            return new MxBool( var != intValue(b) );
        return b.ne( this );
    }
}
```

B.2.10 src/MxMatrix.java

```
import java.io.PrintWriter;
import jamaica.*;

/**
 * the wrapper class for jamaica.Matrix
 *
 * @author Hanhua Feng - hf2048@columbia.edu
 * @version $Id: MxMatrix.java,v 1.17 2003/05/12 23:44:34 hanhua Exp $
 */
class MxMatrix extends MxDataType {
    Matrix mat;
    BitArray mask;

    MxMatrix( Matrix mat ) {
        this.mat = mat;
        mask = null;
    }

    MxMatrix( Matrix mat, BitArray mask ) {
        this.mat = mat;
        this.mask = mask;
    }

    public String typename() {
        return "matrix";
    }

    public MxDataType copy() {
        return new MxMatrix( mat, mask );
    }

    public MxDataType deepCopy() {
        return new MxMatrix( mat.copy(), mask );
    }

    public void print( PrintWriter w ) {
        if ( name != null )
            w.println( name + " = " );
        mat.print( w, 8, 4, 4 );
        if ( mask != null )
        {
            w.print( "  <mask> = " );
            w.println( mask.toString() );
        }
    }

    public void what( PrintWriter w ) {
        w.print( "<" + typename() + ">  " );
        if ( name != null )
            w.print( name + "  " );
        if ( mask != null )
            w.print( " (masked) " );
```

```
    w.println( " " + mat.height() + "*" + mat.width() );
}

final boolean demoteable() {
    return mat.width() == 1 && mat.height() == 1 && mask == null;
}

final MxDouble demote() {
    return new MxDouble( mat.get( 0, 0 ) );
}

public MxDataType transpose() {
    return new MxMatrix( mat.transpose() );
}

public MxDataType uminus() {
    return new MxMatrix( mat.uminus() );
}

public MxDataType plus( MxDataType b ) {
    if ( demoteable() )
        return demote().plus( b );

    if ( b instanceof MxMatrix )
        return new MxMatrix( mat.plus( ((MxMatrix)b).mat ) );

    return error( b, "+" );
}

public MxDataType add( MxDataType b ){
    if ( b instanceof MxMatrix )
    {
        if ( null == mask )
            mat.selfadd( ((MxMatrix)b).mat );
        else
            mat.assign( mask, mat.plus( ((MxMatrix)b).mat ) );
        return this;
    }
    else if ( demoteable() )
    {
        mat.set( 0, 0, mat.get(0,0) + MxDouble.doubleValue(b) );
        return this;
    }

    return error( b, "+=" );
}

public MxDataType minus( MxDataType b ) {
    if ( demoteable() )
        return demote().minus( b );

    if ( b instanceof MxMatrix )
        return new MxMatrix( mat.minus( ((MxMatrix)b).mat ) );
```

```
    return error( b, "-" );
}

public MxDataType sub( MxDataType b ) {
    if ( b instanceof MxMatrix )
    {
        if ( null == mask )
            mat.selfsub( ((MxMatrix)b).mat );
        else
            mat.assign( mask, mat.minus( ((MxMatrix)b).mat ) );
        return this;
    }
    else if ( demoteable() )
    {
        mat.set( 0, 0, mat.get(0,0) - MxDouble.doubleValue(b) );
        return this;
    }

    return error( b, "-=" );
}

public MxDataType times( MxDataType b ) {
    if ( demoteable() )
        return demote().times( b );

    if ( b instanceof MxMatrix && !((MxMatrix)b).demoteable() )
        return new MxMatrix( mat.times( ((MxMatrix)b).mat ) );

    return new MxMatrix( mat.times( MxDouble.doubleValue( b ) ) );
}

public MxDataType mul( MxDataType b ){
    if ( b instanceof MxMatrix && !((MxMatrix)b).demoteable() )
    {
        if ( null == mask )
            mat.selfrmul( ((MxMatrix)b).mat );
        else
            mat.assign( mask, mat.times( ((MxMatrix)b).mat ) );
        return this;
    }

    mat.selfmul( MxDouble.doubleValue( b ) );
    return this;
}

public MxDataType lfracts( MxDataType b ) {
    if ( demoteable() )
        return demote().lfracts( b );

    if ( b instanceof MxMatrix && !((MxMatrix)b).demoteable() )
        return new MxMatrix( mat.ldivide( ((MxMatrix)b).mat ) );
```

```
    return new MxMatrix( mat.times( 1.0 / MxDouble.doubleValue( b ) ) );
}

public MxDataType rfracts( MxDataType b ) {
    if ( demoteable() )
        return demote().rfracts( b );

    if ( b instanceof MxMatrix && !((MxMatrix)b).demoteable() )
        return new MxMatrix( mat.rdivide( ((MxMatrix)b).mat ) );

    return new MxMatrix( mat.times( 1.0 / MxDouble.doubleValue( b ) ) );
}

public MxDataType ldiv( MxDataType b ) {
    if ( b instanceof MxMatrix && !((MxMatrix)b).demoteable() )
    {
        if ( null == mask )
            mat.selfldiv( ((MxMatrix)b).mat );
        else
            mat.assign( mask, mat.ldivide( ((MxMatrix)b).mat ) );
        return this;
    }

    mat.selfmul( 1.0 / MxDouble.doubleValue( b ) );
    return this;
}

public MxDataType rdiv( MxDataType b ) {
    if ( b instanceof MxMatrix && !((MxMatrix)b).demoteable() )
    {
        if ( null == mask )
            mat.selfrdiv( ((MxMatrix)b).mat );
        else
            mat.assign( mask, mat.rdivide( ((MxMatrix)b).mat ) );
        return this;
    }

    mat.selfmul( 1.0 / MxDouble.doubleValue( b ) );
    return this;
}

public MxDataType modulus( MxDataType b ) {
    if ( demoteable() )
        return demote().modulus( b );
    return error( b, "%" );
}

public MxDataType rem( MxDataType b ){
    if ( demoteable() )
    {
        mat.set( 0, 0, mat.get( 0, 0 ) % MxDouble.doubleValue( b ) );
        return this;
    }
```

```
    return error( b, "%=" );
}

public MxDataType gt( MxDataType b ) {
    if ( demoteable() )
        return b.lt( demote() );

    if ( b instanceof MxMatrix && !((MxMatrix)b).demoteable() )
        return new MxBitArray( mat.gt( ((MxMatrix)b).mat ) );

    return new MxBitArray( mat.gt( MxDouble.doubleValue( b ) ) );
}

public MxDataType ge( MxDataType b ) {
    if ( demoteable() )
        return b.le( demote() );

    if ( b instanceof MxMatrix && !((MxMatrix)b).demoteable() )
        return new MxBitArray( mat.ge( ((MxMatrix)b).mat ) );

    return new MxBitArray( mat.ge( MxDouble.doubleValue( b ) ) );
}

public MxDataType lt( MxDataType b ) {
    if ( demoteable() )
        return b.gt( demote() );

    if ( b instanceof MxMatrix && !((MxMatrix)b).demoteable() )
        return new MxBitArray( mat.lt( ((MxMatrix)b).mat ) );

    return new MxBitArray( mat.lt( MxDouble.doubleValue( b ) ) );
}

public MxDataType le( MxDataType b ) {
    if ( demoteable() )
        return b.ge( demote() );

    if ( b instanceof MxMatrix && !((MxMatrix)b).demoteable() )
        return new MxBitArray( mat.le( ((MxMatrix)b).mat ) );

    return new MxBitArray( mat.le( MxDouble.doubleValue( b ) ) );
}

public MxDataType eq( MxDataType b ) {
    if ( demoteable() )
        return b.eq( demote() );

    if ( b instanceof MxMatrix && !((MxMatrix)b).demoteable() )
        return new MxBitArray( mat.eq( ((MxMatrix)b).mat ) );

    return new MxBitArray( mat.eq( MxDouble.doubleValue( b ) ) );
}
```

```
public MxDataType ne( MxDataType b ) {
    if ( demoteable() )
        return b.ne( demote() );

    if ( b instanceof MxMatrix && !((MxMatrix)b).demoteable() )
        return new MxBitArray( mat.ne( ((MxMatrix)b).mat ) );

    return new MxBitArray( mat.ne( MxDouble.doubleValue( b ) ) );
}

private static final int getDim( MxDataType x, boolean rows ) {
    if ( x instanceof MxDouble || x instanceof MxInt )
        return 1;
    if ( x instanceof MxMatrix )
        return (rows ? ((MxMatrix)x).mat.height()
                : ((MxMatrix)x).mat.width());
    x.error( "array [none number/array element]" );
    return 0;
}

public static MxDataType joinVert( MxDataType [] x ) {
    if ( x.length == 0 )
        throw new IllegalArgumentException( "No data in array " );
    int nrow = getDim( x[0], true );
    int ncol = getDim( x[0], false );
    for ( int i=1; i<x.length; i++ )
    {
        if ( getDim( x[i], false ) != ncol )
            return x[i].error( "wrong width" );
        nrow += getDim( x[i], true );
    }

    MxMatrix y = new MxMatrix( new Matrix( nrow, ncol ) );

    int rowp = 0;
    for ( int i=0; i<x.length; i++ )
    {
        if ( x[i] instanceof MxMatrix )
        {
            Matrix b = ((MxMatrix)x[i]).mat;
            int m = b.height();
            y.mat.slice( new Range( rowp, m, 1 ),
                         new Range( 0, ncol, 1 ) ).assign( b );
            rowp += m;
        }
        else
        {
            y.mat.set( rowp, 0, MxDouble.doubleValue( x[i] ) );
            rowp++;
        }
    }

    return y;
```

```
    }

    public static MxDataType joinHori( MxDataType [] x ) {
        if ( x.length == 0 )
            throw new IllegalArgumentException( "no data in array" );
        int nrow = getDim( x[0], true );
        int ncol = getDim( x[0], false );
        for ( int i=1; i<x.length; i++ )
        {
            if ( getDim( x[i], true ) != nrow )
                return x[i].error( "wrong height" );
            ncol += getDim( x[i], false );
        }

        MxMatrix y = new MxMatrix( new Matrix( nrow, ncol ) );

        int colp = 0;
        for ( int i=0; i<x.length; i++ )
        {
            if ( x[i] instanceof MxMatrix )
            {
                Matrix b = ((MxMatrix)x[i]).mat;
                int n = b.width();
                y.mat.slice( new Range( 0, nrow, 1 ),
                             new Range( colp, n, 1 ) ).assign( b );
                colp += n;
            }
            else
            {
                y.mat.set( 0, colp, MxDouble.doubleValue( x[i] ) );
                colp++;
            }
        }

        return y;
    }
}
```

B.2.11 src/MxRange.java

```
import java.io.PrintWriter;
import jamaica.Range;

/**
 * The wrapper class for jamaica.Range
 *
 * @author Hanhua Feng - hf2048@columbia.edu
 * @version $Id: MxRange.java,v 1.13 2003/05/12 23:44:34 hanhua Exp $
 */
class MxRange extends MxDataType {
    Range range;

    MxRange( Range range ) {
        this.range = range;
    }

    public String typename() {
        return "range";
    }

    public MxDataType copy() {
        return new MxRange( range );
    }

    public static MxDataType create( MxDataType v1, MxDataType v2 ) {
        int s = ( null == v1 ? 0 : MxInt.intValue(v1) );

        if ( null != v2 )
        {
            int t = MxInt.intValue(v2);
            if ( t >= 0 )
                return new MxRange( new Range( s, t ) );
            return new MxRange( new Range( s, t, 1 ) );
        }

        return new MxRange( new Range( s, -1, 1 ) );
    }

    public static MxDataType create( MxDataType v1, MxDataType v2, int stride ) {
        if ( v1 instanceof MxInt && v2 instanceof MxInt ) {
            if ( ((MxInt)v2).var <= 0 )
                return v2.error( "invalid range" );
            return new MxRange(
                new Range( ((MxInt)v1).var, ((MxInt)v2).var, stride ) );
        }

        return v1.error( v2, "cast to range components" );
    }

    public Range getRange( int n ) {
        /* Tricky short-cut: if range is empty, the "number" field stores
         * a reverse count: -1 means n-1, -2 means n-2, and so on.
```

```
         */
        if ( range.empty() )
            return new Range( range.first(), n + range.size() );

        return range;
    }

    public void print( PrintWriter w ) {
        if ( name != null )
            w.print( name + " = " );
        w.println( Integer.toString( range.first() ) + ":"
                   + range.size() + ":" + range.interval() );
    }

}
```

B.2.12 src/MxString.java

```
import java.io.PrintWriter;

/**
 * the wrapper class for string
 *
 * @author Hanhua Feng - hf2048@columbia.edu
 * @version $Id: MxString.java,v 1.6 2003/05/12 23:44:34 hanhua Exp $
 */
class MxString extends MxDataType {
    String var;

    public MxString( String str ) {
        this.var = str;
    }

    public String typename() {
        return "string";
    }

    public MxDataType copy() {
        return new MxString( var );
    }

    public void print( PrintWriter w ) {
        if ( name != null )
            w.print( name + " = " );
        w.print( var );
        w.println();
    }

    public MxDataType plus( MxDataType b ) {
        if ( b instanceof MxString )
            return new MxString( var + ((MxString)b).var );

        return error( b, "+" );
    }

    public MxDataType add( MxDataType b ) {
        if ( b instanceof MxString )
        {
            var = var + ((MxString)b).var;
            return this;
        }

        return error( b, "+=" );
    }
}
```

B.2.13 src/MxSymbolTable.java

```
import java.util.*;
import java.io.PrintWriter;

/**
 * Symbol table class: dual parent supported: static and dynamic
 *
 * @author Hanhua Feng - hf2048@columbia.edu
 * @version $Id: MxSymbolTable.java,v 1.8 2003/05/12 23:44:34 hanhua Exp $
 */
class MxSymbolTable extends HashMap {
    MxSymbolTable static_parent, dynamic_parent;
    boolean read_only;

    public MxSymbolTable( MxSymbolTable sparent, MxSymbolTable dparent ) {
        static_parent = sparent;
        dynamic_parent = dparent;
        read_only = false;
    }

    public void setReadOnly() {
        read_only = true;
    }

    public final MxSymbolTable staticParent() {
        return static_parent;
    }

    public final MxSymbolTable dynamicParent() {
        return dynamic_parent;
    }

    public final MxSymbolTable parent( boolean is_static ) {
        return is_static ? static_parent : dynamic_parent;
    }

    public final boolean containsVar( String name ) {
        return containsKey( name );
    }

    private final MxSymbolTable gotoLevel( int level, boolean is_static ) {
        MxSymbolTable st = this;

        if ( level < 0 )
        {
            // global variable
            while ( null != st.static_parent )
                st = st.parent( is_static );
        }
        else
        {
            // local variable
            for ( int i=level; i>0; i-- )
```

```
        {
            while ( st.read_only )
            {
                st = st.parent( is_static );
                assert st != null;
            }

            if ( null != st.parent( is_static ) )
                st = st.parent( is_static );
            else
                break;
        }
    }

    return st;
}

public final MxDataType getValue( String name, boolean is_static,
                                  int level ) {
    MxSymbolTable st = gotoLevel( level, is_static );
    Object x = st.get( name );

    while ( null == x && null != st.parent( is_static ) )
    {
        st = st.parent( is_static );
        x = st.get( name );
    }

    return (MxDataType) x;
}

public final void setValue( String name, MxDataType data,
                            boolean is_static, int level ) {

    MxSymbolTable st = gotoLevel( level, is_static );
    while ( st.read_only )
    {
        st = st.parent( is_static );
        assert st != null;
    }

    st.put( name, data );
}

public void what( PrintWriter output ) {
    for ( Iterator it = values().iterator() ; it.hasNext(); )
    {
        MxDataType d = ((MxDataType)(it.next()));
        if ( !( d instanceof MxFunction && ((MxFunction)d).isInternal() ) )
            d.what( output );
    }
}
```

```
    public void what() {
        what( new PrintWriter( System.out, true ) );
    }
}
```

B.2.14 src/MxVariable.java

```
import java.io.PrintWriter;

/**
 * The wrapper class for unsigned variables
 *
 * @author Hanhua Feng - hf2048@columbia.edu
 * @version $Id: MxVariable.java,v 1.3 2003/05/12 23:44:34 hanhua Exp $
 */
class MxVariable extends MxDataType {
    public MxVariable( String name ) {
        super( name );
    }

    public String typename() {
        return "undefined-variable";
    }

    public MxDataType copy() {
        throw new MxException( "Variable " + name + " has not been defined" );
    }

    public void print( PrintWriter w ) {
        w.println( name + " = <undefined>" );
    }
}
```

B.2.15 src/MxInternalFunction.m4

