MA1710: Week 5

Introduction to using matrices in Matlab

5.1 Introduction to the week 5 session

A matrix is the default type in Matlab and the name Matlab is taken from the first 3 letters of "Matrix Laboratory". It is thus not too surprising that there are a lot of features of the language connected with working with matrices. In this session we consider a few of these and in a few places take as examples things that you do in `MA1720` (Linear Algebra).

5.2 Constructing matrices with `[` and `]`

Row vectors and column vectors were introduced in session 3 and it was remarked then that they are a special case of a matrix. In that session the use of `[` and `]` was described to set-up vectors with all the entries given. We can do the same for a more general matrix and to experiment with what is possible use the editor to create the file `sess05a.m` which contains the following lines and run the file. (The numbers are from question 1 of your week 4 `MA1720` sheet.)

```
format compact
clear
A = [2 -3 1; 3 4 -5; 5 -1 -3]
b = [0; 2; 1]
C = [A, b]
whos
```

As there are no semi-colons at the end of each statement the output is as as follows.

```
A =
     2    -3     1
     3     4    -5
     5    -1    -3
b =
     0
     2
     1
C =
     2    -3     1     0
     3     4    -5     2
     5    -1    -3     1
  Name      Size            Bytes  Class     Attributes

  A         3x3                72  double
  C         3x4                96  double
  b         3x1                24  double
```

Here we have created a 3×3 matrix called `A`, a 3×1 column vector called `b` and a 3×4 matrix called `C`. The entries in each row are separated by spaces (spaces or commas can be used) and the semi-colon indicates the end of a row. As `A` and `b` both have the same number of rows the blocks can be used to create `C` which contains all the numbers involved. The creation of `C` involves a "row vector" with entries which are matrices.

5.3 Using part of a matrix and changing part of the matrix

In the previous example the matrix `C` has 3 rows and 4 columns and we can refer to the $3, 2$ entry by putting

```
C(3, 2)
```

Try this to check that you get `-1`. We can also refer to an entire row or column and to illustrate this add the following lines to `sess05a.m` and save the file as `sess05b.m` .

```
y = C(2, :)
z = C(:, 3)
```

The output when you run it as result of the additional statements is as follows.

```
y =
     3     4    -5     2
z =
     1
    -5
    -3
```

When `:` is given for the row this means all rows and similarly when `:` is given for the column then this means all columns. Thus `y` is everything on row 2 of `C` and `z` is everything in column 3 of `C`.

As well as being able to refer to an entry or a rectangular block of entries we can also change that part of the matrix with an assignment statement. To illustrate this try the following to swap rows 2 and 3.

```
y = C(2, :);
C(2, :) = C(3, :);
C(3, :) = y;
C
```

The output shown for `C` should now be as follows.

```
C =
     2    -3     1     0
     5    -1    -3     1
     3     4    -5     2
```

In fact the swapping operation can be done with just one statement although we first recover the original matrix.

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

To understand this version try typing

```
C([3, 2], :)
```

This gives a 2×3 quantity involving row 3 followed by row 2, i.e. the list of rows is specified by `[3, 2]`. The swapping is achieved by the assignment statement which saves these in the rows `[2, 3]` of `C`.

5.4 Further comments about the shape of a matrix and the storage of a matrix

In the examples given so far we have referred to the entries of a vector with 1 index value and in the matrix examples of this session we have used two index values. It is not actually a requirement that this is done as we only need to give something which is in the range of what is stored and all quantities are actually stored in a ‘one dimensional way’. To illustrate what this means for a matrix use the editor to create `sess05c.m` which contains the following and run it

```
A = [2 -3 1; 3 4 -5; 5 -1 -3];
b = [0; 2; 1];
C = [A, b]
y = C(2, :)
y = y(:)
d = C(:)
```

This generates the following output.

```
C =
     2    -3     1     0
     3     4    -5     2
     5    -1    -3     1
y =
     3     4    -5     2
y =
     3
     4
    -5
     2
d =
     2
     3
     5
    -3
     4
    -1
     1
    -5
    -3
     0
     2
     1
```

The vector `y` is first set-up as a row vector and then the statement `y=y(:)` changes it to a column vector. The second statement for `y` is actually quite useful as it is a means of changing the shape to a column vector whatever the shape was originally. We similarly get a column vector for `d` and it is all the entries of `C` in the order that they are stored and this is column-by-column. This is mentioned here as it is valid to put

```
C(5)
```

which gives the 5th entry in how `C` is stored which in this case is the same as putting `C(2, 2)`.

5.5 Entry-wise operations

As in the case of vectors you can do entry-wise operations which are denoted in Matlab by `.^`, `.*` and `./` for the entry-wise power, multiplication and division respectively. To illustrate this use the editor to create `sess05d.m` which contains the following and run it.

```
A = [2 -3 1; 3 4 -5; 5 -1 -3];
b = [0; 2; 1];
C = [A, b]
C2 = C.^2
CC = C.*C
```

The output is as follows with each entry of `C2` and `CC` being the square of the corresponding entry of `C`.

```
C =
     2    -3     1     0
     3     4    -5     2
     5    -1    -3     1
C2 =
     4     9     1     0
     9    16    25     4
    25     1     9     1
CC =
     4     9     1     0
     9    16    25     4
    25     1     9     1
```

Entry-wise operations involving matrices are useful for setting up data for plotting, for example for surface plots, but we do not consider any examples here.

