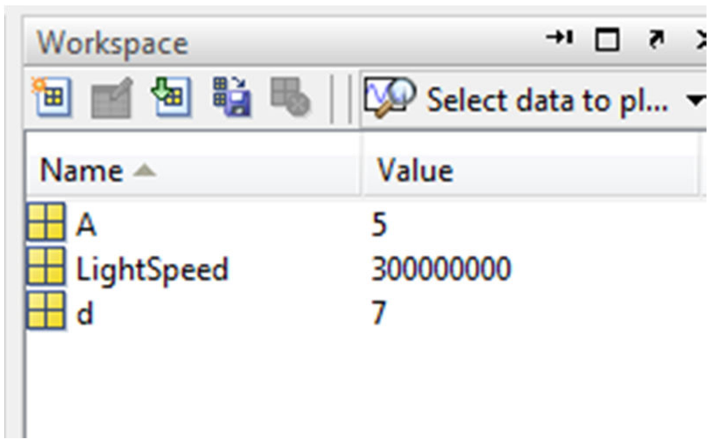
Lecture 2: Variables, Vectors and Matrices in MATLAB

Dr. Mohammed Hawa
Electrical Engineering Department
University of Jordan

Variables in MATLAB

- Just like other programming languages, you can define variables in which to store values.
- All variables can by default hold matrices with scalar or complex numbers in them.
- You can define as many variables as your PC memory can hold.
- Values in variables can be inspected, used and changed
- Variable names are case-sensitive, and show up in the Workspace.

```
>> A = 5
A =
     5

>> d = 7
d =
     7

>> LightSpeed = 3e8
LightSpeed =
   300000000
```



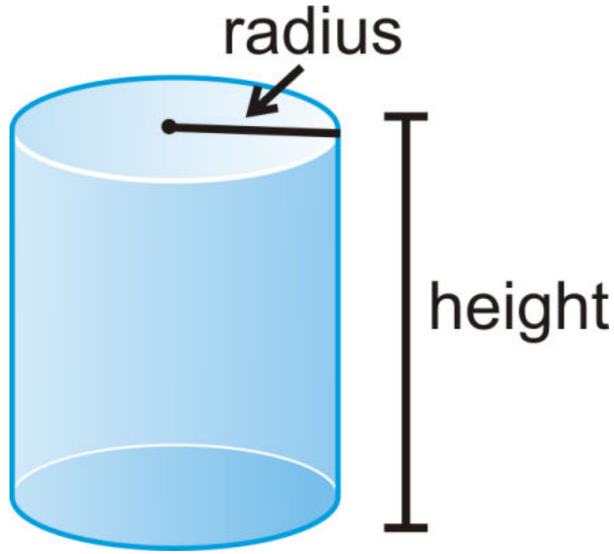
Variables

- You can change the value in the variable by over-writing it with a new value
- Remember that variables are case-sensitive (easy to make a mistake)
- Always left-to right
  >> variable = expression

```
>> a = 7
a =
     7

>> b = 12
b =
    12

>> b = 14
b =
    14

>> B = 88
B =
    88

>> c = a + b
c =
    21

>> c = a / b
c =
    0.5000
```

Exercise

- Develop MATLAB code to find Cylinder volume and surface area.
- Assume radius of 5 m and height of 13 m.



$$V = \pi r^2 h$$

$$A = 2\pi r^2 + 2\pi r h = 2\pi r(r + h)$$

Solution

```
>> r = 5
r =
     5

>> h = 13
h =
    13

>> Volume = pi * r^2 * h
Volume =
  1.0210e+003

>> Area = 2 * pi * r * (r + h)
Area =
  565.4867
```

Useful MATLAB commands

Command	Description
`clc`	Clears the Command window.
`clear`	Removes all variables from memory.
`clear var1 var2`	Removes the variables `var1` and `var2` from memory.
`exist('name')`	Determines if a file or variable exists having the name 'name'.
`quit`	Stops MATLAB.
`who`	Lists the variables currently in memory.
`whos`	Lists the current variables and sizes, and indicates if they have imaginary parts.
`:`	Colon; generates an array having regularly spaced elements.
`,`	Comma; separates elements of an array.
`;`	Semicolon; suppresses screen printing; also denotes a new row in an array.
`...`	Ellipsis; continues a line.

Vectors and Matrices (Arrays)

- So far we used MATLAB variables to store a single value.
- We can also create MATLAB arrays that hold multiple values
  - List of values (1D array) called **Vector**
  - Table of values (2D array) called **Matrix**
- Vectors and matrices are used extensively when solving engineering and science problems.

Row Vector

- Row vectors are special cases of matrices.
- This is a 7-element row vector (1×7 matrix).
- Defined by enclosing numbers within square brackets `[ ]` and separating them by `,` or a space.

```
>> C = [10, 11, 13, 12, 19, 16, 17]

C =
    10    11    13    12    19    16    17


>> C = [10 11 13 12 19 16 17]

C =
    10    11    13    12    19    16    17
```

Column Vector

- Column vectors are special cases of matrices.
- This is a 7-element column vector (7×1 matrix).
- Defined by enclosing numbers within `[ ]` and separating them by semicolon `;`

```
>> R = [10; 11; 13; 12; 19; 16; 17]

R =
    10
    11
    13
    12
    19
    16
    17
```

Matrix

- This is a 3×4-element matrix.
- It has 3 rows and 4 columns (dimension 3×4).
- Spaces or commas separate elements in different columns, whereas semicolons separate elements in different rows.
- A dimension $n \times n$ matrix is called *square* matrix.

```
>> M = [1, 3, 2, 9; 6, 7, 8, 1; 7, 4, 6, 0]
M =
     1     3     2     9
     6     7     8     1
     7     4     6     0


>> M = [1 3 2 9; 6 7 8 1; 7 4 6 0]
M =
     1     3     2     9
     6     7     8     1
     7     4     6     0
```

Transpose of a Matrix

- The transpose operation interchanges the rows and columns of a matrix.
- For an $m \times n$ matrix $\mathbf{A}$ the new matrix $\mathbf{A}^T$ (read "A transpose") is an $n \times m$ matrix.
- In MATLAB, the `A'` command is used for transpose.

$$\mathbf{A} = \begin{bmatrix} -2 & 6 \\ -3 & 5 \end{bmatrix} \qquad \mathbf{A}^T = \begin{bmatrix} -2 & -3 \\ 6 & 5 \end{bmatrix}$$

Exercise

```
>> A = [1 2 3; 5 6 7]
A =
     1     2     3
     5     6     7

>> A'
ans =
     1     5
     2     6
     3     7
```

```
>> B = [5 6 7 8]
B =
     5     6     7     8

>> B'
ans =
     5
     6
     7
     8
```

- What happens to a row vector when transposed?
- What happens to a column vector when transposed?

