Apress ACADEMIC | MATLAB SOLUTIONS SERIES

César Pérez López

MATLAB Matrix Algebra

Springer

Apress®

Contents at a Glance

CHAPTER 1

Matrix and Vector Variables (Numeric and Symbolic)

1.1 Variables

The concept of variable, like the concept of function, is essential when working with mathematical software. Obviously, the theoretical concept of a mathematical variable is fixed and independent of the software package, but how to implement and manage variables is very characteristic of each particular program. MATLAB allows you to define and manage variables, and store them in files, in a very simple way.

When extensive calculations are performed, it is convenient to give names to intermediate results. Each intermediate result is assigned to a variable to make it easier to use. For example, we can define the variable *x* and assign the value *5* to it in the following way:

```
>> x = 5

x =

     5
```

From now on, whenever the variable *x* appears it will be replaced by the value *5*, and it will not change its value until it is redefined.

```
>> x ^ 2

ans =

    25
```

The variable *x* will not change until we explicitly assign another value to it.

```
>> x = 7 + 4

x =

    11
```

From this moment on, the variable *x* will take the value *11*.

It is very important to stress that the value assigned to a variable will remain fixed until it is expressly changed or if the current MATLAB session is closed. It is common to forget the definitions given to variables during a MATLAB session, causing misleading errors when the variables are used later in the session. For this reason, it is convenient to be able to remove the assignment of a value to a variable. This operation is performed by using the command ***clear***. It is also useful to recall the variables we have defined in the present session, which is done using the command ***who***:

- **The expression *x* = *value* assigns the value *value* to the variable *x*.**
- **The command *clear* removes the value assigned to all variables.**
- **The command *clear x* removes the value assigned to the variable *x*.**
- **The command *clear x y* removes the value assigned to the variables *x* and *y*.**
- **The command *who* gives the names of all variables currently in memory (variables in the workspace).**
- **The command *whos* gives the names, sizes, number of items, bytes occupied, and type of all variables currently in memory.**

Here are some examples that use the variable handling commands defined above:

```
>> x = 7, y = 4 + i, z = sqrt (3)

x =

     7

y =

   4.0000 + 1.0000i

z =

    1.7321

>> p=x+y+z

p =

  12.7321 + 1.0000i

>> who

Your variables are:

ans       p         x         y         z

>> whos
              Name Size     Elements  Bytes  Density  Complex

               ANS 1 by 1   1         8      Full     No
                 p 1 by 1   1         16     Full     Yes
                 x 1 by 1   1         8      Full     No
                 y 1 by 1   1         16     Full     Yes
                 z 1 by 1   1         8      Full     No

Grand total is 5 elements using 56 bytes
```

Now we are going to change the value of the variable *y*, and delete the variable *x*.

```
>> y = pi

y =

    3.1416

>> clear x;
>> whos

              Name Size     Elements  Bytes  Density  Complex

               ANS 1 by 1   1         8      Full     No
                 p 1 by 1   1         16     Full     Yes
                 y 1 by 1   1         8      Full     No
                 z 1 by 1   1         8      Full     No

Grand total is 4 elements using 40 bytes
```

We see that the variable *x* has disappeared and that the variable *y* has the new value assigned, but the variable *p* has not changed, despite having changed two of its components. ***For an expression that contains a variable whose value has been changed, to update its value it is necessary to rerun it:***

```
>> p=y+z

p =

    4.8736

>> whos
              Name Size     Elements  Bytes  Density  Complex

               ANS 1 by 1   1         8      Full     No
                 p 1 by 1   1         8      Full     No
                 y 1 by 1   1         8      Full     No
                 z 1 by 1   1         8      Full     No

Grand total is 4 elements using 32 bytes
```

Now all values are updated, including that of *p*.

As for the names that can be given to the variables, the only restriction is that they cannot start with a number or contain punctuation characters that are assigned a special meaning in MATLAB. It is also advisable to name variables with words that begin with lowercase letters, and in general with words completely in lowercase. This avoids collisions with MATLAB functions beginning with an uppercase letter. MATLAB is case sensitive. There can be any number of characters in the name of a variable, but MATLAB will handle only the first 19.

