Universidade Nova de Lisboa
OMNIS CIVITAS CONTRA SE DIVISA NON STABIT
Faculdade de Ciências e Tecnologia

Introduction to Arrays

Vectors, Matrices and Beyond

Pedro Barahona
DI/FCT/UNL

Métodos Computacionais
1º Semestre 2016/2017

Matrices

- An array is a data structure that groups together a set of values of the same type (typically numeric), with the stucture of a *multidimensional* table.
- The most used arrays have 1 dimension (vectors) or 2 dimensions (matrices), but arrays of higher dimension are also possible.
- For example, matrix M represents the multiplication table of the first natural numbers.

Matrix M

$$\begin{array}{c|cccc}
 & 1 & 2 & 3 & 4 \\
\hline
1 & 1 & 2 & 3 & 4 \\
2 & 2 & 4 & 6 & 8 \\
3 & 3 & 6 & 9 & 12
\end{array}$$

- The two dimensions are known as rows and columns. Hence M is a 3×4 M matrix with 3 rows and 4 columns.
- Matrices are the basic data type in MATLAB. A single number is in fact maintained in MATLAB as a 1×1 matrix.

Vectors and Matrices

- A vector is a special case of a matrix where one of the dimensions has size 1.
- For example, we may represent in a vector V, with a single row, the squares of the first 5 natural numbers.

(row) Vector V

1	2	3	4	5
1	4	9	16	25

- Or we may represent in a column a vector U the first 4 prime numbers, .

(column)

Vector U

1	2
2	3
3	5
4	7

Initialisation of Arrays

- Variables may be arrays, and like any variable they must be initialised before being used. There are several ways of initialising an array variable in MATLAB.
- The most general form of initializing an array variable is by means of an explicit declaration of the value of each its elements, thus implicitly defining its dimensions.
- In MATLAB, different values of the same row are separated by commas or spaces, and rows are separated by semi-colons (“;”).
- `V = [1, 4  9, 16 25]`
- `U = [ 2; 3; 5; 7]`
- `M = [1 2 3 4 ; 2, 4, 6, 8 ; 3 6 9 12]`

Vector V

1	2	3	4	5
1	4	9	16	25

Vector U

1	2
2	3
3	5
4	7

Matrix M

	1	2	3	4
1	1	2	3	4
2	2	4	6	8
3	3	6	9	12

Initialisation of Vectors

- In MATLAB, vectors can be initialised as ranges, as already used in iterators in the for loops.

1	2	3	4	5	6
0	2	4	6	8	10

```
>> P = 0:2:10
P =
    0 2 4 6 8 10
```

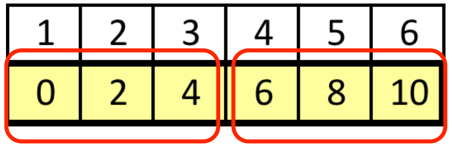
- By default, vectors defined this way are row vectors. But they can be **transposed** to represent column arrays

1	1
2	3
3	5
4	7

```
>> P = [1:2:7]'
P =
    1
    3
    5
    7
```

Initialisation of Matrices

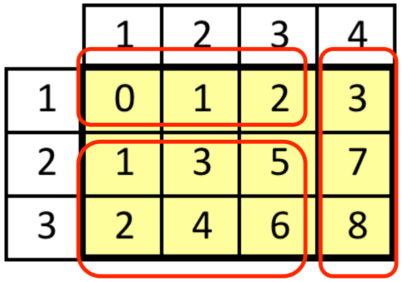
- In MATLAB, once vectors have been defined they can be used to declare parts of other vectors ...



$$\begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 0 & 2 & 4 & 6 & 8 & 10 \end{array}$$

```
>> P = 0:2:4
>> Q = [6,8,10]
>> R = [P,Q]
```

- Vectors and matrices may also be used as parts or components (rows or columns) of matrices that contain them.



$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 2 & 3 \\ 2 & 1 & 3 & 5 & 7 \\ 3 & 2 & 4 & 6 & 8 \end{array}$$

```
>> A = 0:2;
>> B = [1:2:5; 2:2:6]
>> C = [3;7;8]
>> M = [[A;B],C]
```

Initialisation of Matrices

- When matrices are “big” they might be initialised to “0s” or “1s” by predefined instructions, explicitely mentioning the size of each of the dimensions of the matrices.

```
>> V1 = ones(1,5)

>> U1 = zeros(4,1)

>> M1 = ones(3,4)
```

Vector V1

1	2	3	4	5
1	1	1	1	1

Vector U1

1	0
2	0
3	0
4	0

Matrix M1

	1	2	3	4
1	1	1	1	1
2	1	1	1	1
3	1	1	1	1

- **Note**: The initialisation of vectors and matrices (even when the exact values of their elements are not known in advance, as in this case) is useful because it informs the interpreter of the language of the size of memory that it should allocate for the array. Although this size may be dynamically changed in MATLAB, such dynamic changes decrease the efficiency of program execution.