```
divert(-1)
define(FOREACH,‘pushdef(‘$1’, ‘’)pushdef(‘$3’, ‘$4’)dnl
_foreach(‘$1’, ‘$2’, ‘$3’, ‘$4’, ‘$5’)popdef(‘$3’)popdef(‘$1’)’)
define(_arg1,‘$1’)
define(_foreach,
   ‘ifelse(‘$2’, ‘()’, ,
‘define(‘$1’, _arg1$2)$5‘’define(‘$3’,incr($3))dnl
_foreach(‘$1’, (shift$2), ‘$3’, ‘$4’, ‘$5’)’)’)

define(EXCEPTION,‘throw new MxException( "$1() accepts $2 parameter(s)" );’)

define(CHECK_NR_PARAM,‘if ( params.length != $2 )
                EXCEPTION( $1, $2 );’)

define(CASE_MATRIXOP1,‘case f_$1:
            CHECK_NR_PARAM( inv, 1 )
            if ( ! (params[0] instanceof MxMatrix ) )
                return params[0].error( "$1" );
            return new $3( ((MxMatrix)params[0]).mat.$2() );
’)

define(CASE_MINMAX,‘case f_$1:
            if ( params.length > 0 )
            {
                if ( isIntList( params ) )
                {
                    int x = MxInt.intValue( params[0] );
                    for ( int i=1; i<params.length; i++ )
                    {
                        int y = MxInt.intValue( params[i] );
                        if ( y $2 x )
                            x = y;
                    }
                    return new MxInt( x );
                }

                double x = ( (params[0] instanceof MxMatrix)?
                             ((MxMatrix)params[0]).mat.$1()
                             : MxDouble.doubleValue( params[0] ) );
                for ( int i=1; i<params.length; i++ )
                {
                    double y = ( (params[i] instanceof MxMatrix)?
                                 ((MxMatrix)params[i]).mat.$1()
                                 : MxDouble.doubleValue( params[i] ) );
                    if ( y $2 x )
                        x = y;
                }
                return new MxDouble( x );
            }
            EXCEPTION( $1, 1 )
’)
```

```
define(CASE_VECOP1,‘case f_$1:
            CHECK_NR_PARAM( $1, 1 )
            if ( params[0] instanceof MxMatrix )
            {
                Matrix p = ((MxMatrix)params[0]).mat;
                int m = p.height(), n = p.width();
                Matrix x = new Matrix( m, n );
                for ( int j=0; j<n; j++ )
                    for ( int i=0; i<m; i++ )
                        x.set( i, j, $2( p.get(i,j) ) );
                return new MxMatrix( x );
            }
            return new MxDouble(
                $2( MxDouble.doubleValue( params[0] ) ) );
’)

define(CASE_DIM,‘case f_$1:
            CHECK_NR_PARAM( $1, 1 )
            if ( params[0] instanceof MxMatrix )
                return new MxInt( ((MxMatrix)params[0]).mat.$2() );
            return new MxInt( 0 );
’)

define(CASE_MULDIV,‘case f_$1:
            CHECK_NR_PARAM( $1, 2 )
            if ( params[0] instanceof MxMatrix
                 && params[1] instanceof MxMatrix )
            {
                return new MxMatrix(
                    ((MxMatrix)params[0]).mat.$2(
                        ((MxMatrix)params[1]).mat ) );
            }
            throw new MxException( "$1: arguments should be matricies" );
’)

define(FUNCTIONS,‘( print, what, plot, oplot, paint, opaint,
                    colormap, color,
                    load, save, width, height,
                    random, zeros, eye, indgen,
                    inv, det, flip, mirror,
                    min, max,
                    mul, div,
                    count,
                    round, floor, ceil, int, abs,
                    sqrt, exp, log, pow, sin, cos, tan, asin, acos, atan )’)
divert

import java.util.Random;
import java.io.IOException;
import jamaica.*;

/** The internal functions: MxInternalFunction.java is generated by
 * MxInternalFunction.m4
```

```
 *
 * @author Hanhua Feng - hf2048@columbia.edu
 * @version $Id: MxInternalFunction.m4,v 1.13 2003/05/13 23:25:22 hanhua Exp $
 */
class MxInternalFunction {
    static Random random = new Random();
    static int[] rgbtable = Painter.colorMap( Painter.CM_GRAYSCALE );
    static int defcolor = 0;
    static Painter painter = null;
    static Plotter plotter = null;

    /* if there is no rgbtable, generate an length-1 table from defcolor */
    private static int[] getRGBTable() {
        if ( null != rgbtable )
            return rgbtable;
        return new int[]{ defcolor };
    }

FOREACH( NAME, FUNCTIONS, INDEX, 0, `
    final static int f_``NAME = INDEX;define(INDEX,incr(INDEX))')

    public static void register( MxSymbolTable st ) {
FOREACH( NAME, FUNCTIONS, INDEX, `0', `
             st.put( "NAME", new MxFunction( null, f_``NAME ) );' )
        st.put( "E", new MxDouble( Math.E ) );
        st.put( "PI", new MxDouble( Math.PI ) );
    }

    private static boolean isIntList( MxDataType[] params ){
        for ( int i=0; i<params.length; i++ )
            if ( ! ( params[i] instanceof MxInt ) )
                return false;
        return true;
    }

    public static MxDataType run( MxSymbolTable st,
                                  int id, MxDataType[] params ) {
        switch ( id )
        {
        case f_print:
            for ( int i=0; i<params.length; i++ )
                params[i].print();
            return null;

        case f_what:
            if ( params.length > 0 )
            {
                for ( int i=0; i<params.length; i++ )
                        params[i].what();
            }
            else
                st.what();
            return null;
```

```
case f_paint:
    if ( params.length != 1 && params.length != 3 )
        EXCEPTION( paint, 1 or 3 )
    if ( !( params[0] instanceof MxMatrix ) )
        throw new MxException( "Only matrix can be drawn" );
    painter = Painter.create( "Mx Painter", 100, 100 );
    if ( 1 == params.length )
        painter.draw( ((MxMatrix)params[0]).mat, getRGBTable() );
    else
        painter.draw( ((MxMatrix)params[0]).mat,
            MxDouble.doubleValue( params[1] ),
            MxDouble.doubleValue( params[2] ), getRGBTable() );
    return params[0];

case f_opaint:
    if ( null == painter )
        throw new MxException( "No previous painting window" );
    if ( 0 == params.length )
        EXCEPTION( opaint, 1 or more )
    if ( !( params[0] instanceof MxMatrix ) )
        throw new MxException( "Only matrix can be drawn" );
    BitArray bmk = null;
    double min = 0.0, max = 0.0;
    int xpos = 0, ypos = 0;
    int offset = 1;
    if ( offset < params.length
         && ( params[offset] instanceof MxBitArray ) )
        bmk = ((MxBitArray)params[offset++]).ba;

    if ( offset < params.length-1 )
    {
        xpos = MxInt.intValue( params[offset++] );
        ypos = MxInt.intValue( params[offset++] );
    }
    if ( offset < params.length-1 )
    {
        min = MxDouble.doubleValue( params[offset++] );
        max = MxDouble.doubleValue( params[offset++] );
    }
    if ( offset < params.length )
        throw new MxException( "Additional arguments" );
    if ( max > min )
        painter.overdraw( ((MxMatrix)params[0]).mat, bmk, min, max,
                          getRGBTable(), xpos, ypos );
    else
        painter.overdraw( ((MxMatrix)params[0]).mat, bmk,
                         getRGBTable(), xpos, ypos );
    return params[0];

case f_plot:
    if ( 1 != params.length && 2 != params.length )
        EXCEPTION( plot, 1-2 );
```

```
    if ( !( params[0] instanceof MxMatrix ) )
        throw new MxException( "First arg should be a matrix" );
    plotter = Plotter.create( "Mx Plotter", 640, 480 );
    if ( 2 == params.length )
    {
        if ( !( params[1] instanceof MxMatrix ) )
            throw new MxException( "Second arg should be a matrix" );
        Matrix mat = ((MxMatrix)params[1]).mat;
        double[] minmax = new double[6];
        minmax[0] = mat.get(0,0);
        minmax[1] = mat.get(1,0);
        minmax[2] = mat.get(0,1);
        minmax[3] = mat.get(1,1);
        if ( mat.width() >= 3 )
        {
            minmax[4] = mat.get(0,2);
            minmax[5] = mat.get(1,2);
        }
        plotter.draw( ((MxMatrix)params[0]).mat,
                      minmax, defcolor, rgbtable );
    }
    else
        plotter.draw( ((MxMatrix)params[0]).mat, defcolor, rgbtable );
    return params[0];

case f_color:
    CHECK_NR_PARAM( colors, 3 )
    defcolor = Painter.color( MxDouble.doubleValue( params[0] ),
                              MxDouble.doubleValue( params[1] ),
                              MxDouble.doubleValue( params[2] ) );
    return null;

case f_colormap:
    if ( 0 == params.length )
    {
        rgbtable = null;
        return null;
    }

    CHECK_NR_PARAM( colors, 1 )
    if ( params[0] instanceof MxMatrix )
        rgbtable = Painter.colorMap( ((MxMatrix)params[0]).mat );
    else
        rgbtable = Painter.colorMap( MxInt.intValue( params[0] ) );
    return null;

case f_load:
    if ( 4 != params.length && 5 != params.length )
        EXCEPTION( load, 4-5 )
    if ( ! (params[0] instanceof MxString
        || params[1] instanceof MxString ) )
        throw new MxException( "First two args should be strings" );
```

```
    Matrix mat;
    try
    {
        if ( params.length == 4 )
            mat = Matrix.load(
                ((MxString)params[0]).var,
                MxInterpreter.getDataType(((MxString)params[1]).var),
                MxInt.intValue( params[2] ),
                MxInt.intValue( params[3] ) );
        else
            mat = Matrix.load( ((MxString)params[0]).var,
                MxInterpreter.getDataType(((MxString)params[1]).var),
                MxInt.intValue( params[2] ),
                MxInt.intValue( params[3] ),
                MxInt.intValue( params[4] ) );
    }
    catch ( IOException e )
    {
        throw new MxException( "I/O Exception:" + e );
    }

    if ( null == mat )
        throw new MxException( "File not found: "
                               + ((MxString)params[0]).var );
    return new MxMatrix( mat );

case f_save:
    if ( 3 != params.length && 4 != params.length )
        EXCEPTION( save, 3-4 )
    if ( ! (params[0] instanceof MxMatrix) )
        throw new MxException( "Arg 1 should be a matrix" );
    if ( ! (params[1] instanceof MxString
        || params[2] instanceof MxString ) )
        throw new MxException( "Args 2 and 3 should be strings" );
    try
    {
        if ( params.length == 3 )
            ((MxMatrix)params[0]).mat.save(
                ((MxString)params[1]).var,
                MxInterpreter.getDataType(((MxString)params[2]).var) );
        else
            ((MxMatrix)params[0]).mat.save(
                ((MxString)params[1]).var,
                MxInterpreter.getDataType( ((MxString)params[2]).var),
                MxInt.intValue( params[3] ) );
    }
    catch ( IOException e )
    {
        throw new MxException( "I/O Exception:" + e );
    }
    return params[0];

CASE_DIM( width, width )
```

```
CASE_DIM( height, height )

case f_random:
    if ( 0 == params.length )
        return new MxDouble( random.nextDouble() );
    if ( 1 == params.length )
    {
        if ( params[0] instanceof MxDouble )
            return new MxDouble( random.nextDouble()
                                 * MxDouble.doubleValue( params[0] ) );

        if ( params[0] instanceof MxInt )
            return new MxInt( random.nextInt(
                                  MxInt.intValue( params[0] ) ) );
    }
    else if ( 2 == params.length )
        return new MxMatrix( Matrix.random(
                                 MxInt.intValue( params[0] ),
                                 MxInt.intValue( params[1] ) ) );
    EXCEPTION( random, 0-2 )

case f_zeros:
    CHECK_NR_PARAM( zeros, 2 )
    return new MxMatrix(
        new Matrix( MxInt.intValue( params[0] ),
                    MxInt.intValue( params[1] ) ) );

case f_eye:
    CHECK_NR_PARAM( eye, 1 )
    return new MxMatrix( Matrix.eye( MxInt.intValue( params[0] ) ) );

case f_indgen:
    CHECK_NR_PARAM( indgen, 5 );
    return new MxMatrix( Matrix.indgen(
        MxInt.intValue( params[0] ),
        MxInt.intValue( params[1] ),
        MxDouble.doubleValue( params[2] ),
        MxDouble.doubleValue( params[3] ),
        MxDouble.doubleValue( params[4] ) ) );

CASE_MATRIXOP1( inv, invert, MxMatrix )
CASE_MATRIXOP1( det, det, MxDouble )
CASE_MATRIXOP1( flip, flip, MxMatrix )
CASE_MATRIXOP1( mirror, mirror, MxMatrix )

CASE_MINMAX( min, '<' )
CASE_MINMAX( max, '>' )

CASE_MULDIV( mul, etimes )
CASE_MULDIV( div, edivide )

case f_pow:
    CHECK_NR_PARAM( $1, 2 )
```

```
            double pwr = MxDouble.doubleValue( params[1] );

            if ( params[0] instanceof MxMatrix )
            {
                Matrix p = ((MxMatrix)params[0]).mat;
                int m = p.height(), n = p.width();
                Matrix x = new Matrix( m, n );
                for ( int j=0; j<n; j++ )
                    for ( int i=0; i<m; i++ )
                        x.set( i, j, Math.pow( p.get(i,j), pwr ) );
                return new MxMatrix( x );
            }
            return new MxDouble(
                Math.pow( MxDouble.doubleValue( params[0] ), pwr ) );

        case f_count:
            CHECK_NR_PARAM( $1, 1 );
            if ( params[0] instanceof MxBitArray )
                return new MxInt( ((MxBitArray)params[0]).ba.count() );
            throw new MxException( "count() accepts one bitmask argument" );

        CASE_VECOP1( round, Math.round )
        CASE_VECOP1( ceil, Math.ceil )
        CASE_VECOP1( floor, Math.floor )
        CASE_VECOP1( abs, Math.abs )
        CASE_VECOP1( sqrt, Math.sqrt )
        CASE_VECOP1( exp, Math.exp )
        CASE_VECOP1( log, Math.log )
        CASE_VECOP1( sin, Math.sin )
        CASE_VECOP1( cos, Math.cos )
        CASE_VECOP1( tan, Math.tan )
        CASE_VECOP1( asin, Math.asin )
        CASE_VECOP1( acos, Math.acos )
        CASE_VECOP1( atan, Math.atan )

        default:
            throw new MxException( "unknown internal function" );
        }
    }

}
```

B.2.16 src/Makefile

```
# Makefile for the MX programming language
# $Id: Makefile,v 1.16 2003/05/11 16:14:50 hanhua Exp $

GRAMMAR = MxAntlrTokenTypes.java MxAntlrTokenTypes.txt \
          MxAntlrLexer.java MxAntlrParser.java

WALKER = MxAntlrWalker.java \
         MxAntlrWalkerTokenTypes.java MxAntlrWalkerTokenTypes.txt

M4JAVA = MxInternalFunction.java

GENJAVA = $(GRAMMAR) $(WALKER) $(M4JAVA)

CLASSES = MxInternalFunction.class \
          MxAntlrTokenTypes.class MxAntlrLexer.class MxAntlrParser.class \
          MxAntlrWalkerTokenTypes.class MxAntlrWalker.class \
          MxDataType.class MxBool.class MxInt.class MxDouble.class \
          MxMatrix.class MxRange.class MxBitArray.class MxFunction.class\
          MxException.class MxSymbolTable.class MxInterpreter.class \
          MxVariable.class MxString.class MxMain.class

all : $(CLASSES)

$(CLASSES): %.class : %.java $(GENJAVA) jamaica.jar
javac -source 1.4 -g $<

MxInternalFunction.java: MxInternalFunction.m4 Makefile
m4 $< > $@

$(GRAMMAR): grammar.g Makefile
rm -f $(GRAMMAR)
java antlr.Tool $<

$(WALKER): walker.g MxAntlrTokenTypes.txt Makefile
rm -f $(WALKER)
java antlr.Tool $<

jamaica.jar: jamaica/*.java
( cd jamaica ; javac -g *.java )
jar cvf jamaica.jar jamaica/*.class

zip: all
zip -r plt-prj-‘date +%y%m%d‘.zip *.java *.class *.g *.m4 Makefile *.jar jamaica/*.java

clean:
rm -f *.class *~

dist-clean: clean
rm -f MxAntlr* MxInternalFunction.java
```

B.3 Jamaica

B.3.1 src/jamaica/Matrix.java

```
package jamaica;

import java.io.PrintWriter;
import java.io.RandomAccessFile;
import java.io.IOException;
import java.text.NumberFormat;
import java.text.DecimalFormat;
import java.text.DecimalFormatSymbols;
import java.util.Locale;

/**
 * Jamaica - (yet another) Java Matrix Class
 *
 * @author Hanhua Feng - hanhua@cs.columbia.edu
 * @version $Id: Matrix.java,v 1.25 2003/05/12 21:13:54 hanhua Exp $
 */

public class Matrix implements Cloneable {

    /** internal data structure
     * @serial array to store matrix entries
     * @serial number of rows
     * @serial number of columns
     * @serial index (in the array) of the first entry of the matrix
     * @serial index difference if the row number increments
     * @serial index difference if the column number increments
     */
    double [] a;
    int m, n, r, ms, ns;

