Chapter 2

Creating Arrays

MATLAB An Introduction With Applications, 6th Edition
Dr. Amos Gilat
The Ohio State University

Slide deck by
Dr. Greg Reese
Miami University

An *array* is MATLAB's basic data structure

- Can have any number of dimensions. Most common are
  - *vector* - one dimension (a single row or column)
  - *matrix* - two or more dimensions
- Arrays can have numbers or letters

To create a row vector from known numbers, type variable name, then equal sign, then inside square brackets, numbers separated by spaces and/or commas

```
variable_name = [ n1, n2, n3 ]
```

Commas optional

```
>> yr = [1984 1986 1988 1990 1992 1994 1996]
yr =
      1984 1986 1988 1990 1992 1994 1996
```

Note MATLAB displays row vector horizontally

To create a column vector from known numbers

- Method 1 - same as row vector but put semicolon after all but last number

```
variable_name = [ n1; n2; n3 ]
```

```
>> yr = [1984; 1986; 1988 ]
yr =
      1984
      1986
      1988
```

Note MATLAB displays column vector vertically

- Method 2 - same as row vector but put apostrophe (') after closing bracket
  - Apostrophe interchanges rows and columns. Will study later

```
 variable_name = [ n1 n2 n3 ]'
>> yr = [1984 1986 1988 ]'
yr =
1984
1986
1988
```

To create a vector with specified constant spacing between elements

```
variable_name = m:q:n
```

- `m` is first number
- `n` is last number
- `q` is difference between consecutive numbers

```
v = m:q:n
```

means

```
v = [ m m+q m+2q m+3q ... n ]
```

If omit q, spacing is one

```
              v = m:n
```

means

```
 v = [ m m+1 m+2 m+3 ... n ]
```

```
>> x = 1:2:13
x = 1 3 5 7 9 11 13
>> y = 1.5:0.1:2.1
```
Non-integer spacing
```
y = 1.5000 1.6000 1.7000
1.8000 1.9000 2.0000 2.1000
```

```
>> z = -3:7
z = -3 -2 -1 0 1 2 3 4 5 6 7
>> xa = 21:-3:6
```

Negative spacing

```
xa = 21 18 15 12 9 6
```

To create a vector with specified number of terms between first and last

```
v = linspace( xi, xf, n )
```

- `xi` is first number
- `xf` is last number
- `n` is number of terms (= 100 if omitted)

2.1 CREATING A ONE-DIMENSIONAL ARRAY (VECTOR)

```
>> va = linspace( 0, 8, 6 )
```
Six elements

```
va = 0 1.6000 3.2000 4.8000 6.4000 8.0000
>> va = linspace( 30, 10, 11 )
```
Decreasing elements

```
va=30 28 26 24 22 20 18 16 14 12 10
```

`m:q:n` lets you directly specify spacing. `linspace()` lets you directly specify number of terms

Create a two-dimensional matrix like this

```
m = [ row 1 numbers; row 2 numbers;
... ; last row numbers ]
```

- Each row separated by semicolon
- All rows have same number of columns

```
>> a=[ 5 35 43; 4 76 81; 21 32 40]
a =
     5     35     43
     4     76     81
    21     32     40
```

```
>> cd=6; e=3; h=4;
```

Commas optional

```
>> Mat=[e, cd*h, cos(pi/3);...
h^2 sqrt(h*h/cd) 14]

Mat =

     3.0000    24.0000     0.5000
    16.0000     1.6330    14.0000
```

Can also use `m:p:n` or `linspace()` to make rows

- Make sure each row has same number of columns

```
>> A=[1:2:11; 0:5:25;...
linspace(10,60,6); 67 2 43 68 4 13]
A =
       1      3      5      7      9     11
       0      5     10     15     20     25
      10     20     30     40     50     60
      67      2     43     68      4     13
```

What if number of columns different?

$$\texttt{>> B= [}\overbrace{\texttt{1:4}}^{\text{Four columns}}\texttt{; }\overbrace{\texttt{linspace(1,4,5)}}^{\text{Five columns}}\texttt{]}$$

```
??? Error using ==> vertcat
CAT arguments dimensions are
not consistent.
```

`zeros(m,n)` - makes matrix of `m` rows and `n` columns, all with zeros

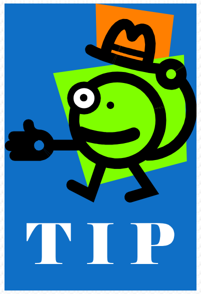
`ones(m,n)` - makes matrix of `m` rows and `n` columns, all with ones

