Matlab Tutorial

Lecture Learning Objectives

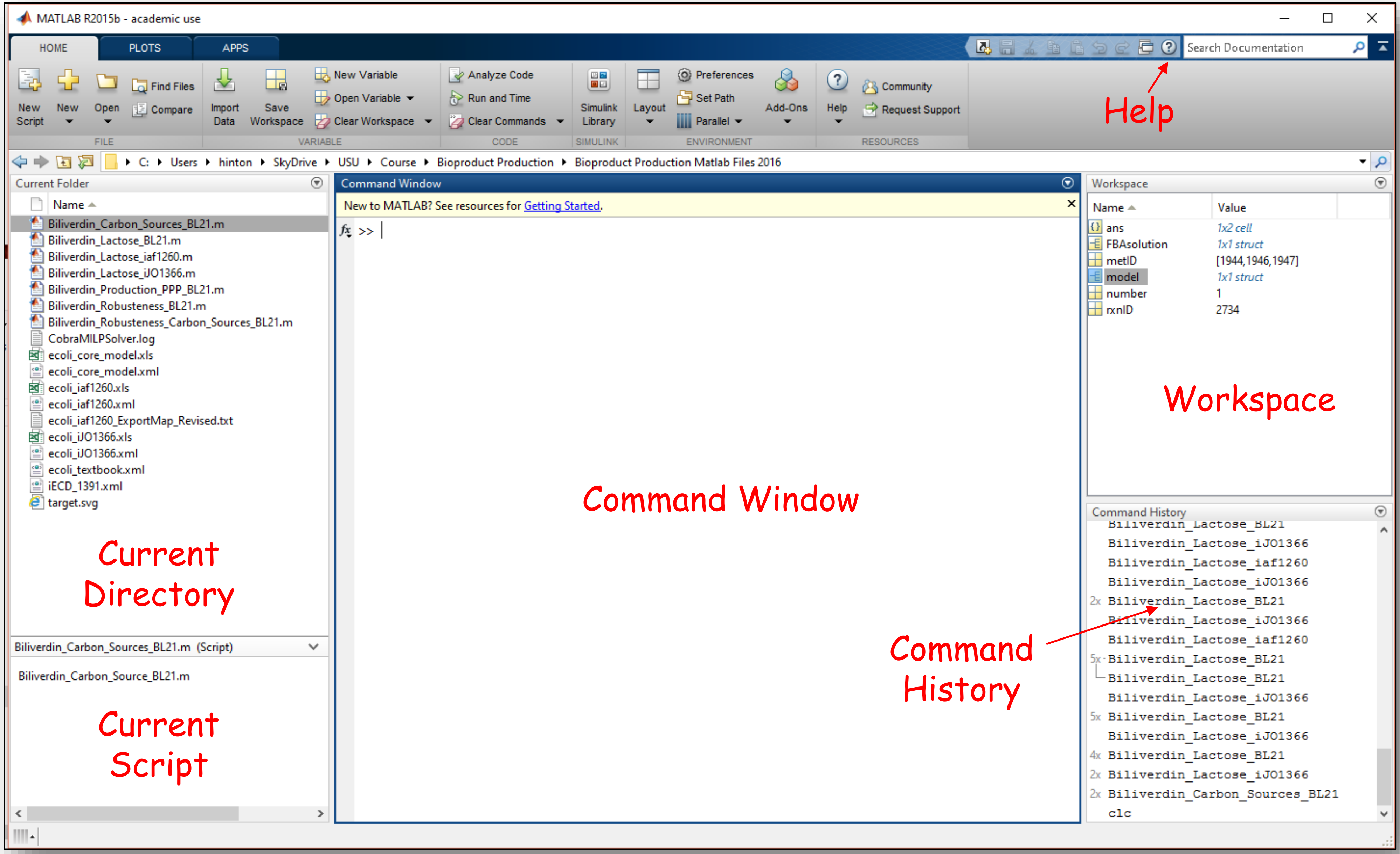
Each student should be able to:

- Describe the Matlab desktop
- Explain the basic use of Matlab variables
- Explain the basic use of Matlab scripts
- Explain the basic mathematical operations in Matlab
- Explain the simple Matlab visualization techniques
- Explain simple Matlab programming
- Explain the basic data structures available in Matlab

Course Introduction

- Desktop
- Variables
- Scripts
- Operations
- Visualization
- Programming
- Data Structures

Matlab Desktop



Biliverdin_Carbon_Sources_BL21.m
Biliverdin_Lactose_BL21.m
Biliverdin_Lactose_iaf1260.m
Biliverdin_Lactose_iJO1366.m
Biliverdin_Production_PPP_BL21.m
Biliverdin_Robusteness_BL21.m
Biliverdin_Robusteness_Carbon_Sources_BL21.m
CobraMILPSolver.log
ecoli_core_model.xls
ecoli_core_model.xml
ecoli_iaf1260.xls
ecoli_iaf1260.xml
ecoli_iaf1260_ExportMap_Revised.txt
ecoli_iJO1366.xls
ecoli_iJO1366.xml
ecoli_textbook.xml
iECD_1391.xml
target.svg

Biliverdin_Carbon_Sources_BL21.m (Script)

Biliverdin_Carbon_Source_BL21.m

New to MATLAB? See resources for Getting Started.

```
>>
```

Name	Value
ans	1x2 cell
FBAsolution	1x1 struct
metID	[1944,1946,1947]
model	1x1 struct
number	1
rxnID	2734

```
   Biliverdin_Lactose_BL21
   Biliverdin_Lactose_iJO1366
   Biliverdin_Lactose_iaf1260
   Biliverdin_Lactose_iJO1366
2x Biliverdin_Lactose_BL21
   Biliverdin_Lactose_iJO1366
   Biliverdin_Lactose_iaf1260
5x Biliverdin_Lactose_BL21
   Biliverdin_Lactose_BL21
   Biliverdin_Lactose_iJO1366
5x Biliverdin_Lactose_BL21
   Biliverdin_Lactose_iJO1366
4x Biliverdin_Lactose_BL21
2x Biliverdin_Lactose_iJO1366
2x Biliverdin_Carbon_Sources_BL21
   clc
```

Help

Workspace

Command Window

Current Directory

Current Script

Command History

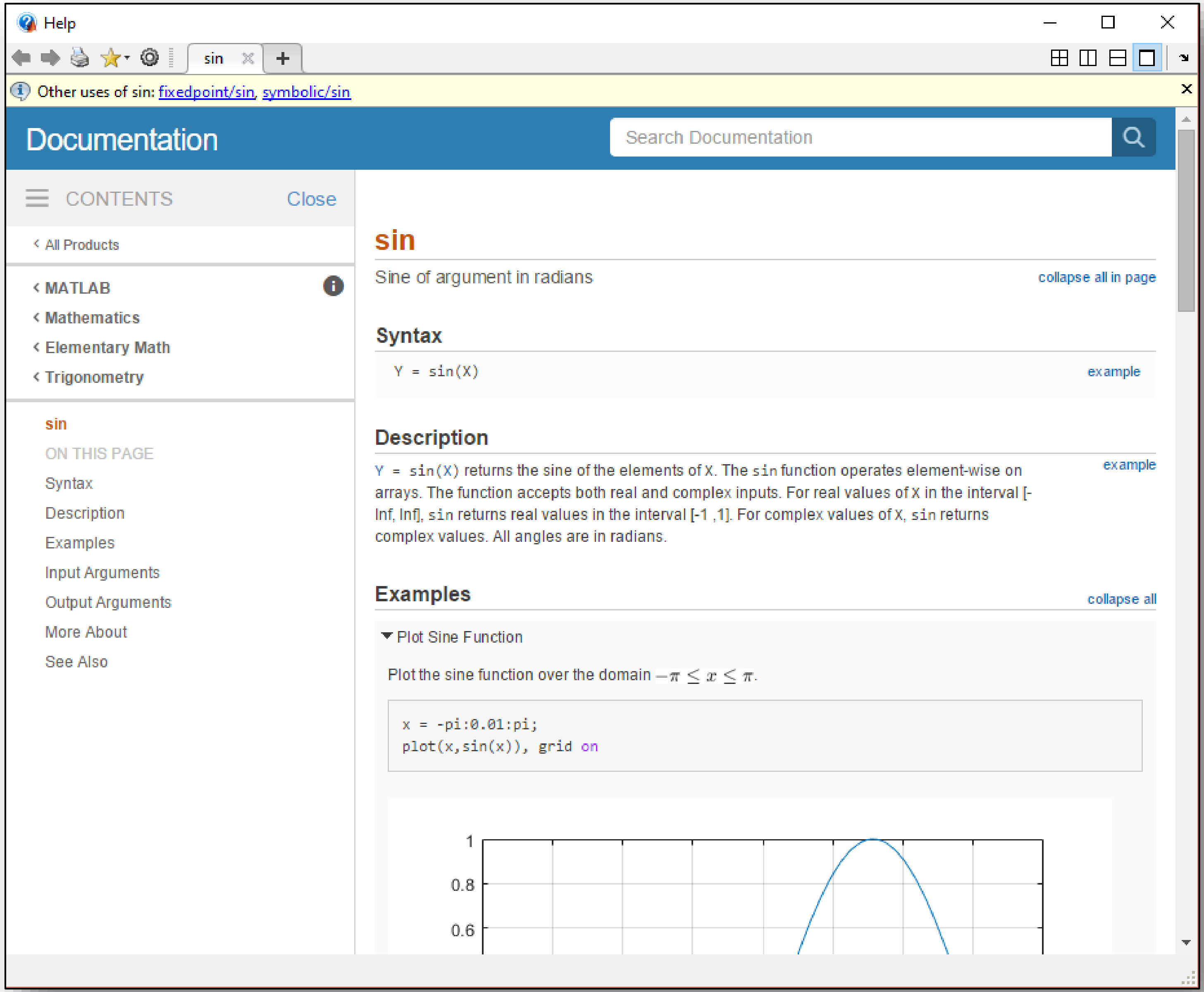
http://www.mathworks.com/products/matlab/

Overview and Help

- MATLAB can be used as a super-powerful graphing calculator
- It is also a programming language
  - ✓ MATLAB is an interpreted language like Java
  - ✓ Commands are executed line by line
- Help/documentation can be found with the doc command
  - ✓ Example: doc sin
  - ✓ Same as "help"

✓ "clc" clears the command window

✓ "clear" clears the workspace



Other uses of sin: fixedpoint/sin, symbolic/sin

sin

Sine of argument in radians

Syntax

```
Y = sin(X)
```

Description

Y = sin(X) returns the sine of the elements of X. The sin function operates element-wise on arrays. The function accepts both real and complex inputs. For real values of X in the interval [-Inf, Inf], sin returns real values in the interval [-1 ,1]. For complex values of X, sin returns complex values. All angles are in radians.