5.6 Matrix operations

You can multiply a matrix by a scalar, you can add a number to every entry and, as in the case of vectors, if two matrices have the same shape then you can add or subtract them. Matrices of compatible sizes can also be multiplied and this is what the operation `*` means when the quantities are matrices. To illustrate this use the editor to create `sess05e.m` which contains the following and run it.

```
A = [2 -3 1; 3 4 -5; 5 -1 -3]
B = 5+2*A
x = [1; 1; 1]
z = A*x
AB = A*B
BA = B*A
```

The output from the statements is as follows.

```
A =
     2    -3     1
     3     4    -5
     5    -1    -3
B =
     9    -1     7
    11    13    -5
    15     3    -1
x =
     1
     1
     1
z =
     0
     2
     1
AB =
     0   -38    28
    -4    34     6
   -11   -27    43
BA =
    50   -38    -7
    36    24   -39
    34   -32     3
```

To explain some of these note that as A has size 3×3 and x has size 3×1 this generates a 3×1 matrix for z, i.e. z is a column vector. You might also note here that z is the same as b used in the earlier exercises and hence x is a solution (actually the unique solution) of the linear equations

$$A\underline{x} = \underline{b}, \quad \text{when} \quad A = \begin{pmatrix} 2 & -3 & 1 \\ 3 & 4 & -5 \\ 5 & -1 & -3 \end{pmatrix} \quad \text{and} \quad \underline{b} = \begin{pmatrix} 0 \\ 2 \\ 1 \end{pmatrix}.$$

The outputs for AB and BA demonstrate that matrix multiplication is not commutative, i.e. in general the products *AB* and *BA* are different. If you wish to check by hand that the products are correct then you can use Matlab to assist in individual computations. For example to get the $3,2$ entry in the product AB we use the 3rd row of A and the 2nd column of B. In Matlab this is the calculation

```
A(3, :)*B(:, 2)
```

Likewise the $3,2$ entry in the product BA is given by

```
B(3, :)*A(:, 2)
```

In both these cases we take a complete row of one of the matrices and the complete column of the other and a row times a column gives a 1×1 matrix, i.e. just a single number. Further, try the following statements.

```
y = [1 2 3]'
y'*y
y*y'
```

The output is as follows.

```
y =
     1
     2
     3
ans =
    14
ans =
     1     2     3
     2     4     6
     3     6     9
```

Here `y'` is the transpose of `y` and thus as `y` is a column vector the transpose is a row vector. The outputs demonstrate again that a 1×3 vector times a 3×1 vector gives a 1×1 matrix and that it also makes sense to multiply in the reverse order to create a 3×3 matrix by multiplying a 3×1 vector with a 1×3 vector.

5.7 Solving linear equations using \

In your linear algebra lectures and seminars you consider techniques to solve linear equations $A\underline{x} = \underline{b}$ involving a square matrix A with the size of A being small (typically 3×3). In addition you also consider cases when you have more equations than unknowns and also cases when there are less equations than unknowns. Matlab can help you with these problems to the extent of checking your final answer and we consider in this section the case when the matrix A is a square matrix. If A is a $n \times n$ non-singular matrix (i.e. an invertible matrix) and $\underline{b}$ is a $n \times 1$ column vector then there exists a unique solution to

$$A\underline{x} = \underline{b}.$$

If in Matlab notation `A` and `b` correspond to A and $\underline{b}$ respectively (we have only changed the fonts used in the description with an italic type font used to display the mathematics and with a typewriter font for the terms used in the programs) then we can obtain `x` by just putting

```
x = A\b
```

To test this on the matrices given in section 5.2 and on other similar examples use the editor to create the file `sess05f.m` which contains the following and run it.

```
A = [2 -3 1; 3 4 -5; 5 -1 -3]
b = [0; 2; 1]
x = A\b
d = b-A*x

A4 = [3 5 -1 2;  5 -1 2 3;  1 3 2 -2;  1 1 1 1]
b4 = [1 4 3 3]'
x4 = A4\b4
d4 = b4-A4*x4
```

The output is as follows.

```
A =
     2    -3     1
     3     4    -5
     5    -1    -3
b =
     0
     2
     1
x =
    1.0000
    1.0000
    1.0000
d =
   1.0e-15 *
   -0.1110
         0
         0
A4 =
     3     5    -1     2
     5    -1     2     3
     1     3     2    -2
     1     1     1     1
b4 =
     1
     4
     3
     3
x4 =
   -0.5000
    0.5000
    2.0000
    1.0000
d4 =
   1.0e-15 *
   -0.4441
         0
   -0.8882
   -0.8882
```

You may have expected `d` and `d4` to both be zero vectors to confirm that respectively `x` and `x4` solve the systems. The reason this is not the case is that floating point arithmetic is used and we get rounding errors and there are some division operations in the computations. The entries in `d` and `d4` are of course very small.

