Introduction to Modelling

2. Logical Vectors and Arrays

Dr. Jim Duggan,
School of Engineering & Informatics
National University of Ireland Galway.

https://github.com/JimDuggan/CT248

Overview

- Element-wise operators
- Logical Vectors
- Matrices

(1) Element-Wise Operations

- Matlab is expecting that dimensions of our matrices agree in some linear algebra sense.
- But what we want is to apply our operation to each element of the array.
- The “dot operator” is used for this

```
>> A

A =

    2    4    6    8

>>
>> 1/A
Error using  /
Matrix dimensions must agree.

>>
>> 1./A

ans =

    0.5000    0.2500    0.1667    0.1250
```

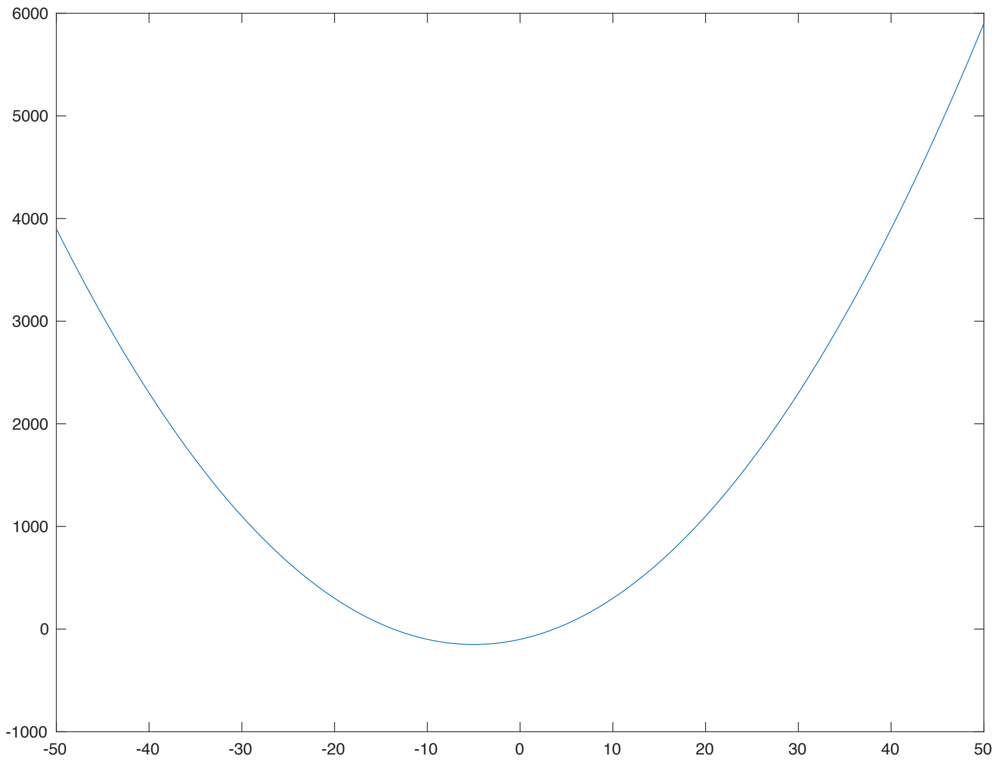
https://math.boisestate.edu/~calhoun/teaching/matlab-tutorials/lab_31/html/lab_31.html

Rules for the dot operator

- **Multiplication:** If both expressions on either side of the mutiplication symbol are arrays, then use the **.*** operator. If one of the expressions is a scalar, then no dot is needed.
- **Division:** If the numerator is a scalar and the denominator is an array, use the **./** operator. If both the numerator and the denominator are arrays, also use the **./** operator. If the numerator is an array, and the denominator is a scalar, then no dot is needed.
- **Exponentiation:** If either the base or the power (or both) is an array, use the **.^** operator. If neither is an array, then no dot is needed.
- **Addition and subtraction:** Dots are never used and are not allowed.