Examples

Plot Sine Function

Plot the sine function over the domain $-\pi \le x \le \pi$.

```
x = -pi:0.01:pi;
plot(x,sin(x)), grid on
```

Course Introduction

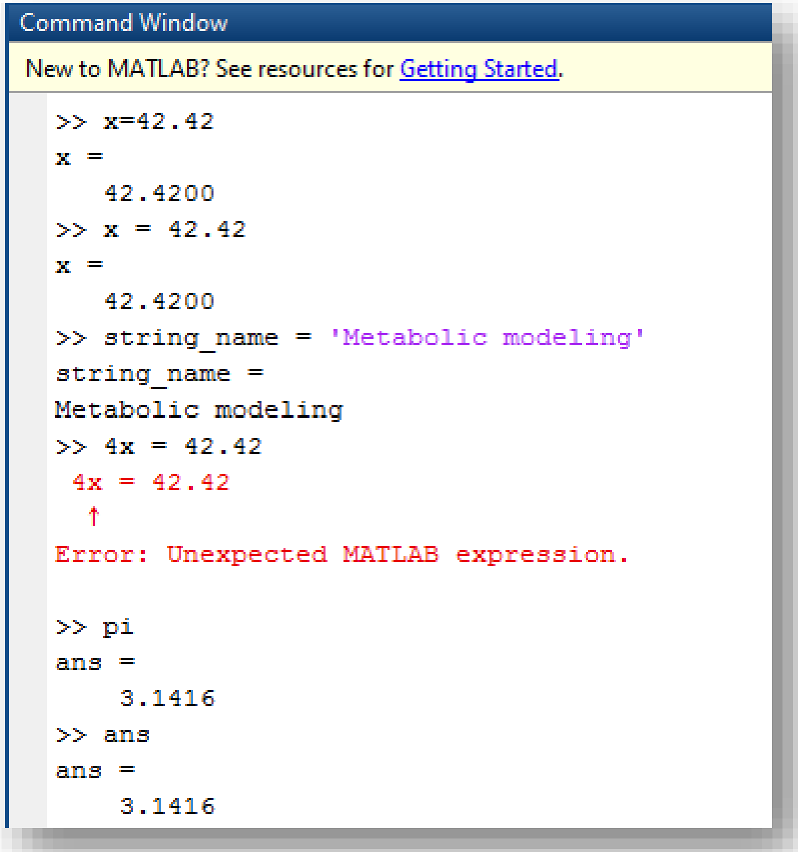
- Desktop
- ➡ Variables
- Scripts
- Operations
- Visualization
- Programming
- Data Structures

Variable Types

- MATLAB is a weakly typed language
  - ✓No need to initialize variables!
- MATLAB supports various types, the most often used are
  - ✓Numbers (42.42) are 64-bit double precision (default)
  - ✓Alphanumeric characters ('b') are 16-bit precision (default)
- Most variables will be vectors or matrices of numbers or alphanumeric characters
- Other types are also supported, including: complex, symbolic, 16-bit and 8 bit integers, etc.

Naming Variables

- To create a variable, simply assign a value to a name:
  - ✓ » `x = 42.42`
  - ✓ » `string = 'name'`
- Variable names
  - ✓ The first character must be a LETTER, after that, any combination of letters, numbers and _
  - ✓ Matlab is CASE SENSITIVE! (x is different from X)
- Built-in variables. Don't use these names!
  - ✓ "i" and "j" are used to indicate complex numbers
  - ✓ "pi" has the value 3.1415926…
  - ✓ "ans" stores the last unassigned value (like on a calculator)
  - ✓ "Inf" and "-Inf" are positive and negative infinity
  - ✓ "NaN" represents "Not a Number"

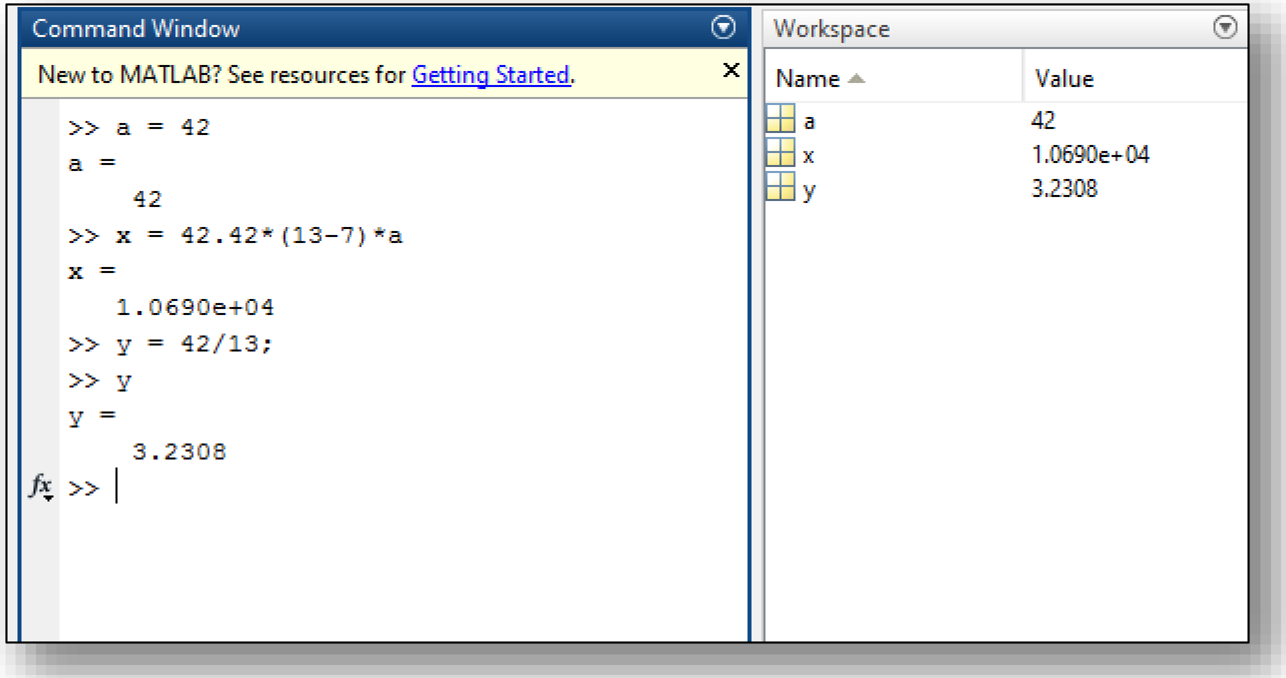


New to MATLAB? See resources for Getting Started.

```
>> x=42.42
x =
   42.4200
>> x = 42.42
x =
   42.4200
>> string_name = 'Metabolic modeling'
string_name =
Metabolic modeling
>> 4x = 42.42
 4x = 42.42
  ↑
Error: Unexpected MATLAB expression.

>> pi
ans =
    3.1416
>> ans
ans =
    3.1416
```

Using Variables

- A variable can be given a value explicitly
  - ✓ » a = 42 (Note that a shows up in workspace!)
- A variable can be used as a function of explicit values and existing variables
  - ✓ » x = 42.42*(13-7)*a
- To suppress the output, end the line with a semicolon
  - ✓ » y = 42/13;



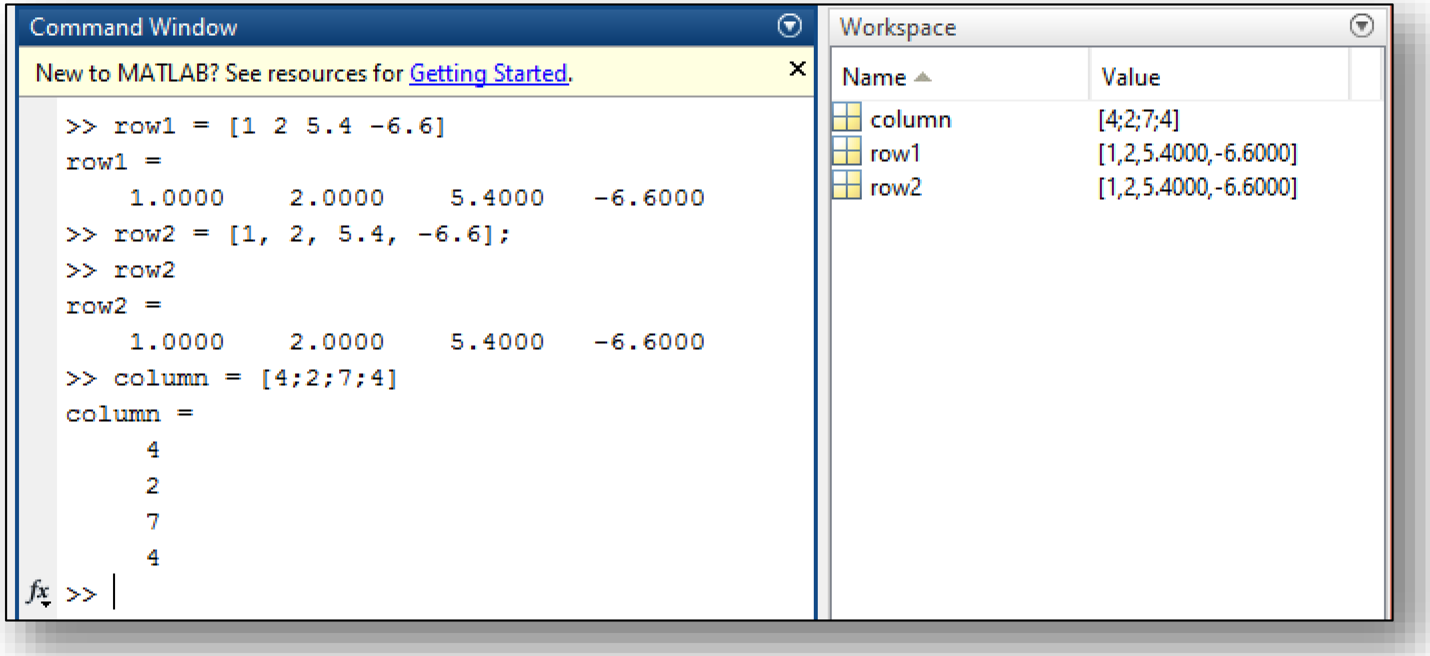
New to MATLAB? See resources for Getting Started.

```
>> a = 42
a =
    42
>> x = 42.42*(13-7)*a
x =
  1.0690e+04
>> y = 42/13;
>> y
y =
    3.2308
>> 
```