Useful Functions

`length(A)`	Returns either the number of elements of A if A is a vector or the largest value of *m* or *n* if A is an $m \times n$ matrix
`size(A)`	Returns a row vector `[m n]` containing the sizes of the $m \times n$ matrix A.
`max(A)`	For vectors, returns the largest element in A. For matrices, returns a row vector containing the maximum element from each column. If any of the elements are complex, `max(A)` returns the elements that have the largest magnitudes.
`[v,k] = max(A)`	Similar to `max(A)` but stores the maximum values in the row vector `v` and their indices in the row vector `k`.

`min(A)`
and
`[v,k] = min(A)`
	Like `max` but returns minimum values.

More Useful Functions

`sort(A)`	Sorts each column of the array `A` in ascending order and returns an array the same size as `A`.
`sort(A,DIM,MODE)`	Sort with two optional parameters: DIM selects a dimension along which to sort. MODE is sort direction ('ascend' or 'descend').
`sum(A)`	Sums the elements in each column of the array `A` and returns a row vector containing the sums.
`sum(A,DIM)`	Sums along the dimension DIM.

Exercises

```
>> X = [4 9 2 5]
X =
      4      9      2      5

>> length(X)
ans =
      4

>> size(X)
ans =
      1      4

>> min(X)
ans =
      2
```

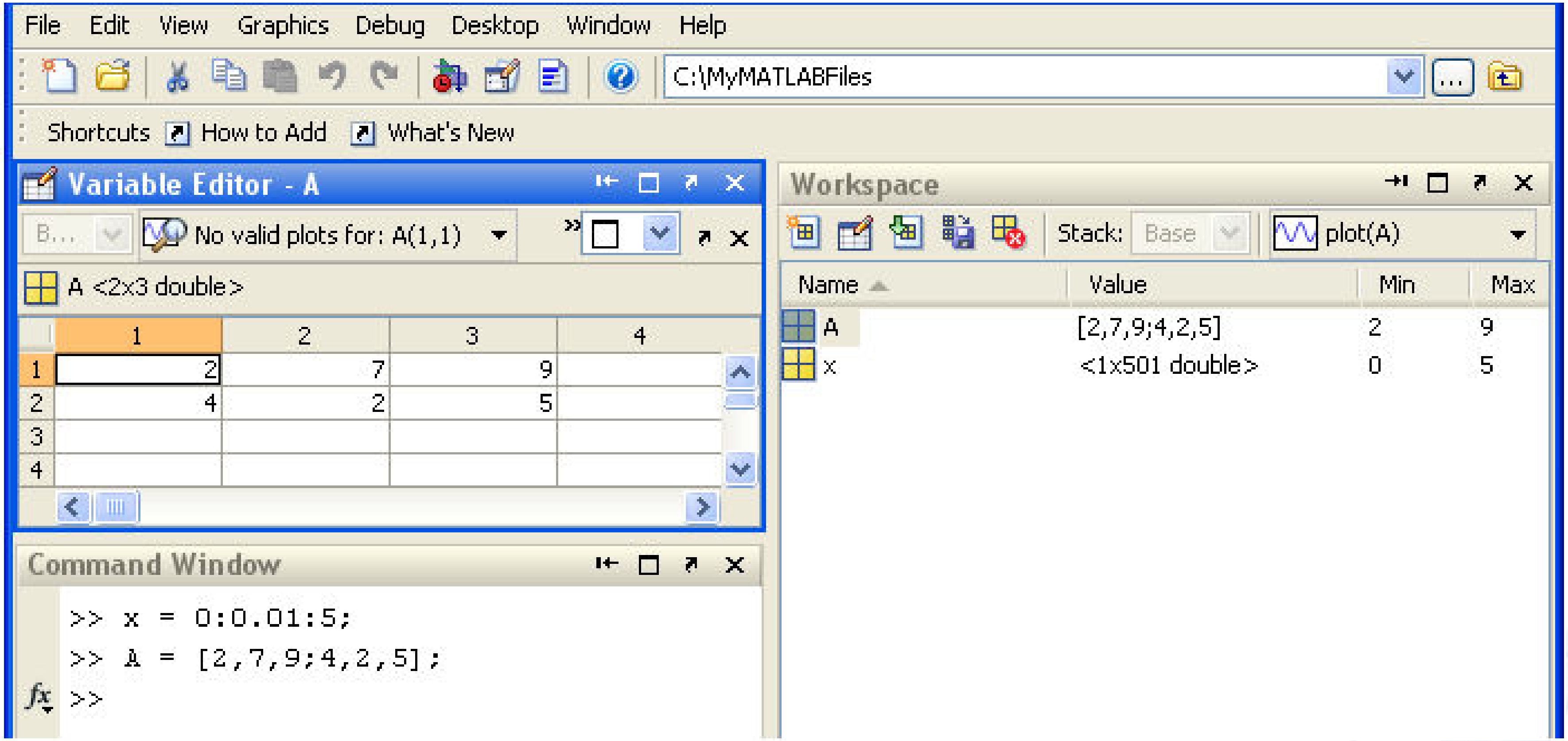
```
>> M = [1 6 4; 3 7 2]

>> size(M)

>> length(M)

>> max(M)

>> [a,b] = max(M)

>> sort(M)

>> sort(M, 1, 'descend')

>> sum(M)

>> sum(M, 2)
```

Solution

```
>> M = [1 6 4; 3 7 2]
M =
     1     6     4
     3     7     2

>> size(M)
ans =
     2     3

>> length(M)
ans =
     3

>> max(M)
ans =
     3     7     4

>> [a,b] = max(M)
a =
     3     7     4
b =
     2     2     1
```

```
>> sort(M)
ans =
     1     6     2
     3     7     4

>> sort(M, 1, 'descend')
ans =
     3     7     4
     1     6     2

>> sum(M)
ans =
     4    13     6

>> sum(M, 2)
ans =
    11
    12
```

The Variable Editor [from Workspace or `openvar('A')`]



Creating Big Matrices

- What if you want to create a Matrix that contains 1000 element (or more)?
- Writing each element by hand is difficult, time-consuming and error-prone.
- MATLAB allows simple ways to quickly create matrices, such as:
- Using the colon `:` operator (very popular).
- Using `linspace()` and `logspace()` functions (less popular, but useful).

