CSC1050 – Hands-On Introduction to MATLAB

Array, Equation System, and Polynomials

Section 2.1: Numeric Arrays: To create a *row* vector, separate the elements by coma. For example,

```
>>p = [3,7,9]
p =
   3   7   9
```

row vector

You can create a *column* vector by using the *transpose* notation (').

```
>>p = [3,7,9]'
p =
   3
   7
   9
```

comma is transpose

You can also create a column vector by separating the elements by semicolons. For example,

```
>>g = [3;7;9]
g =
    3
    7
    9
```

You can create vectors by "appending" one vector to another.

For example, to create the row vector `u` whose first three columns contain the values of `r = [2,4,20]` and whose fourth, fifth, and sixth columns contain the values of `w = [9,-6,3]`, you type `u = [r,w]`. The result is the vector

`u = [2,4,20,9,-6,3].`

use <space> or ',' are ok to separate values

The colon operator (:) easily generates a large vector of regularly spaced elements.

Typing

q: step size

>>`x = [m:q:n]` similar to python string range…

creates a vector x of values with a spacing q. The first value is m. The last value is n if **n - m** is an integer multiple of q. If not, the last value is less than n.

For example, typing `x = [0:2:8]` creates the vector `x = [0,2,4,6,8]`, whereas typing `x = [0:2:7]` creates the vector `x = [0,2,4,6]`.

To create a row vector z consisting of the values from 5 to 8 in steps of 0.1, type `z = [5:0.1:8]`.

If the increment q is omitted, it is presumed to be 1. Thus typing `y = [-3:2]` produces the vector `y = [-3,-2,-1,0,1,2]`.

Magnitude, Length, and Absolute Value of a Vector

Keep in mind the precise meaning of these terms when using MATLAB.

The `length` command gives the *number of elements* in the vector. give dimension

The *magnitude* of a vector **x** having elements x_1, x_2, …, x_n is a scalar, given by $\sqrt{(x_1^2 + x_2^2 + … + x_n^2)}$, and is the same as the vector's geometric length.

The *absolute value* of a vector **x** is a vector whose elements are the absolute values of the elements of **x**.

For example, if `x = [2,-4,5]`,

- its length is 3; (computed from `length(x)`)

- its magnitude is $\sqrt{[2^2 + (-4)^2 + 5^2]} = 6.7082$; (computed from **sqrt(x*x')**)

- its absolute value is `[2,4,5]` (computed from **abs(x)**).

Matrices

A matrix has multiple rows and columns. For example, the matrix

$$\mathbf{M} = \begin{bmatrix} 2 & 4 & 10 \\ 16 & 3 & 7 \\ 8 & 4 & 9 \\ 3 & 12 & 15 \end{bmatrix}$$

has four rows and three columns.

Vectors are special cases of matrices having one row or one column.

Creating Matrices

If the matrix is small you can type it row by row, separating the *elements* in a given row with *spaces* or *commas* and separating the *rows* with semicolons. For example, typing

```
>>A = [2,4,10;16,3,7];
```

creates the following matrix:

$$A = \begin{bmatrix} 2 & 4 & 10 \\ 16 & 3 & 7 \end{bmatrix}$$

Remember, spaces or commas separate elements in different *columns,* whereas semicolons separate elements in different *rows*.

Creating Matrices from Vectors

Suppose `a = [1,3,5]` and `b = [7,9,11]` (row vectors). Note the difference between the results given by `[a b]` and `[a;b]` in the following session:

```
>>c = [a b];
c =
   1   3   5   7   9   11
>>D = [a;b]
D =
   1   3   5
   7   9   11
```

You need not use symbols to create a new array.

For example, you can type

```
>> D = [[1,3,5];[7,9,11]];


D =


     1      3      5

     7      9     11
```

Array Addressing

The colon operator selects individual elements, rows, columns, or "subarrays" of arrays. Here are some examples:

- v(:) represents all the row or column elements of the vector v.
- v(2:5) represents the second through fifth elements; that is v(2), v(3), v(4), v(5).
- A(:,3) denotes all the elements in the third column of the matrix A.
- A(:,2:5) denotes all the elements in the second through fifth columns of A.
- A(2:3,1:3) denotes all the elements in the second and third rows that are also in the first through third columns.