    /** Construct an m-by-n matrix and fill with zeros.
     * @param m     number of rows
     * @param n     number of columns
     */
    public Matrix( int m, int n ) {
        this.m = m;
        this.n = n;
        r = 0;
        ms = 1;     // the matrix is saved column-by-column
        ns = m;
        a = new double[m*n];
    }

    /** Construct an m-by-n matrix and fill with a scalar value.
     * @param m     number of rows
     * @param n     number of columns
     * @param s     the scalar value to be filled
     */
    public Matrix( int m, int n, double s ) {
        this.m = m;
```

```
    this.n = n;
    r = 0;
    ms = 1;
    ns = m;
    a = new double[m*n];
    for ( int i=0; i<a.length; i++ )
        a[i] = s;
}

/** Construct an m-by-n matrix with an array .
 * @param m     number of rows
 * @param n     number of columns
 * @param array the initial array (contructor will not make copies).
 * @param byrow the array is stored by row (default: by column)
 */
public Matrix( int m, int n, double[] array, boolean byrow ) {
    this.m = m;
    this.n = n;
    r = 0;
    if ( byrow )
    {
        ms = n;
        ns = 1;
    }
    else
    {
        ms = 1;
        ns = m;
    }
    a = array;
}

/** Construct a matrix sharing to the same data of another matrix.
 * @param x     original matrix
 */
public Matrix( Matrix x ) {
    m = x.m;
    n = x.n;
    r = x.r;
    ms = x.ms;
    ns = x.ns;
    a = x.a;
}

/** Construct a matrix to be a submatrix of another matrix
 * @param x     original matrix
 * @param mr    row range
 * @param nr    column range
 */
public Matrix( Matrix x, Range mr, Range nr ) {
    if ( ! mr.in( 0, x.m-1 ) || ! nr.in( 0, x.n-1 ) )
    {
        throw new IllegalArgumentException(
```

```
            "Invalid range: " + x.m + "*" + x.n
            + " [" + mr.toString() + "," + nr.toString() + "]" );
    }

    a = x.a;
    m = mr.size();
    n = nr.size();
    r = x.r + mr.first() * x.ms + nr.first() * x.ns;
    ms = x.ms * mr.interval();
    ns = x.ns * nr.interval();
}

/** Return an n-by-n identity matrix
 * @param n     number of rows or columns
 * @return      n-by-n identity matrix
 */
public static Matrix eye( int n ) {
    Matrix x = new Matrix( n, n );
    for ( int i=0; i<n; i++ )
        x.set( i, i, 1.0 );
    return x;
}

/** Return an m-by-n matrix with random entries
 * @param m     number of rows
 * @param n     nubmer of columns
 * @return      m-by-n random matrix
 */
public static Matrix random( int m, int n ) {
    Matrix x = new Matrix( m, n );
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            x.set( i, j, Math.random() );
    return x;
}

/** Return an m-by-n matrix with incremental entries.  All
 * differences between pairs of two vertically adjacent entries
 * are the same, and so for for horizontally pairs.
 * @param m    number of rows
 * @param n    number of columns
 * @param s    the value of entry (0,0)
 * @param dm   vertical increment
 * @param dn   horizontal increment
 * @return     new m-by-n matrix
 */
public static Matrix indgen( int m, int n, double s,
                             double dm, double dn ) {
    Matrix x = new Matrix( m, n );
    for ( int j=0; j<n; j++ )
    {
        double p = s;
        for ( int i=0; i<m; i++, p+=dm )
```

```
            x.set( i, j, p );
        s += dn;
    }
    return x;
}

// Return an string indicating an error for array dimensions
private final String dimErrMsg() {
    return "Invalid dimensions: " + m + "*" + n;
}

private final String dimErrMsg( Matrix b ) {
    return "Dimensions not match: " + m + "*" + n + " <=> "
        + b.m + "*" + b.n;
}

// get correspoding bit in bmk for matrix element (i,j)
final boolean getBitMask( BitArray bmk, int i, int j ) {
    return bmk.get( i+j*m );
}

// set correspoding bit in bmk for matrix element (i,j)
private final void setBitMask( BitArray bmk, int i, int j ) {
    bmk.set( i+j*m );
}

/** Get row dimensions
 * @return      the number of rows
 */
public final int height() {
    return m;
}

/** Get column dimensions
 * @return      the number of rows
 */
public final int width() {
    return n;
}

/** Get an entry of the matrix
 * @param i     row index (zero-based)
 * @param j     column index (zero-based)
 */
public final double get( int i, int j ) {
    return a[r+i*ms+j*ns];
}

/** Assign a value to an entry of the matrix
 * @param i     row index (zero-based)
 * @param j     column index (zero-based)
 * @param x     value to be assigned
 */
```

```
public final void set( int i, int j, double x ) {
    a[r+i*ms+j*ns] = x;
}

/** Add a value to an entry of the matrix
 * @param i     row index (zero-based)
 * @param j     column index (zero-based)
 * @param x     value to be assigned
 */
public final void setAdd( int i, int j, double x ) {
    a[r+i*ms+j*ns] += x;
}

/** Subtract a value to an entry of the matrix
 * @param i     row index (zero-based)
 * @param j     column index (zero-based)
 * @param x     value to be assigned
 */
public final void setSub( int i, int j, double x ) {
    a[r+i*ms+j*ns] -= x;
}

/** Multiply a value to an entry of the matrix
 * @param i     row index (zero-based)
 * @param j     column index (zero-based)
 * @param x     value to be assigned
 */
public final void setMul( int i, int j, double x ) {
    a[r+i*ms+j*ns] *= x;
}

/** Divide a value to an entry of the matrix
 * @param i     row index (zero-based)
 * @param j     column index (zero-based)
 * @param x     value to be assigned
 */
public final void setDiv( int i, int j, double x ) {
    a[r+i*ms+j*ns] /= x;
}

/** swap two rows
 * @param i1   row index 1 ( zero-based )
 * @param i2   row index 2 ( zero-based )
 */
public final void swapRows( int i1, int i2 ) {
    double x;
    for ( int j=0; j<n; j++ )
    {
        x = get( i1, j );
        set( i1, j, get( i2, j ) );
        set( i2, j, x );
    }
}
```

```
/** swap two columns
 * @param i1   column index 1 ( zero-based )
 * @param i2   column index 2 ( zero-based )
 */
public final void swapColumns( int j1, int j2 ) {
    double x;
    for ( int i=0; i<m; i++ )
    {
        x = get( i, j1 );
        set( i, j1, get( i, j2 ) );
        set( i, j2, x );
    }
}

/** Generate a copy of this matrix without share of data
 * @return      copy of this matrix
 */
public final Matrix copy() {
    Matrix x = new Matrix( m, n );
    int c = 0, p = r, q = p;

    for ( int j=0; j<n; j++ )
    {
        q = p;
        for ( int i=0; i<m; i++ )
        {
            x.a[c++] = a[q];
            q += ms;
        }
        p += ns;
    }

    return x;
}

/** Implement the method in interface Cloneable
 * @return      copy of this matrix as an object
 */
public Object clone() {
    return copy();
}

/** Let this matrix be as same as another matrix B (shallow copy)
 * @param b     matrix B
 * @return      this matrix
 */
public Matrix refer( Matrix b ) {
    m = b.m;
    n = b.n;
    r = b.r;
    ms = b.ms;
    ns = b.ns;
```

```
    a = b.a;
    return this;
}

/** Copy all elements of matrix B to this matrix (deep copy)
 * @param b     Matrix B
 * @return      this matrix
 */
public Matrix assign( Matrix b ) {
    if ( m != b.m || n != b.n )
        throw new IllegalArgumentException( dimErrMsg( b ) );
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            set( i, j, b.get( i, j ) );
    return this;
}

/** Copy elements of matrix B to this matrix with bitmasks
 * @param bmk   Bitmask indicating whether permission of assignment
 * @param b     Matrix B
 */
public Matrix assign( BitArray bmk, Matrix b ) {
    if ( m != b.m || n != b.n )
        throw new IllegalArgumentException( dimErrMsg( b ) );
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            if ( getBitMask( bmk, i, j ) )
                set( i, j, b.get( i, j ) );
    return this;
}

/** Set all elements to a scalar
 * @param f     scalar
 * @return      this matrix
 */
public Matrix assign( double f ) {
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            set( i, j, f );
    return this;
}

/** Copy elements to a scalar to this matrix with bitmasks
 * @param bmk   Bitmask indicating whether permission of assignment
 * @param b     Matrix B
 */
public Matrix assign( BitArray bmk, double f ) {
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            if ( getBitMask( bmk, i, j ) )
                set( i, j, f );
    return this;
}
```

```
/** Get the submatrix of this matrix, sharing the same data
 * @param mr    row range
 * @param nr    column range
 * @return      sub-matrix
 */
public Matrix slice( Range mr, Range nr ) {
    return new Matrix( this, mr, nr );

    /* This piece of code is no longer necessary

    Matrix x = new Matrix( this, mr, nr );

    int c1 = x.r;
    int c2 = x.r + (x.m-1)*x.ms;
    int c3 = x.r + (x.n-1)*x.ns;
    int c4 = x.r + (x.m-1)*x.ms + (x.n-1)*x.ns;

    if ( c1 < 0 || c1 >= x.a.length
         || c2 < 0 || c2 >= x.a.length
         || c3 < 0 || c3 >= x.a.length
         || c4 < 0 || c4 >= x.a.length )
    {
        throw new IllegalArgumentException(
            "Invalid range: "
            + m + "*" + n + " [" + mr.toString() + "," + nr.toString()
            + "]" );
    }
    return x;
    */
}

/* Order statistics: minimum
 * @return      the minimum value of a matrix
 */
public double min() {
    double d = Double.MAX_VALUE;

    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
        {
            if ( get(i,j) < d )
                d = get(i,j);
        }
    return d;
}

/* Order statistics: maximum
 * @return      the maximum value of a matrix
 */
public double max() {
    double d = Double.MIN_VALUE;
```

```
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
        {
            if ( get(i,j) > d )
                d = get(i,j);
        }
    return d;
}

/* Order statistics: minimum of a column
 * @param column the column number
 * @return      the minimum value of the matrix column
 */
public double min( int column ) {
    double d = Double.MAX_VALUE;

    for ( int i=0; i<m; i++ )
    {
        if ( get(i,column) < d )
            d = get(i,column);
    }
    return d;
}

/* Order statistics: maximum of a column
 * @param column the column number
 * @return      the maximum value of the matrix column
 */
public double max( int column ) {
    double d = Double.MIN_VALUE;

    for ( int i=0; i<m; i++ )
    {
        if ( get(i,column) > d )
            d = get(i,column);
    }
    return d;
}

/** Transpose this matrix, sharing the same data
 * @return      new tranposed matrix
 */
public Matrix transpose() {
    Matrix x = new Matrix( this );
    int t;
    t = x.m; x.m = x.n; x.n = t;
    t = x.ms; x.ms = x.ns; x.ns = t;
    return x;
}

/** Flip vertically this matrix, sharing the same data
 * @return      new flipped matrix
 */
```

```
public Matrix flip() {
    Matrix x = new Matrix( this );
    x.r += (x.m - 1) * x.ms;
    x.ms = -x.ms;
    return x;
}

/** Mirror horizontally this matrix, sharing the same data
 * @return      new mirrored matrix
 */
public Matrix mirror() {
    Matrix x = new Matrix( this );
    x.r += (x.n - 1) * x.ns;
    x.ns = -x.ns;
    return x;
}

/** Negate all elements of a matrix
 * @return      this matrix
 */
public Matrix selfneg() {
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            set( i, j, -get(i,j) );
    return this;
}

/** Generate a new matrix that is the negative of this matrix
 * @return      new negative matrix
 */
public Matrix uminus() {
    Matrix x = new Matrix( m, n );
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            x.set( i, j, -get(i,j) );
    return x;
}

/** Compute the sum of this matrix and another matrix (C=A+B)
 * @param b     second operand B
 * @return      new sum matrix C
 */
public Matrix plus( Matrix b ) {
    if ( m != b.m || n != b.n )
        throw new IllegalArgumentException( dimErrMsg( b ) );
    Matrix x = new Matrix( m, n );
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            x.set( i, j, get(i,j) + b.get(i,j) );
    return x;
}

/** Add another matrix to this matrix (A+=B)
```

```
 * @param b     matrix B
 * @return      this matrix, which is the sum
 */
public Matrix selfadd( Matrix b ) {
    if ( m != b.m || n != b.n )
        throw new IllegalArgumentException( dimErrMsg( b ) );
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            setAdd( i, j, b.get(i,j) );
    return this;
}

/** Compute the difference between this matrix and another matrix (C=A-B)
 * @param b     matrix B
 * @return      new difference matrix C
 */
public Matrix minus( Matrix b ) {
    if ( m != b.m || n != b.n )
        throw new IllegalArgumentException( dimErrMsg( b ) );
    Matrix x = new Matrix( m, n );
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            x.set( i, j, get(i,j) - b.get(i,j) );
    return x;
}

/** Subtract another matrix from this matrix (A-=B)
 * @param b     matrix B
 * @return      this matrix, which is the difference
 */
public Matrix selfsub( Matrix b ) {
    if ( m != b.m || n != b.n )
        throw new IllegalArgumentException( dimErrMsg( b ) );
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            setSub( i, j, b.get(i,j) );
    return this;
}

/** Compute the product of this matrix by a scalar (C=fA)
 * @param f     scalar operand
 * @return      new product matrix
 */
public Matrix times( double f ) {
    Matrix x = new Matrix( m, n );
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            x.set( i, j, get(i,j) * f );
    return x;
}

/** Compute elemental product of this matrix and another matrix (C=A.*B)
 * @param b     second operand matrix B
```

```
 * @return      new product matrix
 */
public Matrix etimes( Matrix b ) {
    if ( m != b.m || n != b.n )
        throw new IllegalArgumentException( dimErrMsg( b ) );
    Matrix x = new Matrix( m, n );
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            x.set( i, j, get(i,j) * b.get(i,j) );
    return x;
}

/** Compute the product of this matrix and another matrix (C=AB)
 * @param b     second operand matrix B
 * @return      new product matrix
 */
public Matrix times( Matrix b ) {
    if ( n != b.m )
        throw new IllegalArgumentException( dimErrMsg( b ) );
    Matrix x = new Matrix( m, b.n );
    for ( int j=0; j<b.n; j++ )
        for ( int i=0; i<m; i++ )
        {
            double t = 0.0;
            for ( int k=0; k<n; k++ )
                t += get( i, k ) * b.get( k, j );
            x.set( i, j, t );
        }
    return x;
}

/** Multiply this matrix by a scalar (A*=f)
 * @param f     scalar
 * @return      this matrix
 */
public Matrix selfmul( double f ) {
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            setMul( i, j, f );
    return this;
}

/** Elementally multiply this matrix by another matrix (A.*=B)
 * @param b     matrix B
 * @return      this matrix
 */
public Matrix selfemul( Matrix b ) {
    if ( m != b.m || n != b.n )
        throw new IllegalArgumentException( dimErrMsg( b ) );
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            setMul( i, j, b.get(i,j) );
    return this;
```

```
}

/** Left multiply by another matrix (at right) (A=BA)
 * @param b     matrix B
 * @return      this matrix
 */
public Matrix selflmul( Matrix b ) {
    return refer( b.times( this ) );
}

/** Right multiply  by another matrix (at left) (A=AB)
 * @param b     matrix B
 * @return      this matrix
 */
public Matrix selfrmul( Matrix b ) {
    return refer( times( b ) );
}

/** Compute elemental division of this matrix and another matrix (C=A./B)
 * @param b     matrix B
 * @return      new quotient matrix
 */
public Matrix edivide( Matrix b ) {
    if ( m != b.m || n != b.n )
        throw new IllegalArgumentException( dimErrMsg( b ) );
    Matrix x = new Matrix( m, n );
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            x.set( i, j, get(i,j) / b.get(i,j) );
    return x;
}

/** Compute left division of this matrix and another matrix (C=B^-1A)
 * @param b     matrix B
 * @return      new quotient matrix
 */
public Matrix ldivide( Matrix b ) {
    return b.solve( this );
}

/** Compute left division of this matrix and another matrix (C=AB^-1)
 * @param b     matrix B
 * @return      new quotient matrix
 */
public Matrix rdivide( Matrix b ) {
    return b.transpose().solve( transpose() ).transpose();
}

/** Elementally divide this matrix by another matrix (A=A./B)
 * @param b     matrix B
 * @return      this matrix
 */
public Matrix selfediv( Matrix b ) {
```

```
    if ( m != b.m || n != b.n )
        throw new IllegalArgumentException( dimErrMsg( b ) );
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            setDiv( i, j, b.get(i,j) );
    return this;
}

/** Left divide another matrix B from this matrix (A=B^-1A)
 * @param b     matrix B
 * @return      new quotient matrix
 */
public Matrix selfldiv( Matrix b ) {
    return refer( ldivide( b ) );
}