Using the colon operator

- MATLAB command `X = J:D:K` creates vector $X = [J, J+D, \ldots, J+m*D]$ where $m = \mathrm{fix}((K-J)/D)$.
- In other words, it creates a vector X of values **starting** at J, **ending** with K, and with **spacing** D.
- Notice that the last element is K if $K - J$ is an integer multiple of D. If not, the last value is *less than* J.
- MATLAB command `J:K` is the same as `J:1:K`.
- Note:
  - `J:K` is empty if $J > K$.
  - `J:D:K` is empty if $D == 0$, if $D > 0$ and $J > K$, or if $D < 0$ and $J < K$.

Example 1

```
>> x = 0:2:8
x =
     0     2     4     6     8

>> x = 0:2:7
x =
     0     2     4     6

>> x = 4:7
x =
     4     5     6     7

>> x = 7:2
x =
   Empty matrix: 1-by-0
```

Example 2

```
>> x = 7:-1:2
x =
     7     6     5     4     3     2

>> x = 5:0.1:5.9
x =
  Columns 1 through 5
    5.0000    5.1000    5.2000    5.3000    5.4000

  Columns 6 through 10
    5.5000    5.6000    5.7000    5.8000    5.9000

>> y = 5:0.1:5.9; % what happened here?!
>>
>> % now create a 'column' vector from 1 to 10 using :
```

Alternatives to colon

- `linspace` command creates a linearly spaced row vector, but instead you specify the number of values rather than the increment.
- The syntax is `linspace(x1,x2,n)`, where `x1` and `x2` are the lower and upper limits and `n` is the number of points.
- If `n` is omitted, the number of points defaults to 100.
- `logspace` command creates an array of logarithmically spaced elements.
- Its syntax is `logspace(a,b,n)`, where `n` is the number of points between 10^a and 10^b.
- If `n` is omitted, the number of points defaults to 50.

Exercise

```
>> x = linspace(5,8,3)

x =

    5.0000    6.5000    8.0000

>> x = logspace(-1,1,4)

x =

    0.1000    0.4642    2.1544   10.0000
```

Special: ones, zeros, rand

```
>> a = ones(2,4)
a =
      1     1     1     1
      1     1     1     1

>> b = zeros(4, 3) % null matrix
b =
      0     0     0
      0     0     0
      0     0     0
      0     0     0

>> c = rand(2, 4)
c =
    0.8147    0.1270    0.6324    0.2785
    0.9058    0.9134    0.0975    0.5469

% random values drawn from the standard
% uniform distribution on the open
% interval(0,1)
```

Null and Identity Matrix

```
>> eye(4) % identity matrix
ans =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1

>> A = [1 2 3; 4 5 6; 7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9

>> I = eye(3)
I =
     1     0     0
     0     1     0
     0     0     1

>> A*I
ans =
     1     2     3
     4     5     6
     7     8     9
```

$$0A = A0 = 0$$

$$IA = AI = A$$

Matrix Determinant & Inverse

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a\begin{vmatrix} e & f \\ h & i \end{vmatrix} - b\begin{vmatrix} d & f \\ g & i \end{vmatrix} + c\begin{vmatrix} d & e \\ g & h \end{vmatrix}$$

$$= a(ei - fh) - b(di - fg) + c(dh - eg)$$

$$= aei + bfg + cdh - ceg - bdi - afh.$$

$$A^{-1} = \frac{1}{|A|}\begin{bmatrix} \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} & \begin{vmatrix} a_{13} & a_{12} \\ a_{33} & a_{32} \end{vmatrix} & \begin{vmatrix} a_{12} & a_{13} \\ a_{22} & a_{23} \end{vmatrix} \\ \begin{vmatrix} a_{23} & a_{21} \\ a_{33} & a_{31} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{vmatrix} & \begin{vmatrix} a_{13} & a_{11} \\ a_{23} & a_{21} \end{vmatrix} \\ \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} & \begin{vmatrix} a_{12} & a_{11} \\ a_{32} & a_{31} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} \end{bmatrix}.$$

```
>> A = [1 2 3; 2 3 1; 3 2 1]
A =
     1     2     3
     2     3     1
     3     2     1

>> det(A) % determinant
ans =
   -12

>> inv(A) % inverse
ans =
   -0.0833   -0.3333    0.5833
   -0.0833    0.6667   -0.4167
    0.4167   -0.3333    0.0833

>> A^-1
ans =
   -0.0833   -0.3333    0.5833
   -0.0833    0.6667   -0.4167
    0.4167   -0.3333    0.0833
```

Accessing Matrix Elements

```
>> C = [10, 11, 13, 12, 19, 16, 17]

C =
    10    11    13    12    19    16    17

>> C(4)
ans =
    12

>> C(1,4)
ans =
    12

>> C(20)
??? Index exceeds matrix dimensions.
```

Notes

- Use `()` not `[]` to access matrix elements.
- The row and column indices are NOT zero-based, like in C/C++.
- The first is row number, followed by the column number.
- For matrices and vectors, you can use one of three indexing methods: matrix row and column indexing; linear indexing; and logical indexing.
- You can also use ranges (shown later).

