Introduction to Matrix Operations in MATLAB

Gerald W. Recktenwald
Department of Mechanical Engineering
Portland State University
gerry@pdx.edu

Overview

Working with Matrices and Vectors

- Linear algebra
- Vectorized operations
- Array operators

Manipulation of Matrices and Vectors

The name "MATLAB" evolved as an abbreviation of "MATrix LABoratory". The data types and syntax used by MATLAB make it easy to perform the standard operations of linear algebra including addition and subtraction, multiplication of vectors and matrices, and solving linear systems of equations.

Later we will review linear algebra. Here we provide a simple introduction to some operations that are necessary for routine calculation.

- Vector addition and subtraction
- Inner and outer products
- Vectorization
- Array operators

Vector Addition and Subtraction

Vector and addition and subtraction are element-by-element operations.

Example:

```
>> u = [10 9 8];                (u and v are row vectors)
>> v = [1 2 3];
>> u+v
ans =
    11    11    11

>> u-v
ans =
     9     7     5
```

Vector Inner and Outer Products

The inner product combines two vectors to form a scalar

$$\sigma = u \cdot v = u\, v^T \iff \sigma = \sum u_i\, v_i$$

The outer product combines two vectors to form a matrix

$$A = u^T v \iff a_{i,j} = u_i\, v_j$$

Inner and Outer Products in MATLAB

Inner and outer products are supported in MATLAB as natural extensions of the multiplication operator

```
>> u = [10 9 8];          (u and v are row vectors)
>> v = [1 2 3];
>> u*v'                   (inner product)
ans =
    52

>> u'*v                   (outer product)
ans =
    10    20    30
     9    18    27
     8    16    24
```

Vectorization

- **Vectorization** is the use of single, compact expressions that operate on all elements of a vector without explicitly executing a loop. The loop *is* executed by the MATLAB kernel, which is much more efficient at looping than interpreted MATLAB code.
- Vectorization allows calculations to be expressed succintly so that programmers get a high level (as opposed to detailed) view of the operations being performed.
- Vectorization is important to make MATLAB operate efficiently.

Vectorization of Built-in Functions

Most built-in function support *vectorized* operations. If the input is a scalar the result is a scalar. If the input is a vector or matrix, the output is a vector or matrix with the same number of rows and columns as the input.

```
>> x = 0:pi/4:pi      (define a row vector)
x =
    0    0.7854    1.5708    2.3562    3.1416

>> y = cos(x)         (evaluate cosine of each x(i)
y =
    1.0000    0.7071         0   -0.7071   -1.0000
```

Contrast with Fortran implementation:

```
      real x(5),y(5)
      pi = 3.14159624
      dx = pi/4.0
      do 10 i=1,5
          x(i) = (i-1)*dx
          y(i) = sin(x(i))
 10   continue
```

No explicit loop is necessary in MATLAB.

Vector Calculations (1)

More examples

```
>> A = pi*[ 1 2; 3 4]
A =
    3.1416    6.2832
    9.4248   12.5664

>> S = sin(A)
S =
     0     0
     0     0

>> B = A/2
B =
    1.5708    3.1416
    4.7124    6.2832

>> T = sin(B)
T =
     1     0
    -1     0
```

Array Operators

Array operators support element-by-element operations that are not defined by the rules of linear algebra

Array operators are designated by a period prepended to the standard operator

Symbol	Operation
`.*`	element-by-element multiplication
`./`	element-by-element "right" division
`.\`	element-by-element "left" division
`.^`	element-by-element exponentiation

Array operators are a very important tool for writing vectorized code.

Using Array Operators (1)

Examples: Element-by-element multiplication and division

```
>> u = [1 2 3];
>> v = [4 5 6];
>> w = u.*v              (element-by-element product)
w =
     4    10    18

>> x = u./v              (element-by-element division)
x =
    0.2500    0.4000    0.5000

>> y = sin(pi*u/2) .* cos(pi*v/2)
y =
     1     0     1

>> z = sin(pi*u/2) ./ cos(pi*v/2)

Warning: Divide by zero.
z =
     1   NaN     1
```

Using Array Operators (2)

Examples: Application to matrices

```
>> A = [1 2 3 4; 5 6 7 8];
>> B = [8 7 6 5; 4 3 2 1];
>> A.*B
ans =
     8    14    18    20
    20    18    14     8

>> A*B
??? Error using ==> *
Inner matrix dimensions must agree.

>> A*B'
ans =
    60    20
   164    60

>> A.^2
ans =
     1     4     9    16
    25    36    49    64
```