/** Right divide another matrix B from this matrix (A=AB^-1)
 * @param b     matrix B
 * @return      new quotient matrix
 */
public Matrix selfrdiv( Matrix b ) {
    return refer( rdivide( b ) );
}

/** Return a bit array indicating whether elements of this matrix is
 * greater than a scalar value
 * @param f     a scalar
 * @return      new bit array
 */
public BitArray gt( double f ) {
    BitArray bmk = new BitArray( m*n );
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            if ( get(i,j) > f )
                setBitMask( bmk, i, j );
    return bmk;
}

/** Return a bit array indicating whether elements of this matrix is
 * greater than corresponding elements of another matrix B
 * @param b     matrix B
 * @return      new bit array
 */
public BitArray gt( Matrix b ) {
    if ( m != b.m || n != b.n )
        throw new IllegalArgumentException( dimErrMsg( b ) );
    BitArray bmk = new BitArray( m*n );
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            if ( get(i,j) > b.get(i,j) )
                setBitMask( bmk, i, j );
    return bmk;
}
```

```
/** Return a bit array indicating whether elements of this matrix is
 * greater than or equal to a scalar
 * @param f     a scalar
 * @return      new bit array
 */
public BitArray ge( double f ) {
    BitArray bmk = new BitArray( m*n );
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            if ( get(i,j) >= f )
                setBitMask( bmk, i, j );
    return bmk;
}

/** Return a bit array indicating whether elements of this matrix is
 * greater than or equal to corresponding elements of another matrix B
 * @param b     matrix B
 * @return      new bit array
 */
public BitArray ge( Matrix b ) {
    if ( m != b.m || n != b.n )
        throw new IllegalArgumentException( dimErrMsg( b ) );
    BitArray bmk = new BitArray( m*n );
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            if ( get(i,j) >= b.get(i,j) )
                setBitMask( bmk, i, j );
    return bmk;
}


/** Return a bit array indicating whether elements of this matrix is
 * less than a scalar
 * @param f     a scalar
 * @return      new bit array
 */
public BitArray lt( double f ) {
    BitArray bmk = new BitArray( m*n );
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            if ( get(i,j) < f )
                setBitMask( bmk, i, j );
    return bmk;
}

/** Return a bit array indicating whether elements of this matrix is
 * less than corresponding elements of another matrix B
 * @param b     matrix B
 * @return      new bit array
 */
public BitArray lt( Matrix b ) {
    if ( m != b.m || n != b.n )
```

```
        throw new IllegalArgumentException( dimErrMsg( b ) );
    BitArray bmk = new BitArray( m*n );
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            if ( get(i,j) < b.get(i,j) )
                setBitMask( bmk, i, j );
    return bmk;
}

/** Return a bit array indicating whether elements of this matrix is
 * less than or equal to a scalar
 * @param f     a scalar
 * @return      new bit array
 */
public BitArray le( double f ) {
    BitArray bmk = new BitArray( m*n );
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            if ( get(i,j) <= f )
                setBitMask( bmk, i, j );
    return bmk;
}

/** Return a bit array indicating whether elements of this matrix is
 * less than or equal to corresponding elements of another matrix B
 * @param b     matrix B
 * @return      new bit array
 */
public BitArray le( Matrix b ) {
    if ( m != b.m || n != b.n )
        throw new IllegalArgumentException( dimErrMsg( b ) );
    BitArray bmk = new BitArray( m*n );
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            if ( get(i,j) <= b.get(i,j) )
                setBitMask( bmk, i, j );
    return bmk;
}

/** Return a bit array indicating whether elements of this matrix is
 * equal to a scalar
 * @param f     a scalar
 * @return      new bit array
 */
public BitArray eq( double f ) {
    BitArray bmk = new BitArray( m*n );
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            if ( get(i,j) == f )
                setBitMask( bmk, i, j );
    return bmk;
}
```

```
/** Return a bit array indicating whether elements of this matrix is
 * equal to corresponding elements of another matrix B
 * @param b     matrix B
 * @return      new bit array
 */
public BitArray eq( Matrix b ) {
    if ( m != b.m || n != b.n )
        throw new IllegalArgumentException( dimErrMsg( b ) );
    BitArray bmk = new BitArray( m*n );
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            if ( get(i,j) == b.get(i,j) )
                setBitMask( bmk, i, j );
    return bmk;
}

/** Return a bit array indicating whether elements of this matrix is
 * equal to a scalar within a given error
 * @param f     a scalar
 * @param eps   error below which elements can be considered equal.
 * @return      new bit array
 */
public BitArray eq( double f, double eps ) {
    BitArray bmk = new BitArray( m*n );
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            if ( Math.abs( get(i,j) - f ) < eps )
                setBitMask( bmk, i, j );
    return bmk;
}

/** Return a bit array indicating whether elements of this matrix is
 * equal to corresponding elements of another matrix B within a given error
 * @param b     matrix B
 * @param eps   error below which elements can be considered equal.
 * @return      new bit array
 */
public BitArray eq( Matrix b, double eps ) {
    if ( m != b.m || n != b.n )
        throw new IllegalArgumentException( dimErrMsg( b ) );
    BitArray bmk = new BitArray( m*n );
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            if ( Math.abs( get(i,j) - b.get(i,j) ) < eps )
                setBitMask( bmk, i, j );
    return bmk;
}

/** Return a bit array indicating whether elements of this matrix is not
 * equal to a scalar
 * @param f     a scalar
 * @return      new bit array
 */
```

```
public BitArray ne( double f ) {
    BitArray bmk = new BitArray( m*n );
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            if ( get(i,j) != f )
                setBitMask( bmk, i, j );
    return bmk;
}

/** Return a bit array indicating whether elements of this matrix is not
 * equal to corresponding elements of another matrix B
 * @param b     matrix B
 * @return      new bit array
 */
public BitArray ne( Matrix b ) {
    if ( m != b.m || n != b.n )
        throw new IllegalArgumentException( dimErrMsg( b ) );
    BitArray bmk = new BitArray( m*n );
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            if ( get(i,j) != b.get(i,j) )
                setBitMask( bmk, i, j );
    return bmk;
}

/** Return a bit array indicating whether elements of this matrix is not
 * equal to a scalar beyond a given error
 * @param f     a scalar
 * @param eps   error below which elements can be considered equal.
 * @return      new bit array
 */
public BitArray ne( double f, double eps ) {
    BitArray bmk = new BitArray( m*n );
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            if ( Math.abs( get(i,j) - f ) >= eps )
                setBitMask( bmk, i, j );
    return bmk;
}

/** Return a bit array indicating whether elements of this matrix is not
 * equal to corresponding elements of another matrix B beyond a given error
 * @param b     matrix B
 * @param eps   error below which elements can be considered equal.
 * @return      new bit array
 */
public BitArray ne( Matrix b, double eps ) {
    if ( m != b.m || n != b.n )
        throw new IllegalArgumentException( dimErrMsg( b ) );
    BitArray bmk = new BitArray( m*n );
    for ( int j=0; j<n; j++ )
        for ( int i=0; i<m; i++ )
            if ( Math.abs( get(i,j) - b.get(i,j) ) >= eps )
```

```
                setBitMask( bmk, i, j );
    return bmk;
}

/** Solve the linear equations that AX=B.
 * @param b     matrix B
 * @return      solution X
 */
public Matrix solve( Matrix b ) {
    if ( b.m != m )
        throw new IllegalArgumentException( dimErrMsg( b ) );
    Matrix x = new Matrix( b.m, b.n );
    if ( m == n )
    {
        int [] index = new int[m];
        Matrix lum = copy();
        lum.lud( index );

        // exchange rows
        for ( int k=0; k<n; k++ )
        {
            for ( int j=0; j<b.n; j++ )
                x.set( k, j, b.get(index[k],j) );
        }

        // solve L*X=B
        for ( int k=0; k<n; k++ )
        {
            for ( int i=k+1; i<n; i++ )
                for ( int j=0; j<b.n; j++ )
                    x.setSub( i, j,
                              x.get(k,j)*lum.get(i,k) );
        }

        // solve U*X=X
        for ( int k=n-1; k>=0; k-- )
        {
            double d = lum.get( k, k );
            if ( d == 0 )
                throw new RuntimeException( "Singular" );
            d = 1.0 / d;
            for ( int j=0; j<b.n; j++ )
                x.setMul( k, j, d );
            for ( int i=0; i<k; i++ )
                for ( int j=0; j<b.n; j++ )
                    x.setSub( i, j,
                              x.get(k,j)*lum.get(i,k) );
        }
    }
    else
        return null;
    return x;
}
```

```
public double det() {
    if ( m != n )
        throw new IllegalArgumentException( dimErrMsg() );
    int [] index = new int[m];
    Matrix lum = copy();
    double x = (double) lum.lud( index );
    for ( int j=0; j<n; j++ )
        x *= lum.get( j, j );
    return x;
}

public int rank() {
    return 0;
}

public int rank( double eps ) {
    return 0;
}

public Matrix invert() {
    return solve( eye( n ) );
}

/** LU decompose this matrix using Crout's algorithm.  Note that
 * L and U are saved in this matrix as output.
 * @param index   output: original columns' corresponding indexes
 * in the decomposed LU matrix
 * @return        +1 or -1 indicating the sign for det()
 */
public int lud( int[] index ) {
    if ( m < n )
        throw new IllegalArgumentException( dimErrMsg() );

    int sign = 1;
    for ( int i=0; i<m; i++ )
        index[i] = i;

    // for each column do
    for ( int j=0; j<n; j++ )
    {
        double x, sum;
        for ( int i=0; i<j; i++ )
        {
            sum = get( i, j );
            for ( int k=0; k<i; k++ )
                sum -= get( i, k ) * get( k, j );
            set( i, j, sum );
        }

        double maxv = 0.0;
        int maxidx = -1;
```

```
        for ( int i=j; i<m; i++ )
        {
            sum = get( i, j );
            for ( int k=0; k<j; k++ )
                sum -= get( i, k ) * get( k, j );
            set( i, j, sum );
            x = Math.abs( sum );
            if ( x >= maxv )
            {
                maxv = x;
                maxidx = i;
            }
        }

        if ( maxidx != j )
        {
            int t = index[j];
            index[j] = index[maxidx];
            index[maxidx] = t;

            sign = -sign;
            swapRows( j, maxidx );
        }

        if ( get( j, j ) != 0.0 )
        {
            x = 1.0 / get( j, j );
            for ( int i=j+1; i<n; i++ )
                setMul( i, j, x );
        }
    }

    return sign;
}

/** Strassen's matrix multiplication algorithm (test purpose only)
 * @param b     the second matrix B
 * @return      product of this matrix and matrix B
 */
public final Matrix strassen( Matrix b ) {
    if ( m != n || n != b.m || b.m != b.n )
        throw new IllegalArgumentException( dimErrMsg( b ) );

    if ( 1 == n )
        return new Matrix( 1, 1, get(0,0) * b.get(0,0) );

    if ( 0 != (n&1) )
        throw new IllegalArgumentException(
            "Strassen's algorithm is not applicable for matrix "
            + n + "*" + n );

    Range r1 = new Range( 0, n>>1, 1 );
    Range r2 = new Range( n>>1, n>>1, 1 );
```

```
    Matrix x = new Matrix( n, n );

    Matrix a11 = slice( r1, r1 );
    Matrix a12 = slice( r1, r2 );
    Matrix a21 = slice( r2, r1 );
    Matrix a22 = slice( r2, r2 );

    Matrix b11 = b.slice( r1, r1 );
    Matrix b12 = b.slice( r1, r2 );
    Matrix b21 = b.slice( r2, r1 );
    Matrix b22 = b.slice( r2, r2 );

    /* p1 = (a11+a22)*(b11+b22) */
    Matrix p1 = a11.plus( a22 ).strassen( b11.plus( b22 ) );

    /* p2 = (a21+a22)*b11 */
    Matrix p2 = a21.plus( a22 ).strassen( b11 );

    /* p3 = a11*(b12-b22) */
    Matrix p3 = a11.strassen( b12.minus( b22 ) );

    /* p4 = a22*(-b11+b21) */
    Matrix p4 = a22.strassen( b21.minus( b11 ) );

    /* p5 = (a11+a12)*b22 */
    Matrix p5 = a11.plus( a12 ).strassen( b22 );

    /* p6 = (-a11+a21)*(b11+b12) */
    Matrix p6 = a21.minus( a11 ).strassen( b11.plus( b12 ) );

    /* p7 = (a12-a22)*(b21+b22) */
    Matrix p7 = a12.minus( a22 ).strassen( b21.plus( b22 ) );

    /* c11 = p1+p4-p5+p7 */
    x.slice(r1,r1).assign(p1).selfadd(p4).selfsub(p5).selfadd(p7);

    /* c12 = p3+p5 */
    x.slice(r1,r2).assign(p3).selfadd(p5);

    /* c21 = p2+p4 */
    x.slice(r2,r1).assign(p2).selfadd(p4);

    /* c22 = p1+p3-p2+p6 */
    x.slice(r2,r2).assign(p1).selfadd(p3).selfsub(p2).selfadd(p6);

    return x;
}

public final static int DT_BYTE = 1;
public final static int DT_SHORT = 2;
public final static int DT_INT = 3;
public final static int DT_FLOAT = 4;
```

```
public final static int DT_DOUBLE = 5;

private static int[] dt_size = new int[]{ 0, 1, 2, 4, 4, 8 };

/** read a matrix from a binary file.  Note at least one dimension should
 * be positive
 * @param fn      the file name
 * @param type    data type: DT_BYTE, DT_SHORT, DT_INT, DT_FLOAT, DT_DOUBLE
 * @param m       number of rows (determined by file size if <= 0)
 * @param n       number of columns (determined by file size if <=0)
 * @param skip    number of bytes to be skipped from the beginning
 */
public static Matrix load( String fn, int type, int m, int n, long skip )
    throws IOException {

    if ( m <= 0 && n <= 0 )
        return null;

    RandomAccessFile file = new RandomAccessFile( fn, "r" );
    long l = file.length() - skip;
    if ( l < 0 )
    {
        file.close();
        return null;
    }

    l /= dt_size[type];
    if ( m <= 0 )
        m = (int)( l / n );
    else if ( n <= 0 )
        n = (int)( l / m );
    if ( m <= 0 || n <= 0 )
    {
        file.close();
        return null;
    }

    if ( skip != 0 )
        file.seek( skip );

    Matrix x = new Matrix( m, n );

    /* file is default saved by columns */
    switch ( type )
    {
    case DT_BYTE:  /* Java byte is signed!! */
        for ( int j=0; j<n; j++ )
            for ( int i=0; i<m; i++ )
                x.set( i, j, (double) (0xff & (int) file.readByte()) );
        break;
    case DT_SHORT:
        for ( int j=0; j<n; j++ )
            for ( int i=0; i<m; i++ )
```

```
                x.set( i, j, (double) file.readShort() );
        break;
    case DT_INT:
        for ( int j=0; j<n; j++ )
            for ( int i=0; i<m; i++ )
                x.set( i, j, (double) file.readInt() );
        break;
    case DT_FLOAT:
        for ( int j=0; j<n; j++ )
            for ( int i=0; i<m; i++ )
                x.set( i, j, (double) file.readFloat() );
        break;
    case DT_DOUBLE:
        for ( int j=0; j<n; j++ )
            for ( int i=0; i<m; i++ )
                x.set( i, j, file.readDouble() );
        break;
    default:
        file.close();
        throw new IllegalArgumentException( "Illegal data type" );
    }
    file.close();

    return x;
}

/** read a matrix from a binary file.  Note at least one dimension should
 * be positive
 * @param fn      the file name
 * @param type    data type: DT_BYTE, DT_SHORT, DT_INT, DT_FLOAT, DT_DOUBLE
 * @param m       number of rows (determined by file size if <= 0)
 * @param n       number of columns (determined by file size if <=0)
 */
public static Matrix load( String fn, int type, int m, int n )
    throws IOException {
    return load( fn, type, m, n, 0 );
}

/** write a matrix to a binary file.
 * @param fn      the file name
 * @param type    data type: DT_BYTE, DT_SHORT, DT_INT, DT_FLOAT, DT_DOUBLE
 * @param skip    number of bytes to be skipped from the beginning
 */
public void save( String fn, int type, long skip )
    throws IOException {

    RandomAccessFile file = new RandomAccessFile( fn, "rw" );
    if ( skip != 0 )
        file.seek( skip );

    /* file is default saved by columns */
    switch ( type )
    {
```

```
    case DT_BYTE:
        for ( int j=0; j<n; j++ )
            for ( int i=0; i<m; i++ )
                file.writeByte( (byte) (int) get( i, j ) );
        break;
    case DT_SHORT:
        for ( int j=0; j<n; j++ )
            for ( int i=0; i<m; i++ )
                file.writeShort( (short) (int) get( i, j ) );
        break;
    case DT_INT:
        for ( int j=0; j<n; j++ )
            for ( int i=0; i<m; i++ )
                file.writeInt( (int) get( i, j ) );
        break;
    case DT_FLOAT:
        for ( int j=0; j<n; j++ )
            for ( int i=0; i<m; i++ )
                file.writeFloat( (float) get( i, j ) );
        break;
    case DT_DOUBLE:
        for ( int j=0; j<n; j++ )
            for ( int i=0; i<m; i++ )
                file.writeDouble( get( i, j ) );
        break;
    default:
        file.close();
        throw new IllegalArgumentException( "Illegal data type" );
    }
    file.close();
}