1.2 Variables and Special Constants

In many kinds of calculations we need to work with variables and special constants that the program has enabled. Here are some examples:

PI or maple ('PI'): 3.1415926535897...

i or j or maple('i'): imaginary unit (square root of - 1).

inf or maple('infinity') : Infinity, returned for example when presented with 1/0.

NaN (*Not a Number*): Indeterminate, returned for example when presented with 0/0.

realmin: the smallest usable positive real number.

realmax: the greatest usable positive real number.

finite(x): returns 1 if x is finite and zero otherwise.

isinf(x): returns 1 if x is infinity or - infinity, and zero otherwise.

isNaN(x): returns 1 if x is undetermined and zero otherwise.

isfinite(x): returns 1 if x is finite and zero otherwise.

ana: automatically creates a variable to represent the last unmapped processing result which has not been assigned to a variable.

eps: returns the distance from 1.0 to the next largest double-precision number. This is the default tolerance for floating-point operations (floating point relative accuracy). In current IEEE machines its value is 2 ^(-52).

isieee: returns 1 if the machine is IEEE and 0 otherwise.

computer: returns the type of the computer.

flops: returns the number of floating point operations that have been executed in a session (flops(0) resets the operations counter).

version: returns the current version of MATLAB.

why: returns a concise message.

cputime: returns CPU time in seconds used by MATLAB since the beginning of the session.

clock: returns a list consisting of the following 6 items: [year month day hour minutes seconds].

date: returns the current calendar date.

etime: returns the time elapsed between two *clock* type lists (defined above).

tic: enables a temporary counter in seconds that ends with the use of the variable *toc.*

toc: returns the elapsed time in seconds since the variable *tic* was activated.

LastErr: returns the last error message.

See: gives information about the program and its *Toolbox.*

Info: provides information about MATLAB.

subscribe to: gives information about the subscription to MATLAB.

whatsnew: provides information about new undocumented MATLAB features.

Here are some examples:
First we check if our computer is an IEEE machine, what type of computer it is, and find the current date and time:

```
>> isieee

ans =

     1

>> computer

ans =

PCWIN

>> clock

ans =

   1.0e + 003 *

     1.9950 0.0110 0.0140 0.0100 0.0150 0.0079

>> date

ans =

14-mar-99
```

Now we check the CPU time (in seconds) that has passed since the beginning of the MATLAB session, as well as the number of floating-point operations that have occurred during that time:

```
>> cputime

ans =

    23.5100

>> flops

ans =

  1180
```

EXERCISE 1-1

Calculate the time in seconds that the computer takes to return the irrational number π to 50 decimal places.

```
>> tic; vpa 'pi' 50; toc

elapsed_time =

          0.110000000000001
```

EXERCISE 1-2

Calculate the number of floating-point operations required to calculate the numerical value of the square root of the irrational number π to default accuracy. Consider the number π first as a numerical constant, and secondly, as a symbolic constant.

```
>> flops(0);numeric((pi)^(1/2));flops

ans =

   427

>> flops(0);numeric('(pi)^(1/2)');flops

ans =

   6
```

We see that much fewer floating-point operations are required when we consider π as a symbolic constant. The calculations are faster when we work in the symbolic field.

1.3 Symbolic and Numeric Variables

MATLAB deems as symbolic any algebraic expression whose variables have previously been defined as symbolic via the command ***syms***. For example, if we want to treat as symbolic the expression $6ab + 3a^2 + 2a$b in order to simplify it, we need to declare the two variables *a* and *b* as symbolic as shown below:

```
>> syms a b
>> simplify(6*a*b + 3*a^2 + 2*a*b)

ans =

8 * a * b + 3 * a ^ 2
```

The command ***sym*** can be used to transform a numeric expression into a symbolic expression. For example, if we want to simplify the numeric expression 2/5 + 6/10 + 8/20, we first need to transform it into a symbolic expression via *sym(2/5+6/10+8/20),* making the simplification as follows:

```
>> simplify (sym(2/5+6/10+8/20))

ans =

7/5
```

The variables contained in a symbolic expressions must be symbolic. Some commands for working with symbolic and numerical variables are described below:

syms x y z... t: makes the variables *x, y, z,..., t* symbolic.

syms x y z... t real: makes the variables *x, y, z,..., t* symbolic with real values.

syms x y z... t unreal: makes the variables *x, y, z,..., t* symbolic with non-real values.

syms: lists the symbolic variables in the workspace.

x = sym ('x'): *x* becomes a symbolic variable (equivalent to *syms x*).

x = sym ('x', real): *x* becomes a real symbolic variable.

x = sym('x',unreal): *x* becomes a symbolic non-real variable.

