MathLight: Lightweight MATLAB implementation

Boya Song (bs3065),
Chunli Fu (cf2710),
Mingye Chen (mc4414),
Yuli Han (yh2986)

1. Motivation & Introduction

MathLight is a lightweight implementation of MATLAB. As a programming language, it allows basic matrix manipulations, 2D plotting of data and common statistics computations. Our goal is to make it as an easy, fast and flexible language and the basic syntax is similar to C/C++.

In order to make it powerful and swift, we add the matrix data type since it's widely used in the areas of scientific computations. Also, many arithmetic operators and relational operators are designed for matrix manipulations such as matrix multiplication, transpose, equality test and so on. We implemented rich built-in functions to make it user-friendly: statistics computations such as mean, median, variance and more are supported, matrix computations like determinant, rank, trace, eigenvalues are prepared for developers to use, both scatter and line plotting are offered for 2-D data visualization. For built-in I/O library, it supports CSV/XLS loading for the matrix. We also have fine-implemented algorithms like polynomial regression etc. in language's standard library.

2. Syntax

2.1 Data types

Type names	Description
int	32-bit signed integer
double	64- bit double precision float-point number
boolean	8-bit logical value
char	character data
matrix	matrix data

There are five kinds of basic data types in our language. The declaration of them are similar to statically-typed language like C and JAVA. For matrix, the specification of declaration is shown as following.

matrix a = [1, 2, 3, 4; 1, 2, 3, 4];

We use the square brackets to define a matrix. The different rows of a matrix is separated by semicolon. While the different entries in one row is separated by comma. There is only one data type in one matrix. We can use syntax like a[1, 2] to get the value which locate at the intersection of the second row and the third column. We can also use syntax like a[1, 2 : 3] to retrieve part of the matrix to get a new matrix.

For each data type, we could define an array of them. The string is represented as an array of char data type in our language. The syntax is like below:

```
int[] a = {1, 2, 3}
double[5] b
char[] str = “hello world”
```

2.2 Operators

2.2.1 Arithmetic operators

Operators	Description
+	Addition(int, double, char, matrix) .
-	Subtraction(int, double, char, matrix) .
*	Multiplication(int, double, matrix) .
/	Division(int, double, matrix)
^	Power(int, double)
||	Absolute value(int, double)
.*	Special multiplication for matrix
./	Special division for matrix
.^	Special power for matrix
‘	Transpose for matrix

The multiplication and division of the matrix are defined as that in the linear algebra. The dot is a special operator for matrix in our language. The dot means the operator is applied to each elements in the matrix.

2.2.2 Relational operators

Operators	Description
>	Greater than (int, double, char)
<	Small than(int, double, char)
<=	Smaller than or equal to (int, double, char)
>=	Greater than or equal to (int, double, char)
==	Equal to (int, double, char, matrix)

Two matrix are identical if they have the same size and the elements at the corresponding positions are the same.

2.2.2 Logical operators

Operators	Description
!	Not(boolean)
&&	And(boolean)
||	Or(boolean)

2.3 Flow Controls

2.2.1 Condition statements

The definition of a if statement is like below:

```
if  (condition A) {
        Statements
}
else if (condition B) {
        Statements
}
else {
        Statements
}
```

2.2.2 Loop statements

The definition for a for loop is like below:

```
for (int a = 1 : 2) {
        Statements
}

for (int a = 1 ; a < 3; a = a + 1) {
        Statements
}
```

The definition for a while loop is like below:

```
while (condition) {
        Statements
}
```

We can use **break** statement and **continue** statement to jump out from a loop or jump to the beginning of a loop.

2.4 Built-in Functions

Function names	Description

log(int a)
log(double a)
	Compute the logarithm of base 2.

avg(int[] a)
avg(double[] a)
	Compute the average number of an array of numbers.

var(int[] a)
var(double[] a)
	Compute the variance number of an array of numbers.

med(int[] a)
med(double[] a)
	Compute the median number of an array of numbers.

max(int[] a)
max(double[] a)
	Compute the maximum number of an array of numbers.

min(int[] a)
min(double[] a)
	Compute the minimum number of an array of numbers.

inv(matrix m)	Get the Inverse matrix of a given a matrix
det(matrix m)	Compute the determinant of a matrix
rank(matrix m)	Get the rank of a matrix

tr(matrix m)	Compute the trace of a matrix
eig(matrix m)	Compute the eigenvalues of a matrix
zeros(int[] a)	Generate a matrix filled with all zeros, dimension specified by a
ones(int[] a)	Generate a matrix filled with all ones, dimension specified by a
rowSize(matrix m)	Get the number of rows in a matrix
colSize(matrix m)	Get the number of columns in a matrix
addRow(matrix a, int index, matrix m)	Add a row to a specific position to a given matrix. Row is represented as a special matrix. The row should have the same number of columns as the matrix.
addCol(matrix a, int index, matrix m)	Add a column to a specific position to a given matrix. Column is represented as a special matrix.The column should have the same number of rows as the matrix.
addLeft(matrix a, matrix b)	Add a matrix b to the left side of matrix a. The matrix a should have the same number of rows as the matrix b
addRight(matrix a, matrix b)	Add a matrix to the right side of a matrix.The matrix a should have the same number of rows as the matrix b.
addTop(matrix a, matrix b)	Add a matrix above a matrix. The matrix a should have the same number of columns as the matrix b.
addBottom(matrix a, matrix b)	Add a matrix below a matrix. The matrix a should have the same number of columns as the matrix b

print(int a)
print(double a)
print(boolean a)
print(matrix a)
print(int a)
print(char[] a)
	The matrix will be printed row by row.

