

Using MATLAB

in

Linear Algebra

Edward Neuman

Department of Mathematics

Southern Illinois University at Carbondale

One of the nice features of MATLAB is its ease of computations with vectors and matrices. In this tutorial the following topics are discussed: vectors and matrices in MATLAB, solving systems of linear equations, the inverse of a matrix, determinants, vectors in n-dimensional Euclidean space, linear transformations, real vector spaces and the matrix eigenvalue problem. Applications of linear algebra to the curve fitting, message coding and computer graphics are also included.

3.1 Special characters and MATLAB functions

For the reader's convenience we include lists of special characters and MATLAB functions that are used in this tutorial.

Special characters	
;	Semicolon operator
'	Conjugated transpose
.'	Transpose
*	Times
.	Dot operator
^	Power operator
[]	Empty vector operator
:	Colon operator
=	Assignment
==	Equality
\	Backslash or left division
/	Right division
i, j	Imaginary unit
~	Logical not
~=	Logical not equal
&	Logical and
	Logical or
{ }	Cell

Function	Description
acos	Inverse cosine
axis	Control axis scaling and appearance
char	Create character array
chol	Cholesky factorization
cos	Cosine function
cross	Vector cross product
det	Determinant
diag	Diagonal matrices and diagonals of a matrix
double	Convert to double precision
eig	Eigenvalues and eigenvectors
eye	Identity matrix
fill	Filled 2-D polygons
fix	Round towards zero
fliplr	Flip matrix in left/right direction
flops	Floating point operation count
grid	Grid lines
hadamard	Hadamard matrix
hilb	Hilbert matrix
hold	Hold current graph
inv	Matrix inverse
isempty	True for empty matrix
legend	Graph legend
length	Length of vector
linspace	Linearly spaced vector
logical	Convert numerical values to logical
magic	Magic square
max	Largest component
min	Smallest component
norm	Matrix or vector norm
null	Null space
num2cell	Convert numeric array into cell array
num2str	Convert number to string
ones	Ones array
pascal	Pascal matrix
plot	Linear plot
poly	Convert roots to polynomial
polyval	Evaluate polynomial
rand	Uniformly distributed random numbers
randn	Normally distributed random numbers
rank	Matrix rank
reff	Reduced row echelon form
rem	Remainder after division

reshape	Change size
roots	Find polynomial roots
sin	Sine function
size	Size of matrix
sort	Sort in ascending order
subs	Symbolic substitution
sym	Construct symbolic numbers and variables
tic	Start a stopwatch timer
title	Graph title
toc	Read the stopwatch timer
toeplitz	Toeplitz matrix
tril	Extract lower triangular part
triu	Extract upper triangular part
vander	Vandermonde matrix
varargin	Variable length input argument list
zeros	Zeros array

3.2 Vectors and matrices in MATLAB

The purpose of this section is to demonstrate how to create and transform vectors and matrices in MATLAB.

This command creates a row vector

```
a = [1 2 3]
```

```
a =
    1    2    3
```

Column vectors are inputted in a similar way, however, semicolons must separate the components of a vector

```
b = [1;2;3]
```

```
b =
     1
     2
     3
```

The *quote operator* `'` is used to create the *conjugate transpose* of a vector (matrix) while the *dot-quote operator* `.'` creates the *transpose* vector (matrix). To illustrate this let us form a complex vector $\mathbf{a} + i\mathbf{b}'$ and next apply these operations to the resulting vector to obtain

```
(a+i*b)'
```

```
ans =
    1.0000 - 1.0000i
    2.0000 - 2.0000i
    3.0000 - 3.0000i
```

while

```
(a+i*b').'
```

```
ans =
    1.0000 + 1.0000i
    2.0000 + 2.0000i
    3.0000 + 3.0000i
```

Command **length** returns the number of components of a vector

```
length(a)
```

```
ans =
     3
```

The *dot operator*, plays a specific role in MATLAB. It is used for the componentwise application of the operator that follows the dot operator

```
a.*a
```

```
ans =
     1     4     9
```

The same result is obtained by applying the *power operator* $^$ to the vector

```
a.^2
```

```
ans =
     1     4     9
```

Componentwise division of vectors **a** and **b** can be accomplished by using the *backslash operator* \backslash together with the dot operator $.$

```
a.\b'
```

```
ans =
     1     1     1
```