Size of Arrays

- The predefined function **rows(X)** and **columns(X**) can be used, to know the size of the dimensions of an array variable X,

```
>> rows(V)
ans = 1
>> columns(V)
ans = 5
```

```
>> rows(U)
ans = 1
>> columns(U)
ans = 5
```

```
>> rows(M)
ans = 3
>> columns(M)
ans = 4
```

Vector V

1	2	3	4	5
1	4	9	16	25

Vector U

1	2
2	3
3	5
4	7

Matrix M

	1	2	3	4
1	1	2	3	4
2	2	4	6	8
3	3	6	9	12

Size of Arrays

- Since simple numeric variables are stored as 1 × 1 matrices, their size may also be queried.

```
>> a = 5
>> rows(a)
ans = 1
>> columns(a)
ans = 1
```

- For a vector V, be it a row or column vector, the size of its only significant dimension can also be queried by function **length(X)**.

```
>> length(V)
ans = 5
```

```
>> length(U)
ans = 4
```

Vector V

1	2	3	4	5
1	4	9	16	25

Vector U

1	2
2	3
3	5
4	7

Size of Arrays

- The predefined function rows(X) and columns(X) can be used, to know the size of the dimensions of an array variable X,
  - be it a simple variable,
  - or vectors and matrices.

```
>> a = 5
>> rows(a)
ans = 1
>> columns(a)
ans = 1
```

```
>> rows(V)
ans = 1
>> columns(V)
ans = 5
```

```
>> rows(U)
ans = 4
>> columns(UV)
ans = 1
```

Vector V

1	2	3	4	5
1	4	9	16	25

Vector U

1	2
2	3
3	5
4	7

Matrix M

	1	2	3	4
1	1	2	3	4
2	2	4	6	8
3	3	6	9	12

Multidimensional Arrays

- Arrays with more than 2 dimensions are also possible in MATLAB where they are called multidimensional arrays.
- There is no special syntax to initialise them directly, i.e. there is no syntactical separator to “enter the 3rd dimension”.
- They can nonetheless be initialised with the zeros and ones functions (and have their value updated later).

```
>> X = zeros(3,4,2)
X =
ans(:,:,1) =
    0    0    0    0
    0    0    0    0
    0    0    0    0
ans(:,:,2) =
    0    0    0    0
    0    0    0    0
    0    0    0    0
```

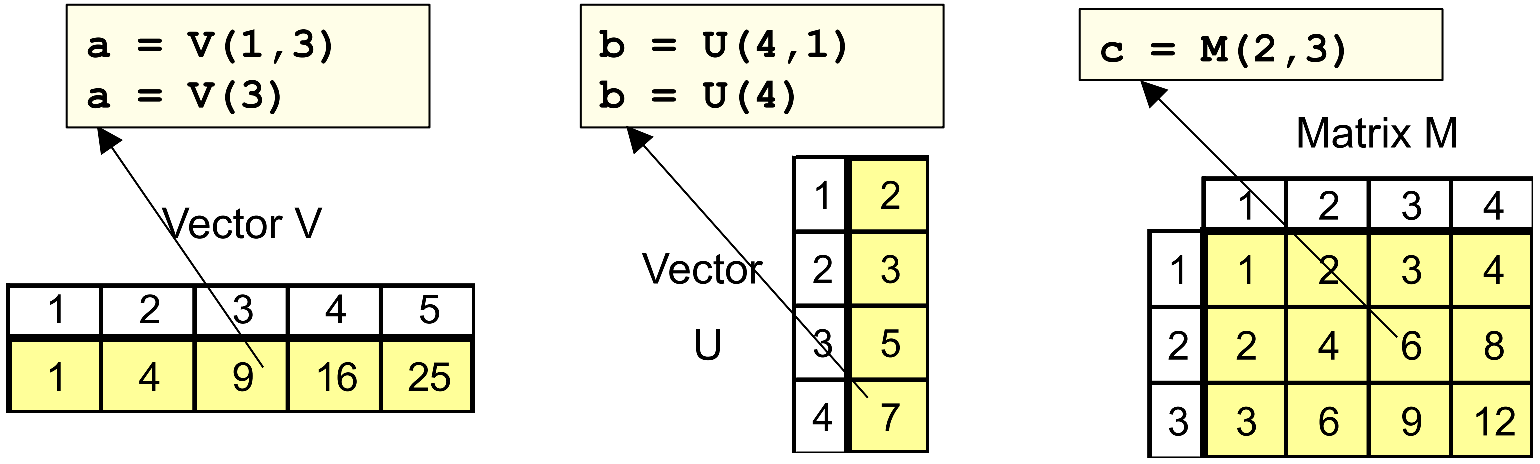
- The number of their dimensions may also be known by using the predefined functions size and ndims.

```
>> size(X)
ans =    3    4    2

>> ndims(X)
ans =   3>>
```