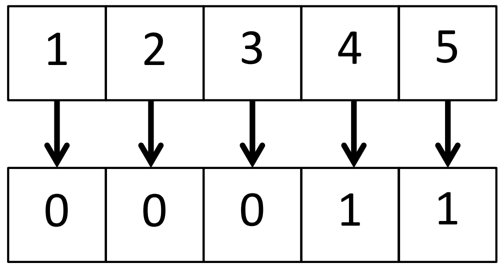
Challenge 2.1

- Generate the following plot for a quadratic equation. a = 2 , b = 20, c = -100.
- Use plot(x,y)



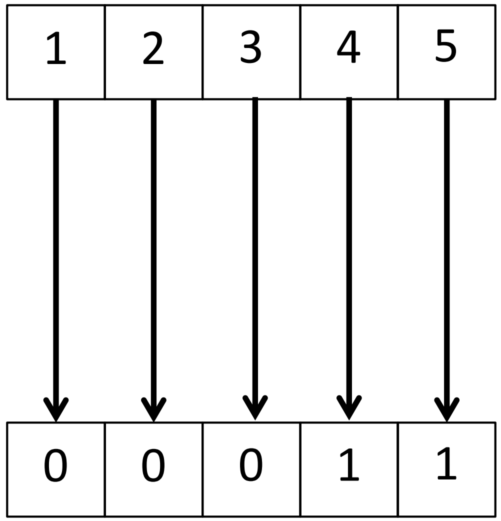
(2) Vectors and Logical Expressions

- When a vector is involved in a logical expression, the comparison is carried out element-by-element
- If the comparison is true, the resulting vector (a logical vector) has a 1 in the corresponding position, otherwise it has a 0



Consider the following code

```
>>  v = 1:5

v =

    1     2     3     4     5

>>
>> v > 3

ans =

  1×5 logical array

   0   0   0   1   1
```



Counting dice outcomes

- If rand() is in the range [0,1), 6*rand() will be in the range [0,6)
- Therefore 6*rand() + 1 will be in the range [1,7), i.e. between 1 and 6.9999999
- By using floor, we can discard the decimal point.

```
d  = floor(6*rand(1,20)) + 1
```

```
N=20

d = floor(6*rand(1,N))+1

sum(d ==6)
```

Command Window

```
>> Dice
N =

    20


d =

  Columns 1 through 19

     6     5     4     2     2     6     1     3     2     6     5     4     3     1     5     1     1     4     1

  Column 20

     5


ans =

     3

>> |
```

Logical Operators

Operator	Meaning
~	NOT
&	AND
|	OR

lex1	lex2	~lex1	lex1 & lex2	lex1 | lex2	xor(lex1, lex2)
F	F	T	F	F	F
F	T	T	F	T	T
T	F	F	F	T	T
F	F	F	T	T	F

Operator Precedence (update)

Precedence	Operators
1.	()
2.	^ .^ ‘ .’ (pure transpose)
3.	+ (unary plus) – (unary minus) ~ (NOT)
4.	* / \ .* ./ .\
5.	+ (addition) – (subtraction)
6.	:
7.	> >= < <= == ~=
8.	& (AND)
9.	| (OR)

Subscripting with logical vectors

- A logical vector v may be a subscript of another vector x
- Only the elements of x corresponding to 1s in v are returned
- x and v must be the same size
- The function logical(v) returns a logical vector, with elements which are 1 or 0 according as the elements of v are non-zero or 0.