For the purpose of the next example let us change vector **a** to the column vector

```
a = a'
```

```
a =
     1     2     3
```

The *dot product* and the *outer product* of vectors **a** and **b** are calculated as follows

```
dotprod = a'*b
```

```
dotprod = 14
```

```
outprod = a*b'
```

```
outprod =
     1     2     3
     2     4     6
     3     6     9
```

The *cross product* of two three-dimensional vectors is calculated using command `cross`. Let the vector **a** be the same as above and let

```
b = [-2 1 2];
```

Note that the semicolon after a command avoids display of the result. The cross product of **a** and **b** is

```
cp = cross(a,b)
```

```
cp =
     1    -8     5
```

The cross product vector **cp** is perpendicular to both **a** and **b** [`cp*a cp*b'`]

```
ans =
     0     0
```

We will now deal with operations on matrices. Addition, subtraction, and scalar multiplication are defined in the same way as for the vectors.

This creates a 3-by-3 matrix

```
A = [1 2 3;4 5 6;7 8 10]
```

```
A =
     1     2     3
     4     5     6
     7     8    10
```

Note that the *semicolon operator*; separates the rows. To extract a submatrix **B** consisting of rows 1 and 3 and columns 1 and 2 of the matrix **A** do the following

```
B = A([1 3],[1 2])
```

```
B =
     1     2
     7     8
```

To interchange rows 1 and 3 of **A** use the vector of row indices together with the colon operator

```
C = A([3 2 1],:)
```

```
C =
```

```

7     8    10
4     5     6
1     2     3

```

The *colon operator*: stands for *all columns* or *all rows*. For the matrix **A** from the last example the following command

```
A(:)
```

```
ans =
     1
     4
     7
     2
     5
     8
     3
     6
    10

```

creates a vector version of the matrix **A**. We will use this operator on several occasions. To delete a row (column) use the *empty vector operator*

```
A(:, 2) = []
```

```
A =
     1     3
     4     6
     7    10

```

Second column of the matrix **A** is now deleted. To insert a row (column) we use the technique for creating matrices and vectors

```
A = [A(:,1) [2 5 8]' A(:,2)]
```

```
A =
     1     2     3
     4     5     6
     7     8    10

```

Matrix **A** is now restored to its original form.

Using MATLAB commands one can easily extract those entries of a matrix that satisfy an imposed condition. Suppose that one wants to extract all entries of that are greater than one. First, we define a new matrix **A**

```
A = [-1 2 3; 0 5 1]
```

```
A =
    -1     2     3
     0     5     1

```

Command **A > 1** creates a matrix of zeros and ones

```
A > 1
```

```
ans =
```

```

0 1 1
0 1 0

```

with ones on these positions where the entries of **A** satisfy the imposed condition and zeros everywhere else. This illustrates *logical addressing* in MATLAB. To extract those entries of the matrix **A** that are greater than one we execute the following command

```
A(A > 1)
```

```
ans =
    2 5 3
```

The dot operator `.` works for matrices too. Let now

```
A = [1 2 3; 3 2 1] ;
```

The following command `A.*A`

```
ans =
    1    4    9
    9    4    1
```

computes the entry-by-entry product of **A** with **A**. However, the following command `A*A`

```
"???" Error using ==> *
Inner matrix dimensions must agree.
```

generates an error message.

Function `diag` will be used on several occasions. This creates a *diagonal matrix* with the diagonal entries stored in the vector **d**

```
d = [1 2 3];
```

```
D = diag(d)
```

```
D =
    1    0    0
    0    2    0
    0    0    3
```

To extract the main diagonal of the matrix **D** we use function `diag` again to obtain `d = diag(D)`

```
d =
    1
    2
    3
```

What is the result of executing of the following command?

```
diag(diag(d));
```

In some problems that arise in linear algebra one needs to calculate a *linear combination* of several matrices of the same dimension. In order to obtain the desired combination both the coefficients and the matrices must be stored in *cells*. In MATLAB a cell is inputted using curly braces `{ }`. This

```
c = {1,-2, 3}
```

```
c =
```

```
    [1]    [-2]    [3]
```

is an example of the cell.