Accessing Matrix Elements

```
>> M = [1, 3, 2, 9; 6, 7, 8, 1; 7, 4, 6, 0]
M =
     1     3     2     9
     6     7     8     1
     7     4     6     0

>> M(2, 3)
ans =
     8

>> M(3, 1)
ans =
     7

>> M(0, 1)
??? Subscript indices must either be real
positive integers or logicals.

>> M(9)
ans =
     6
```

Matrix Linear Indexing

A =

		Columns (n)				
		1	2	3	4	5
Rows (m)	1	4 1	10 6	1 11	6 16	2 21
	2	8 2	1.2 7	9 12	4 17	25 22
	3	7.2 3	5 8	7 13	1 18	11 23
	4	0 4	0.5 9	4 14	5 19	56 24
	5	23 5	83 10	13 15	0 20	10 25

A (2,4)

A (17)

A = 5 x 5 matrix.

Rectangular Matrix:
Scalar: 1-by-1 array
Vector: m-by-1 array
1-by-n array
Matrix: m-by-n array

Indexing: Sub-matrix

- `v(2:5)` represents the second through fifth elements
  - i.e., v(2), v(3), v(4), v(5).
- `v(2:end)` represents the second till last element of v.
- `v(:)` represents all the row or column elements of vector v.

- `A(:,3)` denotes all elements in the third column of matrix A.
- `A(:,2:5)` denotes all elements in the second through fifth columns of A.
- `A(2:3,1:3)` denotes all elements in the second and third rows that are also in the first through third columns.
- `A(end,:)` all elements of the last row in A.
- `A(:,end)` all elements of the last column in A.
- `v = A(:)` creates a vector v consisting of all the columns of A stacked from first to last.

Exercise

```
>> v = 10:10:70
v =
    10    20    30    40    50    60    70

>> v(2:5)
ans =
    20    30    40    50

>> v(2:end)
ans =
    20    30    40    50    60    70

>> v(:)
ans =
    10
    20
    30
    40
    50
    60
    70
```

Exercise

```
>> A = [4 10 1 6 2; 8 1.2 9 4 25; 7.2 5 7 1
11; 0 0.5 4 5 56; 23 83 13 0 10]

A =
    4.0000   10.0000    1.0000    6.0000    2.0000
    8.0000    1.2000    9.0000    4.0000   25.0000
    7.2000    5.0000    7.0000    1.0000   11.0000
         0    0.5000    4.0000    5.0000   56.0000
   23.0000   83.0000   13.0000         0    0.0000

>> A(:,3)
ans =
     1
     9
     7
     4
    13

>> A(:,2:5)
ans =
   10.0000    1.0000    6.0000    2.0000
    1.2000    9.0000    4.0000   25.0000
    5.0000    7.0000    1.0000   11.0000
    0.5000    4.0000    5.0000   56.0000
   83.0000   13.0000         0   10.0000

>> A(2:3,1:3)
ans =

    8.0000    1.2000    9.0000
    7.2000    5.0000    7.0000
```

```
>> A(end,:)
ans =
    23    83    13     0    10

>> A(:,end)
ans =
     2
    25
    11
    56
    10

>> v = A(:)
v =
    4.0000
    8.0000
    7.2000
         0
   23.0000
   10.0000
    1.2000
    5.0000
    0.5000
   83.0000
    1.0000
    9.0000
    7.0000
    4.0000
   13.0000
    6.0000
    4.0000
    1.0000
    5.0000
         0
    2.0000
   25.0000
   11.0000
   56.0000
   10.0000
```

Linear indexing: *Advanced*

```
>> A = 5:5:50
A =
    5   10   15   20   25   30   35   40   45   50

>> A([1 3 6 10])
ans =
     5    15    30    50

>> A([1 3 6 10]')
ans =
     5    15    30    50

>> A([1 3 6; 7 9 10])
ans =
     5    15    30
    35    45    50
% indexing into a vector with a nonvector,
the shape of the indices is honored
```

Linear indexing is useful: `find`

```
>> A = [1 2 3; 4 5 6; 7 8 9]
A =
      1     2     3
      4     5     6
      7     8     9

>> B = find(A > 5) % returns linear index
B =
      3
      6
      8
      9

>> A(B) % same as A( find(A > 5) )
ans =
      7
      8
      6
      9
```

Advanced: Logical indexing

```
>> A = [1 2 3; 4 5 6; 7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9

>> B = logical([0 1 0; 1 0 1; 0 0 1])
B =
     0     1     0
     1     0     1
     0     0     1

>> A(B)
ans =
     4
     2
     6
     9
```

Logical indexing is also useful!

```
>> A = [1 2 3; 4 5 6; 7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9

>> B = (A > 5) % true or false
B =
     0     0     0
     0     0     1
     1     1     1

>> A(B) % same as A( A > 5 )
ans =
     7
     8
     6
     9
```

Subscripting Examples

A =

	1	2	3	4	5
1	$4\ ^{1}$	$10\ ^{6}$	$1\ ^{11}$	$6\ ^{16}$	$2\ ^{21}$
2	$8\ ^{2}$	$1.2\ ^{7}$	$9\ ^{12}$	$4\ ^{17}$	$25\ ^{22}$
3	$7.2\ ^{3}$	$5\ ^{8}$	$7\ ^{13}$	$1\ ^{18}$	$11\ ^{23}$
4	$0\ ^{4}$	$0.5\ ^{9}$	$4\ ^{14}$	$5\ ^{19}$	$56\ ^{24}$
5	$23\ ^{5}$	$83\ ^{10}$	$13\ ^{15}$	$0\ ^{20}$	$10\ ^{25}$

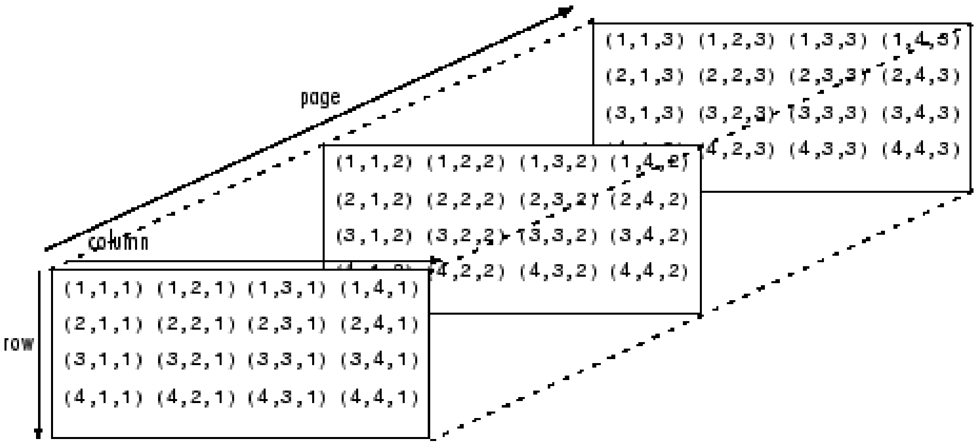
A(3,1)
A(3)

