# Mathematical Functions and Matrix Operations

## 1 Mathematical functions

MATLAB offers many predefined mathematical expressions which contain a large set of mathematical functions. The commands `doc elfun` and `doc specfun` call up full lists of elementary and special functions respectively.

There is a long list of mathematical functions that are built into MATLAB. These functions are called built-ins. Many standard mathematical functions, such as sin(x), cos(x), tan(x), $e^x$, ln(x), are evaluated by the functions **`sin, cos, tan, exp`**, and `log` respectively in MATLAB. Table 1 lists some commonly used functions, where variables x and y can be numbers, vectors, or matrices.

Table 1: Elementary functions

| | | | |
|---|---|---|---|
| cos(x) | Cosine | abs(x) | Absolute value |
| sin(x) | Sine | max(x) | Maximum value |
| tan(x) | Tangent | min(x) | Minimum value |
| acos(x) | Arc cosine | ceil(x) | Round toward $+\infty$ |
| asin(x) | Arc sine | floor(x) | Round toward $-\infty$ |
| atan(x) | Arc tangent | round(x) | Round to nearest integer |
| exp(x) | Exponential | rem(x) | Remainder after division |
| sqrt(x) | Square root | angle(x) | Phase angle |
| log(x) | Natural logarithm | conj(x) | Complex conjugate |
| log10(x) | Decimal logarithm | | |

In addition to the elementary functions shown in Table 1, MATLAB includes a number of predefined constant values. A list of the most common values is given in Table 2.

Table 2: Predefined Constant Values

| | |
|---|---|
| pi | $\pi$(3.14159...) |
| i,j | The imaginary unit i, $\sqrt{-1}$ |
| Inf | $\infty$ (infinity) |
| NaN | Not a number |

## 2 Introduction to Matrices (continued)

Matrices are the basic elements of the MATLAB environment. A matrix is a two-dimensional array consisting of m rows and n columns, i.e. A(m,n). Special cases are column vectors (n=1) and row vectors (m=1).

### 2.1 Linear spacing

There is a command to generate linearly spaced vectors, called `linspace`. It is similar to the colon operator (:), but gives direct control over the number of points. For example,

```
y = linspace(a,b)
```

generates a row vector `y` of 100 points spaced between and including `a` and `b`. In addition,

```
y = linspace(a,b,n)
```

generates a row vector `y` of `n` points linearly spaced between and including `a` and `b`. This is useful when we want to divide an interval into a number of subintervals of the same length. For example,

```
>> theta = linspace(0,2*pi,101)
```

divides the interval [0,2$\pi$] into 100 equal subintervals, thus creating a vector of 101 elements.

## 2.2 Creating a sub-matrix

For the rest of the lab, we will refer to the matrix A introduced in the previous lab.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

The sub-matrix comprising the intersection of rows p to q and columns r to s in the matrix A is denoted by A(p:q,r:s).

As a special case, a colon (:) as the row or column specifier covers all entries in that row or column; thus

- A(:,j) is the jth column of A, while
- A(i,:) is the ith row, and
- A(end,:) picks out the last row of A.

The keyword `end`, used in A(end,:), denotes the last index in the specified dimension. Here are some examples.

```
>> A
A  =
   1   2   3
   4   5   6
   7   8   9

>> A(2:3,2:3)
ans =
    5   6
    8   9
```

Note that when we start from the last row and went towards the first, we have to use a negative increment.

```
>> A(end:-1:1,end)
ans =
    9
    6
    3
```

To extract a sub-matrix B consisting of rows 2 and 3 and columns 1 and 2 of the matrix A, do the following,

```
>> B = A([2 3],[1 2])
B    =
      4   5
      7   8
```

To interchange rows 1 and 2 of A, use the vector of row indices together with the colon operator.

```
>> C = A([2 1 3],:)
C    =
      4   5   6
      1   2   3
      7   8   9
```

It is important to note that the colon operator (:) stands for all columns or all rows.