Name	Value
a	42
x	1.0690e+04
y	3.2308

Arrays

- Like other programming languages, arrays are an important part of MATLAB
- There are two types of arrays
  - ✓ Matrix of numbers (either double or complex)
  - ✓ Cell array of objects (advanced data structure)
- Row vectors: Use a comma or space to separate values between brackets
  - ✓ » `row1 = [1 2 5.4 -6.6]`
  - ✓ » `row2 = [1, 2, 5.4, -6.6];`
- Column vectors: Use a semicolon to separate values between brackets
  - ✓ » `column = [4;2;7;4]`



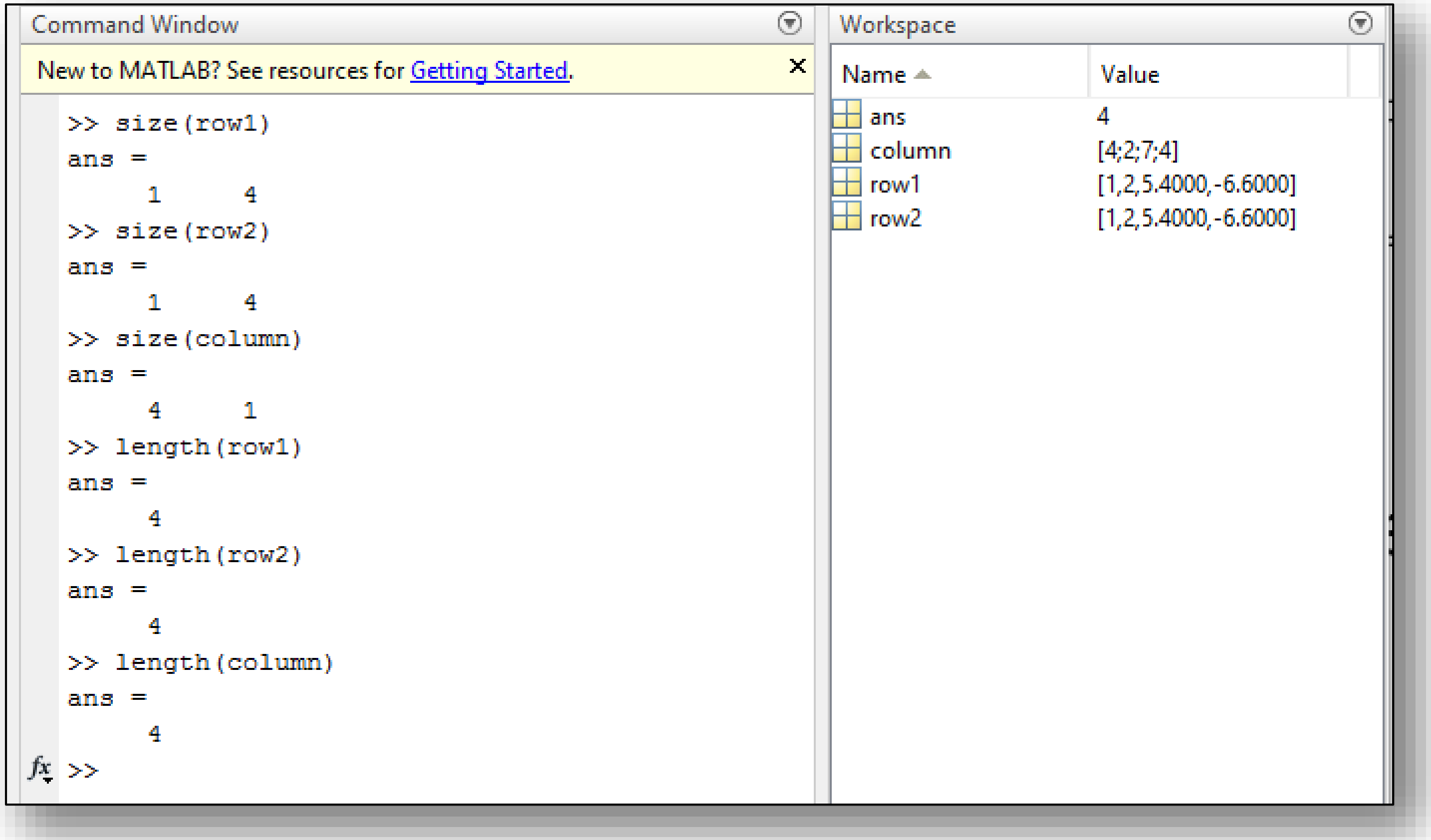
New to MATLAB? See resources for Getting Started.

```
>> row1 = [1 2 5.4 -6.6]
row1 =
    1.0000    2.0000    5.4000   -6.6000
>> row2 = [1, 2, 5.4, -6.6];
>> row2
row2 =
    1.0000    2.0000    5.4000   -6.6000
>> column = [4;2;7;4]
column =
     4
     2
     7
     4
>> 
```

Name	Value
column	[4;2;7;4]
row1	[1,2,5.4000,-6.6000]
row2	[1,2,5.4000,-6.6000]

Size & Length

- The difference between a row and a column vector can be seen by:
  - ✓ Looking at the workspace
  - ✓ Displaying the variable in the command window
  - ✓ Using the size function
- Use the length function to get a vector's length

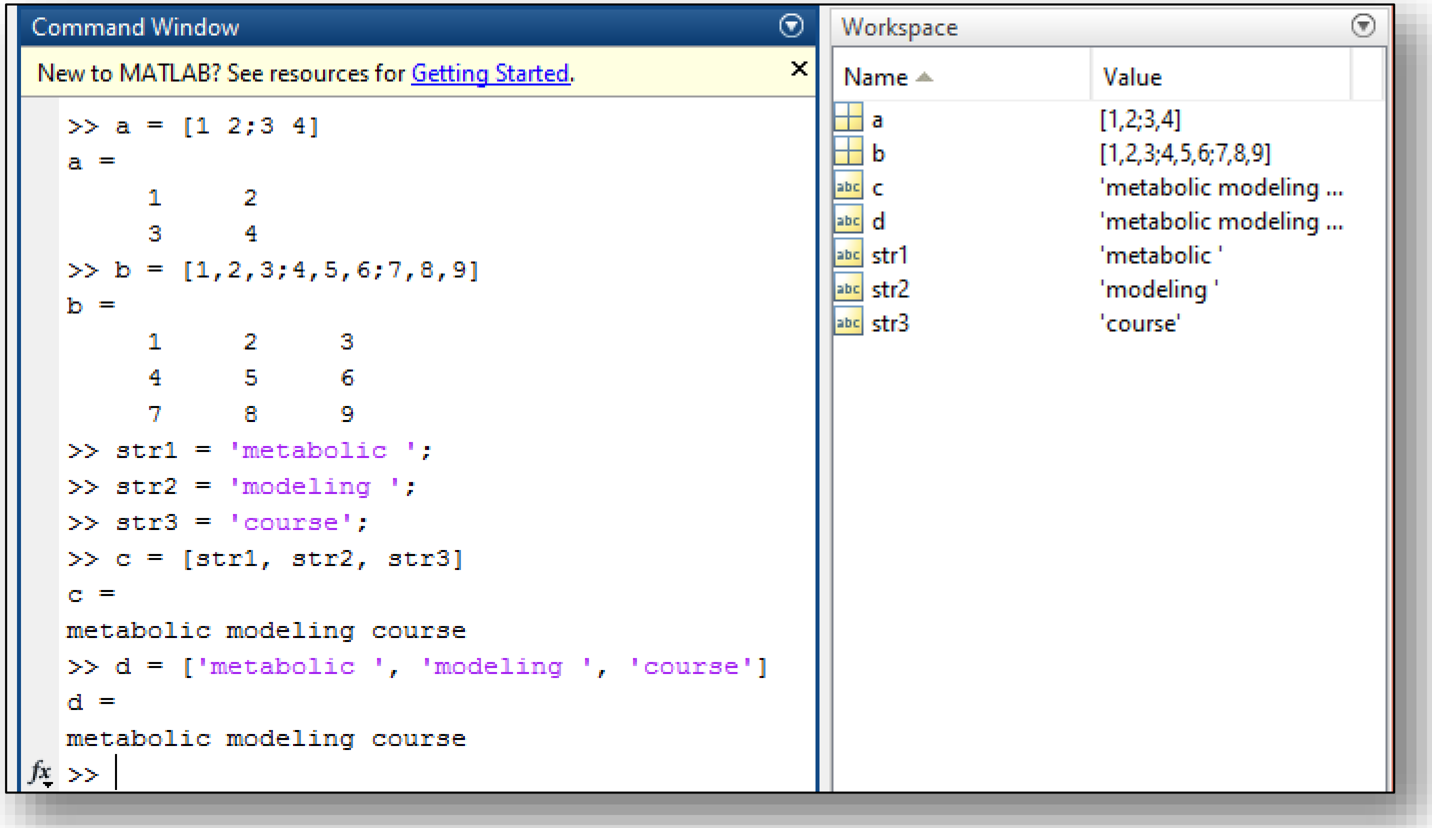


```
>> size(row1)
ans =
     1     4
>> size(row2)
ans =
     1     4
>> size(column)
ans =
     4     1
>> length(row1)
ans =
     4
>> length(row2)
ans =
     4
>> length(column)
ans =
     4
>>
```

Name	Value
ans	4
column	[4;2;7;4]
row1	[1,2,5.4000,-6.6000]
row2	[1,2,5.4000,-6.6000]

Matrices

- Make matrices like vectors
- Construct matrix element by element
  - ✓ » a = [1 2;3 4]
  - ✓ » b = [1,2,3;4,5,6;7,8,9]
- Strings are character vectors
  - ✓ » str1 = 'metabolic ';
  - ✓ » str2 = 'modeling ';
  - ✓ » str3 = 'course';
  - ✓ » c = [str1, str2, str3]
  - ✓ » d = ['metabolic ', 'modeling ', 'course']



New to MATLAB? See resources for Getting Started.

```
>> a = [1 2;3 4]
a =
     1     2
     3     4
>> b = [1,2,3;4,5,6;7,8,9]
b =
     1     2     3
     4     5     6
     7     8     9
>> str1 = 'metabolic ';
>> str2 = 'modeling ';
>> str3 = 'course';
>> c = [str1, str2, str3]
c =
metabolic modeling course
>> d = ['metabolic ', 'modeling ', 'course']
d =
metabolic modeling course
>>
```