S = sym(A): creates a symbolic variable S from A, where A can be a string, a scalar, an array, a numeric expression, etc.

S = sym(A, 'option'): converts the array, scalar or numeric expression A to a symbolic variable S according to the specified option. The option can be 'f' for floating point, 'r' for rational, 'e' for error format and 'd' for decimal.

numeric(x): makes the variable or expression *x* numeric with double precision.

sym2poly(poly): converts the symbolic polynomial *poly* to a vector whose components are its coefficients.

poly2sym(vector): creates a symbolic polynomial whose coefficients are the components of the vector.

poly2sym(vector, 'v'): converts a symbolic polynomial in the variable *v* whose coefficients are the components of the vector.

digits(d): gives symbolic variables to an accuracy of *d* significant figures.

digits: returns the current accuracy for symbolic variables.

vpa(expr): returns the numerical result of the expression to an accuracy determined by *digits.*

vpa(expr, n): returns the numerical result of the expression to *n* significant figures.

vpa('expr', n): returns the numerical result of the expression to *n* significant figures.

pretty(expr): returns the symbolic expression in the form of standard mathematical script.

EXERCISE 1-3

Solve the equation $ax^2 + bx + c = 0$ assuming that the variable is *x*. Solve it when the variables are *a*, *b* or *c*, respectively.

Since by default MATLAB considers *x* to be the only symbolic variable, to solve the equation in *x* we don't need to declare *x* as symbolic. We simply use the command *solve* as follows:

```
>> solve('a*x^2+b*x+c=0')

ans =

[1/2/a*(-b+(b^2-4*a*c)^(1/2))]
[1/2/a*(-b-(b^2-4*a*c)^(1/2))]
```

But to solve the equation with respect to the variables *a*, *b* or *c* respectively, it is necessary to first specify them as symbolic variables:

```
>> syms a
>> solve('a*x^2+b*x+c=0',a)

ans =

-(b*x+c)/x^2

>> syms b
>> solve('a*x^2+b*x+c=0',b)

ans =

-(a*x^2+c)/x

>> syms c
>> solve('a*x^2+b*x+c=0',c)

ans =

-a * x ^ 2-b * x
```

EXERCISE 1-4

Find the roots of the polynomial $x^4 - 8x^2 + 16 = 0$ obtaining the result to default accuracy, to 20 significant figures and with double-precision. Also generate the vector of coefficients associated with the polynomial.

```
>> p = solve('x^4-8*x^2-16=0')

p =

[ 2*(2^(1/2)+1)^(1/2)]
[-2*(2^(1/2)+1)^(1/2)]
[ 2*(1-2^(1/2))^(1/2)]
[-2*(1-2^(1/2))^(1/2)]

>> vpa(p)

ans =

[     3.1075479480600746146883179061262]
[    -3.1075479480600746146883179061262]
[   1.2871885058111652494708868748364*i]
[  -1.2871885058111652494708868748364*i]

>> numeric(p)

ans =

   3.1075
  -3.1075
        0 + 1.2872i
        0 - 1.2872i

>> vpa(p,20)

ans =

[    3.1075479480600746146]
[   -3.1075479480600746146]
[  1.2871885058111652495*i]
[ -1.2871885058111652495*i]

>>  syms x
>>  sym2poly(x^4-8*x^2-16)

ans =

     1   0   -8   0   -16
```