`eye(n)` - makes square matrix of `n` rows and columns. Main diagonal (upper left to lower right) has ones, all other elements are zero

2.2.1 The `zeros`, `ones` and, `eye` Commands

```
>> zr=zeros(3,4)
zr = 0   0   0   0
     0   0   0   0
     0   0   0   0

>> ne=ones(4,3)
ne = 1   1   1
     1   1   1
     1   1   1
     1   1   1
```

```
>> idn=eye(5)

idn = 1   0   0   0   0

      0   1   0   0   0

      0   0   1   0   0

      0   0   0   1   0

      0   0   0   0   1
```



To make a matrix filled with a particular number, multiply `ones(m,n)` by that number

```
>> z=100*ones(3,4)

z =

   100   100   100   100
   100   100   100   100
   100   100   100   100
```

- All variables are arrays
  - *Scalar* - array with only one element
  - *Vector* - array with only one row or column
  - *Matrix* - array with multiple rows and columns
- Assigning to variable specifies its dimension
  - Don't have to define variable size before assigning to it, as you do in many programming languages
- Reassigning to variable changes its dimension to that of assignment

Transpose a variable by putting a single quote after it, e.g., `x'`

- In math, transpose usually denoted by superscript "T", e.g., x^T

- Converts a row vector to a column vector and vice-versa
- Switches rows and columns of a matrix, i.e., first row of original becomes first column of transposed, second row of original becomes second column of transposed, etc.

```
>> aa=[3 8 1]
aa =        3        8        1
>> bb=aa'
bb =  3
      8
      1
```

```
>> C=[2 55 14 8; 21 5 32 11; 41 64 9 1]
C =   2      55      14       8
     21       5      32      11
     41      64       9       1
>> D=C'
D =   2      21      41
     55       5      64
     14      32       9
      8      11       1
```

Can access (read from or write to) elements in array (vector or matrix) individually or in groups

- Useful for changing subset of elements
- Useful for making new variable from subset of elements

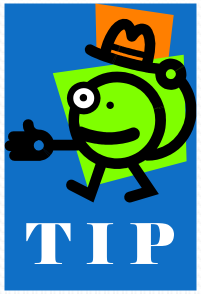
Address of element is its position in the vector

- "address" often called *index*
- Addresses always start at 1 (not 0)
  - Address 1 of row vector is leftmost element
  - Address 1 of column vector is topmost element
- To access element of a vector represented by a variable, follow variables name by address inside parentheses, e.g., `v(2)=20` sets second element of vector `v` to `20`

```
>> VCT=[35 46 78 23 5 14 81 3 55]
VCT = 35 46 78 23 5 14 81 3 5
>> VCT(4)
ans = 23
>> VCT(6)=273
VCT = 35 46 78 23 5 273 81 3 5
>> VCT(2)+VCT(8)
ans = 49
>> VCT(5)^VCT(8)+sqrt(VCT(7))
ans = 134
```

Address of element in a matrix is given by row number and column number. Address often called *index* or *subscript*

- Addresses always start at 1 (not 0)
  - Row 1 is top row
  - Column 1 is left column
- If variable `ma` is a matrix, `ma(k,p)` is element in row `k` and column `p`



In MATLAB, left index always refers to row, right index to column

```
>> MAT=[3 11 6 5; 4 7 10 2; 13 9 0 8]
```

Column 1

```
MAT =   3      11       6       5
        4       7      10       2
       13       9       0       8
```

Element in row 3 and column 1 →

Row 3

```
>> MAT(3,1)
ans = 13
>> MAT(3,1)=20
```

Assign new value to element in row 3 and column 1

```
MAT = 3      11       6       5
      4       7      10       2
     20       9       0       8
```