/** write a matrix to a binary file.
 * @param fn      the file name
 * @param type    data type: DT_BYTE, DT_SHORT, DT_INT, DT_FLOAT, DT_DOUBLE
 */
public void save( String fn, int type ) throws IOException {
    save( fn, type, 0 );
}

/** print a matrix with specified format to the standard output
 * @param width    field width of each elements
 * @param pecision number of fraction digits
 * @param indent   number of additional spaces at the beginning of a line
 */
public void print( int width, int precision, int indent ) {
    print( new PrintWriter( System.out, true ), width, precision, indent );
}

/** print a matrix with user-defined number format to the standard output
 * @param format   user-defined number format
 * @param width    field width of each elements
 * @param indent   number of additional spaces at the beginning of a line
```

```
     */
    public void print( NumberFormat format, int width, int indent ) {
        print( new PrintWriter( System.out, true ), format, width, indent );
    }

    /** print a matrix with specified format
     * @param output   the printing device
     * @param width    field width of each elements
     * @param pecision number of fraction digits
     * @param indent   number of additional spaces at the beginning of a line
     */
    public void print( PrintWriter output, int width, int precision,
                       int indent ) {
        DecimalFormat fmt = new DecimalFormat();
        fmt.setDecimalFormatSymbols( new DecimalFormatSymbols(Locale.US) );
        fmt.setMinimumIntegerDigits( 1 );
        fmt.setMaximumFractionDigits( precision );
        fmt.setMinimumFractionDigits( precision );
        fmt.setGroupingUsed( false );
        print( output, fmt, width, indent );
    }

    /** print a matrix with user-defined number format
     * @param output   the printing device
     * @param format   user-defined number format
     * @param width    field width of each elements
     * @param indent   number of additional spaces at the beginning of a line
     */
    public void print( PrintWriter output, NumberFormat format,
                       int width, int indent ) {
        for ( int i=0; i<m; i++ )
        {
            for ( int j=0; j<n; j++ )
            {
                String str = format.format( get(i,j) );
                int space = width - str.length();
                if ( 0 == j )
                {
                    space += indent;
                }
                else
                {
                    output.print( ' ' );
                    space--;
                }
                for ( ; space>0; space-- )
                    output.print( ' ' );
                output.print( str );
            }
            output.println();
        }
    }
}
```

B.3.2 src/jamaica/BitArray.java

```
package jamaica;

/**
 * The bit array class that functs as an array of boolean, in which
 * each elements takes one bit.  The return type of matrix comparison.
 *
 * @author Hanhua Feng - hanhua@cs.columbia.edu
 * @version $Id: BitArray.java,v 1.9 2003/05/13 00:13:13 hanhua Exp $
 */
public class BitArray implements Cloneable {
    int size;
    int[] barray;

    /** Constructor
     * @param n    size of the bit array
     */
    public BitArray( int n ) {
        size = n;
        barray = new int[(size+31)/32];
    }

    /** Constructor from existing integer array
     * @param n    size of the bit array
     * @param a    integer array containing all bits
     */
    private BitArray( int n, int[] a ) {
        size = n;
        barray = a;
    }

    /** Return a new copy of the bit array
     * @return     copy of this bit array
     */
    public final BitArray copy() {
        return new BitArray( size, (int[]) barray.clone() );
    }

    /** Implements the method in Cloneable
     * @return     copy of this bit array as Object
     */
    public Object clone() {
        return copy();
    }

    /** get the length of the bit array
     */
    public final int length() {
        return size;
    }

    /** Set one bit in the bit array
     * @param bit  index of this bit
```

```
 */
public final void set( int bit ) {
    barray[bit/32] |= 1<<(bit%32);
}

/** Clear one bit in the bit array
 * @param bit  index of this bit
 */
public final void clear( int bit ) {
    barray[bit/32] &= ~(1<<(bit%32));
}

/** Flip one bit in the bit array
 * @param bit  index of this bit
 */
public final void flip( int bit ) {
    barray[bit/32] ^= 1<<(bit%32);
}

/** Clear all barray of the bit array
 */
public final void clear() {
    for ( int i=0; i<barray.length; i++ )
        barray[i] = 0;
}

/** Check a bit in the bit array.
 * @param b    index of this bit
 * @return     boolean value indicating whether this bit has been set.
 */
public final boolean get( int bit ) {
    return 0 != ( barray[bit/32] & ( 1<<(bit%32) ) );
}

/** Return a new bit array whose bits are exactly the reverse.
 * @return    new bit array
 */
public final BitArray not() {
    BitArray x = copy();
    for ( int i=0; i<barray.length; i++ )
        x.barray[i] = ~barray[i];
    return x;
}

/** Return a new bit array whose bits are the bitwise ors.
 * @return    new bit array
 */
public final BitArray or( BitArray b ) {
    BitArray x, y;
    int xq, xr;
    if ( size >= b.size )
    {
        x = this.copy();
```

```
        y = b;
        xq = b.size / 32;
        xr = b.size % 32;
    }
    else
    {
        x = b.copy();
        y = this;
        xq = size / 32;
        xr = size % 32;
    }

    for ( int i=0; i<xq; i++ )
        x.barray[i] |= y.barray[i];
    if ( 0 != xr )
        x.barray[xq] |= y.barray[xq] & ((1<<xr)-1);

    return x;
}

/** Return a new bit array whose bits are the bitwise ands.
 * @return    new bit array
 */
public final BitArray and( BitArray b ) {
    BitArray x, y;
    int xq, xr;
    if ( size >= b.size )
    {
        x = this.copy();
        y = b;
        xq = b.size / 32;
        xr = b.size % 32;
    }
    else
    {
        x = b.copy();
        y = this;
        xq = size / 32;
        xr = size % 32;
    }

    for ( int i=0; i<xq; i++ )
        x.barray[i] &= y.barray[i];
    if ( 0 != xr )
        x.barray[xq] &= y.barray[xq] | ~((1<<xr)-1);

    return x;
}

/** Return a new bit array whose bits are the bitwise exclusive or.
 * @return    new bit array
 */
public final BitArray xor( BitArray b ) {
```

```
    BitArray x, y;
    int xq, xr;
    if ( size >= b.size )
    {
        x = this.copy();
        y = b;
        xq = b.size / 32;
        xr = b.size % 32;
    }
    else
    {
        x = b.copy();
        y = this;
        xq = size / 32;
        xr = size % 32;
    }

    for ( int i=0; i<xq; i++ )
        x.barray[i] ^= y.barray[i];
    if ( 0 != xr )
        x.barray[xq] ^= y.barray[xq] & ((1<<xr)-1);

    return x;
}

final static int countBits( int x ) {
    int y = x;
    y = ( ( y & 0xaaaaaaaa ) >>> 1 )
        + ( y & 0x55555555 );
    y = ( ( y & 0xcccccccc ) >>> 2 )
        + ( y & 0x33333333 );
    y = ( ( y & 0xf0f0f0f0 ) >>> 4 )
        + ( y & 0x0f0f0f0f );
    y = ( ( y & 0xff00ff00 ) >>> 8 )
        + ( y & 0x00ff00ff );
    y = ( ( y & 0xffff0000 ) >>> 16 )
        + ( y & 0x0000ffff );

    return (int) y;
}

/** count the number of 1's in the bit array.
 * @return    the count of 1's
 */
public final int count() {
    int cnt = 0;
    int xq = size / 32;
    int xr = size % 32;
    for ( int i=0; i<xq; i++ )
        cnt += countBits( barray[i] );
    if ( 0 != xr )
        cnt += countBits( barray[xq] & ((1<<xr)-1) );
```

```
        return cnt;
    }

    /** convert bit array to a string for printing.
     */
    public String toString() {
        StringBuffer str = new StringBuffer();
        for ( int i=0; i<size; i++ )
            if ( get(i) )
                str.append( '1' );
            else
                str.append( '0' );
        return str.toString();
    }
}
```

B.3.3 src/jamaica/Range.java

```
package jamaica;

/**
 * The Range class representing a linear set of integers (usually indexes)
 *
 * Range is basically a triple (number,base,stride), where number
 * means the number of integers, base is the start integer, and
 * stride is the distance between to consecutive integers.
 *
 * @author Hanhua Feng - hanhua@cs.columbia.edu
 * @version $Id: Range.java,v 1.11 2003/05/13 00:13:13 hanhua Exp $
 */
public class Range implements Cloneable {
    private int number;
    private int base;
    private int stride;

    /** constructor
     * @param first   the start integer of a range
     * @param last    the end integer of a range
     */
    public Range( int first, int last ) {
        this.base = first;
        number = last - base + 1;
        if ( number < 0 )
        {
            number = -number;
            stride = -1;
        }
        else
            stride = 1;
    }

    /** constructor
     * @param first   the start integer of a range
     * @param size    the number of integers in the range
     * @param interval the distance between two consecutive integers
     */
    public Range( int first, int size, int interval ) {
        this.base = first;
        this.number = size;
        this.stride = interval;
    }

    /** the first integer of the range
     */
    public final int first() {
        return base;
    }

    /** the number of integers in the range
     */
```

```
public final int size() {
    return number;
}

/** the last integer of the range
 */
public final int last() {
    return base + (number-1) * stride;
}

/** the next integer of the current
 * @param n    current integer
 */
public final int next( int n ) {
    return n + stride;
}

/** the distance between two consecutive integers
 */
public final int interval() {
    return stride;
}

/** does a range contain a specified integer?
 * @param n    the specified integer
 * @return     a boolean value indicating the result
 */
public final boolean contain( int n ) {
    n -= base;
    if ( stride >= 0 )
        return n >= 0 && n < number * stride && 0 == n % stride;
    return n <= 0 && n > number * stride && 0 == (-n) % (-stride);
}

/** does a range is in bounded region
 * @param lower   the lower bound
 * @param upper   the upper bound
 * @return        a boolean value indicating the result
 */
public final boolean in( int lower, int upper ) {
    if ( stride >= 0 )
        return base >= lower && ( number - 1 ) * stride + base <= upper;
    return base <= upper && ( number - 1 ) * stride + base >= lower;
}

/** The smallest (not necessarily the first) integer in a range
 */
public int inf() {
    if ( stride >= 0 )
        return base;
    return base + (number-1) * stride;
}
```

```
/** The largest (not necessarily the last) integer in a range
 */
public int sup() {
    if ( stride <= 0 )
        return base;
    return base + (number-1) * stride;
}

/** is a range empty? (contains zero integers)
 * @return     a boolean value indicating the result
 */
public boolean empty() {
    return number <= 0;
}

// interesting methods starting here, although not implemented

/** intersection of two ranges.
 *  Note: the intersection of two ranges are still a range
 *  NOT implemented yet
 */
public Range intersect( Range x ) {
    throw new UnsupportedOperationException( "intersect" );
}

/** get the closure range of the union of two ranges.
 * Note: union of ranges is usually not a range, but there is
 * a minimum range, namely closure range, covering it.
 * NOT implemented yet
 */
public Range unionClosure( Range x ) {
    throw new UnsupportedOperationException( "unionClosure" );
}

/** subtract another range from this range
 * The subtraction is still a range.
 * NOT implemented yet
 */
public Range sub( Range x ) {
    throw new UnsupportedOperationException( "sub" );
}

/** convert a range to string for printing
 */
public String toString() {
    if ( 1 == stride )
    {
        if ( 1 == number )
            return Integer.toString( base );
        return Integer.toString( base ) + ":" + last();
    }

    return Integer.toString( base ) + ":"
```

```
            + ( stride>=0 ? "+" + stride : "-" + (-stride) )
            + ":" + number;
    }
}
```

B.3.4 src/jamaica/Painter.java

```
package jamaica;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;

/**
 * The painting class that draws the contents of a matrix in a window.
 * This class is derived from class JPanel in Java Swing package.
 *
 * @author Hanhua Feng - hanhua@cs.columbia.edu
 * @version $Id: Painter.java,v 1.13 2003/05/12 21:13:54 hanhua Exp $
 */
public class Painter extends JPanel {
    private static int[] grayscale = null;
    private JFrame frame;
    private Image image = null;

    public static final int CM_BLACKWHITE = 0;
    public static final int CM_GRAYSCALE = 1;
    public static final int CM_COLORS = 2;
    public static final int CM_DARKCOLORS = 3;
    public static final int CM_HOTCOLORS = 4;
    public static final int CM_WARMCOLORS = 5;
    public static final int CM_COLDCOLORS = 6;

    public static int frame_counter = 0;

    /** create a painting panel
     * @param frame    the parent frame where the painting panel is located
     */
    public Painter( JFrame frame ) {
        this.frame = frame;
    }

    /** create a color represented by an integer
     * @param r       red value, 0.0-1.0
     * @param g       green value, 0.0-1.0
     * @param b       blue value, 0.0-1.0
     * @return        the color represented by an integer
     */
    public final static int color( double r, double g, double b ) {
        return ( 0xff & (int)(b * 255) )
            + ( ( 0xff & (int)(g * 255) ) << 8 )
            + ( ( 0xff & (int)(r * 255) ) << 16 );
    }

    /** create corresponding color map
     * @param type     the id of color maps
     * @return      the color map integer array
     */
```

```
public final static int[] colorMap( int type ) {
    int[] cm;

    switch ( type )
    {
    case CM_BLACKWHITE:
        cm = new int[]{0, 0xFFFFFF};
        break;
    case CM_GRAYSCALE:
        cm = new int[256];
        for ( int i=0; i<256; i++ )
            cm[i] = (i<<16) | (i<<8) | i;
        break;
    case CM_COLORS:
        cm = new int[192];
        for ( int i=0; i<64; i++ )
            cm[i] = (i<<18) | ((63-i)<<10) | 0xff;
        for ( int i=0; i<64; i++ )
            cm[i+64] = (i<<10) | ((63-i)<<2) | 0xff0000;
        for ( int i=0; i<64; i++ )
            cm[i+128] = (i<<2) | ((63-i)<<18) | 0xff00;
        break;
    case CM_DARKCOLORS:
        cm = new int[192];
        for ( int i=0; i<64; i++ )
            cm[i] = (i<<18) | ((63-i)<<10);
        for ( int i=0; i<64; i++ )
            cm[i+64] = (i<<2) | ((63-i)<<18);
        for ( int i=0; i<64; i++ )
            cm[i+128] = (i<<10) | ((63-i)<<2);
        break;
    case CM_HOTCOLORS:
        cm = new int[128];
        for ( int i=0; i<64; i++ )
            cm[i] = 0xff0000 | (i<<10);
        for ( int i=0; i<64; i++ )
            cm[i+64] = 0xffff00 | (i<<2);
        break;
    case CM_WARMCOLORS:
        cm = new int[64];
        for ( int i=0; i<64; i++ )
            cm[i] = 0xff00 | (i<<2);
        break;
    case CM_COLDCOLORS:
        cm = new int[32];
        for ( int i=0; i<32; i++ )
            cm[i] = ((63-i)<<18) | (i<<2);
        break;
    default:
        return new int[] {0};
    }

    return cm;
```

```
}

/** create a color map form a matrix by columns.  The entries of this
 * matrix is between 0.0 and 1.0.
 * @param map   a matrix whose rows are blue, green and red, respectively
 * @return      the color map integer array
 */
public final static int[] colorMap( Matrix map ) {
    int n = map.width();
    int[] cm = new int[n];
    for ( int i=0; i<n; i++ )
        cm[i] = color( map.get(2,i), map.get(1,i), map.get(0,i) );
    return cm;
}


/** create an painting panel inside a new painting frame window
 * @param title  the title of the frame
 * @param width  the width of the painting panel
 * @param height the height of the painting panel
 * @return       the painter panel
 */
public static Painter create( String title, int width, int height ) {
    try {
        UIManager.setLookAndFeel(
            UIManager.getCrossPlatformLookAndFeelClassName() );
    } catch (Exception e) {}

    JFrame frame = new JFrame( title );
    Painter pane = new Painter( frame );
    frame.getContentPane().add( pane, BorderLayout.CENTER );

    frame_counter++;

    frame.addWindowListener(
        new WindowAdapter()
        {
            public void windowClosing( WindowEvent e ) {
                frame_counter--;
                /* System.exit(0); */
            }
        });

    pane.newsize( width, height );

    frame.setLocationRelativeTo( null );
    frame.setVisible( true );

    return pane;
}

/* change the dimensions of the drawing panel */
void newsize( int width, int height ) {
```

```
    setPreferredSize( new Dimension( width, height ) );
    image = new BufferedImage( width, height, BufferedImage.TYPE_INT_RGB );
    frame.pack();
}