Matlab start from '1'
CompSci start from '0'

You can use array indices to extract a smaller array from another array. For example, if you first create the array **B**

$$\mathbf{B} = \begin{bmatrix} 2 & 4 & 10 & 13 \\ 16 & 3 & 7 & 18 \\ 8 & 4 & 9 & 25 \\ 3 & 12 & 15 & 17 \end{bmatrix}$$

then type `C = B(2:3,1:3)`, you can produce the following array:

$$\mathbf{C} = \begin{bmatrix} 16 & 3 & 7 \\ 8 & 4 & 9 \end{bmatrix}$$

Additional Array Functions (Table 2.1–1)

`[u,v,w] = find(A)`   Computes the arrays `u` and `v`, containing the row and column indices of the nonzero elements of the matrix `A`, and the array `w`, containing the values of the nonzero elements. The array `w` may be omitted.

`length(A)`   Computes either the number of elements of `A` if `A` is a vector or the largest value of `m` or `n` if `A` is an $m \times n$ matrix.

Additional Array Functions (Table 2.1–1)

`max(A)`   Returns the algebraically largest element in `A` if **A** is a vector.

Returns a row vector containing the largest elements in each column if **A** is a matrix.

If any of the elements are complex, `max(A)` returns the elements that have the largest magnitudes.

Additional Array Functions (Table 2.1–1)

`[x,k] =`
 `max(A)`

Similar to `max(A)` but stores the maximum values in the row vector `x` and their indices in the row vector `k`.

`min(A)`
and
`[x,k] =`
 `min(A)`

Like `max` but returns minimum values.

Additional Array Functions (Table 2.1–1)

`size(A)`	Returns a row vector `[m n]` containing the sizes of the m x n array `A`.
`sort(A)`	Sorts each column of the array `A` in ascending order and returns an array the same size as `A`.
`sum(A)`	Sums the elements in each column of the array `A` and returns a row vector containing the sums.

The function `size(A)` returns a row vector `[m n]` containing the sizes of the $m \times n$ array **A**. The `length(A)` function computes either the number of elements of **A** if `A` is a vector or the largest value of m or n if **A** is an $m \times n$ matrix.

For example, if

$$\mathbf{A} = \begin{bmatrix} 6 & 2 \\ -10 & -5 \\ 3 & 0 \end{bmatrix}$$

then `max(A)` returns the vector `[6,2]`; `min(A)` returns the vector `[-10, -5]`; `size(A)` returns `[3, 2]`; and `length(A)` returns `3`.

Section 2.2: Multidimensional Arrays

Consist of two-dimensional matrices "layered" to produce a third dimension. Each "layer" is called a *page*.

`cat(n,A,B,C, ...)` Creates a new array by concatenating the arrays `A,B,C,` and so on along the dimension `n`.

Section 2.3: Element-by-Element Operations

For example: **Array Addition and Subtraction**

$$\begin{bmatrix} 6 & -2 \\ 10 & 3 \end{bmatrix} + \begin{bmatrix} 9 & 8 \\ -12 & 14 \end{bmatrix} = \begin{bmatrix} 15 & 6 \\ -2 & 17 \end{bmatrix} \qquad (2.3\text{-}1)$$

Array subtraction is performed in a similar way.
  The addition shown in equation 2.3–1 is performed in MATLAB as follows:

```
>>A = [6,-2;10,3];
>>B = [9,8;-12,14]
>>A+B
ans =
    15       6
    -2       17
```

Multiplying a matrix **A** by a scalar *w* produces a matrix whose elements are the elements of **A** multiplied by *w*. For example:

$$3 \begin{bmatrix} 2 & 9 \\ 5 & -7 \end{bmatrix} = \begin{bmatrix} 6 & 27 \\ 15 & -21 \end{bmatrix}$$

This multiplication is performed in MATLAB as follows:

```
>>A = [2, 9; 5,-7];
>>3*A
ans =
     6  27
    15 -21
```

Multiplication of an array by a scalar is easily defined and easily carried out.

However, multiplication of two *arrays* is not so straightforward.

MATLAB uses two definitions of multiplication:

(1) array multiplication (also called *element-by-element* multiplication), and

(2) matrix multiplication.

- Division and exponentiation must also be carefully defined when you are dealing with operations between two arrays.

- MATLAB has two forms of arithmetic operations on arrays. Next we introduce one form, called *array* operations, which are also called *element-by-element* operations. Then we will introduce *matrix* operations. Each form has its own applications.

