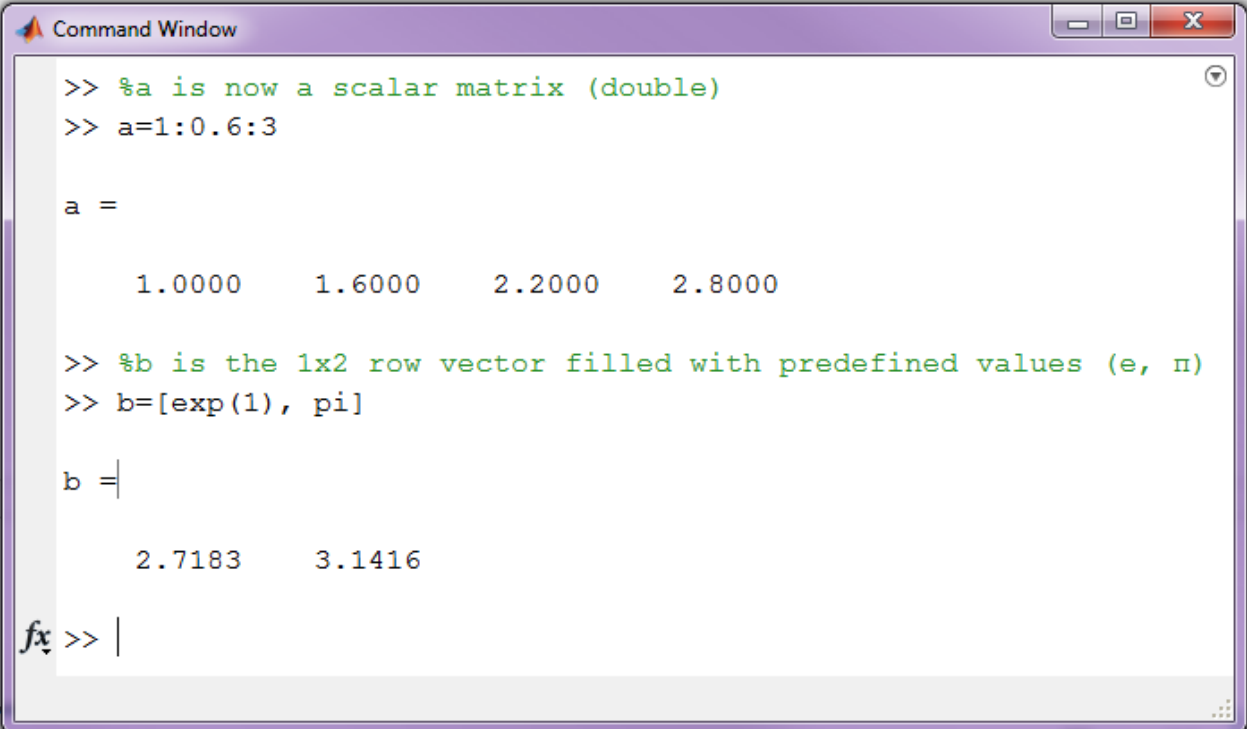


Matrix

Matlab Objects

Matlab provides users with a large variety of basic objects starting with numbers and character strings up to more sophisticated objects such as booleans, polynomials, and structures. A Matlab object is a basic object or a set of basic objects arranged in a vector, a matrix, a hypermatrix, or a structure (list).

As already mentioned, Matlab is devoted to scientific computing, and the basic object is a two-dimensional matrix with floating-point double-precision number entries. In fact, a real scalar is nothing but a 1×1 matrix. The next Matlab session illustrates this type of object. Note also that `a:b:c` gives numbers starting from `a` to `c` spaced `b` units apart.



```
Command Window
>> %a is now a scalar matrix (double)
>> a=1:0.6:3

a =

    1.0000    1.6000    2.2000    2.8000

>> %b is the 1x2 row vector filled with predefined values (e, pi)
>> b=[exp(1), pi]

b =

    2.7183    3.1416

fx >> |
```