/** draw a matrix in a panel with a color table and adjust the
 * dimensions of the panel accordingly
 * @param mat    the matrix
 * @param min    the value of the first color
 * @param max    the value of the last color
 * @param rgbtable the color table for intermediate values
 */
public void draw( Matrix mat, double min, double max, int[] rgbtable ) {

    int n = mat.width(), m = mat.height();
    int [] row = new int[n+n+n];

    newsize( n, m );

    WritableRaster raster = ((BufferedImage)image).getRaster();

    double factor = (max > min) ? rgbtable.length / ( max - min ) : 0;

    for ( int i=0; i<m; i++ )
    {
        for ( int j=0; j<n; j++ )
        {
            int k = (int)Math.round((mat.get(i,j)-min)*factor);
            if ( k < 0 )
                k = 0;
            if ( k >= rgbtable.length )
                k = rgbtable.length - 1;
            row[j+j+j] = ( rgbtable[k] >> 16 ) & 0xff;
            row[j+j+j+1] = ( rgbtable[k] >> 8 ) & 0xff;
            row[j+j+j+2] = ( rgbtable[k] ) & 0xff;
        }
        raster.setPixels( 0, i, n, 1, row );
    }

    repaint();
}

private int[] getGrayScale() {
    if ( null == grayscale )
        grayscale = colorMap( CM_GRAYSCALE );

    return grayscale;
}

/** draw a matrix in a panel as a grayscaled image and adjust the
 * dimensions of the panel accordingly
 * @param mat    the matrix
 * @param min    the value of white
```

```
 * @param max    the value of black
 */
public void draw( Matrix mat, double min, double max ) {
    draw( mat, min, max, getGrayScale() );
}

/** draw a matrix in a panel with a color table and adjust the
 * dimensions of the panel accordingly. The min and max values of
 * the matrix are mapped to white and black, respectively.
 * @param mat    the matrix
 * @param rgbtable the color table for intermediate values
 */
public void draw( Matrix mat, int[] rgbtable ) {
    draw( mat, mat.min(), mat.max(), rgbtable );
}

/** draw a matrix in a panel as a grayscaled image and adjust the
 * dimensions of the panel accordingly
 * @param mat    the matrix
 */
public void draw( Matrix mat ) {
    draw( mat, mat.min(), mat.max() );
}

/** draw a matrix in a panel over an existing image at some place,
 * with a color table and a pixel mask.
 * @param mat    the matrix
 * @param bmk    the mask indicating whether a matrix entry is drawn.
 * @param min    the value of the first color
 * @param max    the value of the last color
 * @param rgbtable the color table for intermediate values
 * @param x      the X-coordinate
 * @param y      the Y-coordinate
 */
public void overdraw( Matrix mat, BitArray bmk, double min, double max,
                      int[] rgbtable, int x, int y ) {

    WritableRaster raster = ((BufferedImage)image).getRaster();
    int n = mat.width(), m = mat.height();
    int[] rgb = new int[3];
    double factor = (max > min) ? rgbtable.length / ( max - min ) : 0;

    for ( int i=0; i<m; i++ )
    {
        for ( int j=0; j<n; j++ )
        {
            if ( bmk != null && !mat.getBitMask(bmk,i,j) )
                continue;
            int k = (int)Math.round((mat.get(i,j)-min)*factor);
            if ( k < 0 )
                k = 0;
            if ( k >= rgbtable.length )
                k = rgbtable.length - 1;
```

```
            rgb[0] = ( rgbtable[k] >> 16 ) & 0xff;
            rgb[1] = ( rgbtable[k] >> 8 ) & 0xff;
            rgb[2] = ( rgbtable[k] ) & 0xff;
            raster.setPixel( x+j, y+i, rgb );
        }
    }

    repaint();
}

/** draw a matrix in a panel over an existing image at some place,
 * with a color table and a pixel mask.  The min and max values of
 * the matrix are mapped to white and black, respectively.
 * @param mat    the matrix
 * @param bmk    the mask indicating whether a matrix entry is drawn.
 * @param rgbtable the color table for intermediate values
 * @param x      the X-coordinate
 * @param y      the Y-coordinate
 */
public void overdraw( Matrix mat, BitArray bmk, int[] rgbtable,
                      int x, int y ) {
    overdraw( mat, bmk, mat.min(), mat.max(), rgbtable, x, y );
}

/** draw a matrix in a panel over an existing image at some place,
 * with a color table.  The min and max values of the matrix are
 * mapped to white and black, respectively.
 * @param mat    the matrix
 * @param rgbtable the color table for intermediate values
 * @param x      the X-coordinate
 * @param y      the Y-coordinate
 */
public void overdraw( Matrix mat, int[] rgbtable, int x, int y ) {
    overdraw( mat, null, mat.min(), mat.max(), rgbtable, x, y );
}

/** draw a matrix in a panel over an existing image at the orgin,
 * with a pixel mask.  The min and max values of the matrix are
 * mapped to white and black, respectively.
 * @param mat    the matrix
 * @param bmk    the mask indicating whether a matrix entry is drawn.
 * @param rgbtable the color table for intermediate values
 */
public void overdraw( Matrix mat, BitArray bmk, int[] rgbtable ) {
    overdraw( mat, bmk, mat.min(), mat.max(), rgbtable, 0, 0 );
}

public void paintComponent( Graphics g ) {
    super.paintComponent (g);

    Insets insets = getInsets();
    int width = getWidth() - insets.left - insets.right;
    int height = getHeight() - insets.top - insets.bottom;
```

```
        if ( null != image )
            g.drawImage( image, 0, 0, width, height, this );
    }

    /** return the number of plotter windows */
    public static int frameCount() {
        return frame_counter;
    }

    /* the test program */
    public static void main( String[] args ) {
        create( "test", 200, 200 ).draw( Matrix.random( 128, 512 ) );
    }
}
```

B.3.5 src/jamaica/Plotter.java

```
package jamaica;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;

/**
 * The Plotter class which draw a curve in a window.
 *
 * Functions expected in the future:
 *   (1) multiplot: plot multiple curves by a single matrix
 *       in different colors
 *   (2) overplot: plot another curve in the same window
 *
 * @author Hanhua Feng - hanhua@cs.columbia.edu
 * @version $Id: Plotter.java,v 1.11 2003/05/12 21:13:54 hanhua Exp $
 */
public class Plotter extends JPanel {
    JFrame frame;
    Font tick_font;
    Font label_font;

    final static int nr_curve = 16;

    Insets border = new Insets( 32, 64, 32, 64 );

    int tick_width = 6;
    int grid_density = 100;
    int tick_density = 5;

    Color tick_color = new Color( 0x555555 );
    Color grid_color = new Color( 0xAAAAAA );
    Color border_color = new Color( 0xBBBB00 );
    Color[] def_color = new Color[nr_curve];

    String xlabel = null;
    String ylabel = null;

    float[] range;

    float[][] curve = new float[nr_curve][];
    Color[][] curve_color = new Color[nr_curve][];
    int[] curve_flags = new int[nr_curve];

    final static int FL_LINESTYLE = 0x80;
    final static int FL_RANGEMASK = 3;

    public static int frame_counter = 0;

    public Plotter( JFrame frame ) {
        this.frame = frame;
```

```
    tick_font = new Font( "Monospaced", Font.PLAIN, 10 );
    label_font = new Font( null, Font.PLAIN, 12 );
    range = new float[24];
    for ( int i=0; i<24; i+=2 )
    {
        range[i] = 0f;
        range[i+1] = 1f;
    }
}


public static Plotter create( String title, int width, int height ) {
    try {
        UIManager.setLookAndFeel(
            UIManager.getCrossPlatformLookAndFeelClassName() );
    } catch (Exception e) {}

    JFrame frame = new JFrame( title );
    Plotter pane = new Plotter( frame );
    frame.getContentPane().add( pane, BorderLayout.CENTER );

    frame_counter++;

    frame.addWindowListener(
        new WindowAdapter()
        {
            public void windowClosing( WindowEvent e ) {
                frame_counter--;
                /* System.exit(0); */
            }
        });

    pane.newsize( width, height );

    frame.setLocationRelativeTo( null );
    frame.setVisible( true );

    return pane;
}

/* change the dimensions of the drawing panel */
void newsize( int width, int height ) {
    setPreferredSize( new Dimension( width, height ) );
    frame.pack();
}

boolean addData( Matrix mat, int idx ) {
    int m = mat.height(), n = mat.width();
    if ( 1 == n )
        return false;
    curve[idx] = new float[m+m+m];
    for ( int i=0; i<m; i++ )
    {
```

```
        curve[idx][i+i+i] = (float) mat.get( i, 0 );
        curve[idx][i+i+i+1] = (float) mat.get( i, 1 );
        if ( n == 3 )
            curve[idx][i+i+i+2] = (float) mat.get( i, 2 );
    }
    return true;
}

void addColorTable( int[] colortbl, int idx ) {
    if ( null != colortbl )
    {
        curve_color[idx] = new Color[colortbl.length];
        for ( int i=0; i<colortbl.length; i++ )
            curve_color[idx][i] = new Color( colortbl[i] );
    }
    else
        curve_color = null;
}

public void draw( Matrix mat, double[] rgs,
                  int def_color, int[] colortbl  ) {
    removeAllCurve();
    if ( !addData( mat, 0 ) )
        return;
    setAllRanges( (float) rgs[0], (float) rgs[1], 0 );
    setAllRanges( (float) rgs[2], (float) rgs[3], 1 );
    if ( mat.width() >= 3 )
    {
        setAllRanges( (float) rgs[4], (float) rgs[5], 2 );
        addColorTable( colortbl, 0 );
    }
    this.def_color[0] = new Color( def_color );
    curve_flags[0] = FL_LINESTYLE;
    repaint();
}

public void draw( Matrix mat, int def_color, int[] colortbl ) {
    double[] rgs = new double[6];
    int n = mat.width();
    if ( n > 3 )
        n = 3;
    for ( int i=0; i<n; i++ )
    {
        rgs[i+i] = mat.min( i );
        rgs[i+i+1] = mat.max( i );
    }

    draw( mat, rgs, def_color, colortbl );
}

public void draw( Matrix mat, double[] rgs, int def_color ) {
    draw( mat, rgs, def_color, null );
}
```

```
public void draw( Matrix mat, int def_color ) {
    draw( mat, def_color, null );
}

void removeAllCurve() {
    for ( int i=0; i<nr_curve; i++ )
    {
        curve[i] = null;
        curve_color[i] = null;
    }
}

private final void setAllRanges( float min, float max, int dimid ) {
    range[dimid*8+0] = range[dimid*8+2]
        = range[dimid*8+4] = range[dimid*8+6] =  min;
    range[dimid*8+1] = range[dimid*8+3]
        = range[dimid*8+5] = range[dimid*8+7] =  max;
}

private final void setRange( float min, float max,
                             int dimid, int rangeid ) {
    range[dimid*8+rangeid*2] = min;
    range[dimid*8+rangeid*2+1] = max;
}

private final static double log10 = Math.log( 10 );

private final static int closestUnit( double x ) {
    double y = Math.log(x) / log10;
    int order = (int) Math.floor( y );
    y -= (double) order;
    if ( y < 0.1 )
        return ( order << 8 ) + 1;
    if ( y < 0.4 )
        return ( order << 8 ) + 2;
    if ( y < 0.8 )
        return ( order << 8 ) + 5;
    return ( (order+1) << 8 ) + 1;
}

private final static double decimalToDouble( int n, int order ) {
    return Math.pow( 10, order ) * n;
}

private final static String decimalToString( int n, int order ) {
    if ( 0 == n )
        return new String( "0" );

    while ( 0 == n % 10 )
    {
        n /= 10;
        order++;
```

```
    }

    StringBuffer str = new StringBuffer( 64 );

    if ( order > 4 )
        str.append(n).append( 'E' ).append( order );
    else if ( order >= 0 )
    {
        str.append(n);
        for ( int i=0; i<order; i++ )
            str.append( '0' );
    }
    else
    {
        str.append( Math.abs(n) );
        int j = str.length();
        if ( j > -order )
            str.insert( j+order, '.' );
        else if ( j > -order-3 )
        {
            for ( ; j < -order; j++ )
                str.insert( 0, '0' );
            str.insert( 0, "0." );
        }
        else
        {
            if ( j > 1 )
                str.insert( 1, "." );
            str.append( 'E' ).append( order+j-1 );
        }
        if ( n < 0 )
            str.insert( 0, '-' );
    }
    return str.toString();
}

public void paintRulers( Graphics g, int x, int y, int length, int breath,
                         int breath_extend,
                         double min, double max, double density,
                         boolean is_vert, boolean with_label ) {

    FontMetrics fm = g.getFontMetrics();
    int label_ypos;

    if ( is_vert )
        label_ypos = ( fm.getAscent() - fm.getDescent() ) / 2;
    else
    {
        if ( breath > 0 )
            label_ypos = breath_extend + fm.getAscent();
        else
            label_ypos = breath_extend - fm.getDescent();
    }
```

```
double factor = length / (max-min);
int unit = closestUnit( density / factor );
int order = unit >> 8;
int lsd = unit & 0xf;
int larger1 = ( 5 == lsd ) ? 2 : 5;
int larger2 = 10;
double dt = decimalToDouble( lsd, order );

int i = (int) Math.ceil( min/dt );
for ( double t = i * dt; t <= max; t += dt, i++ )
{
    int offset = (int) Math.round( (t-min) * factor );
    int ext = 0, pe = 0, ne = 0;

    if ( 0 == i % larger2 )
    {
        ext = breath_extend;
        if ( with_label )
        {
            String label = decimalToString( i*lsd, order );
            if ( is_vert )
            {
                int w;

                if ( breath_extend < 0 )
                    w = -fm.stringWidth( label )-1;
                else
                    w = 2;
                g.drawString( label, x + breath_extend + w,
                              y + length - 1 - offset + label_ypos );
            }
            else
            {
                g.drawString( label,
                              x + offset - fm.stringWidth( label )/2,
                              y + label_ypos );
            }
        }
    }
    else if ( 0 == i % larger1 )
        ext = (breath_extend+breath)/2;
    else
        ext = breath;

    if ( is_vert )
    {
        int y1 = y + length - 1 - offset;
        g.drawLine( x, y1, x+ext, y1 );
    }
    else
        g.drawLine( x+offset, y, x+offset, y+ext );
```

```
    }
}
public void paintBorder( Graphics g,  int win_width, int win_height ) {

    int x0 = border.left-1, x1 = win_width-border.right;
    int y0 = border.top-1, y1 = win_height-border.bottom;
    g.drawLine( x0, y0, x1, y0 );
    g.drawLine( x1, y0, x1, y1 );
    g.drawLine( x1, y1, x0, y1 );
    g.drawLine( x0, y1, x0, y0 );
}

public void paintCurve(
    Graphics g, float[] curve, float[] data_range,
    Color[] color, Color def_color,
    int width, int height, boolean line ) {

    if ( data_range[0] >= data_range[1] || data_range[2] >= data_range[3] )
        return;

    double xfactor = width / (data_range[1]-data_range[0]);
    double yfactor = height / (data_range[3]-data_range[2]);
    double zfactor = 0;
    if ( null != color )
    {
        if ( data_range[5] <= data_range[4] )
            return;
        zfactor = color.length / (data_range[5]-data_range[4]);
    }

    g.setColor( def_color );

    int x = border.left
        + (int) Math.round( (curve[0]-data_range[0])*xfactor );
    int y = border.top + height - 1
        - (int) Math.round( (curve[1]-data_range[2])*yfactor );
    if ( !line )
        g.drawArc( x-1, y-1, 2, 2, 0, 360 );

    for ( int i=3; i<curve.length-2; i+=3 )
    {
        int x1 = border.left + (int) Math.round(
            (curve[i]-data_range[0])*xfactor );
        int y1 = border.top + height - 1 - (int) Math.round(
            (curve[i+1]-data_range[2])*yfactor );
        double z = line ? curve[i+2] : (curve[i+2]+curve[i-1])*0.5;
        if ( null != color )
        {
            int ci = (int) Math.round(
                ( z-data_range[4] ) * zfactor );
            if ( ci < 0 )
                ci = 0;
            else if ( ci >= color.length )
```

```
                ci = color.length-1;
            g.setColor( color[ci] );
        }

        if ( line )
            g.drawLine( x, y, x1, y1 );
        else
            g.drawArc( x1-1, y1-1, 2, 2, 0, 360 );
        x = x1;
        y = y1;
    }
}

public static void measureFont( Graphics g ) {
}

public void paintComponent( Graphics g ) {
    super.paintComponent (g);
    Insets insets = getInsets();
    int win_width = getWidth() - insets.left - insets.right;
    int win_height = getHeight() - insets.top - insets.bottom;
    int width = win_width - border.left - border.right;
    int height = win_height - border.top - border.bottom;

    g.setColor( border_color );
    paintBorder( g, win_width, win_height );

    g.setColor( tick_color );
    g.setFont( tick_font );
    paintRulers( g, border.left, border.top-1,
                 width, -tick_width, -2*tick_width,
                 range[2], range[3], tick_density, false, true );
    paintRulers( g, border.left, win_height - border.bottom,
                 width, tick_width, 2*tick_width,
                 range[4], range[5], tick_density, false, true );
    paintRulers( g, border.left-1, border.top,
                 height, -tick_width, -2*tick_width,
                 range[10], range[11], tick_density, true, true );
    paintRulers( g, win_width - border.right, border.top,
                 height, tick_width, 2*tick_width,
                 range[12], range[13], tick_density, true, true );