A(4:5,2:3)
A([9 14;10 15])

A(1:5,5)
A(:,5)
A(21:25)

A(1:end,end)
A(:,end)
A(21:end)’

More dimensions possible



- The first index references array dimension 1, the row.
- The second index references dimension 2, the column.
- The third index references dimension 3, the page.

```
>> rand(4,4,3)

ans(:,:,1) =

    0.7431    0.7060    0.0971    0.9502
    0.3922    0.0318    0.8235    0.0344
    0.6555    0.2769    0.6948    0.4387
    0.1712    0.0462    0.3171    0.3816


ans(:,:,2) =

    0.7655    0.4456    0.2760    0.1190
    0.7952    0.6463    0.6797    0.4984
    0.1869    0.7094    0.6551    0.9597
    0.4898    0.7547    0.1626    0.3404


ans(:,:,3) =

    0.5853    0.5060    0.5472    0.8407
    0.2238    0.6991    0.1386    0.2543
    0.7513    0.8909    0.1493    0.8143
    0.2551    0.9593    0.2575    0.2435
```

Extending Matrices

- You can add extra elements to a matrix by creating them directly using `()`
- Or by concatenating (appending) them using `[ , ]` or `[ ; ]`
- If you don't assign array elements, MATLAB gives them a default value of 0

```
>> h = [12 11 14 19 18 17]
h =
   12   11   14   19   18   17

>> h = [h 13]
h =
   12   11   14   19   18   17   13

>> h(10) = 1
h =
   12   11   14   19   18   17   13    0    0    1
```

Example

```
>> a = [2 4 20]
a =
     2     4    20

>> b = [9, -3, 6]
b =
     9    -3     6

>> [a b]
ans =
     2     4    20     9    -3     6

>> [a, b]
ans =
     2     4    20     9    -3     6

>> [a; b]
ans =
     2     4    20
     9    -3     6
```

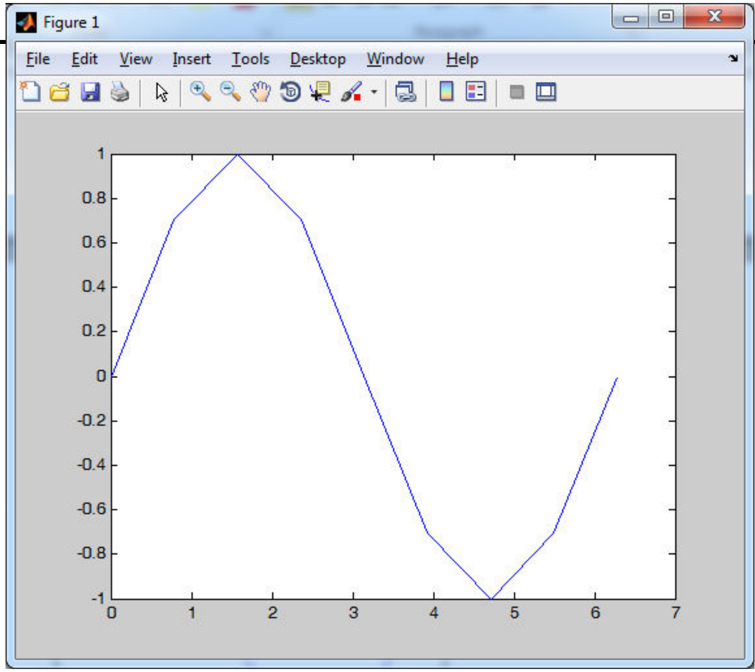
Functions on Arrays

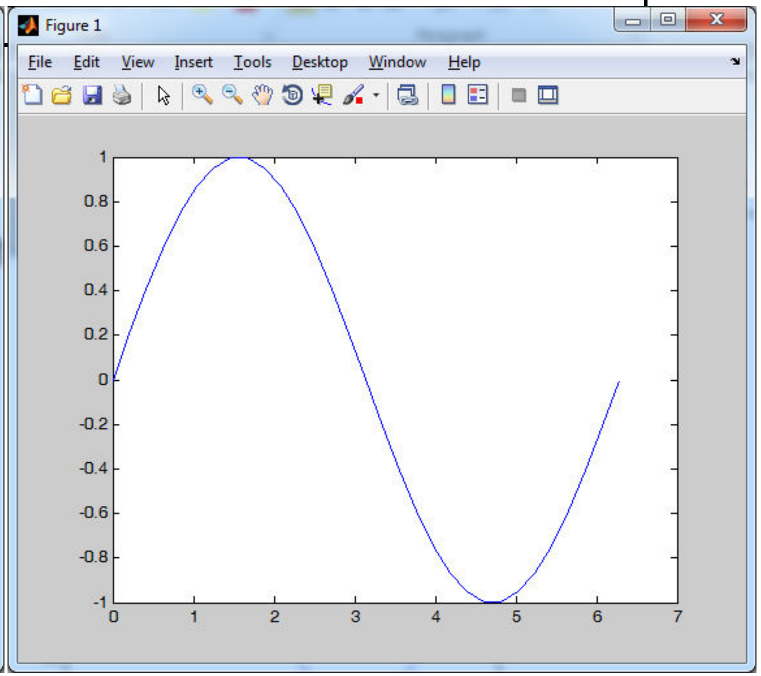
- Standard MATLAB functions (sin, cos, exp, log, etc) can apply to vectors and matrices as well as scalars.
- They operate on array arguments to produce an array result the same size as the array argument x.
- These functions are said to be vectorized functions.
- In this example `y` is [sin(1), sin(2), sin(3)]
- So, when writing functions (later lectures) remember input might be a vector or matrix.

```
>> x = [1, 2, 3]
x =
     1     2     3

>> y = sin(x)
y =
    0.8415    0.9093    0.1411
```