New types have been added as Matlab has evolved, but the matrix aspect has always been kept. String matrices, boolean matrices, sparse matrices, integer matrices (`int8`, `int16`, `int32`), polynomial matrices, and rational matrices are now available in the standard Matlab environment. Complex structures called lists (`list`, `tlist`, and `mlist`) are also available. Note also that functions in Matlab are considered as objects as well.

```
Command Window
>> %a 1x1 string matrix
>> a='Matlab'

a =

Matlab

>> %a matrix
>> b=rand(2,2)

b =

    0.8147    0.1270
    0.9058    0.9134

>> %a boolean matrix
>> b= b>= 0.5

b =

     1     0
     1     1

>> %a structure implemented using mlist
>> A.x=32; A.y='t'

A =

    x: 32
    y: 't'
```

Matrix Construction and Manipulation

As already pointed out, one of the goals of Matlab is to give access to matrix operations for any kind of matrix types. In this section we highlight general functions and operators that are common to all matrix types.

A matrix in Matlab refers to one- or two-dimensional arrays, which are internally stored as a one-dimensional array (two-dimensional arrays are stored in column order). It is therefore always possible to access matrix entries with one or two indices. Vectors and scalars are stored as matrices.

Multidimensional matrices can also be used in Matlab. They are called hypermatrices.

Elementary construction operators, which are overloaded for all matrix types, are the row concatenation operator “;” and the column concatenation operator “,”. These two operators perform the concatenation operation when used in a matrix context, that is, when they appear between “[” and “]”. All the associated entries must be of the same

All the associated entries must be of the same type. Note that in the same matrix context a white space means the same thing as “,” and a line feed means the same thing as “;”. However, this equivalence can lead to confusion when, for example, a space appears inside an entry, as illustrated in the following:

```

>> %here A=[1,2,3,+5] with a unary +
>> A=[1,2,3 +5]

A =

     1     2     3     5

>> %here A=[1,2,3*5] with a binary *
>> A=[1,2,3 *5]

A =

     1     2    15

>> A=[A,0; 1,2,3,4]

A =

     1     2    15     0
     1     2     3     4
fx >> |

```

Table 2.1 describes frequently used matrix functions that can be used to create special matrices.

Table 2.1. A set of functions for creating matrices.

accumarray	Construct array with accumulation
blkdiag	Construct block diagonal matrix from input arguments
diag	Create diagonal matrix or get diagonal elements of matrix
eye	Identity matrix
false	Logical 0 (false)
freqspace	Frequency spacing for frequency response

linspace	Generate linearly spaced vector
logspace	Generate logarithmically spaced vector
meshgrid	Rectangular grid in 2-D and 3-D space
ndgrid	Rectangular grid in N-D space
ones	Create array of all ones
rand	Uniformly distributed random numbers
true	Logical 1 (true)
zeros	Create array of all zeros

These matrix functions are illustrated in the following examples:

```

Command Window
>> A= [eye(2,1), 3*ones(2,3); linspace(3,9,4); zeros(1,4)]

A =

     1     3     3     3
     0     3     3     3
     3     5     7     9
     0     0     0     0

>> %main diagonal of A as a row matrix
>> d=diag(A) '

d =

     1     3     7     0

>> %builds a diagonal matrix
>> B=diag(d)

B =

     1     0     0     0
     0     3     0     0
     0     0     7     0
     0     0     0     0

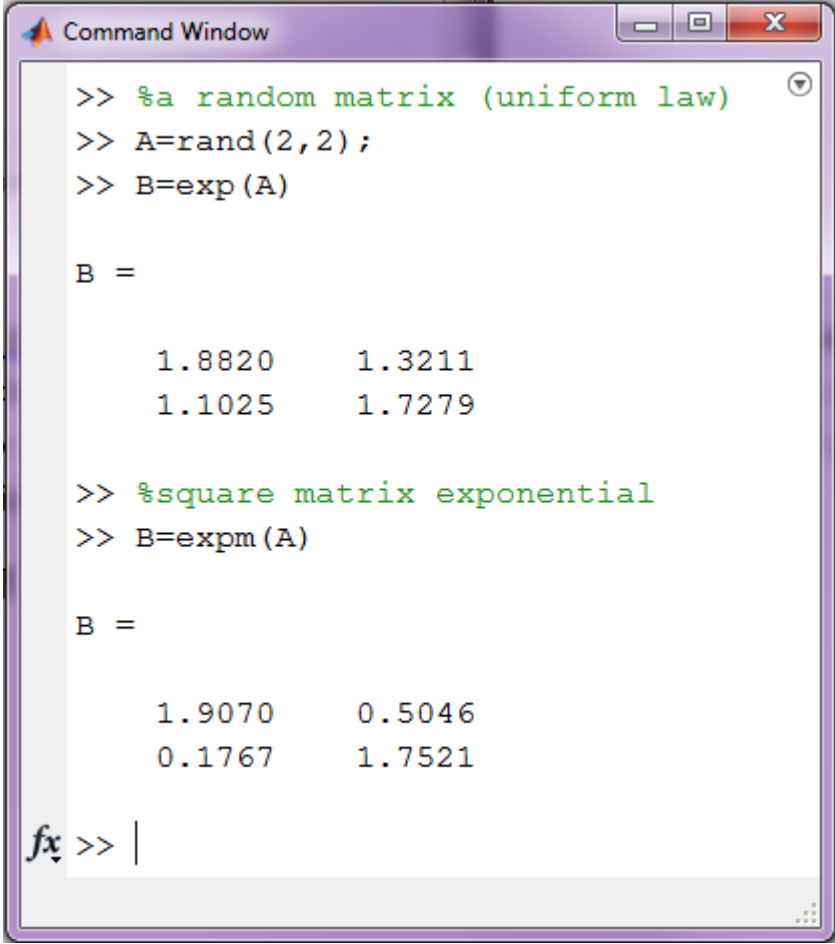
>> %reshape vector d
>> C=reshape(d,2,2)

C =

     1     7
     3     0