Access to Array Elements

- Elements of an array are referred by their position in the matrix, i.e their row and column indices (the number of the row and the column they occupy in the matrix).
- Contrary to other languages, in MATLAB row and column indices of a matrix **always** start at 1.



```
a = V(1,3)
a = V(3)
```

```
b = U(4,1)
b = U(4)
```

```
c = M(2,3)
```

Vector V

1	2	3	4	5
1	4	9	16	25

Vector U

1	2
2	3
3	5
4	7

Matrix M

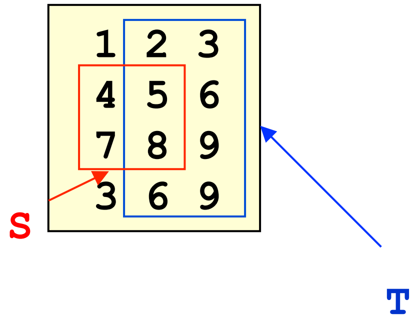
	1	2	3	4
1	1	2	3	4
2	2	4	6	8
3	3	6	9	12

- For convenience, MATLAB allows that the row/column of a row/column vector is ommited in the reference.

Access to Sub-Arrays

- Arrays can be "extracted" from larger arrays by defining the ranges of the elements one is interested in.

```
>> V = [1 2 3; 4 5 6; 7 8 9; 3 6 9]
>> S = V(2:3,1:2)
S =
   4   5
   7   8
>> T = V(:,2:3)
T =
   2   3
   5   6
   8   9
   6   9
>> T(2,2)
ans =   6
```



- Notice that the selection of a whole dimension may be specified by the full range ":"
- Also the indices of the subarrays are automatically adjusted so that they start in 1.

Array Updates

- Once defined, arrays can be updated, i.e. have the value of their elements changed.
- This is of course quite useful when the initialisation was made by means of the zeros and ones function.
- For example, two elements of a matriz M may be swapped, as shown,
- Notice that, like in any other swap, one auxiliary variable is used for temporarily store one of the variables.

```
>> M = [1,2,3;4,5,6]
M =
     1     2     3
     4     5     6
>> x = M(2,3)
x =   6
>> M(2,3) = M(1,2)
M =
     1     2     3
     4     5     2
>> M(1,2) = x
M =
     1     6     3
     4     5     2
```

Array Updates

- In general, one may be interested in changing all the elements of an array.
- In its more general form, this can be done by means of nested for loops, one for each dimension.
- For example, one may want to impose that all elements of a vector V take as values some function f (for example the decimal logarithm) of their indices, running a script

```
V = ones(1,10)
for i = 1:10
   V(i) = log10(i);
endfor
V
```

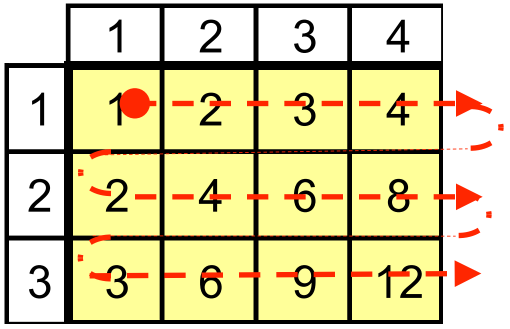
```
V =   1   1   1   1   1   1   1   1   1   1
V =
 Columns 1 through 7:
   0.00000   0.30103   0.47712   0.60206   0.69897   0.77815   0.84510
 Columns 8 through 10:
   0.90309   0.95424   1.00000
```