EXERCISE 1-5

Find the numerical value to default precision of the abscissa of the intersection point in the first quadrant of the curves *y* = sin(*x*) and *y* = cos(*x*). Find the symbolic solution. Find the abscissa to 12 significant figures.

```
>> p = numeric(solve ('sin(x) = cos(x)'))

p =

    0.7854

>> q = sym(p)

q =

PI/4

>> digits(12);r = numeric(solve('sin(x)=cos(x)'))

r =

.785398163398
```

EXERCISE 1-6

Simplify the following expressions as much as possible:

1 / 2m - 1 / 3m + 1 / 4 m + 1 / 5m + 1 / 6m

1/2 - 1/3 + 1/4 + 1/5 + 1/6

```
>> syms m
>> simplify(1/(2*m) - 1/(3*m) + 1/(4*m) + 1/(5*m) + 1/(6*m))

ans =

47/60m

>> pretty(simplify(1/(2*m) - 1/(3*m) + 1/(4*m) + 1/(5*m) + 1/(6*m)))

 47
---
60m
>> sym(1/2 - 1/3 + 1/4 + 1/5 + 1/6)

ans =

47/60
```

1.4 Vector Variables

A variable that represents a vector of length *n* can be defined in MATLAB in the following ways:

```
variable = [e1, e2, e3,..., en]
variable = [e1 e2 e3... en]
```

Therefore, to define a vector variable, simply insert the vector elements between brackets separated by commas or blank spaces.

When you apply most MATLAB commands and functions to a vector variable, the result obtained is that found by applying the command or function to each element of the vector:

```
>> vector1 = [1,3,5,2.3,1/2]

vector1 =

    1.0000 3.0000 5.0000 2.3000 0.5000

>> sin(vector1)

ans =

    0.8415 0.1411 - 0.9589 0.7457 0.4794

>> exp(vector1)

ans =

    2.7183 20.0855 148.4132 9.9742 1.6487

>> log(vector1)

ans =

    0 1.0986 1.6094 0.8329 - 0.6931
```

There are different ways of defining a vector variable without explicitly bracketing all its elements, separated by commas or blank spaces.

variable = [first_element:last_element]: Defines the vector whose first and last elements are specified, and the intermediate elements differ by one unit.

variable = [first_element:increase:last_element]: Defines the vector whose first and last elements are specified, and the intermediate elements differ by the amount specified by the increase.

variable = linspace (first_element, last_element, n): Defines the vector whose first and last elements are specified, and which has in total n evenly spaced elements.

variable = logspace (a,b,n): Defines the vector whose first and last elements are 10^a and 10^b, and which has in total n evenly logarithmically spaced elements.

Here are some examples:

```
>> vector2 = [0:5:20]

vector2 =

     0  5  10  15  20
```

We have obtained the numbers between 0 and 20 separated by 5 units.

```
>> vector3 = [0:20]

vector3 =

  Columns 1 through 12

     0     1     2     3     4     5     6     7     8     9    10    11

  Columns 13 through 21

    12  13  14  15  16  17  18  19  20
```

We have obtained the numbers between 0 and 20 separated by units.

```
>> vector4 = linspace(0,10,11)

Vector4 =

     0     1     2     3     4     5     6     7     8     9    10
```

We have obtained the numbers between 0 and 10 separated by units.

```
>> vector5 = linspace (0,20,6)

vector5 =

     0  4  8  12  16  20
```

We have obtained 6 equally spaced numbers between 0 and 20.

```
>> vector6 = logspace (0,2,6)

vector6 =

    1.0000  2.5119  6.3096  15.8489  39.8107  100.0000
```

We have obtained 6 evenly logarithmically spaced numbers between 10^0 and 10^2.

We can also consider row and column vectors in MATLAB. A column vector is obtained by separating its elements by semicolons, or by transposing a row vector using a single apostrophe at the end of its definition.

```
>> a = [1;2;3;4]

a =

        1
        2
        3
        4

>> a = [1:4];b=a'

b =

        1
        2
        3
        4

>> c = (a')'

c =

        1  2  3  4
```

You can also select an element of a vector or a subset of elements.

x(n): returns the n-th element of the vector x.

x(a:b): returns the a-th through b-th elements of the vector x, both inclusive.

x(a:p:b): returns the a-th through b-th elements of the vector x, both inclusive, each separated from the next by p units (a < b).

x(b:-p:a): returns the b-th through a-th elements of the vector x, both inclusive, each separated from the next by p units and starting with the b-th (b > a).