```

The majority of Matlab functions are implemented in such a way that they accept matrix arguments. Most of the time this is implemented by applying mathematical functions elementwise. For example, the exponential function `exp` applied to a matrix returns the elementwise exponential and differs from the important matrix exponential function `expm`.



```
Command Window

>> %a random matrix (uniform law)
>> A=rand(2,2);
>> B=exp(A)

B =

    1.8820    1.3211
    1.1025    1.7279

>> %square matrix exponential
>> B=expm(A)

B =

    1.9070    0.5046
    0.1767    1.7521

fx >> |
```

Extraction, Insertion, and Deletion

To specify a set of matrix entries for the matrix A we use the syntax $A(B)$ or $A(B,C)$, where B and C are numeric or boolean matrices that are used as indices. The interpretation of $A(B)$ and $A(B,C)$ depends on whether it is on the left- or right-hand side of an assignment $=$ if an assignment is present.

If we have $A(B)$ or $A(B,C)$ on the left, and the right-hand side evaluates to a nonnull matrix, then an assignment operation is performed. In that case the left member expression stands for a submatrix description whose entries are replaced by the ones of the right matrix entries. Of course, the right and left submatrices must have compatible sizes, that is, they must have the same size, or the right-hand-side matrix must be a scalar.

If the right-hand-side expression evaluates to an empty matrix, then the operation is a deletion operation. Entries of the left-hand-side matrix expression are deleted. Assignment or deletion can change dynamically the size of the left-hand-side matrix.

```
Command Window

>> %assigns 1 to (2,4) entry of A
>> clear ('A');
>> A(2,4)=1

A =

     0     0     0     0
     0     0     0     1

>> %assignment for changing a submatrix of A
>> A([1,2],[1,2])=round(5*rand(2,2))

A =

     1     4     0     0
     1     1     0     1

>> %submatrix deletion
>> A(:,[1 3]) = []

A =

     4     0
     1     1

>> %":" stands for all indices, here all the rows of A
>> A(:,1)= 8

A =

     8     0
     8     1
```

fx

```

Command Window
>> %deletion, "end" stands for the last index
>> A(:,end)=[]

A =

     8
     8

>> A(:,end+1)=[4;5]

A =

     8     4
     8     5

fx >> |

```

When an expression A(B) or A(B,C) is not the left member of an assignment, then it stands for a submatrix extraction and its evaluation builds a new matrix.

```

Command Window
>> %round integer part
>> A = round(10*rand(3,7));
>> %extracts a submatrix using row 1 and 3
>> %and the two last columns of A
>> A

A =

     8     6     3     3     6     7     1
     9     5     2     5     3     7     2
     1     0     8     2     7     5     9

>> B=A([1,3],end-1:end)

B =

     7     1
     5     9

fx