5.8 Some special matrices: `zeros`, `ones` and `eye`

There are several functions in Matlab which generate matrices and we just consider three of these here. To illustrate what these generate create the file `sess05g.m` containing the following and run it.

```
Z = zeros(3, 5)
Z3 = zeros(3)
O = ones(3)
I3 = eye(3)
I4 = eye(4, 4)
```

This generates the following output.

```
Z =
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
Z3 =
     0     0     0
     0     0     0
     0     0     0
O =
     1     1     1
     1     1     1
     1     1     1
I3 =
     1     0     0
     0     1     0
     0     0     1
I4 =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

In the case of `Z` and `I4` the shape is specified by the arguments. In the case of `Z3`, `O` and `I3` we get a square matrix and not a row or column vector. The function `eye` gives the identity matrix and in your linear algebra module you will we usually write this as I.

5.9 Some matrix functions: `size`, `det`, `inv` and `rref`

For the last part of this session (before some exercises) we illustrate a few functions which relate to things that you do in the first part of your linear algebra module. Use the editor to create `sess05h.m` which contains the following and run it.

```
C = [2 -3 1 -1;  3 4 -5 -4;  5 -1 -3 -6]
s3 = size(C)
A3 = C(:, 1:3)
d3 = det(A3)
A3I = inv(A3)
A3j = d3*A3I
rC = rref(C)
E = A3*A3I-eye( size(A3) )
```

The output generated is shown below.

```
C =
     2    -3     1    -1
     3     4    -5    -4
     5    -1    -3    -6
s3 =
     3     4
A3 =
     2    -3     1
     3     4    -5
     5    -1    -3
d3 =
   -9.0000
A3I =
    1.8889    1.1111   -1.2222
    1.7778    1.2222   -1.4444
    2.5556    1.4444   -1.8889
A3j =
  -17.0000  -10.0000   11.0000
  -16.0000  -11.0000   13.0000
  -23.0000  -13.0000   17.0000
rC =
     1     0     0     1
     0     1     0     2
     0     0     1     3
E =
   1.0e-15 *
   -0.4441   -0.2220         0
         0         0         0
    0.8882         0    0.8882
```

The output shows that the function `size` gives a row vector containing the number of rows and the number of columns. This is used in the statement for `E` so that we get an identity matrix of the same size as `A3`. The function `det` gives the determinant of the matrix.

The function `inv` gives the inverse of the matrix and the statement for `A3j` is one way to generate the adjoint matrix. The function `rref` generates the reduced row echelon form and in this case the last column is the unique solution to the linear system with `A3` as the matrix and `C(:, 4)` as the right hand side vector. To confirm this add the following two lines.

```
b3 = C(:, 4)
x3 = A3\b3
```

To just solve a linear system when there is a unique solution you only need to use \ as described in section 5.7 and this is the recommended thing to do in Matlab. When the problem does not have a unique solution then the output from `rref` helps describes the answer.

5.10 Exercises

Exercise 5.10.1

Use Matlab to get the solution of the following equations.

$$\begin{pmatrix} 2 & -3 & 1 \\ 3 & 4 & -5 \\ 5 & 1 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \\ 1 \end{pmatrix}.$$

To show the output in rational number format you should type the following statement.

```
format rat
```

and later to return to the more usual format for showing the output you need to put

```
format short
```

Exercise 5.10.2

Use the function `rref` to determine if the following systems have a solution and when there is a solution describe all possible solutions.

(a)

$$\begin{pmatrix} 2 & -3 & 1 \\ 3 & 4 & -5 \\ 5 & 1 & -4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \\ 2 \end{pmatrix}.$$

(b)

$$\begin{pmatrix} 3 & 5 & -1 & 2 \\ 5 & -1 & 2 & 3 \\ 1 & 11 & -4 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 4 \\ 2 \end{pmatrix}.$$

Exercise 5.10.3

Matrix multiplication is associative, i.e. provided matrices A, B and C have compatible sizes for the matrix products to exists then

$$A(BC) = (AB)C$$

and we we can just write ABC. The outcome is independent of which matrix multiplication we do first. Try the following to verify that this is true for a few randomly generated matrices with entries which are integers.

```
rand('seed', 1)
for k = 1:3
  fprintf('\n...test when k = %d\n', k);
  A = randi([1, 4], 3, 3)
  B = randi([1, 4], 3, 3)
  C = randi([1, 4], 3, 3)
  M1 = A*(B*C)
  M2 = (A*B)*C
  dif = M1-M2
end
```

Here the line which contains `seed` is so that we get the same matrices each time the script is run. The statements with `randi` generate 3×3 matrices (the last 2 arguments) with the entries being integers between 1 and 4.