Here are some examples:

```
>> x = (1:10)

x =

        1      2      3      4      5      6      7      8      9     10

>> x(6)

ans =

        6
```

We have obtained the sixth element of the vector *x*.

```
>> x(4:7)

ans =

    4 5 6 7
```

We have obtained the elements of the vector *x* located between the fourth and the seventh elements, both inclusive.

```
>> x(2:3:9)

ans =

    2 5 8
```

We have obtained the elements of the vector *x* located between the second and ninth elements, both inclusive, but separated from each other by three units.

```
>> x(9:-3:2)

ans =

    9 6 3
```

We have obtained the elements of the vector *x* located between the ninth and second elements, both inclusive, but separated from each other by three units and starting at the ninth.

Simple mathematical operations between scalars and vectors scale each element of the vector according to the defined operation, and simple operations between vectors are performed elementwise.

Below is a summary of these operations:

a = {a1, a2,..., an}, b = {b1, b2,..., bn}, c = scalar

a + c = [a1 + c, a2 + c,..., an + c]: sum of a scalar and a vector

a * c = [a1 * c, a2 * c,..., an * c]: product of a scalar and a vector

a + b = [a1 + b1, a2 + b2,... an + bn]: sum of two vectors

a. * b = [a1 * b1, a2 * b2,... , an * bn]: product of two vectors

a. / b = [a1/b1 a2/b2... an/bn]: right ratio of two vectors

a. \ b = [a1\b1 a2\b2... an\bn]: left ratio of two vectors

a. ^ c = [a1 ^ c, a2 ^ c,..., an ^ c]: scalar power of a vector

c. ^ a = [c ^ a1, c ^ a2,... ,c ^ an]: vector power of a scalar

a. ^ b = [a1 ^ b1, a2 ^ b2,... ,an ^ bn]: vector power of a vector

It must be borne in mind that the vectors must be of the same length, and that in the product, quotient, and power the first operand is followed by a point.

On the other hand, you can also set the vector variables to be symbolic using the command *syms*.

```
>> syms t
>> A=sym([sin(t),cos(t)])

A =

[sin (t), cos (t)]
```

EXERCISE 1-7

Given the vector variables a = [π, 2π, 3π, 4π, 5π] and b = [e, 2e, 3e, 4e, 5e] calculate c = sin (a) + b, d = cos (a), e = ln (b), f = c * d, g = c/d, h = d ^ 2, i = d ^ 2-e ^ 2 and j = 3d ^ 3-2e ^ 2.

```
>> a = [pi, 2 * pi, 3 * pi, 4 * pi, 5 * pi], b = [exp (1), 2 * exp (1), 3 * exp (1), 4 * exp (1),
5 * exp (1)], c=sin(a)+b, d=cos(a), e=log(b), f=c.*d, g=c./d, h=d.^2, i=d.^2-e.^2, j = 3 * d.
^ 3-2 * e ^ 2

a =

    3.1416 6.2832 9.4248 12.5664 15.7080

b =

    2.7183 5.4366 8.1548 10.8731 13.5914

c =

    2.7183 5.4366 8.1548 10.8731 13.5914

d =

    -1      1     -1      1     -1
e =

    1.0000 1.6931 2.0986 2.3863 2.6094

f =

   -2.7183 5.4366 - 8.1548 10.8731 - 13.5914

g =

   -2.7183 5.4366 - 8.1548 10.8731 - 13.5914

h =

     1      1      1      1      1
```

```
i =

0 - 1.8667 - 3.4042 - 4.6944 - 5.8092

j =

   -5.0000 - 2.7335 - 11.8083 - 8.3888 - 16.6183
```

1.5 Matrix Variables

MATLAB defines arrays by inserting in brackets all its row vectors separated by a semicolon. Vectors can be entered by separating their components by spaces or by commas, as we already know. For example, a 3 × 3 matrix variable can be entered in the following two ways:

M = [a_{11} a_{12} a_{13}; a_{21} a_{22} a_{23}; a_{31} a_{32} a_{33}]
M = [a_{11},a_{12} ,a_{13};a_{21} ,a_{22} ,a_{23};a_{31},a_{32},a_{33}]

Similarly we can define an array of variable dimension *(M×N)*. Once a matrix variable has been defined, MATLAB enables many ways to insert, extract, renumber, and generally manipulate its elements. The following table shows different ways to define matrix variables.