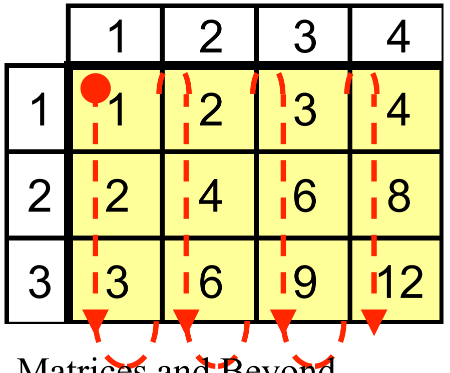
Array Updates

- Or one may be interested in filling a times table with two nesteded loop, one for each dimension.
- Notice that, in this case, the nesting of the loops makes no difference in the final matrix , but the order in which the elements are computed is diffeernt.

```
M = ones(3,4)
for i = 1:3
    for j = 1:4
        M(i,j) = i*j;
    endfor
endfor
```

```
V = ones(3,4)
for j = 1:4
    for i = 1:3
        M(i,j) = i*j;
    endfor
endfor
```





Array Updates

- In general, acesses to the elements of an array should not go beyond their current sizes.
- For this purpose, and if the size of the arrays is not known, it can be questioned by function size, that returns an array with the size of each dimension.

```
M = zeros(3,4,2);
S = size(M)
for i = 1:S(1)
   for j = 1:S(2)
      for k = 1:S(3)
         M(i,j,k) = i*j*k;
      end
   end
end
M
```

```
S =    3    4    2
M =
ans(:,:,1) =
    1    2    3    4
    2    4    6    8
    3    6    9   12
ans(:,:,2) =
    2    4    6    8
    4    8   12   16
    6   12   18   24
```

Matrix Operations

- In MATLAB, since the matrix is the basic representation for variables, all matrix (or vector operations) are available as defined in algebra.
- The simplest case is the **sum** / **difference** of two matrices.

```
>> M = [1 2 3; 4,5,6]
M =    1    2    3    4    5    6
>> N = [2,4,6; 0,1,2]
N =    2    4    6    0    1    2
>> P = M+N
P =    3    6    9    4    6    8
```

- Notice that suming arrays of difefernt sizes originaites an error

```
>> V = [1 2 3];
>> U = [1,2,3,4];
>> X = U+V
error: operator +: nonconformant arguments (op1 is 1x4, op2 is 1x3)
```

Matrix Operations

- In MATLAB, since the matrix is the basic representation for variables, all matrix (or vector operations) are available as defined in algebra.
- The simplest case is the **sum** / **difference** of two matrices.

```
>> M = [1 2 3; 4,5,6]
M =    1    2    3    4    5    6
>> N = [2,4,6; 0,1,2]
N =    2    4    6    0    1    2
>> P = M+N
P =    3    6    9    4    6    8
```

- Notice that suming arrays of difefernt sizes originaites an error

```
>> V = [1 2 3];
>> U = [1,2,3,4];
>> X = U+V
error: operator +: nonconformant arguments (op1 is 1x4, op2 is 1x3)
```

- The product is more complicated and will be dicussed later.

Matrix Dot Operations

- In many cases we are interested in applying the same operation to all corresponding elements of two arrays, i.e. to apply pointwise operations.
- In MATLAB, this is possible, by using dot notation, i.e. by using the wnated operator preceded by a dot.

```
>> M = [1 2 3; 4,5,6]
M =
   1   2   3
   4   5   6
>> N = [2,4,6; 0,1,2]
N =
   2   4   6
   0   1   2
>> P = M .* N
P =
    2    8   18
    0    5   12
```

- Notice that although there is no difference between sums and dotted sumsm this is not the case with multiplication!

Transposition

- A useful operation on vectors and matrices is the transposition that changes rows and columns.

```
>> M = [1 2 3; 4,5,6]
M =
   1   2   3
   4   5   6
>> N = M'
N =
   1   4
   2   5
   3   6
>> P = M(:,3)'
P =
   3   6
```

Distribution Operations

- In many cases we are interested in applying an operation between a scalar and all the elements of a matrix.
- In MATLAB, by default this is what happens when the operation is specified in the natural way.

```
>> M = [1 2 3; 4,5,6]
M =
   1   2   3
   4   5   6
>> a = 5
a =  5
>> Q = a*M
Q =
    5   10   15
   20   25   30
```

- In oher languages, this distribution must be done through nested loops!