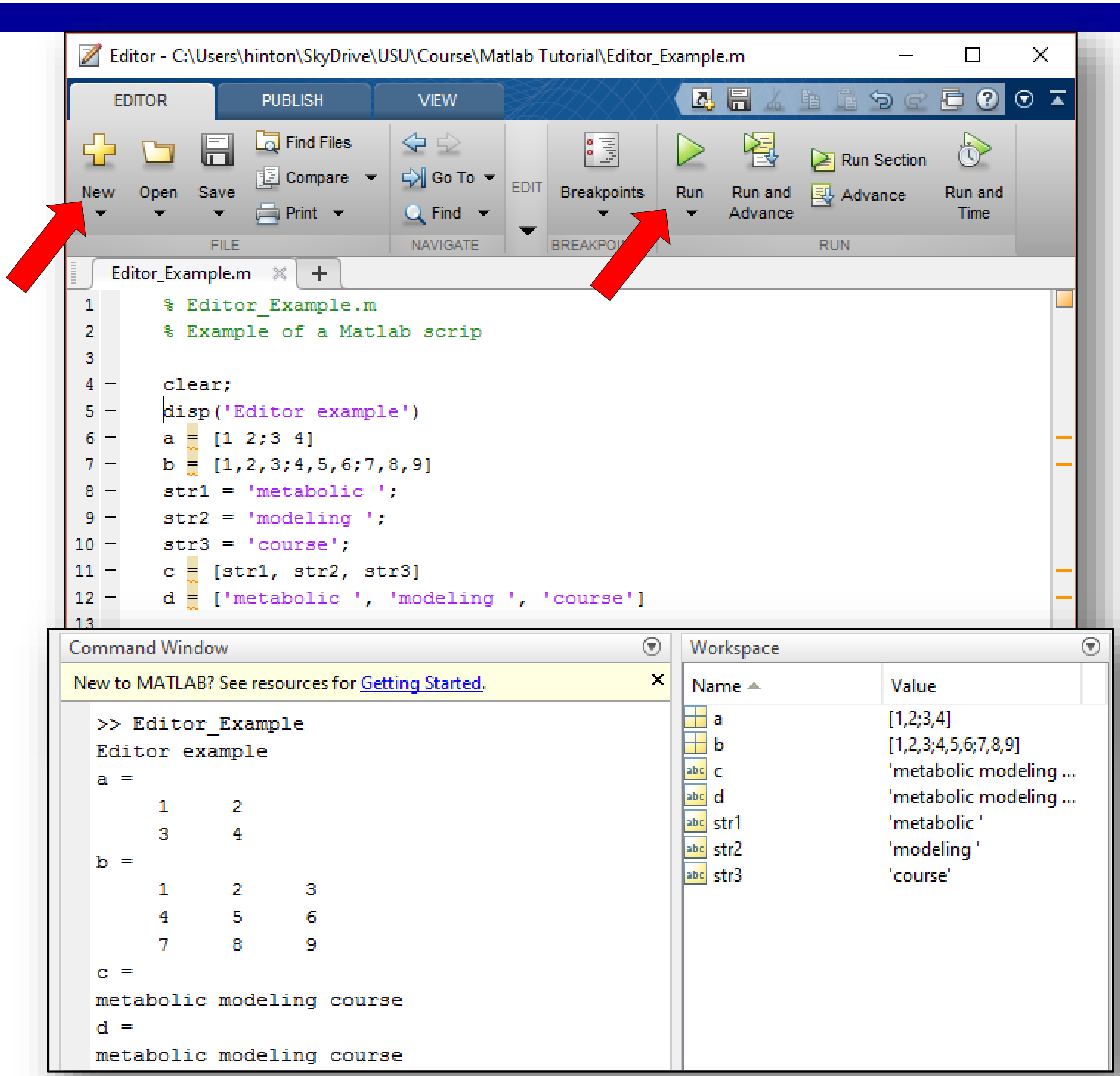
Name	Value
a	[1,2;3,4]
b	[1,2,3;4,5,6;7,8,9]
c	'metabolic modeling ...
d	'metabolic modeling ...
str1	'metabolic '
str2	'modeling '
str3	'course'

Course Introduction

- Desktop
- Variables
- ➡ Scripts
- Operations
- Visualization
- Programming
- Data Structures

Editor & Scripts

- The Matlab desktop includes an editor that can be used to create scripts which are composed of Matlab commands stored in a Matlab ".m" file.
- "%" assigns whatever text that follows on that line as a comment
- "clear" clear's the workspace
- The "disp()" command can be used to display strings



```
% Editor_Example.m
% Example of a Matlab scrip

clear;
disp('Editor example')
a = [1 2;3 4]
b = [1,2,3;4,5,6;7,8,9]
str1 = 'metabolic ';
str2 = 'modeling ';
str3 = 'course';
c = [str1, str2, str3]
d = ['metabolic ', 'modeling ', 'course']
```

New to MATLAB? See resources for Getting Started.

```
>> Editor_Example
Editor example
a =
     1     2
     3     4
b =
     1     2     3
     4     5     6
     7     8     9
c =
metabolic modeling course
d =
metabolic modeling course
```

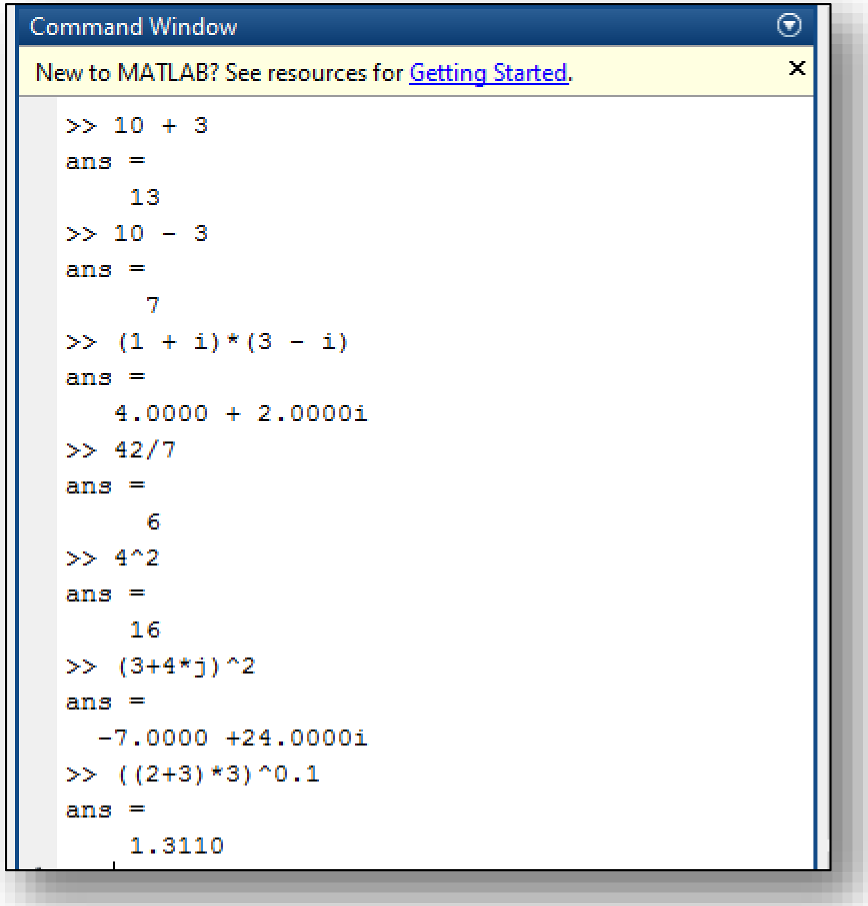
Name	Value
a	[1,2;3,4]
b	[1,2,3;4,5,6;7,8,9]
c	'metabolic modeling ...
d	'metabolic modeling ...
str1	'metabolic '
str2	'modeling '
str3	'course'

Course Introduction

- Desktop
- Variables
- Scripts
- ➡ Operations
- Visualization
- Programming
- Data Structures

Scalar Operations

- Arithmetic operations (+,-,*,/)
  - ✓» 10 + 3
  - ✓» 10 - 3
  - ✓» (1+i)*(3-i)
  - ✓» 42/7
- Exponentiation (^)
  - ✓» 4^2
  - ✓» (3+4*j)^2
- Complicated expressions, use parentheses
  - ✓» ((2+3)*3)^0.1

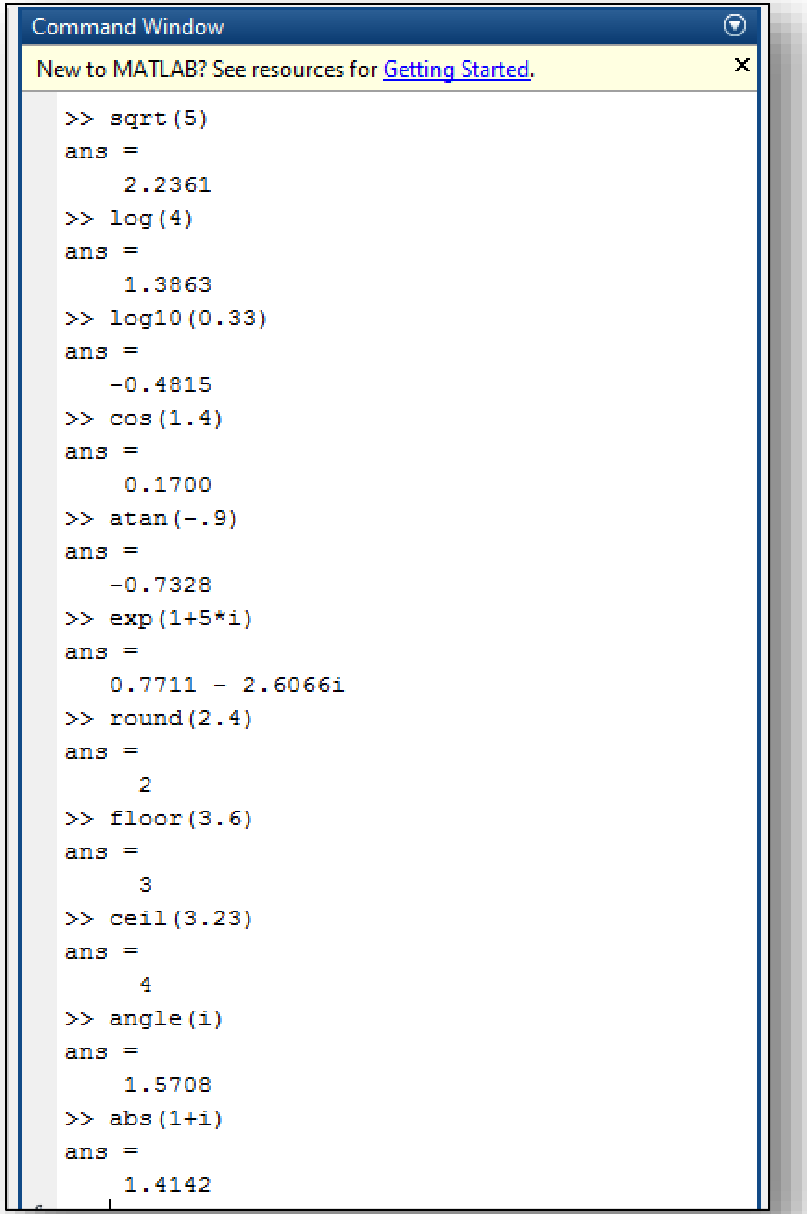


New to MATLAB? See resources for Getting Started.

```
>> 10 + 3
ans =
    13
>> 10 - 3
ans =
     7
>> (1 + i)*(3 - i)
ans =
   4.0000 + 2.0000i
>> 42/7
ans =
     6
>> 4^2
ans =
    16
>> (3+4*j)^2
ans =
  -7.0000 +24.0000i
>> ((2+3)*3)^0.1
ans =
    1.3110
```