3.3 Solving systems of linear equations

MATLAB has several tools needed for computing a solution of the system of linear equations.

Let \mathbf{A} be an m -by- n matrix and let \mathbf{b} be an m -dimensional (column) vector. To solve the linear system $\mathbf{Ax} = \mathbf{b}$ one can use the *backslash operator* `\`, which is also called the *left division*.

1. Case $m = n$

In this case MATLAB calculates the exact solution (modulo the roundoff errors) to the system in question.

Let

```
A = [1 2 3; 4 5 6; 7 8 10]
```

```
A =
```

```
    1    2    3
    4    5    6
    7    8   10
```

and let

```
b = ones(3,1);
```

Then

```
x = A\b
```

```
x =
```

```
-1.0000
 1.0000
 0.0000
```

In order to verify correctness of the computed solution let us compute the *residual vector* \mathbf{r}

```
r = b - A*x
```

```
r =
```

```
1.0e-015 *
```



```
0.1110
0.6661
0.2220
```

Entries of the computed residual \mathbf{r} theoretically should all be equal to zero. This example illustrates an effect of the roundoff errors on the computed solution.

2. Case $m > n$

If $m > n$, then the system $\mathbf{Ax} = \mathbf{b}$ is *overdetermined* and in most cases system is inconsistent. A solution to the system $\mathbf{Ax} = \mathbf{b}$, obtained with the aid of the backslash operator `\`, is the *least-squares solution*.

Let now

```
A = [2 -1; 1 10; 1 2];
```

and let the vector of the right-hand sides will be the same as the one in the last example. Then

```
x = A\b
```

```
x =
    0.5849
    0.0491
```

The residual \mathbf{r} of the computed solution is equal to $\mathbf{r} = \mathbf{b} - \mathbf{A}*\mathbf{x}$

```
r =
   -0.1208
   -0.0755
    0.3170
```

Theoretically the residual \mathbf{r} is orthogonal to the *column space* of \mathbf{A} . We have

```
r'*A
```

```
ans =
    1.0e-014 *
    0.1110    0.6994
```

3. Case $m < n$

If the number of unknowns exceeds the number of equations, then the linear system is *underdetermined*. In this case MATLAB computes a *particular solution* provided the system is consistent. Let now

```
A = [1 2 3; 4 5 6];
```

```
b = ones(2,1);
```

Then

```
x = A\b
```

```
x =
  -0.5000
         0
   0.5000
```

A *general solution* to the given system is obtained by forming a linear combination of \mathbf{x} with the columns of the *null space* of \mathbf{A} . The latter is computed using MATLAB function `null`

```
z = null(A)
```

```
z =
   0.4082
  -0.8165
   0.4082
```

3.4 The inverse of a matrix

MATLAB function `inv` is used to compute the inverse matrix. Let the matrix \mathbf{A} be defined as follows

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

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

Then

```
B = inv(A)
```

```
B =
 -0.6667   -1.3333    1.0000
 -0.6667    3.6667   -2.0000
  1.0000   -2.0000    1.0000
```

In order to verify that \mathbf{B} is the inverse matrix of \mathbf{A} it suffices to show that $\mathbf{A}*\mathbf{B}=\mathbf{I}$ and $\mathbf{B}*\mathbf{A}=\mathbf{I}$, where \mathbf{I} is the 3-by-3 identity matrix.

References

- [1] B.D. Hahn, *Essential MATLAB for Scientists and Engineers*, John Wiley & Sons, New York, NY, 1997.
- [2] D.R. Hill and D.E. Zitarelli, *Linear Algebra Labs with MATLAB*, Second edition, Prentice Hall, Upper Saddle River, NJ, 1996.
- [3] B. Kolman, *Introductory Linear Algebra with Applications*, Sixth edition, Prentice Hall, Upper Saddle River, NJ, 1997.
- [4] R.E. Larson and B.H. Edwards, *Elementary Linear Algebra*, Third edition, D.C. Heath and Company, Lexington, MA, 1996.
- [5] S.J. Leon, *Linear Algebra with Applications*, Fifth edition, Prentice Hall, Upper Saddle River, NJ, 1998.
- [6] G. Strang, *Linear Algebra and Its Applications*, Second edition, Academic Press, Orlando, FL, 1980.