To create a vector version of matrix A that has the all of its values in consecutive order starting from the first row and first column of A and ending at the last row and last column, do the following,

```
>> A(:)
ans =
    1
    4
    7
    2
    5
    8
    3
    6
    9
```

## 2.3 Exercise 1

Make a submatrix that contains only the first and third rows of A and only the second and third columns.

## 2.4 Deleting row or column

To delete a row or column of a matrix, use the empty vector operator, [].

```
>> A(3,:) = []
A  =
   1   2   3
   4   5   6
```

The third row was deleted.

## 2.5 Dimension

To determine the dimensions of a matrix or vector, use the command `size`. For example, to find the dimensions of A, type,

```
>> size(A)
ans   =
      3   4
```

which means A has 3 rows and 3 columns.

Or more explicitly with,

```
>> [m,n] = size(A)
m = 3
n = 3
```

Also, if you just want to find how many rows there are, you can type, `size(A,1)` or to find the number of columns, type, `size(A,2)`. The second argument is the dimension we want to know the size of, i.e. 1 for rows, 2 for columns.

## 2.6 Transposing a matrix

The transpose operation is denoted by an apostrophe or a single quote ('). It turns all the rows into columns and vice versa. Thus,

```
>> A'
ans =
    1   4   7
    2   5   8
    3   6   0
```

By using linear algebra notation, the transpose of mxn real matrix A is the nxm matrix that results from interchanging the rows and columns of A. The transpose matrix in linear algebra is generally denoted as $A^T$.

## 2.7 Concatenating matrices

Matrices can be composed by using sub-matrices as building blocks. Here is an example.

```
   A  =
   1   2   3
   4   5   6
   7   8   9
```

Now we want to create a new matrix B that will consist of 4 concatenated matrices. It will be composed of the matrix A, a matrix where all the entries are 10 times those of A, another matrix with negatives of the entries of A, and a 3x3 identity matrix. The new matrix B will be:

```
>> B = [A 10*A; -A [1 0 0; 0 1 0; 0 0 1]]
B  =
   1   2   3   10   20   30
   4   5   6   40   50   60
   7   8   9   70   80   90
  -1  -2  -3    1    0    0
  -4  -5  -6    0    1    0
  -7  -8  -9    0    0    1
```

Try to think of concatenating matrices the same way you think about creating a normal matrix. Instead of entering the values in each row and column, you are putting entire matrices together. Imagine the matrix B being divided into four equal sections (top left, top right, bottom left, and bottom right). It is as if each of these sections is a value, and this is essentially what you entered when creating B.

## 2.8 Matrix generators

MATLAB provides functions that generate elementary matrices. The matrix of zeros, the matrix of ones, and the identity matrix are returned by the functions `zeros`, `ones`, and `eye`, respectively.

Table 3: Elementary Matrices

| | |
|---|---|
| eye(m,n) | Returns an m-by-n matrix with 1 on the main diagonal |
| eye(n) | Returns an n-by-n square identity matrix |
| zeros(m,n) | Returns an m-by-n matrix of zeros |
| ones(m,n) | Returns an m-by-n matrix of ones |
| diag(A) | Extracts the diagonal of matrix A |
| rand(m,n) | Returns an m-by-n matrix of random numbers between 0 and 1 |

## 2.9 Exercise 2

Refer to Table 3 to complete the following exercises.

a. Make a 3x1 vector of ones.

b. Make a 3x3 identity matrix.

c. Make a 2x3 matrix of zeros.

For a complete list of elementary matrices, type `help elmat` or `doc elmat`.

## 2.10 More on Matrix Generators

In addition, matrices can be constructed in a block form or by concatenation (as was introduced in Section 2.7). With C defined by C = [1 2; 3 4], we may create a matrix D as follows,

```
>> D = [C zeros(2); ones(2) eye(2)]
D =
  1   2   0   0
  3   4   0   0
  1   1   1   0
  1   1   0   1
```

It is important to remember that the three elementary matrix operations of addition (+), subtraction (-), and multiplication (*) apply also to matrices whenever the dimensions are compatible.