Built-in Matlab Functions

- MATLAB has a large library of built-in functions
  - ✓ https://www.mathworks.com/help/matlab/functionlist.html?s_cid=doc_ftr
- To use the functions, call using parentheses which passes the parameters to function
  - ✓ » sqrt(5)
  - ✓ » log(4), log10(0.33)
  - ✓ » cos(1.4), atan(-.9)
  - ✓ » exp(1+5*i)
  - ✓ » round(2.4), floor(3.6), ceil(3.23)
  - ✓ » angle(i); abs(1+i);

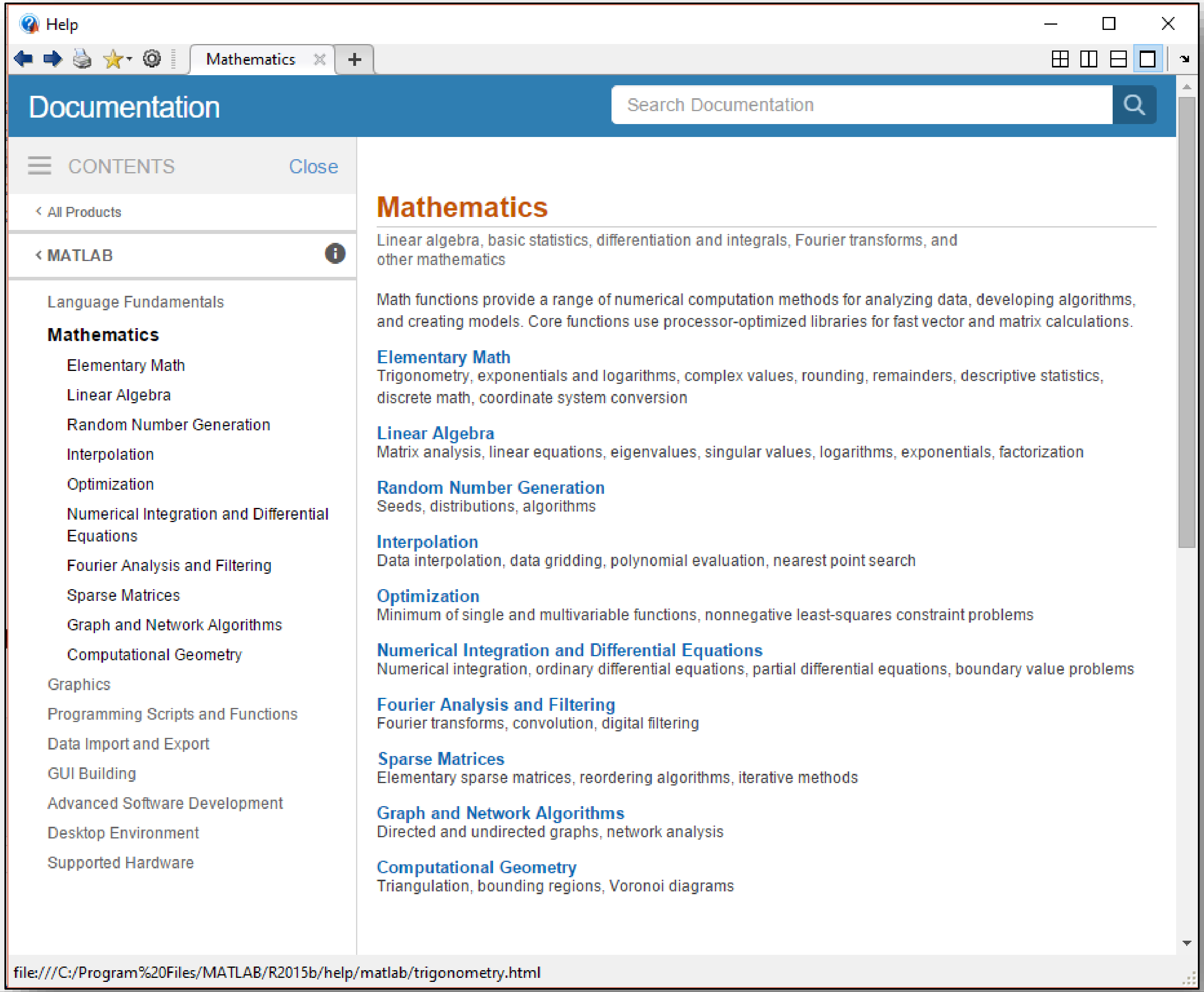


New to MATLAB? See resources for Getting Started.

```
>> sqrt(5)
ans =
    2.2361
>> log(4)
ans =
    1.3863
>> log10(0.33)
ans =
   -0.4815
>> cos(1.4)
ans =
    0.1700
>> atan(-.9)
ans =
   -0.7328
>> exp(1+5*i)
ans =
   0.7711 - 2.6066i
>> round(2.4)
ans =
     2
>> floor(3.6)
ans =
     3
>> ceil(3.23)
ans =
     4
>> angle(i)
ans =
    1.5708
>> abs(1+i)
ans =
    1.4142
```

Matlab Functions

- See Matlab documentation



CONTENTS Close

‹ All Products

‹ MATLAB

Language Fundamentals

Mathematics

Elementary Math

Linear Algebra

Random Number Generation

Interpolation

Optimization

Numerical Integration and Differential Equations

Fourier Analysis and Filtering

Sparse Matrices

Graph and Network Algorithms

Computational Geometry

Graphics

Programming Scripts and Functions

Data Import and Export

GUI Building

Advanced Software Development

Desktop Environment

Supported Hardware

Mathematics

Linear algebra, basic statistics, differentiation and integrals, Fourier transforms, and other mathematics

Math functions provide a range of numerical computation methods for analyzing data, developing algorithms, and creating models. Core functions use processor-optimized libraries for fast vector and matrix calculations.

Elementary Math
Trigonometry, exponentials and logarithms, complex values, rounding, remainders, descriptive statistics, discrete math, coordinate system conversion

Linear Algebra
Matrix analysis, linear equations, eigenvalues, singular values, logarithms, exponentials, factorization

Random Number Generation
Seeds, distributions, algorithms

Interpolation
Data interpolation, data gridding, polynomial evaluation, nearest point search

Optimization
Minimum of single and multivariable functions, nonnegative least-squares constraint problems

Numerical Integration and Differential Equations
Numerical integration, ordinary differential equations, partial differential equations, boundary value problems

Fourier Analysis and Filtering
Fourier transforms, convolution, digital filtering

Sparse Matrices
Elementary sparse matrices, reordering algorithms, iterative methods

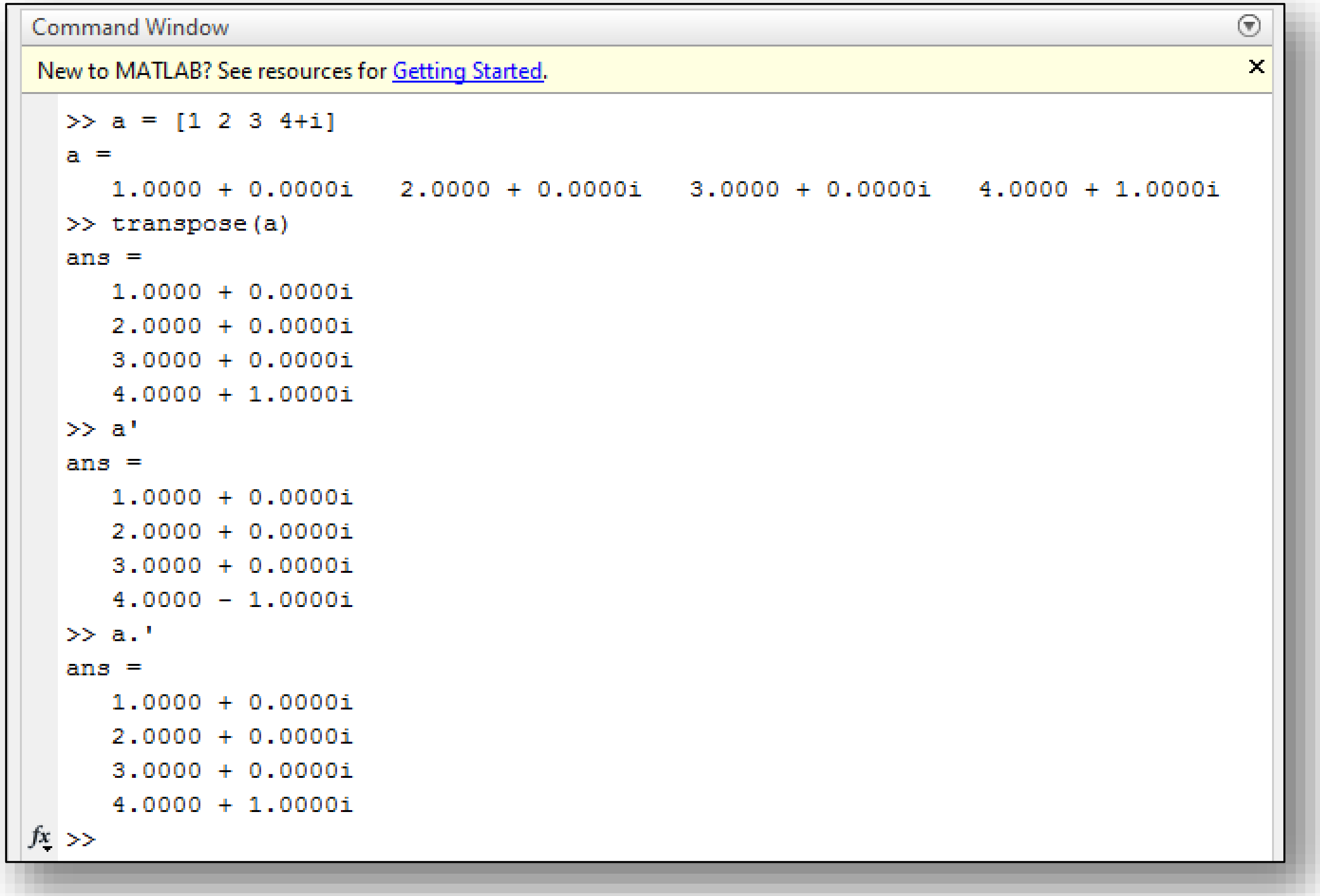
Graph and Network Algorithms
Directed and undirected graphs, network analysis