A(m,n)	*Defines the (m, n)-th element of the matrix A (row m and column n).*
A(a:b,c:d)	*Defines the subarray of A formed between the a-th and the b-th rows and between the c-th and the d-th columns, inclusive.*
A(a:p:b,c:q:d)	*Defines the subarray of A formed by every p-th row between the a-th and the b-th rows, inclusive, and every q-th column between the c-th and the d-th columns, inclusive.*
A([a b],[c d])	*Defines the subarray of A formed by the intersection of the a-th through b-th rows and c-th through d-th columns, inclusive.*
A([a b c...],[e f g...])	*Defines the subarray of A formed by the intersection of rows a, b, c,...and columns e, f, g,...*
A(:,c:d)	*Defines the subarray of A formed by all the rows in A and the c-th through to the d-th columns.*
A(:,[c d e...])	*Defines the subarray of A formed by all the rows in A and columns c, d, e,...*
A(a:b,:)	*Defines the subarray of A formed by all the columns in A and the a-th through to the b-th rows.*
A([a b c...],:)	*Defines the subarray of A formed by all the columns in A and rows a, b, c,...*
A(a,:)	*Defines the a-th row of the matrix A.*
A(:,b)	*Defines the b-th column of the matrix A.*
A(:)	*Defines a column vector whose elements are the columns of A placed in order below each other.*
A(:,:)	*This is equivalent to the entire matrix A.*
[A, B, C,...]	*Defines the matrix formed by the matrices A, B, C,...*
S_A = [ ]	*Clears the subarray of the matrix A, SA, and returns the remainder.*

(*continued*)

diag (v)	*Creates a diagonal matrix with the vector v in the diagonal.*
diag (A)	*Extracts the diagonal of the matrix as a column vector.*
eye (n)	*Creates the identity matrix of order n.*
eye (m, n)	*Creates an m×n matrix with ones on the main diagonal and zeros elsewhere.*
zeros (m, n)	*Creates the zero matrix of order m×n.*
ones (m, n)	*Creates the matrix of order m×n with all its elements equal to 1.*
rand (m, n)	*Creates a uniform random matrix of order m×n.*
randn (m, n)	*Creates a normal random matrix of order m×n.*
flipud (A)	*Returns the matrix whose rows are those of A but placed in reverse order (from top to bottom).*
fliplr (A)	*Returns the matrix whose columns are those of A but placed in reverse order (from left to right).*
rot90 (A)	*Rotates the matrix A counterclockwise by 90 degrees.*
reshape(A,m,n)	*Returns an m×n matrix formed by taking consecutive entries of A by columns.*
size (A)	*Returns the order (size) of the matrix A.*
find (cond$_A$)	*Returns all A items that meet a given condition.*
length (v)	*Returns the length of the vector v.*
tril (A)	*Returns the lower triangular part of the matrix A.*
triu (A)	*Returns the upper triangular part of the matrix A.*
A'	*Returns the transpose of the matrix A.*
Inv (A)	*Returns the inverse of the matrix A.*

Here are some examples:
We consider first the *2 × 3* matrix whose rows are the first six consecutive odd numbers:

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

A =

1 3 5
7 9 11
```

Now we are going to change the *(2,3)-th* element, i.e. the last element of *A*, to zero:

```
>> A(2,3) = 0

A =

1 3 5
7 9 0
```

We now define the matrix *B* to be the transpose of *A*:

```
>> B = A'

B =

1  7
3  9
5  0
```

We now construct a matrix *C*, formed by attaching the identity matrix of order 3 to the right of the matrix *B*:

```
>> C = [B eye (3)]

