Maths Skills (MTCS) G5071

Lecture 6

Kingsley Sage
Room 5C16, Pevensey III
khs20@sussex.ac.uk

Lecture 6

- Using MATLAB matrices and vectors.
- MATLAB *matrix* is a rectangular array of numbers, 1 by 1 matrices are *scalars* and matrices with only one row or column are *vectors*.

Using MATLAB vectors and matrices

- You can enter matrices into MATLAB in several different ways:
  - Enter an explicit list of elements
  - Load matrices from external data files
  - Generate matrices using built-in functions
  - Create matrices with your own functions in M-files
- Start by entering a matrix as a list of elements. You have only to follow a few basic conventions:
  - Separate the elements of a row with blanks or commas
  - Use a semi-colon, ; , to indicate the end of a row
  - Surround the entire list of elements with square brackets, [ ]

Dürer's matrIx

- An example matrix appears in Renaissance engraving Melancholia I by the German artists and amateur mathematician Albrecht Dürer. To enter Dürer's matrix, simply type:

```
A = [16 3 2 13; 5 10 11 8, 9 6 7 12; 4 15 14 1]
```

- MATLAB displays the matrix you just entered:

```
A =
    16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1
```

Dürer's matrix

- Once you have entered the matrix, it is automatically remembered in the MATLAB workspace. You can refer to it simply as ***A***.
- Now that we have ***A*** in the workspace, take a look at what makes it so interesting. Why is it magic?
- The special properties of a magic square have to do with the ways of summing its elements.
- They are related to Sudoko puzzles.

Matrix operations

- If you take the sum along any row or column, or along either of the two main diagonals, you will always get the same number.
- The first MATLAB statement to try is:

```
>> sum(A)

ans =
   34  34  34  34
```

- When you don't specify an output variable, MATLAB uses the variable `ans`, short for answer, to store the results of a calculation.

Matrix operations

- You have computed a row vector containing the sum of the columns of ***A*** and each of the columns has the same sum, the magic sum, 34.
- How about the row sums? MATLAB has a preference for working with the columns of a matrix.
- The easiest way to get the row sums is to transpose the matrix, compute the column sums of the transpose, and then transpose the result.

Transpose matrices

- The transpose operation is denoted by the apostrophe or single quote, ‘. It flips a matrix about its main diagonal and it turns a row vector into a column vector and vice versa. So:

```
>> A'

ans =
   16   5   9   4
    3  10   6  15
    2  11   7  14
   13   8  12   1
```

```
>> sum(A')'

ans =
    34
    34
    34
    34
```

Diag function

- The sum of the elements on the main diagonal is easily obtained with the help of the `diag` function that picks off that diagonal:

```
>> diag A

ans =
   16
   10
    7
    1
```

```
>> sum(diag(A))

ans =
    34

>> sum(diag(fliplr(A)))

ans =
    34
```

- The other diagonal (the anti-diagonal) is not important mathematically, but a function originally intended for use in graphics, `fliplr`, flips a matrix from left to right.

Subscripts

- The element in row *i* and column *j* of ***A*** is denoted by *A(i,j)*. For example, *A(4,2)* is the number in the fourth row and second column.
- For our magic square, *A(4,2)* is 14. So it is possible to compute the sum of the elements in the fourth column of ***A*** by:

```
>> A(1,4)+A(2,4)+A(3,4)+A(4,4)

ans =
    34
```

- It is also possible to refer to elements of a matrix with a single subscript, A(k) – see what it does …

Subscripts

- We can also apply the colon operator, : , to a two-dimensional matrix, in which case the array is regarded as one long column vector formed from the columns of the original matrix.
- So for our magic square, *A(8)* is another way of referring to the value 15 stored in *A(4,2)* and *A(:)* is:

  A = [16; 5; 9; 4; 3; 10; 6; 15; 2; 11; 7; 14; 13; 8; 12; 1]

- MATLAB actually has a built-in function that creates magic squares.

Errors

- If you try to use the value of an element outside of a matrix, it is an error:

```
>> t=A(4,5)

  Index exceeds matrix dimensions
```

- On the other hand, if you store a value in an element outside of the matrix, the size increases to accommodate the newcomer.

Colon operator

- The colon operator, : , is one of MATLAB's most important operators. It occurs in several different forms. The expression *1:10* is a row vector containing the integers from 1 to 10

```
>> 1:10

ans =
   1  2  3  4  5  6  7  8  9  10
```

- To obtain non unit spacing, specify an increment. For example:

```
>> 100:-7:50

ans =
   100  93  86  79  72  65  58  51
```

Generate matrices using built-in functions

- MATLAB provides four functions that generate basic matrices:
  - `zeros` – all zeros
  - `ones` – all ones
  - `rand` – Uniformly distributed random elements
  - `randn` – Normally distributed random elements
- Some examples:

```
>> F=5*ones(3,3)

  F =
     5  5  5
     5  5  5
     5  5  5
```

```
>> R=randn(4,4)

  R =
    1.0668   0.2944 -0.6918 -1.4410
    0.0593 -1.3362  0.8580  0.5711
   -0.0956  0.7143  1.2540 -0.3999
   -0.8323  1.6236 -1.5937  0.6900
```

Next time …

- Vector applications
  - Velocity fields
  - Flow patterns
  - Co-ordinate systems