Computational Geometry
Triangulation, bounding regions, Voronoi diagrams

file:///C:/Program%20Files/MATLAB/R2015b/help/matlab/trigonometry.html

Linear Algebra

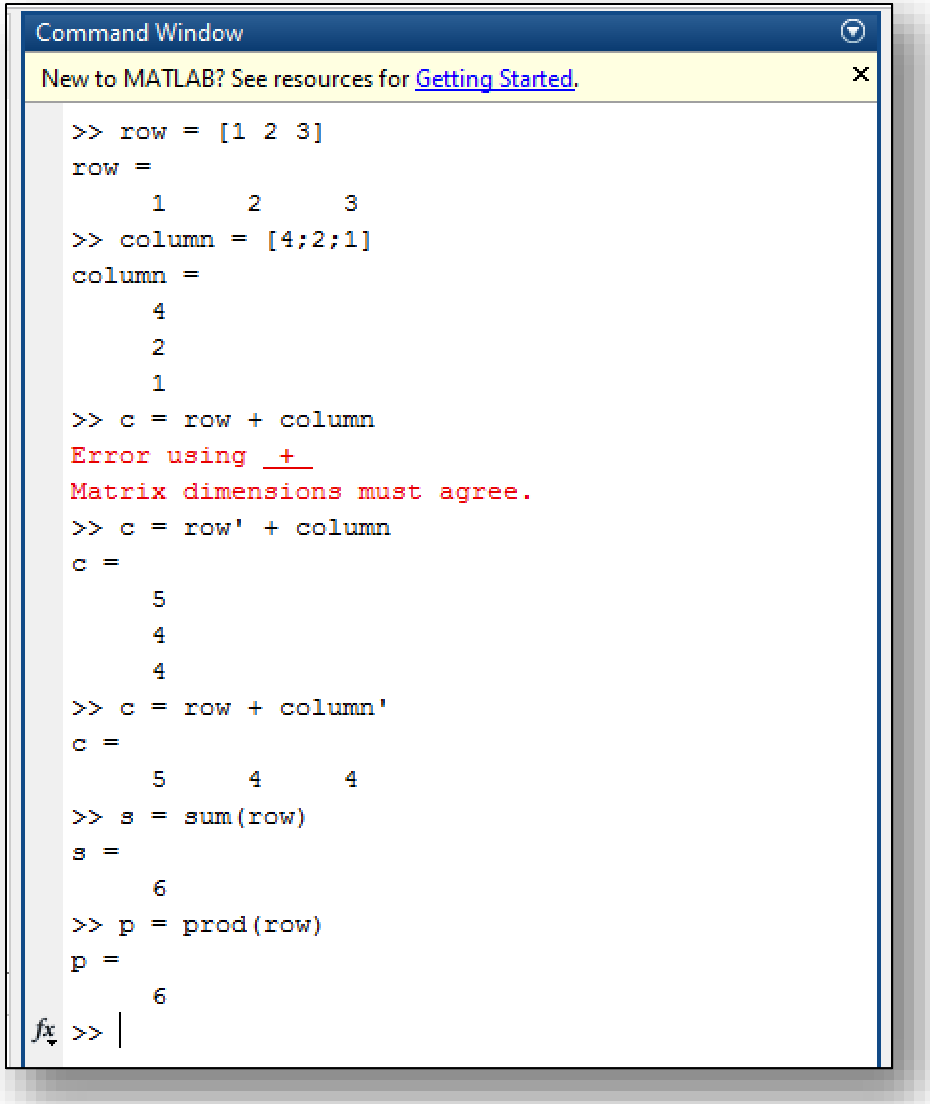
- Transpose
  - ✓ The transpose operators turns a column vector into a row vector and vice versa
    - `» a = [1 2 3 4+i]`
    - `» transpose(a)`
    - `» a'`
    - `» a.'`
  - ✓ The ' gives the Hermitian-transpose, i.e. transposes and conjugates all complex numbers
  - ✓ For vectors of real numbers .' and ' give same result



```
>> a = [1 2 3 4+i]
a =
   1.0000 + 0.0000i   2.0000 + 0.0000i   3.0000 + 0.0000i   4.0000 + 1.0000i
>> transpose(a)
ans =
   1.0000 + 0.0000i
   2.0000 + 0.0000i
   3.0000 + 0.0000i
   4.0000 + 1.0000i
>> a'
ans =
   1.0000 + 0.0000i
   2.0000 + 0.0000i
   3.0000 + 0.0000i
   4.0000 - 1.0000i
>> a.'
ans =
   1.0000 + 0.0000i
   2.0000 + 0.0000i
   3.0000 + 0.0000i
   4.0000 + 1.0000i
>>
```

Adding and Subtracting Arrays

- Addition and subtraction are element-wise; sizes must match (unless one is a scalar):
  - ✓ » `row = [1 2 3]`
  - ✓ » `column = [4;2;1]`
  - ✓ » `c = row + column % Error`
- Use the transpose to make sizes compatible
  - ✓ » `c = row'+ column`
  - ✓ » `c = row + column'`
- Can sum up or multiply elements of vector
  - ✓ » `s = sum(row);`
  - ✓ » `p = prod(row);`

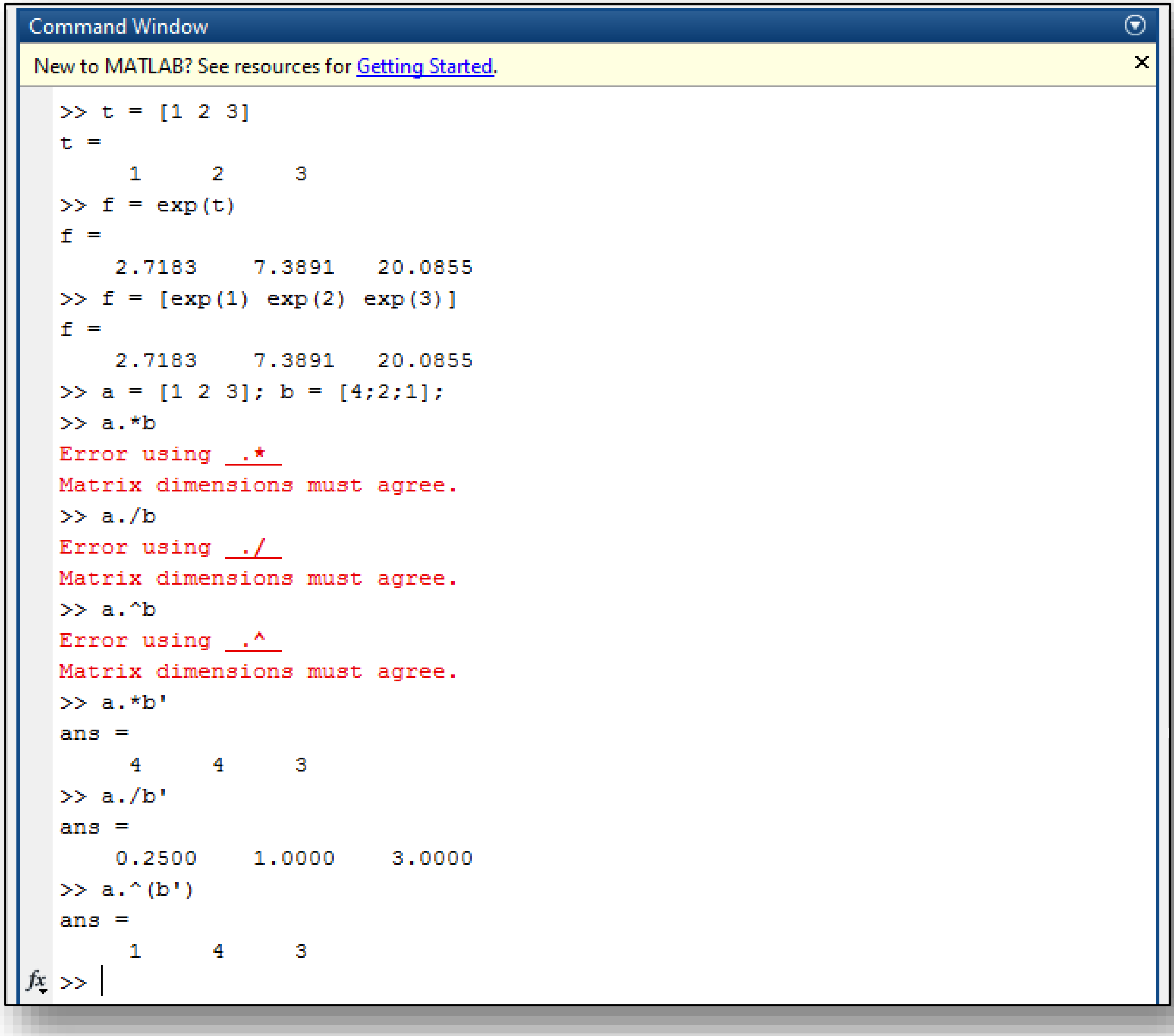


New to MATLAB? See resources for Getting Started.

```
>> row = [1 2 3]
row =
          1      2      3
>> column = [4;2;1]
column =
          4
          2
          1
>> c = row + column
Error using  + 
Matrix dimensions must agree.
>> c = row' + column
c =
          5
          4
          4
>> c = row + column'
c =
          5      4      4
>> s = sum(row)
s =
          6
>> p = prod(row)
p =
          6
>> 
```