Only this element changed

```
>> MAT(2,4)-MAT(1,2)
ans = -9
```

The colon : lets you address a range of elements

- Vector (row or column)
  - `va(:)` - all elements
  - `va(m:n)` - elements `m` through `n`
- Matrix
  - `A(:,n)` - all rows of column `n`
  - `A(m,:)` - all columns of row `m`
  - `A(:,m:n)` - all rows of columns `m` through `n`
  - `A(m:n,:)` - all columns of rows `m` through `n`
  - `A(m:n,p:q)` - columns `p` through `q` of rows `m` through `n`

```
>> A=[1:2:11; 2:2:12; 3:3:18; 4:4:24; 5:5:30]
A =   1      3      5      7      9     11
      2      4      6      8     10     12
      3      6      9     12     15     18
      4      8     12     16     20     24
      5     10     15     20     25     30
>> B=A(:,3)
```
All rows of column 3

```
B =   5
      6
      9
     12
     15
```

```
>> C=A(2,:)
```
All columns of row 2

```
C =   2     4     6     8     10     12
```

```
>> E=A(2:4,:)
```
All columns of rows two through four

```
E =   2     4     6     8     10     12
      3     6     9    12     15     18
      4     8    12    16     20     24
```

```
>> F=A(1:3,2:4)
```
Columns two through four of rows one through three

```
F =   3     5     7
      4     6     8
      6     9    12
```

Can replace vector index or matrix indices by vectors in order to pick out specific elements. For example, for vector `v` and matrix `m`

- `v([a b c:d])` returns elements `a`, `b`, and `c` through `d`
- `m([a b],[c:d e])` returns columns `c` through `d` and column `e` of rows `a` and `b`

```
>> v=4:3:34
v = 4 7 10 13 16 19 22 25 28 31 34

>> u=v([3, 5, 7:10])
u = 10 16 22 25 28 31
```

```
>> A=[10:-1:4; ones(1,7); 2:2:14; zeros(1,7)]
A = 10     9     8     7     6     5     4
     1     1     1     1     1     1     1
     2     4     6     8    10    12    14
     0     0     0     0     0     0     0

>> B=A([1 3],[1 3 5:7])
B = 10     8     6     5     4
     2     6    10    12    14
```

Two ways to add elements to existing variables

1. Assign values to indices that don't exist
   - MATLAB expands array to include indices, puts specified values in assigned elements, fills any unassigned new elements with zeros
2. Add values to ends of variables
   - Adding to ends of variables is called *appending* or *concatenating*
   - "end" of vector is right side of row vector or bottom of column vector
   - "end" of matrix is right column or bottom row

Assigning to undefined indices of vectors

```
>> DF=1:4
DF = 1 2 3 4
>> DF(5:10)=10:5:35
DF = 1 2 3 4 10 15 20 25 30 35
>> AD=[5 7 2]
AD = 5 7 2
>> AD(8)=4
AD = 5 7 2 0 0 0 0 4
>> AR(5)=24
AR = 0 0 0 0 24
```

Unassigned elements set to zero

Appending to vectors

- Can only append row vectors to row vectors and column vectors to column vectors
  - If `r1` and `r2` are any row vectors, `r3 = [r1 r2]` is a row vector whose left part is r`1` and right part is `r2`
  - If `c1` and `c2` are any column vectors, `c3 = [c1; c2`] is a column vector whose top part is `c1` and bottom part is `c2`

```
>> RE=[3 8 1 24];
>> GT=4:3:16;
>> KNH=[RE GT]
KNH = 3 8 1 24 4 7 10 13 16
>> KNV=[RE'; GT']
KNV = 3
      8
      1
     24
      4
      7
     10
     13
     16
```

Assigning to undefined indices of matrices

```
>> AW=[3 6 9; 8 5 11]
AW = 3     6     9
     8     5    11
>> AW(4,5)=17
AW = 3     6     9     0     0
     8     5    11     0     0
     0     0     0     0     0
     0     0     0     0    17
>> BG(3,4)=15
BG = 0     0     0     0
     0     0     0     0
     0     0     0    15
```

AW doesn't have a fourth row or fifth column

Now it does!

Unassigned elements set to zero

Appending to matrices

- If appending one matrix to right side of other matrix, both must have same number of rows
- If appending one matrix to bottom of other matrix, both must have same number of columns

2.7 ADDING ELEMENTS TO EXISTING VARIABLES

```
>> A2=[1 2 3; 4 5 6]
A2 = 1      2      3
     4      5      6
>> B2=[7 8; 9 10]
B2 = 7      8
     9     10
>> C2=eye(3)
C2 = 1      0      0
     0      1      0
     0      0      1
```

```
>> Z=[A2 B2]
Z =   1       2       3       7       8
      4       5       6       9      10
>> Z=[A2; C2]
Z =   1       2       3
      4       5       6
      1       0       0
      0       1       0
      0       0       1
>> Z=[A2; B2]
??? Error using ==> vertcat
CAT arguments dimensions are not
consistent.
```