Exercise

```
>> x = linspace(0, 2*pi, 9) % OR x = linspace(0, 2*pi, 31)
x =
         0    0.7854    1.5708    2.3562    3.1416    3.9270    4.7124    5.4978    6.2832

>> y = sin(x)
y =
         0    0.7071    1.0000    0.7071    0.0000   -0.7071   -1.0000   -0.7071   -0.0000

>> plot(x,y)
```





Matrix vs. Array Arithmetic

- Multiplying and dividing vectors and matrices is different than multiplying and dividing scalars (or arrays of scalars).
- This is why MATLAB has two types of arithmetic operators:
  - **Array** operators: where the arrays operated on have the same size. The operation is done element-by-element (for all elements).
  - **Matrix** operators: dedicated for matrices and vectors. Operations are done using the matrix as a whole.

Matrix vs. Array Operators

Symbol	Operation	Symbol	Operation
+	Matrix addition	+	Array addition
−	Matrix subtraction	−	Array subtraction
*	Matrix multiplication	.*	Array multiplication
/	Matrix division	./	Array division
\	Left matrix division	.\	Left array division
^	Matrix power	.^	Array power

* idivide() allows integer division with rounding options

Matrix/Array Addition/Subtraction

- Matrices and arrays are treated the same when adding and subtracting.
- The two matrices should have identical size.
- Their sum or difference has the same size, and is obtained by adding or subtracting the corresponding elements.
- Addition and subtraction are associative and commutative.

$$\begin{bmatrix} 6 & -2 \\ 10 & 3 \end{bmatrix} + \begin{bmatrix} 9 & 8 \\ -12 & 14 \end{bmatrix} = \begin{bmatrix} 15 & 6 \\ -2 & 17 \end{bmatrix}$$

```
>>A = [6,-2;10,3];
>>B = [9,8;-12,14]
>>A+B
ans =
      15    6
      -2    17
```

$$(\mathbf{A} + \mathbf{B}) + \mathbf{C} = \mathbf{A} + (\mathbf{B} + \mathbf{C})$$

$$\mathbf{A} + \mathbf{B} + \mathbf{C} = \mathbf{B} + \mathbf{C} + \mathbf{A} = \mathbf{A} + \mathbf{C} + \mathbf{B}$$

More …

- A scalar value at either side of the operator is expanded to an array of the same size as the other side of the operator.

$$[6,3]+2=[8,5]$$
$$[8,3]-5=[3,-2]$$
$$[6,5]+[4,8]=[10,13]$$
$$[6,5]-[4,8]=[2,-3]$$

Array Multiplication

- Element-by-element multiplication.
- Only for arrays that are the same size.
- Use the `.*` operator not the `*` operator.
- Not the same as matrix multiplication.
- Useful in programming, but students make the mistake of using *

$$\mathbf{A} = \begin{bmatrix} 11 & 5 \\ -9 & 4 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} -7 & 8 \\ 6 & 2 \end{bmatrix}$$

```
C = A.*B
```

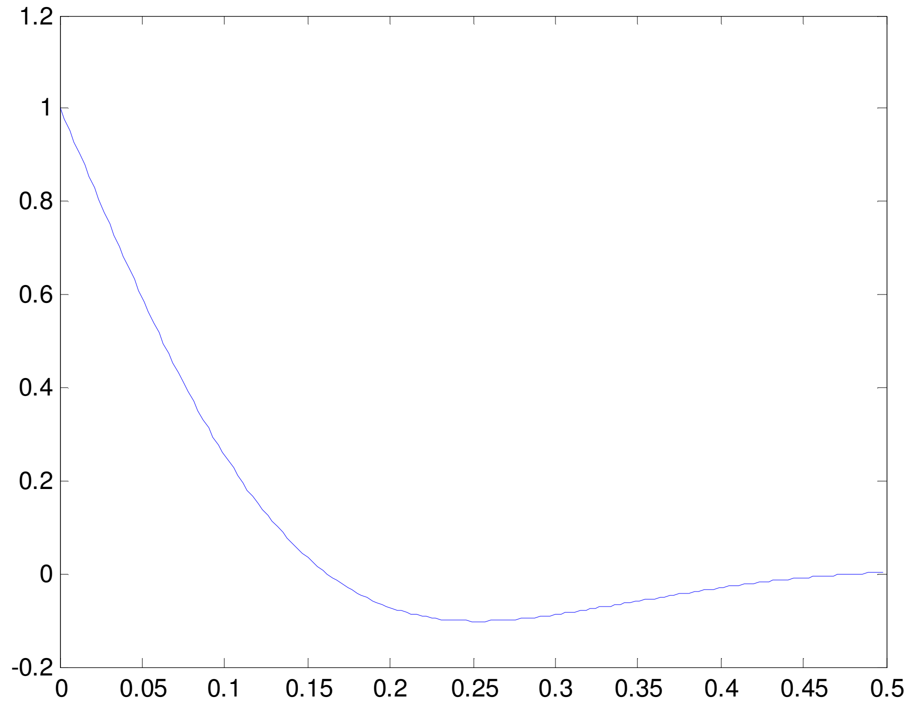
$$\mathbf{C} = \begin{bmatrix} 11(-7) & 5(8) \\ -9(6) & 4(2) \end{bmatrix} = \begin{bmatrix} -77 & 40 \\ -54 & 8 \end{bmatrix}$$

Using Array Multiplication (Plot)

- Plot the following function:
- Notice the use of `.*` operator

```
>> t = 0:0.003:0.5;
>> y = exp(-8*t).*sin(9.7*t+pi/2);
>> plot(t,y)
```

$$y(t) = e^{-8t}\sin\left(9.7t + \frac{\pi}{2}\right)$$



Matrix Multiplication

- If A is an $n \times m$ matrix and B is a $m \times p$ matrix, their matrix product AB is an $n \times p$ matrix, in which the m entries across the rows of A are multiplied with the m entries down the columns of B.
- In general, AB ≠ BA for matrices. Be extra careful.