scatter(int[] x, int[] y, char c)
scatter(double[] x, double[] y, char c)
	Draw a scatter plot picture based on an array of position[x, y]

polyfit(int[] x, int[] y, int n)	Returns the coefficients for a polynomial p(x) of

polyfit(double[] x, double[] y, int n)	degree n that is a best fit (in a least-squares sense) for the data in y
polyval(int[] x, int[] coef)	Evaluates the polynomial p at each point in x.

plot(int[] x, int[] y)
plot(double[] x, double[] y)
	Creates a 2-D line plot of the data in y versus the corresponding values in x.

load(char[] filename, boolean header)	Load a csv/xls file as a matrix
show()	Open the display console and display the figure
imread(char[] filename)	Load an image as a matrix
imshow(matrix img)	Open the display console and display an image

2.5 Function definition

The user-defined function could be declared as below:

```
func int name(int a, int b)  {
        statements
}
```

3. Example

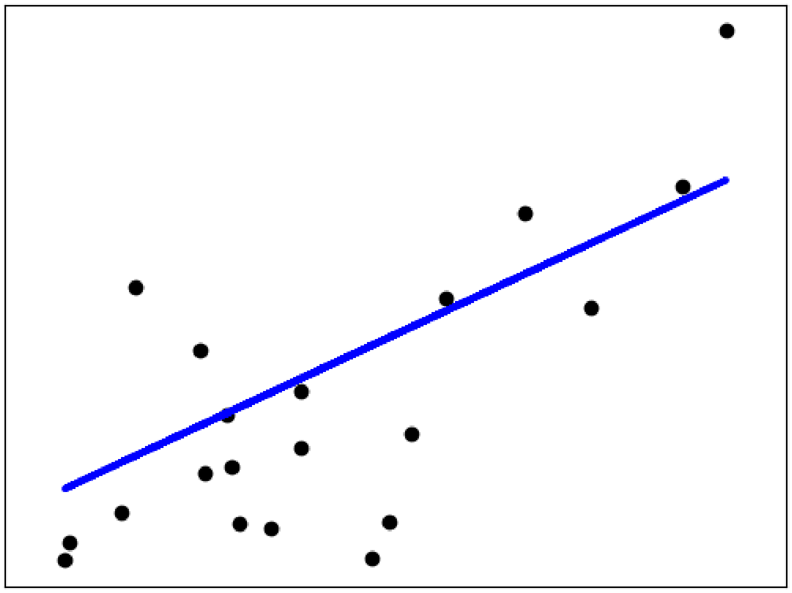
3.1 Linear Regression

Linear regression problem is a very common problem in statistics and machine learning. Here, for simplicity, we select dataset with one dimension data. The program below uses normal equation to solve the problem. It loads the data first, preprocess the data, calculate the coefficients and make the prediction. It also draws a scatter plot and a regression line to visualize the result.

```
// load two column csv file as a matrix
matrix data = load('diabetes.csv');
// matrix slicing
matrix x = data[:, 0];
matrix y = data[:, 1];
// draw scatter figure
scatter(x, y);
// data preprocess
```

```
matrix x_new = addLeft(x, ones(rowSize(x), 1));
// calculate coefficients
matrix w = inv(x_new' * x_new) * x_new' * y;
// make prediction
matrix y_lr = x_new * w;
// print result
print(w)
// draw regression result
plot(x, y_lr);
show();
```

```
Output: [152.91886182616167, 938.23786125]
```



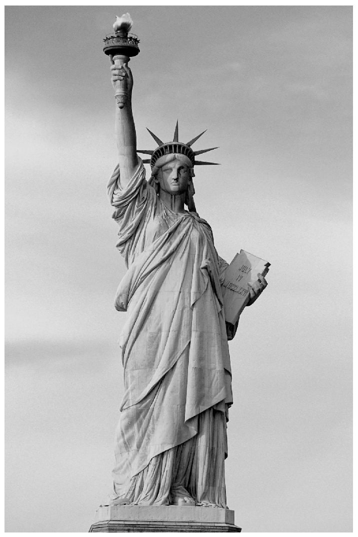
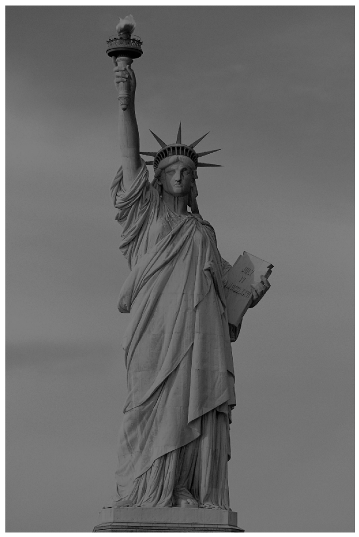
3.2 Image Processing

Image can be represented by matrix in our language. We can load image and do some transformation with it. For example, we can load an image of statue of liberty and change the color of it.

```
// load an image
matrix img = imread('liberty.jpg');
// show the image
imshow(img)
// subtract 30 from every pixel in the image, the image becomes darker
matrix img2 = img - 30
// show the new image
imshow(img2)
```

```
// every pixel in the image is divided by 2, the image becomes darker
matrix img3 = img / 2
// show the new image
imshow(img3)
```

img, img2 and img3 are as follow:

References

1. Scikit learn linear regression.
   http://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html