Element-by-element operations: Table 2.3–1

Symbol	Operation	Form	Examples
+	Scalar-array addition	`A + b`	`[6,3]+2=[8,5]`
–	Scalar-array subtraction	`A - b`	`[8,3]-5=[3,-2]`
+	Array addition	`A + B`	`[6,5]+[4,8]=[10,13]`
–	Array subtraction	`A - B`	`[6,5]-[4,8]=[2,-3]`
.*	Array multiplication	`A.*B`	`[3,5].*[4,8]=[12,40]`
./	Array right division	`A./B`	`[2,5]./[4,8]=[2/4,5/8]`
.\	Array left division	`A.\B`	`[2,5].\[4,8]=[2\4,5\8]`
.^	Array exponentiation	`A.^B`	`[3,5].^2=[3^2,5^2]`
			`2.^[3,5]=[2^3,2^5]`
			`[3,5].^[2,4]=[3^2,5^4]`

Array or *Element-by-element* multiplication is defined only for arrays having the same size. The definition of the product `x.*y`, where `x` and `y` each have *n* elements, is

`x.*y = [x(1)y(1), x(2)y(2), ..., x(n)y(n)]`

if `x` and `y` are row vectors. For example, if

x = [2, 4, − 5], y = [− 7, 3, − 8] (2.3–4)

then **z = x.*y** gives

$$z = [2(− 7), 4 (3), −5(−8)] = [−14, 12, 40]$$

If x and y are column vectors, the result of $x.*y$ is a column vector. For example $z = (x').*(y')$ gives

$$z = \begin{bmatrix} 2(-7) \\ 4(3) \\ -5(-8) \end{bmatrix} = \begin{bmatrix} -14 \\ 12 \\ 40 \end{bmatrix}$$

Note that x' is a column vector with size 3 × 1 and thus does not have the same size as y, whose size is 1 × 3.

Thus for the vectors x and y the operations $x'.*y$ and $y.*x'$ are not defined in MATLAB and will generate an error message.

The array operations are performed between the elements in corresponding locations in the arrays. For example, the array multiplication operation `A.*B` results in a matrix `C` that has the same size as `A` and `B` and has the elements $c_{ij} = a_{ij} b_{ij}$. For example, if

$$\mathbf{A} = \begin{bmatrix} 11 & 5 \\ -9 & 4 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} -7 & 8 \\ 6 & 2 \end{bmatrix}$$

then `C = A.*B` gives this result:

$$\mathbf{C} = \begin{bmatrix} 11(-7) & 5(8) \\ -9(6) & 4(2) \end{bmatrix} = \begin{bmatrix} -77 & 40 \\ -54 & 8 \end{bmatrix}$$

The built-in MATLAB functions such as `sqrt(x)` and `exp(x)` automatically operate on array arguments to produce an array result the same size as the array argument `x`.

Thus these functions are said to be *vectorized* functions.

For example, in the following session the result `y` has the same size as the argument `x`.

```
>>x = [4, 16, 25];
>>y = sqrt(x)
y =
    2   4   5
```

However, when multiplying or dividing these functions, or when raising them to a power, you must use element-by-element operations if the arguments are arrays.

For example, to compute $z = (e^y \sin x) \cos^2 x$, you must type

```
z = exp(y).*sin(x).*(cos(x)).^2.
```

You will get an error message if the size of `x` is not the same as the size of `y`. The result `z` will have the same size as `x` and `y`.

Array Division

The definition of array division is similar to the definition of array multiplication except that the elements of one array are divided by the elements of the other array. Both arrays must have the same size. The symbol for array right division is ./. For example, if

$$\mathbf{x} = [8, 12, 15] \qquad \mathbf{y} = [-2, 6, 5]$$

then $z = x./y$ gives

$$\mathbf{z} = [8/(-2), 12/6, 15/5] = [-4, 2, 3]$$

Also, if

$$\mathbf{A} = \begin{bmatrix} 24 & 20 \\ -9 & 4 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} -4 & 5 \\ 3 & 2 \end{bmatrix}$$

then `C = A./B` gives

$$\mathbf{C} = \begin{bmatrix} 24/(-4) & 20/5 \\ -9/3 & 4/2 \end{bmatrix} = \begin{bmatrix} -6 & 4 \\ -3 & 2 \end{bmatrix}$$

Array Exponentiation

MATLAB enables us not only to raise arrays to powers but also to raise scalars and arrays to *array* powers.

To perform exponentiation on an element-by-element basis, we must use the `.^` symbol.

For example, if `x = [3, 5, 8]`, then typing `x.^3` produces the array $[3^3, 5^3, 8^3] = [27, 125, 512]$.

We can raise a scalar to an array power. For example, if `p = [2, 4, 5]`, then typing `3.^p` produces the array $[3^2, 3^4, 3^5]$ = [9, 81, 243].