$$\begin{bmatrix} 2 & 7 \\ 6 & -5 \end{bmatrix}\begin{bmatrix} 3 \\ 9 \end{bmatrix} = \begin{bmatrix} 2(3) + 7(9) \\ 6(3) - 5(9) \end{bmatrix} = \begin{bmatrix} 69 \\ -27 \end{bmatrix}$$

$$\begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix}\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = u_1 w_1 + u_2 w_2 + u_3 w_3$$

Matrix Multiplication

$$\begin{bmatrix} 6 & -2 \\ 10 & 3 \\ 4 & 7 \end{bmatrix} \begin{bmatrix} 9 & 8 \\ -5 & 12 \end{bmatrix} = \begin{bmatrix} (6)(9) + (-2)(-5) & (6)(8) + (-2)(12) \\ (10)(9) + (3)(-5) & (10)(8) + (3)(12) \\ (4)(9) + (7)(-5) & (4)(8) + (7)(12) \end{bmatrix}$$

$$= \begin{bmatrix} 64 & 24 \\ 75 & 116 \\ 1 & 116 \end{bmatrix} \qquad (2.4\text{–}4)$$

```
>> A = [6,-2;10,3;4,7];
>> B = [9,8;-5,12];
>> A*B
ans =
   64    24
   75   116
    1   116
```

$$3 \begin{bmatrix} 2 & 9 \\ 5 & -7 \end{bmatrix} = \begin{bmatrix} 6 & 27 \\ 15 & -21 \end{bmatrix}$$

```
>>A = [2,9;5,-7];
>>3*A
```

Array Division

- Element-by-element division.
- Only for arrays that are the same size.
- Use the `./` operator not the `/` operator.
- Not the same as matrix division.
- Useful in programming, but students make the mistake of using /

$$\mathbf{A} = \begin{bmatrix} 24 & 20 \\ -9 & 4 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} -4 & 5 \\ 3 & 2 \end{bmatrix}$$

```
C = A./B
```

$$\mathbf{C} = \begin{bmatrix} 24/(-4) & 20/5 \\ -9/3 & 4/2 \end{bmatrix} = \begin{bmatrix} -6 & 4 \\ -3 & 2 \end{bmatrix}$$

Matrix Division

- An $n \times n$ square matrix **B** is called invertible (also nonsingular) if there exists an $n \times n$ matrix $\mathbf{B}^{-1}$ such that their multiplication is the identity matrix.

$$\frac{\mathbf{A}}{\mathbf{B}} = \mathbf{A}\,\mathbf{B}^{-1}$$

$$\mathbf{B}\,\mathbf{B}^{-1} = \mathbf{I}$$

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 2 & 1 & 3 \end{bmatrix} \qquad B = \begin{bmatrix} 4 & 5 & 6 \\ 6 & 5 & 4 \\ 4 & 6 & 5 \end{bmatrix}$$

$$B^{-1} = \begin{bmatrix} \frac{1}{30} & \frac{11}{30} & \frac{-1}{3} \\ \frac{-7}{15} & \frac{-2}{15} & \frac{2}{3} \\ \frac{8}{15} & \frac{-2}{15} & \frac{-1}{3} \end{bmatrix}$$

Matrix Division

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 2 & 1 & 3 \end{bmatrix} \quad B = \begin{bmatrix} 4 & 5 & 6 \\ 6 & 5 & 4 \\ 4 & 6 & 5 \end{bmatrix}$$

$$A . B^{-1}$$

$$= \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 2 & 1 & 3 \end{bmatrix} . \begin{bmatrix} \frac{1}{30} & \frac{11}{30} & \frac{-1}{3} \\ \frac{-7}{15} & \frac{-2}{15} & \frac{2}{3} \\ \frac{8}{15} & \frac{-2}{15} & \frac{-1}{3} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{7}{10} & \frac{-3}{10} & 0 \\ \frac{-3}{10} & \frac{7}{10} & 0 \\ \frac{6}{5} & \frac{1}{5} & -1 \end{bmatrix}$$

```
>> A = [1 2 3; 3 2 1; 2 1 3];
>> B = [4 5 6; 6 5 4; 4 6 5];
>> A/B
ans =
    0.7000   -0.3000         0
   -0.3000    0.7000    0.0000
    1.2000    0.2000   -1.0000

>> format rat
>> A/B
ans =
     7/10          -3/10            0
    -3/10           7/10            *
     6/5            1/5            -1
```

Matrix Left Division

- Use the left division operator (\) (back slash) to solve sets of linear algebraic equations.
- If A is $n \times n$ matrix and B is a column vector with n elements, then x = A\B is the solution to the equation Ax = B.
- A warning message is displayed if A is badly scaled or nearly singular.

$$6x + 12y + 4z = 70$$
$$7x - 2y + 3z = 5$$
$$2x + 8y - 9z = 64$$

```
>>A = [6,12,4;7,-2,3;2,8,-9];
>>B = [70;5;64];
>>Solution = A\B
Solution =
      3
      5
     -2
```

The solution is $x = 3$, $y = 5$, and $z = -2$.

Homework: Mesh Analysis

KVL @ mesh 2:

$$1(i_2 - i_1) + 2i_2 + 3(i_2 - i_3) = 0$$

KVL @ supermesh 1/3:

$$-7 + 1(i_1 - i_2) + 3(i_3 - i_2) + 1i_3 = 0$$

@ current source:

$$7 = i_1 - i_3$$

Three equations:

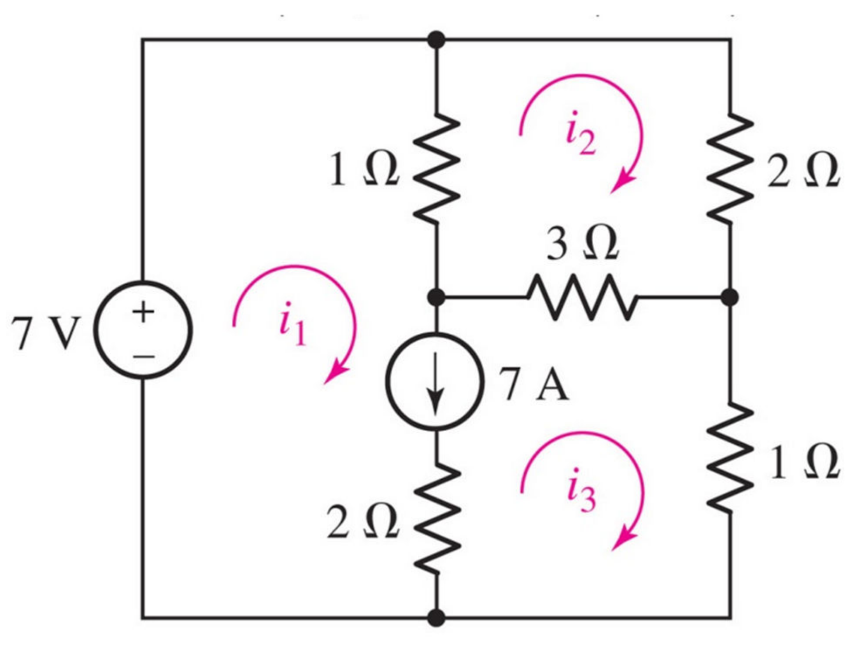
$-i_1 + 6i_2 - 3i_3 = 0$

$i_1 - 4i_2 + 4i_3 = 7$

$i_1 - i_3 = 7$

Solution:

$i_1 = 9\text{A}, i_2 = 2.5\text{A}, i_3 = 2\text{A}$



Just between us…

- Matrix division and matrix left division are related in MATLAB by the equation:

```
B/A = (A'\B')' % reversing
```

- To see the details, type: `doc mldivide` or type: `doc mrdivide`

Array Left Division

- The array left division `A.\B` (back slash) divides each entry of B by the corresponding entry of A.
- Just like `B./A`
- A and B must be arrays of the same size.
- A scalar value for either A or B is expanded to an array of the same size as the other.

```
>> A = [-4 5; 3 2];
>> B = [24 20; -9 4];

>> A.\B % notice the back slash
ans =
      -6              4
      -3              2

>> B./A
ans =
      -6              4
      -3              2
```

Array Power

```
B = A.^3
```

$$\mathbf{B} = \begin{bmatrix} 4^3 & (-5)^3 \\ 2^3 & 3^3 \end{bmatrix} = \begin{bmatrix} 64 & -125 \\ 8 & 27 \end{bmatrix}$$

```
[3,5].^2=[3^2,5^2]
2.^[3,5]=[2^3,2^5]
[3,5].^[2,4]=[3^2,5^4]
```

```
p = [2, 4, 5]
```

```
3.^p
3.0.^p
3..^p
(3).^p
3.^[2,4,5]
```

Matrix Power

- `A^k` computes matrix power (exponent).
- In other words, it multiplies matrix **A** by itself *k* times.
- The exponent *k* requires a positive, real-valued integer value.
- Remember: this is repeated *matrix* multiplication

```
>> A = [1 2; 3 4];
>> A^3
ans =
    37    54
    81   118

>> A*A*A
ans =
    37    54
    81   118
```

Matrix Manipulation Functions

- `diag`: Diagonal matrices and diagonal of a matrix.
- `det`: Matrix determinant
- `inv`: Matrix inverse
- `cond`: Matrix condition number (for inverse)
- `fliplr`: Flip matrices left-right
- `flipud`: Flip matrices up and down
- `repmat`: Replicate and tile a matrix

Matrix Manipulation Functions

- `rot90`: rotate matrix 90^{o}
- `tril`: Lower triangular part of a matrix
- `triu`: Upper triangular part of a matrix
- `cross`: Vector cross product
- `dot`: Vector dot product
- `eig`: Evaluate eigenvalues and eigenvectors
- `rank`: Rank of matrix

Exercise

```
>> A = [1 2 3; 4 5 6; 7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9

>> diag(A)
ans =
     1
     5
     9

>> det(A)
ans =
  6.6613e-016
```

```
>> fliplr(A)
ans =
     3     2     1
     6     5     4
     9     8     7

>> flipud(A)
ans =
     7     8     9
     4     5     6
     1     2     3

>> rot90(A)
ans =
     3     6     9
     2     5     8
     1     4     7
```

Exercise

```
>> A = [1 2 3; 4 5 6; 7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9

>> tril(A)
ans =
     1     0     0
     4     5     0
     7     8     9

>> triu(A)
ans =
     1     2     3
     0     5     6
     0     0     9
```

```
>> [V, D] = eig(A)

V =
  -0.2320   -0.7858    0.4082
  -0.5253   -0.0868   -0.8165
  -0.8187    0.6123    0.4082


D =
  16.1168         0         0
        0   -1.1168         0
        0         0   -0.0000
```

Exercise

- Define matrix A of dimension 2 by 4 whose (i,j) entries are A(i,j) = i+j
- Extract two 2 by 2 matrices A1 and A2 out of matrix A.
  - A1 contains the first two columns of A
  - A2 contains the last two columns of A
- Compute matrix B to be the sum of A1 and A2
- Compute the eigenvalues and eigenvectors of B
- Solve the linear system B x = b, where b has all entries = 2
- Compute the determinant of B, inverse of B, and the condition number of B
- NOTE: Use only MATLAB native functions for all above.

Solution

```
>> A =[0 1 2 3; 1 2 3 4]
A =
     0     1     2     3
     1     2     3     4

>> A1 = A(:,1:2)
A1 =
     0     1
     1     2

>> A2 = A(:,3:4)
A2 =
     2     3
     3     4

>> B = A1 + A2
B =
     2     4
     4     6
```

```
>> b = [2; 2]
b =
     2
     2

>> B\b
ans =
   -1.0000
    1.0000

>> det(B)
ans =
    -4

>> inv(B)
ans =
   -1.5000    1.0000
    1.0000   -0.5000

>> cond(B)
ans =
   17.9443
```

Homework

- Solve as many problems from Chapter 1 as you can
- Suggested problems:
- 1.3, 1.8, 1.15, 1.26, 1.30
- Solve as many problems from Chapter 2 as you can
- Suggested problems:
- 2.3, 2.10, 2.13, 2.25, 2.26