C =

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

We are going to build a matrix *D* by extracting the odd columns of the matrix *C*, a matrix *E* formed by taking the intersection of the first two rows of *C* and its third and fifth columns, and a matrix *F* formed by taking the intersection of the first two rows and the last three columns of the matrix *C*:

```
>> D = C(:,1:2:5)

D =

1  1  0
3  0  0
5  0  1

>> E = C([1 2],[3 5])

E =

1  0
0  0

>> F = C([1 2],3:5)

F =

1  0  0
0  1  0
```

Now we build the diagonal matrix *G* such that the elements of the main diagonal are the same as those of the main diagonal of *D*:

```
>> G = diag(diag(D))

G =

1  0  0
0  0  0
0  0  1
```

We then build the matrix *H*, formed by taking the intersection of the first and third rows of *C* and its second, third and fifth columns:

```
>> H = C([1 3],[2 3 5])

H =

7  1  0
0  0  1
```

Now we build an array *I* formed by the identity matrix of order *5* × *4*, appending the zero matrix of the same order to its right and to the right of that, the unit matrix, again of the same order. Then we extract the first row of *I* and, finally, form the matrix *J* comprising the odd rows and even columns of *I* and calculate its order (size).

```
>> I = [eye(5,4) zeros(5,4) ones(5,4)]

ans =

1     0     0     0     0     0     0     0     1     1     1     1
0     1     0     0     0     0     0     0     1     1     1     1
0     0     1     0     0     0     0     0     1     1     1     1
0     0     0     1     0     0     0     0     1     1     1     1
0     0     0     0     0     0     0     0     1     1     1     1

>> I(1,:)

ans =

1     0     0     0     0     0     0     0     1     1     1     1

>> J = I(1:2:5,2:2:12)

J =

0     0     0     0     1     1
0     0     0     0     1     1
0     0     0     0     1     1

>> size(J)

ans =

3  6
```

We now construct a random matrix *K* of order *3* ×*4*, reverse the order of the rows of *K*, reverse the order of the columns of *K* and then perform both operations simultaneously. Finally, we find the matrix *L* of order *4* × *3* whose columns are obtained by taking the elements of *K* sequentially by columns.

```
>> K = rand(3,4)

K =

0.5269    0.4160    0.7622    0.7361
0.0920    0.7012    0.2625    0.3282
0.6539    0.9103    0.0475    0.6326

>> K(3:-1:1,:)

ans =

0.6539    0.9103    0.0475    0.6326
0.0920    0.7012    0.2625    0.3282
0.5269    0.4160    0.7622    0.7361

>> K(:,4:-1:1)

ans =

0.7361    0.7622    0.4160    0.5269
0.3282    0.2625    0.7012    0.0920
0.6326    0.0475    0.9103    0.6539

>> K(3:-1:1,4:-1:1)

ans =

0.6326    0.0475    0.9103    0.6539
0.3282    0.2625    0.7012    0.0920
0.7361    0.7622    0.4160    0.5269

>> L = reshape(K,4,3)

L =

0.5269 0.7012 0.0475
0.0920 0.9103 0.7361
0.6539 0.7622 0.3282
0.4160 0.2625 0.6326
```

EXERCISE 1-8

Given the square matrix of order 3 whose entries are the first nine natural numbers, find its inverse, its transpose and its diagonal. Transform it into a lower triangular matrix and an upper triangular matrix and rotate it by 90 degrees counterclockwise. Find the sum of the elements in the first row and the sum of the diagonal elements. Extract the subarray whose diagonal is formed by the elements at $_{11}$ and $_{22}$ and also remove the subarray whose diagonal elements are at $_{11}$ and $_{33}$.

```
>> M = [1,2,3;4,5,6;7,8,9]

M =

     1     2     3
     4     5     6
     7     8     9

>> A = inv(M)
```

Warning: Matrix is close to singular or badly scaled.

Results may be inaccurate. RCOND = 2.937385e-018

```
A =

  1.0e + 016 *

    0.3152 - 0.6304   0.3152
   -0.6304   1.2609 - 0.6304
    0.3152 - 0.6304   0.3152

>> B = M'

B =

     1 4 7
     2 5 8
     3 6 9

>> V = diag(M)

V =

     1
     5
     9
```