Remember that `.^` is a *single* symbol. The dot in `3.^p` is *not* a *decimal point* associated with the number 3. The following operations, with the value of `p` given here, are equivalent and give the correct answer:

```
3.^p
3.0.^p
3..^p
(3).^p
3.^[2,4,5]
```

Section 2.4: Matrix Operations

Matrix-Matrix Multiplication: In the product of two matrices **AB**, the number of *columns* in **A** must equal the number of *rows* in **B**. The row-column multiplications form column vectors, and these column vectors form the matrix result. The product **AB** has the same number of *rows* as **A** and the same number of *columns* as **B**.

$$\begin{bmatrix} 6 & -2 \\ 10 & 3 \\ 4 & 7 \end{bmatrix} \begin{bmatrix} 9 & 8 \\ -5 & 12 \end{bmatrix} = \begin{bmatrix} (6)(9) + (-2)(-5) & (6)(8) + (-2)(12) \\ (10)(9) + (3)(-5) & (10)(8) + (3)(12) \\ (4)(9) + (7)(-5) & (4)(8) + (7)(12) \end{bmatrix}$$

$$= \begin{bmatrix} 64 & 24 \\ 75 & 116 \\ 1 & 116 \end{bmatrix}$$

(2.4–4)

Use the operator * to perform matrix multiplication in MATLAB. The following MATLAB session shows how to perform the matrix multiplication shown in (2.4–4).

```
>>A = [6,-2;10,3;4,7];
>>B = [9,8;-5,12];
>>A*B
ans =
     64    24
     75    116
      1    116
```

Matrix multiplication does not have the commutative property; that is, in general, **AB** ≠ **BA**. A simple example will demonstrate this fact:

$$\mathbf{AB} = \begin{bmatrix} 6 & -2 \\ 10 & 3 \end{bmatrix} \begin{bmatrix} 9 & 8 \\ -12 & 14 \end{bmatrix} = \begin{bmatrix} 78 & 20 \\ 54 & 122 \end{bmatrix}$$

whereas

$$\mathbf{BA} = \begin{bmatrix} 9 & 8 \\ -12 & 14 \end{bmatrix} \begin{bmatrix} 6 & -2 \\ 10 & 3 \end{bmatrix} = \begin{bmatrix} 134 & 6 \\ 68 & 65 \end{bmatrix}$$

Reversing the order of matrix multiplication is a common and easily made mistake.

Special Matrices

Two exceptions to the noncommutative property are the *null* or *zero* matrix, denoted by **0** and the *identity,* or *unity,* matrix, denoted by **I**.

The null matrix contains all zeros and is not the same as the *empty* matrix [ ], which has no elements.

These matrices have the following properties:

$$0A = A0 = 0$$

$$IA = AI = A$$

The identity matrix is a square matrix whose diagonal elements are all equal to one, with the remaining elements equal to zero.

For example, the 2×2 identity matrix is

$$\mathbf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

The functions `eye(n)` and `eye(size(A))` create an $n \times n$ identity matrix and an identity matrix the same size as the matrix `A`.

Sometimes we want to initialize a matrix to have all zero elements. The `zeros` command creates a matrix of all zeros.

Typing `zeros(n)` creates an $n \times n$ matrix of zeros, whereas typing `zeros(m,n)` creates an $m \times n$ matrix of zeros.

Typing `zeros(size(A))` creates a matrix of all zeros having the same dimension as the matrix **A**. This type of matrix can be useful for applications in which we do not know the required dimension ahead of time.

The syntax of the `ones` command is the same, except that it creates arrays filled with ones.

Section 2.5: Linear Algebraic Equations

$6x + 12y + 4z = 70$
$7x - 2y + 3z = 5$
$2x + 8y - 9z = 64$

```
>>A = [6,12,4;7,-2,3;2,8,-9];
>>B = [70;5;64];
>>Solution = A\B
Solution =
     3
     5
    -2
```

The solution is $x = 3, y = 5,$ and $z = -2.$

Matrix notation enables us to represent multiple equations as a single matrix equation. For example, consider the following set:

$$2x_1 + 9x_2 = 5 \qquad\qquad\qquad (6.2–1)$$

$$3x_1 - 4x_2 = 7 \qquad\qquad\qquad (6.2–2)$$

This set can be expressed in vector-matrix form as

$$\begin{bmatrix} 2 & 9 \\ 3 & -4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 7 \end{bmatrix}$$

which can be represented in the following compact form

$$\mathbf{Ax} = \mathbf{b} \qquad\qquad\qquad (6.2–3)$$

For the equation set $\mathbf{Ax} = \mathbf{b}$,

● if $|\mathbf{A}| = 0$, then there is no unique solution.