    Shape clip = g.getClip();
    g.setClip( border.left, border.top, width, height );

    g.setColor( grid_color );
    paintRulers( g, border.left, border.top, width, height, height,
                 range[0], range[1], grid_density, false, false );
    paintRulers( g, border.left, border.top, height, width, width,
                 range[8], range[9], grid_density, true, false );

    float[] rg = new float[6];
```

```
        for ( int i=0; i<nr_curve; i++ ) {
            if ( null == curve[i] )
                continue;
            int idx = (curve_flags[i] & FL_RANGEMASK) << 1;
            rg[0] = range[idx];
            rg[1] = range[idx+1];
            rg[2] = range[idx+8];
            rg[3] = range[idx+9];
            rg[4] = range[idx+16];
            rg[5] = range[idx+17];
            paintCurve( g, curve[i], rg, curve_color[i], def_color[i],
                        width, height,
                        0 != ( curve_flags[i] & FL_LINESTYLE ) );
        }

        g.setClip( clip );

    }

    /** return the number of plotter windows */
    public static int frameCount() {
        return frame_counter;
    }

    public static void main( String[] args ) {
        Matrix mat = new Matrix( 1024, 2 );
        for ( int i=0; i<1024; i++ ) {
            mat.set( i, 0, Math.sin( 0.1*i )*1000000 );
            mat.set( i, 1, Math.sin( 0.115*i )*0.00075+10 );
            // mat.set( i, 2, Math.sin( 0.02*i )*2 );
        }
        create( "test", 512, 384 ).draw(mat,0x678910,
                                        Painter.colorMap(Painter.CM_DARKCOLORS));
    }
}
```

B.3.6 src/jamaica/MatrixTest.java

```
package jamaica;

import java.io.PrintWriter;

/**
 * The testing program for the Matrix class
 *
 * @author Hanhua Feng - hanhua@cs.columbia.edu
 * @version $Id: MatrixTest.java,v 1.7 2003/05/12 21:13:54 hanhua Exp $
 */
public class MatrixTest {
    static Range xr;
    static Range yr;

    final static int rand( int n ) {
        return (int) Math.floor( Math.random() * n );
    }

    final static void randRange( int rows, int cols ) {
        int i;
        i = rand(cols-1);
        xr = new Range( i, rand(cols-i)+1, 1 );
        i = rand(rows-1);
        yr = new Range( i, rand(rows-i)+1, 1 );
    }

    public static void verifyMatrix( Matrix x, int rows, int cols ) {
        if ( rows != x.m || cols != x.n )
            throw new TestError( "Invalid dimensions: "
                                 + x.m + "*" + x.n
                                 + " [should be "
                                 + rows + "*" + cols + " ]" );
        if ( x.a == null )
            throw new TestError( "Null data" );
        int[] corners = { x.r,
                          x.r+(rows-1)*x.ms,
                          x.r+(cols-1)*x.ns,
                          x.r+(rows-1)*x.ms+(cols-1)*x.ns };
        for ( int i=0; i<4; i++ )
            if ( corners[i] < 0 || corners[i] >= x.a.length )
                throw new TestError( "Boundary error: "
                                     + x.m + "*" + x.n + ", "
                                     + "length=" + x.a.length
                                     + ", r=" + x.r
                                     + ", ms=" + x.ms
                                     + ", ns=" + x.ns );
    }

    public static void compareMatrix( Matrix x, Matrix y, int nd ) {
        verifyMatrix( x, x.height(), x.width() );
        verifyMatrix( y, y.height(), y.width() );
        int n = x.ne(y,1E-10).count();
```

```
    int nx = x.eq(y,1E-10).count();
    if ( n != nd || nx != x.height()*x.width() - nd )
    {
        System.out.println( "X = " );
        x.print( 10, 4, 2 );
        System.out.println( "Y = " );
        y.print( 10, 4, 2 );

        BitArray ba = ( n != nd ) ? x.ne( y, 1E-10 ) : x.eq( y, 1E-10 );
        System.out.print( (n != nd) ? "  NE: " : "  EQ: " );
        for ( int i=0; i<ba.length(); i++ )
            System.out.print( ba.get(i)? "1" : "0" );
        System.out.println();

        throw new TestError( "Two matrices are different: "
                             + n + " diff (" + nx + " same) [should be "
                             + nd + " diff ("
                             + (x.height()*x.width() - nd) + " same)" );
    }
}

public static String testConstructor( int rows, int cols ) {
    Matrix A = new Matrix( rows, cols );
    verifyMatrix( A, rows, cols );
    Matrix B = new Matrix( rows, cols, 3.0 );
    verifyMatrix( B, rows, cols );
    Matrix C = new Matrix( A );
    verifyMatrix( C, rows, cols );
    Matrix D = Matrix.random( rows, cols );
    verifyMatrix( C, rows, cols );
    A.assign( B );
    compareMatrix( C, B, 0 );
    C.assign( 3.0 );
    compareMatrix( A, B, 0 );
    A.assign( D );
    compareMatrix( C, D, 0 );
    compareMatrix( D, D.copy(), 0 );

    A.refer( D );
    C.refer( D.copy() );
    A.set( rows/2, cols/2, 0.73 );
    compareMatrix( A.copy(), C, 1 );

    return null;
}

public static String testReadWrite( int rows, int cols ) {
    Matrix A = new Matrix( rows, cols );
    Matrix B = Matrix.random( rows, cols );
    for ( int j=0; j<cols; j++ )
        for ( int i=0; i<rows; i++ )
            A.set( i, j, B.get( i, j ) );
    for ( int j=0; j<cols; j++ )
```

```
        for ( int i=0; i<rows; i++ )
            if ( A.get( i, j ) != B.get( i, j ) )
                return "Read/Write failed";
    compareMatrix( A, B, 0 );
    for ( int j=0; j<cols; j++ )
        for ( int i=0; i<rows; i++ )
        {
            A.setAdd( i, j, A.get( i, j ) );
            B.setMul( i, j, 2.0 );
        }
    compareMatrix( A, B, 0 );
    for ( int j=0; j<cols; j++ )
        for ( int i=0; i<rows; i++ )
            A.setSub( i, j, B.get( i, j ) );
    compareMatrix( A, new Matrix( A.m, A.n ), 0 );
    return null;
}

public static String testSlicing( int rows, int cols ) {
    Matrix A = Matrix.random( rows, cols );
    randRange( rows, cols );
    compareMatrix( A.slice( yr, xr ),
                   A.transpose().slice( xr, yr ).transpose(), 0 );
    compareMatrix( A.transpose().copy().slice( xr, yr ),
                   A.copy().slice( yr, xr ).transpose(), 0 );

    Matrix B = A.copy();
    A.slice( yr, xr ).assign( 3.4 );
    compareMatrix( A, B, xr.size() * yr.size() );
    B.transpose().slice( xr, yr ).assign( -3.4 ).selfneg();
    compareMatrix( A, B, 0 );

    B.slice( yr, xr ).assign( 0 );
    compareMatrix( A, B, xr.size() * yr.size() );
    for ( int i=xr.first(); i<=xr.last(); i++ )
        for ( int j=yr.first(); j<=yr.last(); j++ )
            if ( B.get( j, i ) != 0 )
                return "Wrong result [" + j + "," + i + "] = " + B.get(j,i)
                    + " should be 0.";
    return null;
}

public static String testArithmetic( int rows, int cols ) {
    Matrix A = Matrix.random( rows, cols );
    Matrix B = A.uminus();
    compareMatrix( A.selfneg(), B, 0 );
    compareMatrix( A, B.assign( 1.23 ), rows*cols );

    B = A.copy();
    B.selfadd( A );
    B.selfadd( B );
    compareMatrix( A.plus( A.times(3.0) ), B, 0 );
```

```
    B.selfsub( B.minus( A ) );
    compareMatrix( A, B, 0 );

    B.selfemul( new Matrix(rows,cols,3.0) );
    compareMatrix( A.minus( A.times(2.0).uminus() ), B, 0 );

    B = Matrix.random( cols, rand( rows+cols ) + 1 );
    int n = Math.max( Math.max( rows, cols ), B.width() );
    int i = 1;
    while ( i < n )
        i <<= 1;
    n = i;

    Matrix A1 = new Matrix( n, n );
    Matrix B1 = new Matrix( n, n );
    A1.slice( new Range(0,rows,1), new Range(0,cols,1) ).assign( A );
    B1.slice( new Range(0,cols,1), new Range(0,B.width(),1) ).assign( B );
    Matrix C = A.transpose().copy();
    compareMatrix( A.selfrmul( B ),
                   A1.strassen(B1).slice( new Range(0,rows,1),
                                          new Range(0,B.width(),1) ), 0 );
    compareMatrix( C.selflmul( B.transpose() ).transpose(),
                   B1.transpose().strassen(A1.transpose()).slice(
                       new Range(0,B.width(),1),
                       new Range(0,rows,1) ).transpose(), 0 );
    return null;
}

public static String testInversion( int rows, int cols ) {
    Matrix B = Matrix.random( rows, cols );
    Matrix A = Matrix.random( rows, rows );
    compareMatrix( B, A.times( A.ldivide(B) ), 0 );
    return null;
}

public static String testRelational( int rows, int cols ) {
    Matrix A = Matrix.random(rows,cols);
    A.set(0,cols-1,0.5);
    compareMatrix( A.copy().assign( A.gt( 0.5 ), 1.0 ),
                   A.copy().assign( A.ge( 0.5 ), 1.0 ), 1 );
    compareMatrix( A.copy().assign( A.lt( 0.5 ), 1.0 ),
                   A.copy().assign( A.le( 0.5 ), 1.0 ), 1 );
    compareMatrix( A.copy().assign( A.eq( 0.5 ), 10.0 ),
                   A.copy().assign( A.ne( 0.5 ), 10.0 ), rows*cols );
    Matrix B = A.copy();
    B.assign( A.gt( 0.5 ), 0.0 );
    B.assign( A.lt( 0.5 ), 0.0 );
    compareMatrix( B, new Matrix( rows, cols ), 1 );
    B = A.times(3.0);
    B.assign( A.eq( 0.5 ), 0.0 );
    B.assign( A.ne( 0.5 ), 0.0 );
    compareMatrix( B, new Matrix( rows, cols ), 0 );
```

```
        B = Matrix.random( rows, cols );
        B.set( rows/2,cols/2,A.get(rows/2,cols/2) );
        compareMatrix( B.copy().assign( B.gt(A).not(), -1.0 ),
                       B.copy().assign( B.le(A), -1.0 ), 0 );
        compareMatrix( B.copy().assign( B.gt(A).not(), -1.0 ),
                       B.copy().assign( B.lt(A), -1.0 ), 1 );
        compareMatrix( B.copy().assign( B.ge(A).not(), -1.0 ),
                       B.copy().assign( B.lt(A), -1.0 ), 0 );
        compareMatrix( B.copy().assign( B.ne(A).not(), -1.0 ),
                       B.copy().assign( B.eq(A), -1.0 ), 0 );
        compareMatrix( B.copy().assign( B.ne(A,0.1).not(), -1.0 ),
                       B.copy().assign( B.eq(A,0.1), -1.0 ), 0 );
        return null;
    }

    public static String testException( int rows, int cols ) {
        return null;
    }

    public static void main( String[] args ) {
        for ( int rows=1; rows<100; rows += rand(rows)+1 )
            for ( int cols=1; cols<100; cols += rand(cols)+1 )
            {
                System.out.println( "Test " + rows + "*" + cols + "..." );
                String r;
                r = testConstructor( rows, cols );
                if ( r != null )
                    System.out.println( r );
                r = testReadWrite( rows, cols );
                if ( r != null )
                    System.out.println( r );
                r = testSlicing( rows, cols );
                if ( r != null )
                    System.out.println( r );
                r = testArithmetic( rows, cols );
                if ( r != null )
                    System.out.println( r );
                r = testInversion( rows, cols );
                if ( r != null )
                    System.out.println( r );
                r = testRelational( rows, cols );
                if ( r != null )
                    System.out.println( r );
            }
    }
}

class TestError extends RuntimeException {
    TestError( String msg ) {
        System.out.println( "Test error: " + msg );
    }
}
```

B.4 Testing script and database

B.4.1 test/mxtest.tcl

```
#!/usr/bin/tclsh
#
# Mx: testing script for awk-like test database
#
# Hanhua Feng - hanhua@cs.columbia.edu
#
# $Id: mxtest.tcl,v 1.10 2003/05/13 23:24:23 hanhua Exp $
#
# Usage:  ./mxtest.tcl <your-test-database>.tdb
#
# Database format:
#
# The test database file contains one or more entries of the following:
#
#    <expected-result> { <Mx-program> }
#
# Note that both <expected-result> and <Mx-program> can span multiple
# lines.  Everything in the same line after the last } will be ignored.
#
# <expected-result> contains one or more items separated by space/newline.
# Each item can have the following format:
#
#    !              switch stdin and stderr (default: matching stdin)
#    ...            there are more items can be ignored
#    /<regexp>/     use regular expression to match (use ^ and $ if needed!)
#    [<num>,<num>]  check whether the result is a number in the range
#    <other>        do exactly match.  number comparision for numbers.
#
# ; starts a line of comments
# ;; starts a line of comments that will be printed during testing
# ;;; starts a line of comments that will be immediately printed
#
# Example: the test for the following program will succeed:
#
#    ;;; My test script
#    ;; my test 1
#    1 2 [4,7] /^[0-9.]+$/  { return [1,2;5,3]; }
#    ;; my test 2
#    3                      { return 1+2; }

# set the source code directory here
set src ../src

# java should be in the path, otherwise add the full path
set javaexec java

# check operating system: windows or unix
if { [string first ":" $env(PWD)] >= 0 || ( [info exists env(OS)] \
           && [string first windows [string tolower $env(OS)]] >= 0 ) } {
    set osd ";"
```

```
} else {
    set osd ":"
}

proc count { line cnt } {
    set pos -1
    while { 1 } {
        incr pos
        set pos [string first "\{" $line $pos]
        if { $pos < 0 } break;
        incr cnt
    }

    set pos -1
    while { 1 } {
        incr pos
        set pos [string first "\}" $line $pos]
        if { $pos < 0 } break;
        incr cnt -1
    }

    return $cnt
}

proc remove_empty { l } {
    set rv {}

    foreach item $l {
        if { [string length $item] > 0 } {
            lappend rv $item
        }
    }

    return $rv
}

proc matchresult { pattern string error } {
    global linenum

    set pattern [remove_empty $pattern]
    set string [remove_empty $string]
    set error [remove_empty $error]

#    puts "Matching $pattern with $string..."

    foreach item $pattern {
        if { [string equal "!" $item ] } {
            set t $string
            set string $error
            set error $t
            continue
        }
```

```
        set data [lindex $string 0]
        set string [lrange $string 1 end]

        if { [string equal -length 1 "/" $item ] } {
            if { ![string equal [string index $item end] "/"] } {
                puts stderr "Error ($linenum): regexp must be ended by /"
                exit 1
            }

            if { ![regexp [string range $item 1 end-1] $data] } {
                puts "Test failed: expect $item, got $data"
                return -1
            }
        } elseif { [string equal -length 1 "\[" $item ] } {
            if { ![regexp "^\\\[(\[0-9.Ee+-\]+),(\[0-9.Ee+-\]+)\\\]$" \
                    $item dummy min max] } {
                puts stderr "Error($linenum): range format \[<num>,<num>\]."
                exit 1
            }

            if { ![string is double $data] || $data < $min || $data > $max } {
                puts "Test failed: expect $item, got $data"
                return -1;
            }
        } elseif { [string is double $data] } {
            if { $data != $item } {
                puts "Test failed: expect $item, got $data"
                return -1;
            }
        } elseif { [string equal "..." $item] } {
            set string {}
            set error {}
            break;
        } else {
            if { ![string equal $item $data] } {
                puts "Test failed: expect $item, got $data"
                return -1;
            }
        }

#         puts "Match succeeded: $item: $data"
    }

    if { [llength $string] > 0 } {
        puts "Test failed: additional data in output: $string"
        return -1;
    }

    if { [llength $error] > 0 } {
        puts "Test failed: additional data in error: $error"
        return -1;
    }
```