Standard and Element-Wise Operators

- Operators (* / ^) have two modes of operation
  - ✓ Standard (* / ^)
  - ✓ Element-wise (.* ./ .^)
- All the functions that work on scalars also work on vectors
  - ✓ » `t = [1 2 3];`
  - ✓ » `f = exp(t)` is the same as
  - ✓ » `f = [exp(1) exp(2) exp(3)];`
- For element-wise operations, use the dot: (.*, ./, .^). Both dimensions must match (unless one is scalar)!
  - ✓ » `a =[1 2 3]; b=[4;2;1];`
  - ✓ » `a.*b, a./b, a.^b` are all errors
  - ✓ » `a.*b', a./b', a.^(b')` are all valid



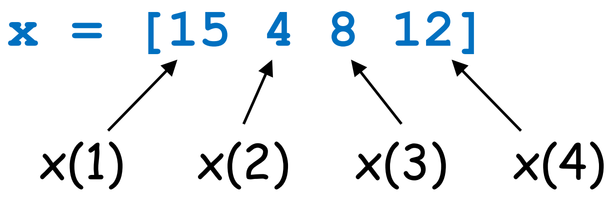
```
>> t = [1 2 3]
t =
     1     2     3
>> f = exp(t)
f =
    2.7183    7.3891   20.0855
>> f = [exp(1) exp(2) exp(3)]
f =
    2.7183    7.3891   20.0855
>> a = [1 2 3]; b = [4;2;1];
>> a.*b
Error using  .*
Matrix dimensions must agree.
>> a./b
Error using  ./
Matrix dimensions must agree.
>> a.^b
Error using  .^
Matrix dimensions must agree.
>> a.*b'
ans =
     4     4     3
>> a./b'
ans =
    0.2500    1.0000    3.0000
>> a.^(b')
ans =
     1     4     3
fx >> 
```

Operator Guidelines

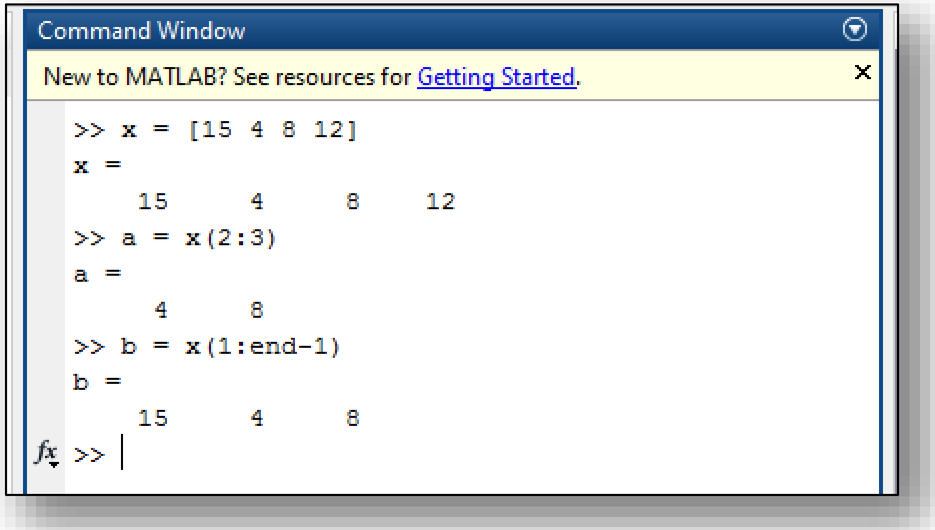
- Multiplication can be done in a standard way or element-wise
- Standard multiplication (*) is either a dot-product or an outer-product
  - ✓Remember from linear algebra: inner dimensions must MATCH!!
- Standard exponentiation (^) can only be done on square matrices or scalars
- Left and right division (/ \) is same as multiplying by inverse
  - ✓Recommendation: just multiply by inverse

Indexing Vectors

- MATLAB indexing of arrays starts with 1, not 0
- x(n) returns the nth element of the array



- The index argument can be a vector. In this case, each element is looked up individually, and returned as a vector of the same size as the index vector.
  - ✓ » `x = [15 4 8 12]`
  - ✓ » `a = x(2:3)`
  - ✓ » `b = x(1:end-1)`

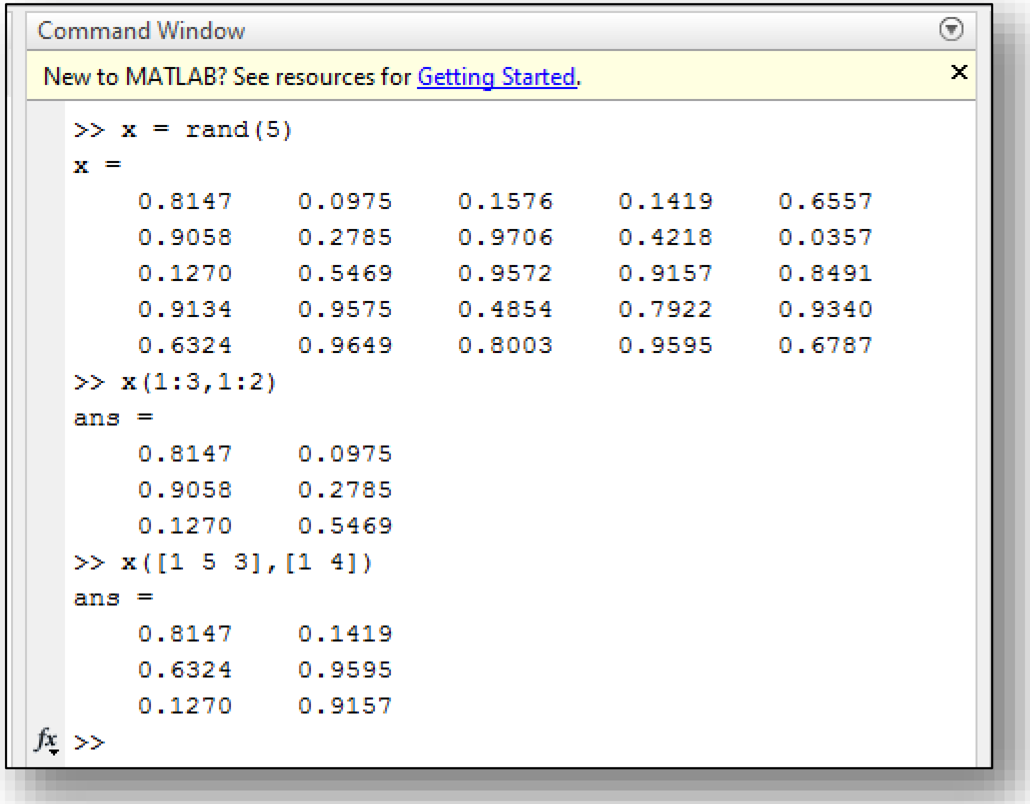


```
>> x = [15 4 8 12]
x =
    15     4     8    12
>> a = x(2:3)
a =
     4     8
>> b = x(1:end-1)
b =
    15     4     8
>> |
```

Indexing Matrices

- Matrices can be indexed in two ways
  - ✓ Using subscripts (row and column)
  - ✓ Using linear indices (as if matrix is a vector)
- Matrix indexing: subscripts or linear indices

$$\begin{matrix} x(1,1) \rightarrow \\ x(2,1) \rightarrow \end{matrix} \begin{bmatrix} 22 & 34 \\ 4 & 5 \end{bmatrix} \begin{matrix} \leftarrow x(1,2) \\ \leftarrow x(2,2) \end{matrix} \qquad \begin{matrix} x(1) \rightarrow \\ x(2) \rightarrow \end{matrix} \begin{bmatrix} 22 & 34 \\ 4 & 5 \end{bmatrix} \begin{matrix} \leftarrow x(3) \\ \leftarrow x(4) \end{matrix}$$



```
>> x = rand(5)
x =
    0.8147    0.0975    0.1576    0.1419    0.6557
    0.9058    0.2785    0.9706    0.4218    0.0357
    0.1270    0.5469    0.9572    0.9157    0.8491
    0.9134    0.9575    0.4854    0.7922    0.9340
    0.6324    0.9649    0.8003    0.9595    0.6787
>> x(1:3,1:2)
ans =
    0.8147    0.0975
    0.9058    0.2785
    0.1270    0.5469
>> x([1 5 3],[1 4])
ans =
    0.8147    0.1419
    0.6324    0.9595
    0.1270    0.9157
>>
```

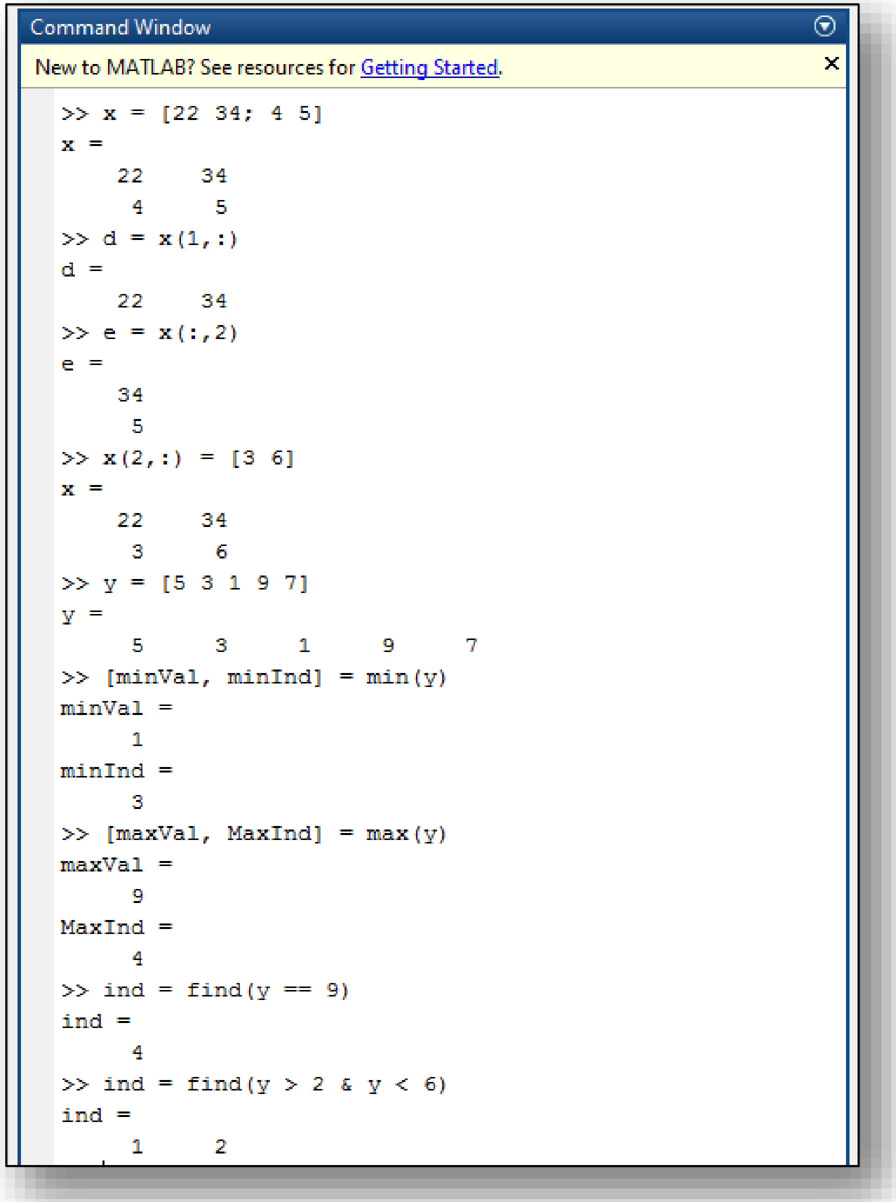
- Picking submatrices
  - ✓ » `x = rand(5) % Uniformly distributed random numbers`
  - ✓ » `x(1:3,1:2) % Specify contiguous submatrix`
  - ✓ » `x([1 5 3], [1 4]) % Specify rows and columns`