Example of subscripting…

```
v = 1:5;

lv = v > 3;

disp('Using logical vector to filter v');
disp(v);
disp(lv);
disp(v(lv));
```

```
>> LogicalVector
Using logical vector to filter v
   1   2   3   4   5

  0  0  0  1  1

   4   5
```

Using the logical() function

```
v = 1:5;

lv = logical([0 1 0 1 0]);

disp('Using logical vector from logical() to filter v');
disp(v);
disp(lv);
disp(v(lv));
```

```
>> LogicalVector2
Using logical vector from logical() to filter v
   1   2   3   4   5

  0  1  0  1  0

   2   4
```

Logical Functions

- MATLAB has a number of useful logical functions that operate on scalars, vectors and matrices
  - any(x) returns the scalar 1 (true) if any element of x is non-zero.
  - all(x) returns the scalar 1 if all the elements of x are non-zero
  - exists('a') returns 1 if a is a workspace variable
  - find(x) returns a vector containing the subscripts of the non-zero (true) elements of x

Sample code

```
v = [1 2 3 4 3 4 5 0];

disp(v);
disp(any(v));
disp(all(v));
disp(find(v==3));
```

```
>> log_functions
   1   2   3   4   3   4   5   0

  1

  0

   3   5
```

Challenge 2.2

- Write code that removes all the duplicated values in a vector

Challenge 2.3

- For a vector of 100 random numbers, filter out all those that are less than the mean
- Build on this to create a function called partition, which takes a vector and splits it into two vectors, one containing all numbers less than the mean, the other containing all numbers greater than or equal to the mean.

(3) Matrices

- MATLAB – MATrix LABoratory
- The word matrix has two distinct meanings:
  - An arrangement of data in rows or columns e.g. a table
  - A mathematical object, for which partiuclar mathematical operations are defined (e.g. matrix multiplication)

Creating matrices

- Bigger matrices can be constructed from smaller ones

```
a = [1 2; 3 4];
x = [5 6];

a = [a; x]
```

```
>> CreateM

a =

    1    2
    3    4
    5    6
```

Subscripts

- Individual elements of a matrix are referenced with two subscripts, the first for the row the second for the column
- Less commonly, a single subscript can be used (column-based order)

```
a1 = a(3,2);
a2 = a(5);

disp(['a(3,2) = ' num2str(a1)]);
disp(['a(5) = ' num2str(a2)]);
```

```
>> CreateM

a =

   1   2
   3   4
   5   6

a(3,2) = 6
a(5) = 4
```

Transpose

- The transpose operator turns rows into columns and vice-versa

```
a = [1:3;4:6]
b = a'
```

```
a =

   1   2   3
   4   5   6


b =

   1   4
   2   5
   3   6
```

The colon operator

- The colon operator is extremely powerful, and provides for very efficient ways of handling matrices

```
a =

   1   2   3
   4   5   6
   7   8   9

>>
>> a(2:3,1:2)

ans =

   4   5
   7   8
```

```
>> a(3,:)

ans =

   7   8   9

>> a(:,3)

ans =

   3
   6
   9
```

Tiling: Duplicating rows and columns

- Sometimes it is useful to generate a matrix where all the rows and columns are the same. **repmat** can be used.

```
b =  [1:3]

repmat(b, 3 , 1)
repmat(b, 3 , 2)
```

```
>> Colon

b =

   1   2   3


ans =

   1   2   3
   1   2   3
   1   2   3
```

```
ans =

   1   2   3   1   2   3
   1   2   3   1   2   3
   1   2   3   1   2   3
```

Deleting rows and columns

- The colon operator and the empty array can be used to delete entire rows or columns

```
a = [1:3; 4:6; 7:9]
a(:,2) = [ ]
a(1,:) = [ ]
```

```
a =

   1   2   3
   4   5   6
   7   8   9

a =

   1   3
   4   6
   7   9

a =

   4   6
   7   9
```

Elementary matrices

- There are a group of functions to generate 'elementary' matrices that are used in a number of applications.
- With a single argument, they generate n x n square matrices, with two arguments they generate n x m matrices
- The function eye(n) generates an n x n identify matrix

```
>> ones(3)

ans =

   1   1   1
   1   1   1
   1   1   1

>> zeros(2,3)

ans =

   0   0   0
   0   0   0
```