```
    return 0
}

if { [llength $argv] != 1 } {
    puts "Usage: $argv0 <test-script>.tdb"
    exit 0
}

set file [open [lindex $argv 0] r]

set env(CLASSPATH) $env(CLASSPATH)$osd$src$osd$src/matrix.jar

# puts $env(CLASSPATH)

set linenum 0

set testnum 1

for { set index 1 } { ![eof $file] } { incr index } {

    set expected ""
    set msg "\#$index"

    while { [gets $file line] >= 0 } {
        incr linenum
        set line [string trim $line]
        if { ![string equal -length 1 ";" $line] } {
            if { [string first "\{" $line] >= 0 } break
            append expected " " $line
        } elseif { [string equal -length 2 ";;" $line] } {
            if { [string equal -length 3 ";;;" $line] } {
                puts [string range $line 3 end]
            } else {
                set msg "\#$index:[string range $line 2 end]"
            }
        }
    }

    if { [string length $line] == 0 } {
        if { [string length [string trim $expected]] > 0 } {
            puts "Error line ($linenum): no action: $expected"
        }
        break;
    }

    set cnt [count $line 0]

    if { 0 == $cnt } {
        if { ![ regexp "^(\[^\{\]*)\{(.*)\} *$" $line dummy rexp cmd ] } {
            puts "Program Internal Error: handling line $linenum: $line"
            exit 1
        }
        append expected " " $rexp
```

```
        append cmd "\n"
    } else {
        if { ![regexp "^(\[^\{\]*)\{(.*)$" $line dummy rexp cmd ] } {
            puts "Program Internal Error: handling line $linenum: $line"
            exit 1
        }

        append expected " " $rexp
        append cmd "\n"

        set cnt [count $cmd 1]
        while { [gets $file line] >= 0 } {
            incr linenum
            set cnt [count $line $cnt]
            if { $cnt > 0 } {
                append cmd $line "\n"
            } else {
                set endpos [string last "\}" $line]
                incr endpos -1
                append cmd [string range $line 0 $endpos ] "\n"
                break
            }
        }
    }

    puts -nonewline $msg

    set result ""
    if { ![catch { set result [exec -- $javaexec MxMain -b << $cmd] } error] } {
        set error ""
    }


    if { [matchresult [split $expected] [split $result] [split $error]] < 0 } {
        puts " \[FAILED\]"
        puts "\[Program\]"
        puts $cmd
        puts "\[Result\]"
        puts $result
        if { [string length $error] > 0 } {
            puts "\[Error Message\]"
            puts $error
        }
        puts "\[Expecting\]"
        puts $expected

        exit 1
    } else {
        puts " \[PASSED\]"
    }

    incr testnum
}
```

```
puts "All passed!"
exit 0
```

B.4.2 test/bool.tdb

```
;; not true
false { return not  true; }
;; not false
true { return not false; }

;; not true variable
false {
        a = true;
        return not a;
}

;; not false variable
true {
        a = false;
        return not a;
}

;; and true and true
true { return true and true; }

;; and true and false
false { return true and false; }

;; and false and true
false { return false and true; }

;; and false and false
false { return false and false; }

;; or true and true
true { return true or true; }

;; or true and false
true { return true or false; }

;; or false and true
true { return false or true; }

;; or false and false
false { return false or false; }

;; mixed and/or
true { return true or true and false; }

;; mixed and/or
false { return (false or true) and false;}

;; mixed and/or
false { return true and false or true and false; }

;; mixed not/and/or
true { return true and not false and ( not true or true or false ); }
```

```
;; ==
false { return true==false; }

;; ==
true { return false==false; }

;; ==
true { return true==true; }

;; ==
false { return false == true; }

;; !=
true { return true!=false; }

;; !=
false { return false!=false; }

;; !=
false { return true!=true; }

;; !=
true { return false != true; }

;; use variables
true {
        a = true;
        b = not a;
        c = a and b;
        d = a or b;
        return a or b and c or d;
}

;; use variables
false {
        a = true;
        b = not a;
        c = a and b;
        d = a or b;
        return a and b or c and d;
}
```

B.4.3 test/double.tdb

```
;; +
4 { return 2.0+2; }
;; +
4 { return +2++2.0;}
;; -
0 { return 2.0-2; }
;; -
4 { return 2--2.0;}
;; -
0 { return -2.0--2.0;}
;; *
12 { return 3.0*4; }
;; *-
-12 { return 3*-4.0;}
;; /
4.5 { return 9.0/2; }
;; /'
4.5 { return 9/'2.0; }
;; %
1 { return 9%2.0; }
;; +=
8 { a=3.0; a+=5; return a; }
;; -=
-2 { a=3.0; a-=5.0; return a; }
;; *=
15 { a=3.0; a*=5.0; return a; }
;; /=
0.6 { a=3.0; a/=5; return a; }
;; /= (no type change)
0 { a=3; a/=5.0; return a; }
;; /'=
0.6 { a=3.0; a/'=5.0; return a; }
;; %=
3 { a=3.0; a%=5; return a; }
;; %= (no type change)
3 { a=3; a%=5.1; return a; }
;; >=
true { return 5>=3.0; }
;; >=
true { return 3.0>=3.0; }
;; >=
false { return 3.0>=5; }
;; <=
true { return 3<=5.0; }
;; <=
true { return 3.0<=3; }
;; <=
false { return 5<=3.0; }
;; >
true { return 5>3.0; }
;; >
false { return 3.0>3; }
```

```
;; >
false { return 3>5.0; }
;; <
true { return 3.0<5; }
;; <
false { return 3.0<3; }
;; <
false { return 5<3.0; }
;; ==
false { return 5.0==3; }
;; ==
true { return 3==3.0; }
;; ==
false { return 3.0==5; }
;; !=
true { return 3!=5.0; }
;; !=
false { return 3!=3.0; }
;; !=
true { return 5.0!=3; }
```

B.4.4 test/infunc.tdb

```
;; width
200   { return width(zeros(100,200)); }
;; height
100   { return height(zeros(100,200)); }
;; random/count
0      { return count(random(100,100)>1.0); }
;; random/count
10000  { return count(random(100,100)>=0.0); }
;; random/count, are we lucky? law of large numbers
[4900,5100]  { return count(random(100,100)>=0.5); }
;; zeros/count
100 { return count(zeros(4,25)==0); }
;; eye abs
0 { return count( abs(inv(eye(5))-eye(5))>0.0001 ); }
;; abs
3 { return abs(-3); }
;; indgen/count
4 { return count( indgen(4,4,1,1,2)<3.5 ); }
;; flip/mirror
10 { A=random(2,5); return count(flip(mirror(A))==mirror(flip(A))); }
;; mul
2 2 4 100 { return mul([1,2;2,4],[2,1;2,25]);}
;; div
1 1 2 2 { return div([3,4;2,6],[3,4;1,3]); }
;; max
3  { return max([1,3;2,-1]); }
;; min
-1  { return min([1,3,1,-1]); }
;; multiple min
-10 { return min(5,7,0.3,[2,-10]); }
;; pow
[0.999,1.001] [3.9999,4.000] { return pow([1,2],2); }
;; floor
-4 4 5 6 { return floor([-3.1,4.2;5.3,6.9]); }
;; sin
[0.8414,0.8416] { return sin(1.0); }
;; cos
[0.5402,0.5404] { return cos(1.0); }
;; tan matrix
[1.5573,1.5575]  [-2.1851,-2.1849]
[-0.1426,-0.1424] [1.1577,1.1579]
     { return tan([1,2;3,4]); }
```

B.4.5 test/int.tdb

```
;; +
4 { return 2+2; }
;; +
4 { return +2++2;}
;; -
0 { return 2-2; }
;; -
4 { return 2--2;}
;; -
0 { return -2--2;}
;; *
12 { return 3*4; }
;; *-
-12 { return 3*-4;}
;; /
4 { return 9/2; }
;; /'
4 { return 9/'2; }
;; %
1 { return 9%2; }
;; +=
8 { a=3; a+=5; return a; }
;; -=
-2 { a=3; a-=5; return a; }
;; *=
15 { a=3; a*=5; return a; }
;; /=
0 { a=3; a/=5; return a; }
;; /'=
0 { a=3; a/'=5; return a; }
;; %=
3 { a=3; a%=5; return a; }
;; >=
true { return 5>=3; }
;; >=
true { return 3>=3; }
;; >=
false { return 3>=5; }
;; <=
true { return 3<=5; }
;; <=
true { return 3<=3; }
;; <=
false { return 5<=3; }
;; >
true { return 5>3; }
;; >
false { return 3>3; }
;; >
false { return 3>5; }
;; <
true { return 3<5; }
```

```
;; <
false { return 3<3; }
;; <
false { return 5<3; }
;; ==
false { return 5==3; }
;; ==
true { return 3==3; }
;; ==
false { return 3==5; }
;; !=
true { return 3!=5; }
;; !=
false { return 3!=3; }
;; !=
true { return 5!=3; }
```

B.4.6 test/samples.tdb

```
;;; samples.tdb: a sample of testing
;;; Hanhua Feng
;;; database for the Mx

;; assign
1 2 3 4 { a=[1,2;3,4]; return a; }
;; transpose
1 3 2 4 { a=[1,2;3,4]'; return a; }
;; transpose transpose
1 2 3 4 { a=[1,2;3,4]''; return a; }
;; triple transposes
1 2 4 { return [1;2;4]'''; }
;; vector element
3 { a=[1,2,3]; return a[2]; }
;; matrix element
3 { a = [1,2;3,4;5,6]; return a[1,0];}
;; matrix add
1 2 3 4 { return [1,1;1,1]+[0,1;2,3]; }
;; matrix sub
1 1 1 1 { return [2,3;4,5]-[1,2;3,4]; }
;; matrix +=
1 2 3 4 { a=[4,3;2,1];
          a+=[-3,-1;1,3];
          return a; }
;; matrix -=
1 2 3 4 { a=[2,2;4,6];
          a-=[1,0;1,2];
          return a; }
;; matrix unary -
3 -4 5 -6 { return -[-3,4,-5,6]; }
;; matrix element
4 { a = [1,2,3]; return a[2]+1; }
;; matrix element
4 { a = [1;2;3]; return 1+a[2]; }
;; matrix element
4 { a = [1;2;3]; return a[0]+a[2]; }
;; Error
! Error: ... { return true+3; }
;; Error
! Error: ... { return [1,2]*[1,2]; }
```

B.4.7 test/tzfor.tdb

```
; Tiantian Zhou - tz2002@columbia.edu

;; for

1 1 1 1 1 1 {
       b = [0, 0; 1, 1; 2, 2];
for (i = 0 : 2){
   for (j = 0 : 1){
b[i::1, j::1] = 1;
   }
}
return b;
}
```

B.4.8 test/tzfunc.tdb

```
; Tiantian Zhou - tz2002@columbia.edu

;;func () =
3 {a = 1;
b = 2;
func assign () = a + b;
return assign();
}


false {a = true;
b = false;
func assign () = a and b;
return assign();
}

;; static scope
4 {a = 1;
b = 2;
c = a + b;
func assign (a) {
a += 1;
return a;
}
return assign(c);
}

-1 {a = 1;
b = 2;
func assign (a, b) {
return a - b;
}
return assign(a, b);
}
```

B.4.9 test/tzloop.tdb

```
; Tiantian Zhou - tz2002@columbia.edu

;; loop

1 0 1 2 {
        i = 0;
        b = [0, 0; 1, 1 ];
        loop {
    b[i, i] += 1;
       i += 1;
    if (i == 2)
  break;
        }

        return b;
}
```

B.4.10 test/tzmisc.tdb

```
; Tiantian Zhou - tz2002@columbia.edu
;

;; sum +
3 5 7 9 {a = [1, 2; 3, 4];
  b = [2, 3; 4, 5];
  c = a + b;
  return c;
}

;; diff -
-1 -1 -1 -1 {a = [1, 2; 3, 4];
b = [2, 3; 4, 5];
  c = a - b;
  return c;
}

;; product *
10 13 22 29 {a = [1, 2; 3, 4];
  b = [2, 3; 4, 5];
  c = a * b;
  return c;
}

;; div /
1.5 -0.5 0.5 0.5 {a = [1, 2; 3, 4];
  b = [2, 3; 4, 5];
  c = a / b;
  return c;
}

;; div2 /'
2 1 -1 0 {a = [1, 2; 3, 4];
  b = [2, 3; 4, 5];
  c = a /' b;
  return c;
}

;; mod % (only used on numbers, does not support maxtrix)
1  { a = 15;
b = 2;
c = a % b;
return c;}

;; func //not sure if {} in {} works
3 5 7 9 {a = [1, 2; 3, 4];
  b = [2, 3; 4, 5];
func calSum(a, b){
return a + b;
}
sum = calSum(a, b);
return sum;
```

```
}

;; if else //question: isnt true a keyword:
;1 {true = 1;
; if (true == 1)
; a = 1;
; else
; a = 2;
; return a;
; }

;; and both
1 {a = 1;
b = 2;
if ((a == 1) and (b == 2))
c = 1;
else
c = 2;
return c;
}

;; and one
2 {a = 1;
b = 2;
if ((a == 2) and (b == 2))
c = 1;
else
c = 2;
return c;
}

;; or both
1 {a = 1;
b = 2;
if ((a == 1) or (b == 2))
c = 1;
else
c = 2;
return c;
}

;; or match one
1 {a = 1;
b = 2;
if ((a == 2) or (b == 2))
c = 1;
else
c = 2;
return c;
}

;; or match none
2 {a = 1;
```

```
b = 2;
if ((a == 10) or (b == 10))
c = 1;
else
c = 2;
return c;
}


;; <
1 {a = 1;
b = 2;
if (a < b)
c = 1;
return c;
}

;; >
2 {a = 2;
b = 1;
if (a > b)
c = 2;
return c;
}

;; <=
1 {a = 1;
b = 2;
if (a <= b)
c = 1;
return c;
}

;; >=
1 {a = 2;
b = 1;
if (a >= b)
c = 1;
return c;
}

;; !=
1 {a = 1;
b = 2;
if (a != b)
c = 1;
return c;
}

;; func =
3 {a = 1;
b = 2;
func assign () = a + b;
```

```
return assign();
}

;; E
0.11 {return 11e-2;}

;; e
1.11 {return 11.1E-1;}


;; +=
2 { a = 1;
a += 1;
return a;
}

;; -=
0 { a = 1;
a -= 1;
return a;
}

;; *=
18 { a = 9;
a *= 2;
return a;
}


;; /= //
4.5 { a = 9.0;
a /= 2;
return a;
}

;; /= //
4       { a = 9;
        a /= 2;
        return a;
        }
```

B.4.11 test/tzmx.tdb

```
; Tiantian Zhou - tz2002@columbia.edu

1  {a = [1];
return a;
}

3  {a = [1];
b = [2];
return a + b;
}

-1  {a = [1];
b = [2];
return a - b;
}

2  {a = [1];
b = [2];
return a * b;
}

0.5  {a = [1];
b = [2];
return a / b;
}


3 5 7 9 {a = [1, 2; 3, 4];
  b = [2, 3; 4, 5];
  c = a + b;
  return c;
}

;; diff -
-1 -1 -1 -1 {a = [1, 2; 3, 4];
b = [2, 3; 4, 5];
  c = a - b;
  return c;
}

;; product *
10 13 22 29 {a = [1, 2; 3, 4];
  b = [2, 3; 4, 5];
  c = a * b;
  return c;
}

;; div /
1.5 -0.5 0.5 0.5 {a = [1, 2; 3, 4];
  b = [2, 3; 4, 5];
  c = a / b;
  return c;
```

```
}

;; div2 /'
2 1 -1 0 {a = [1, 2; 3, 4];
  b = [2, 3; 4, 5];
  c = a /' b;
  return c;
}
```

B.4.12 test/tzrange.tdb

```
; Tiantian Zhou - tz2002@columbia.edu

3 4 5 6 {a = [1,2,3,4,5,6,7,8];
b = a[:,2:5];
return b;
}

3 4 5 6 {a = [1,2,3,4,5,6,7,8];
b = a[2:5];
return b;
}

3 4 5 6 {a = [1,2,3,4,5,6,7,8];
b = a[0,2:5];
return b;
}

4 4 4 4 4 4 4 4 {a = [1, 2; 1, 2; 1, 2; 1, 2];
a[:, :] = 4;
return a;
}

4  {a = [1];
a[:, :] = 4;
return a;
}

4 4 {a = [1;2];
a[:, :] = 4;
return a;
}

4 2 4 2 4 2 4 2 {a = [1, 2; 1, 2; 1, 2; 1, 2];
a[:, 0 :: 1] = 4;
return a;
}


4 2 1 2 1 2 1 2 {a = [1, 2; 1, 2; 1, 2; 1, 2];
a[0 :: 1, 0 :: 1] = 4;
return a;
}


1 {a = [1, 2; 1, 2; 1, 2; 1, 2];
b = a[0 :: 1, 0 :: 1];
return b;
}
```

B.4.13 test/tzstring.tdb

```
; Tiantian Zhou - tz2002@columbia.edu

;; plain
tiantian {a = "tiantian";
return a;
}

;; string + string
tiantianzhou {a = "tiantian";
b = "zhou";
return a + b;
}
```

B.4.14 test/runall.sh

```
#!/bin/sh

for F in *.tdb
do
   if ./mxtest.tcl $F
   then echo ==== $F: test passed ====
   else exit
   fi
done
```