```

When B and C are boolean vectors and A is a numerical matrix, A(B) and A(B,C) specify a submatrix of matrix A where the indices of the submatrix are those for which B and C take the boolean value T. We shall see more on that in the section on boolean matrices.


```
Command Window
>> %transpose ' and usual matrix product *
>> A=(1:3) '*ones(1,3)

A =

     1     1     1
     2     2     2
     3     3     3

>> %multiplication tables: using a term-by-term product
>> A.*A'

ans =

     1     2     3
     2     4     6
     3     6     9

>> %term-by-term exponentiation to build
>> %a Vandermonde matrix A(i, j) = t_i^j-1
>> t=(1:3)'; m=size(t,1); n=3;
>> m

m =

     3

>> A=(t*ones(1,n+1)).^(ones(m,1)*[0:n])

A =

     1     1     1     1
     1     2     4     8
     1     3     9    27
```

```
Command Window
>> A=eye(2,2).*[1,2;3,4]
A =
     1     0
     0     4
>> A=[1,2;3,4];b=[5;6];
>> % \ for solving a linear system Ax=b.
>> x = A \ b ; norm(A*x -b)
ans =
     0
>> % underdetermined system: a solution with minimum norm is returned
>> A1=[A,zeros(2)]; x = A1 \ b
x =
   -4.0000
    4.5000
         0
         0
>> %overdetermined system: a least squared
>> %solution is returned
>> A1=[A;A]; x = A1 \ [b;7;8]
x =
   -5.0000
    5.5000
```

Symbolic matrices and operations on them

In Matlab, you can set a symbolic matrix, i.e. the matrix which-elements are string type. Remember! The string elements must be enclosed in double or single quotes.

```
Command Window
>> M=['a','b'; 'c' 'd']

M =

ab
cd

>> P=['1', '2'; '3' '4']

P =

12
34

>> P1=[1 2; 3 4]

P1 =

    1    2
    3    4
```

The Symbolic matrix can be added (the result of addition - Addition of numeric character codes) and transpose:

```
Command Window
>> M+P

ans =

    146    148
    150    152

>> M'

ans =

ac
bd
```

```
Command Window
>> S = 'computer'

S =

computer

>> X = double(S)

X =

    99    111    109    112    117    116    101    114

>> ischar(S)

ans =

    1

>> c{1,1}='My ';
>> c{1,2}='home ';
>> c{1,3}='computer ';
>> c

c =

    'My '    'home '    'computer '
```

fx >> |

```

Command Window
>> clear
>> syms a b c d
>> M=[a b; c d]

M =

[ a, b]
[ c, d]

>> syms x
>> f(x)=x

f(x) =

x

>> P=f([1 2;3 4])

P =

[ 1, 2]
[ 3, 4]

>> M+P

ans =

[ a + 1, b + 2]
fx [ c + 3, d + 4]

```

In addition, the operations of addition and multiplication can be carried out on the individual elements of the symbolic matrix by using functions `addf(a, b)` and `mulf(a, b)`:

```

Command Window
>> M(1,2)+P(2,1)

ans =

b + 3

>> M(1,1)*P(2,2)

ans =

fx 4*a

```

Solving systems of linear algebraic equations

The System of linear algebraic equations (SLAE) is the system of m equations with n unknowns and has the form:


```
Editor - P:\DeryushevaVN\MatLabR2014b\MM_matlab\Cramer.m
Cramer.m x +
1 %the coefficient matrix
2 A=[2 1 -5 1; 1 -3 0 -6; 0 2 -1 2; 1 4 -7 6];
3 b=[8; 9; -5; 0]; %Vector of free coefficients
4 A1=A; A1(:,1)=b; %The first auxiliary matrix
5 A2=A; A2(:,2)=b; %The second auxiliary matrix
6 A3=A; A3(:,3)=b; %The third auxiliary matrix
7 A4=A; A4(:,4)=b; %The fourth auxiliary matrix
8 D=det(A); %Main determinant
9 %Determinants auxiliary matrix:
10 d(1)=det(A1); d(2)=det(A2);
11 d(3)=det(A3); d(4)=det(A4);
12 x=d/D %The vector of unknowns
13 P=A*x'-b %Checking

Command Window
>> Cramer

x =

    3.0000   -4.0000   -1.0000    1.0000

P =

    1.0e-14 *
         0
         0
    -0.0888
     0.2665

fx
```