MATLAB functions

- When a MATLAB mathematical or trigonometric function has a matrix argument, the function operates on every element of the matrix
- However, some MATLAB functions operate on matrices ***column by column***
- To test if all the elements of a matrix are true, two steps needed

```
a = [1 0 1; 1 1 1; 0 0 1];

all(a)

all(all(a))
```

```
>> Functions
ans =

  1×3 logical array

   0   0   1

ans =

  logical

   0
```

Manipulating matrices

- Here are functions for manipulating matrices
  - diag extracts or creates a diagonal
  - fliplr flips from left to right
  - flipud flips from top to bottom
  - rot90 rotates
  - tril extracts the lower triangular part

Element by element operations

- If a is a matrix a*2 multiplies each element of a by 2, and a .* 2 also does

```
>> a
a =

   1   0   1
   1   1   1
   0   0   1

>> 2*a

ans =

   2   0   2
   2   2   2
   0   0   2
```

```
>> a
a =

   1   0   1
   1   1   1
   0   0   1

>> 2 .* a

ans =

   2   0   2
   2   2   2
   0   0   2
```

Matrix Operations: Multiplication

- Multiplication is probably the most important matrix operation
- Widely used in: network theory, solution of linear systems, population modeling
- AB is not equal to BA
- If A is an n x m matrix and B is an m x p matrix, their product C will be an n x p matrix, and the general element c_{ij} is given by:

$$c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}$$

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \times \begin{pmatrix} 5 & 6 \\ 0 & -1 \end{pmatrix} = \begin{pmatrix} 5 & 4 \\ 15 & 14 \end{pmatrix}$$

```
a = [1 2; 3 4];
b = [5 6; 0 -1];

c = a*b

c =

   5   4
  15  14
```

Matrix Exponentiation

- A^2 means A x A, where A must be a square matrix
- The operator ^ is used for matrix exponentiation
- Note: why is A ^ 2 different from A .^ 2?

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

a ^ 2


ans =

    7    10
   15    22
```

Other (more advanced) Matrix functions

- det – determinant
- eig – eigenvalue decomposition
- inv – inverse
- expm – exponential matrix
- lu – LU factorisation
- qr – orthogonal factorisation
- svd – singular value decomposition

Challenge 2.4

- Given the following matrices A and B, calculate results for the following operations in MATLAB, and explain the basis for your results.

$$A = \begin{pmatrix} 2 & 4 \\ 6 & 8 \end{pmatrix} \qquad B = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$$

```
A * B;
A .* B;
A + B;
A .^ B;
```

An Example

- A ready mix company has three factories (F1, F2 and F3) which must supply 3 building sites (S1, S2 and S2)
- The costs of transporting a load of concrete from any factory to any given site are given by the following table:

	S1	S2	S3
F1	3	12	10
F2	17	18	35
F3	7	10	24

Transportation Scheme

- Suppose the factory manager proposes the following transportation scheme (each entry represents the number of concrete loads to be transported along this particular route).

	S1	S2	S3
F1	4	0	0
F2	6	6	0
F3	0	3	5

Solution

- Each entry in the solution table must be multiplied by the corresponding element in the cost table.
- The values should then be summed
- Note, the solution is not the mathematical operation of matrix multiplication

	S1	S2	S3
F1	3	12	10
F2	17	18	35
F3	7	10	24

	S1	S2	S3
F1	4	0	0
F2	6	6	0
F3	0	3	5

MATLAB Code

```
C = [3 12 10; 17 18 35; 7 10 24];

X = [4 0 0; 6 6 0; 0 3 5];

disp(C);
disp(X);

total = C .* X;

disp(total);

costs = sum(sum(total));

disp(['Total costs = '
num2str(costs)])
```

```
>> Concrete
   3   12   10
  17   18   35
   7   10   24

   4    0    0
   6    6    0
   0    3    5

  12    0    0
 102  108    0
   0   30  120

Total costs = 372
```