● Depending on the values in the vector $\mathbf{b}$, there may be no solution at all, or an infinite number of solutions.

MATLAB provides the *left-division* method for solving the equation set **Ax** = **b**. The left-division method is based on Gauss elimination.

To use the left-division method to solve for **x**, type `x = A\b`. For example,

```
>> A = [6, -10; 3, -4]; b = [2; 5];
>> x = A\b
x =
      7      4
```

This method also works in some cases where the number of unknowns does not equal the number of equations.

If the number of equations equals the number of unknowns and if $|\mathbf{A}| \neq 0$, then the equation set has a solution and it is unique.

If $|\mathbf{A}| = 0$ or if the number of equations does not equal the number of unknowns, then you must use the methods presented in Section 2.5 in textbook.

The MATLAB command `inv(A)` computes the inverse of the matrix **A**. The following MATLAB session solves the following equations using MATLAB.

$$2x + 9y = 5$$
$$3x - 4y = 7$$

```
>>A = [2,9;3,-4];
>>b = [5;7]
>>x = inv(A)*b
x =
   2.3714
   0.0286
```

If you attempt to solve a singular problem using the `inv` command, MATLAB displays an error message.

A simple example of an underdetermined systems is the equation

$$x + 3y = 6$$

All we can do is solve for one of the unknowns in terms of the other; for example, $x = 6 - 3y$. An infinite number of solutions satisfy this equation.

When there are more equations than unknowns, the left-division method will give a solution with some of the unknowns set equal to zero. For example,
```
>>A = [1, 3]; b = 6;
>>solution = A\b
solution =
     0
     2
```
which corresponds to $x = 0$ and $y = 2$.

Solving Linear Equations: Summary

If the number of equations in the set *equals* the number of unknown variables, the matrix **A** is square and MATLAB provides two ways of solving the equation set **Ax** = **b**:

1. The matrix inverse method; solve for **x** by typing `x = inv(A)*b`.
2. The matrix left-division method; solve for **x** by typing `x = A\b`.

(continued …)

Solving Linear Equations: Summary

If **A** is square and if MATLAB does not generate an error message when you use one of these methods, then the set has a unique solution, which is given by the left-division method.

You can always check the solution for `x` by typing `A*x` to see if the result is the same as `b`.

If you receive an error message, the set is underdetermined, and either it does not have a solution or it has more than one solution.

Section 2.6: Polynomial Operations: Multiplication and Division

The function `conv(a,b)` computes the product of the two polynomials described by the coefficient arrays `a` and `b`. The two polynomials need not be the same degree. The result is the coefficient array of the product polynomial.

The function `[q,r] = deconv(num,den)` computes the result of dividing a numerator polynomial, whose coefficient array is `num`, by a denominator polynomial represented by the coefficient array `den`. The quotient polynomial is given by the coefficient array `q`, and the remainder polynomial is given by the coefficient array `r`.

Polynomial Multiplication and Division: Examples

```
>>a = [9,-5,3,7];
>>b = [6,-1,2];
>>product = conv(a,b)
product =
    54   -39    41    29    -1    14
>>[quotient, remainder] = deconv(a,b)
quotient =
    1.5   -0.5833
remainder =
    0    0   -0.5833    8.1667
```

Polynomial Roots

The function `roots(a)` computes the roots of a polynomial specified by the coefficient array `a`. The result is a *column* vector that contains the polynomial's roots.

For example,

```
>>r = roots([2, 14, 20])
r =
    -2
    -7
```

Polynomial Coefficients

The function `poly(r)` computes the coefficients of the
polynomial whose roots are specified by the vector `r`.
The result is a *row* vector that contains the polynomial's
coefficients arranged in descending order of power.

For example,

```
>>c = poly([-2, -5])
c =
    1     7     10
```

Plotting Polynomials

The function `polyval(a,x)` evaluates a polynomial at specified values of its independent variable x, which can be a matrix or a vector. The polynomial's coefficients of descending powers are stored in the array a. The result is the same size as x.

Example of Plotting a Polynomial

To plot the polynomial $f(x) = 9x^3 - 5x^2 + 3x + 7$ for $-2 \le x \le 5$, you type

```
>>a = [9,-5,3,7];
>>x = [-2:0.01:5];
>>f = polyval(a,x);
>>plot(x,f),xlabel('x'),ylabel('f(x)')
```