To delete elements in a vector or matrix, set range to be deleted to empty brackets

```
>> kt=[2 8 40 65 3 55 23 15 75 80]
kt = 2 8 40 65 3 55 23 15 75 80
>> kt(6)=[]
```

Delete sixth element (55)

```
kt = 2 8 40 65 3 23 15 75 80
```

55 gone

```
>> kt(3:6)=[]
```

Delete elements 3 through 6 of current `kt`, not original `kt`

```
kt = 2 8 15 75 80
```

To delete elements in a vector or matrix, set range to be deleted to empty brackets

```
>> mtr=[5 78 4 24 9; 4 0 36 60 12; 56 13 5
89 3]
mtr = 5      78      4      24      9
      4       0     36      60     12
     56      13      5      89      3
>> mtr(:,2:4)=[]
mtr = 5       9
      4      12
     56       3
```

MATLAB has many built-in functions for working with arrays. Some common ones are:

- `length(v)` - number of elements in a vector
- `size(A)` - number of rows and columns in a matrix or vector
- `reshape(A,m,n)` - changes number of rows and columns of a matrix or vector while keeping total number of elements the same. For example, changes 4x4 matrix to 2x8 matrix

- `diag(v)` - makes a square matrix of zeroes with vector in main diagonal
- `diag(A)` - creates vector equal to main diagonal of matrix

For more functions, click on the Help icon, then in the Help window click on MATLAB, then on “MATLAB functions”, then on “By Category", then scroll down to the section labeled "Matrices and Arrays"

A *string* is an array of characters

Strings have many uses in MATLAB

- Display text output
- Specify formatting for plots
- Input arguments for some functions
- Text input from user or data files

- Create a string by typing characters within single quotes (')
  - Many programming languages use the quotation mark (") for strings. Not MATLAB!

- When typing in string
  - Color of text changes to maroon when type first single quote
  - Color of text changes to purple when type last single quote

- Can have letters, digits, symbols, spaces
  - To type single quote in string, use two consecutive single quotes, e.g., make the string of English "Greg's car" by typing `'Greg''s car'`
  - Examples: `'ad ef'`, `'3%fr2'`, `'edcba:21!'`, `'MATLAB'`

Can assign string to a variable, just like numbers

```
>> name = 'Sting'
name =
    Sting
>> police = 'New York''s
finest'
police =
    New York's finest
```

In a string variable

- Numbers are stored as an array
- A one-line string is a row vector
  - Number of elements in vector is number of characters in string

```
>> name = 'Howard the Duck';
>> size( name )
ans =
   1 15
```

Strings are indexed the same way as vectors and matrices

- Can read by index
- Can write by index
- Can delete by index

Example

```
>> word = 'dale';
>> word(1)
ans = d
>> word(1) = 'v'
word = vale
>> word(end) = []
word = val
>> word(end+1:end+3) = 'ley'
word = valley
```

MATLAB stores strings with multiple lines as an array. This means each line must have the same number of columns (characters)

```
>> names = [ 'Greg'; 'John' ]
names =
      Greg
      John
>> size( names )
ans =
      2 4
```

Problem

4 characters 3 characters

```
>> names = [ 'Greg'; 'Jon' ]???
Error using ==> vertcat
CAT arguments dimensions are not
consistent.
```

Must put in extra characters (usually spaces) by hand so that all rows have same number of characters

```
>> names = [ 'Greg'; 'Jon ' ]
    Greg
    Jon
```

$\uparrow$

Extra space

Making sure each line of text has the same number of characters is a big pain. MATLAB solves problem with `char` function, which *pads* each line on the right with enough spaces so that all lines have the same number of characters

```
char('string 1', 'string 2', 'string 3')
```

EXAMPLE

```
>> question=char('Romeo,
Romeo,',...
'Wherefore art thou', 'Romeo?' )
question =
    Romeo, Romeo,
    Wherefore art thou
    Romeo?
>> size( question )
ans =
    3 18
```

R	o	m	e	o	,		R	o	m	e	o	,					
W	h	e	r	e	f	o	r	e		a	r	t		t	h	o	u
R	o	m	e	o	?												

Three lines of text stored in a 3x18 array

MATLAB makes all rows as long as longest row

- First and third rows above have enough space characters added on ends to make each row 18 characters long