Advanced Indexing

- To select rows or columns of a matrix, use ":"

$$x = \begin{bmatrix} 22 & 34 \\ 4 & 5 \end{bmatrix}$$

  - ✓ » d = x(1,:) % list elements of row 1
  - ✓ » e = x(:,2) % list elements of column 2
  - ✓ » x(2,:) = [3 6]; % replaces second row of x
- Functions that can help you find desired values within a vector or matrix
  - ✓ » y = [5 3 1 9 7]
- To get the minimum value and its index:
  - ✓ »[minVal,minInd] = min(y);
  - ✓ »[maxVal,maxInd] = max(y);
- To find any the indices of specific values or ranges
  - ✓ » ind = find(y == 9);
  - ✓ » ind = find(y > 2 & y < 6);



New to MATLAB? See resources for Getting Started.

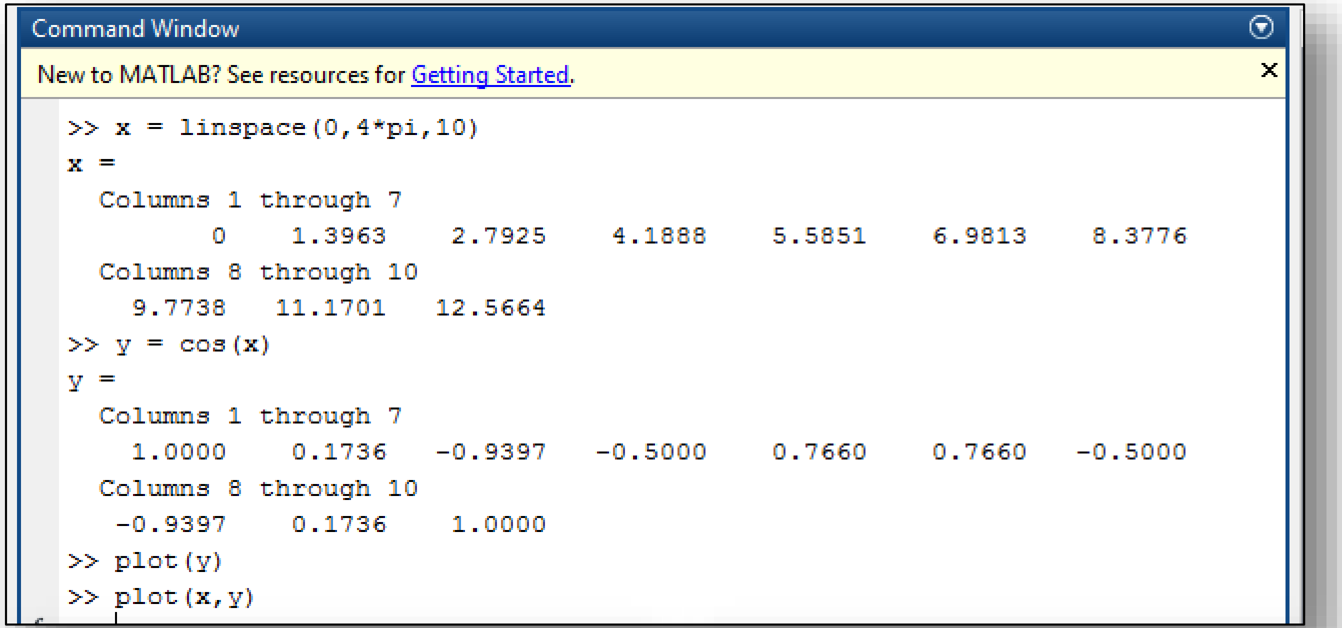
```
>> x = [22 34; 4 5]
x =
    22    34
     4     5
>> d = x(1,:)
d =
    22    34
>> e = x(:,2)
e =
    34
     5
>> x(2,:) = [3 6]
x =
    22    34
     3     6
>> y = [5 3 1 9 7]
y =
     5     3     1     9     7
>> [minVal, minInd] = min(y)
minVal =
     1
minInd =
     3
>> [maxVal, MaxInd] = max(y)
maxVal =
     9
MaxInd =
     4
>> ind = find(y == 9)
ind =
     4
>> ind = find(y > 2 & y < 6)
ind =
     1     2
```

Course Introduction

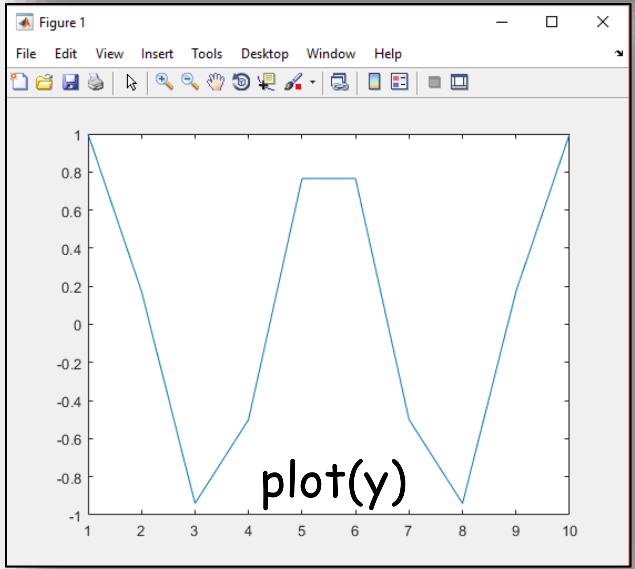
- Desktop
- Variables
- Scripts
- Operations
- ➡ Visualization
- Programming
- Data Structures

Simple Plotting

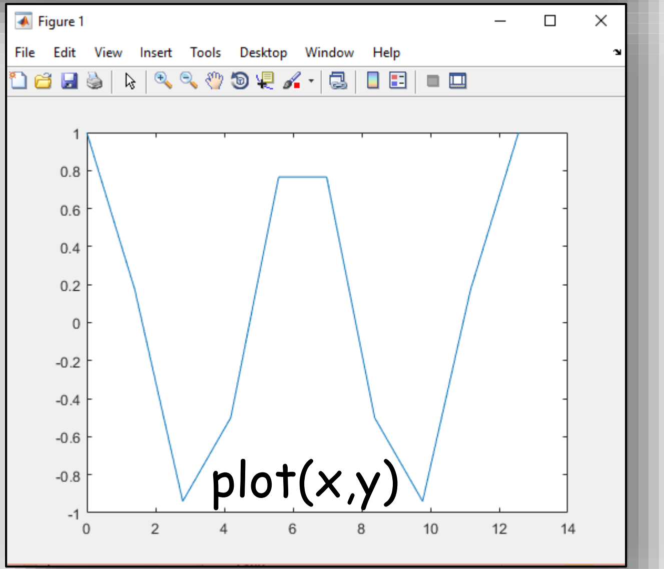
- Simple example
  - ✓ » `x = linspace(0,4*pi,10)`
  - ✓ » `y = cos(x)`
- Plot values against their index
  - ✓ » `plot(y);`
- Plotting y versus x
  - ✓ » `plot(x,y);`
- "plot" generates dots at each (x,y) pair and then connects the dots with a line
- Default is plotting 10 points



```
>> x = linspace(0,4*pi,10)
x =
  Columns 1 through 7
         0    1.3963    2.7925    4.1888    5.5851    6.9813    8.3776
  Columns 8 through 10
    9.7738   11.1701   12.5664
>> y = cos(x)
y =
  Columns 1 through 7
    1.0000    0.1736   -0.9397   -0.5000    0.7660    0.7660   -0.5000
  Columns 8 through 10
   -0.9397    0.1736    1.0000
>> plot(y)
>> plot(x,y)
```



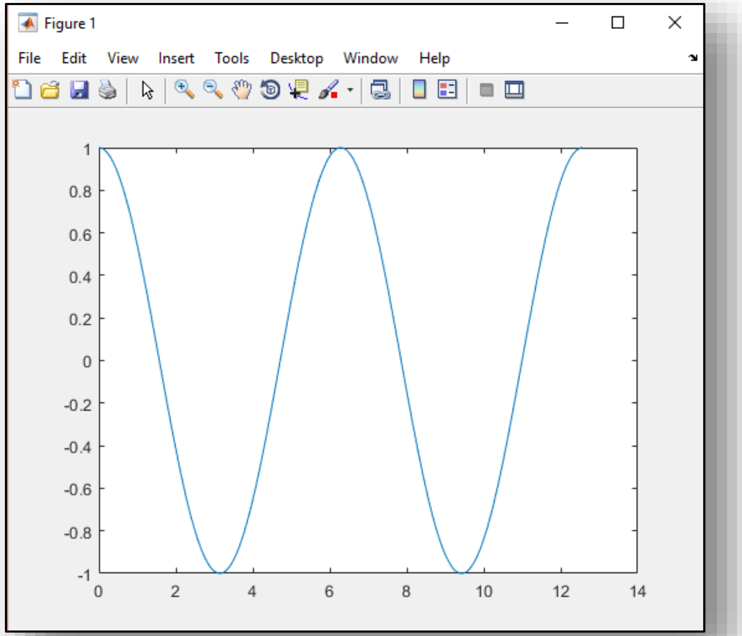
plot(y)

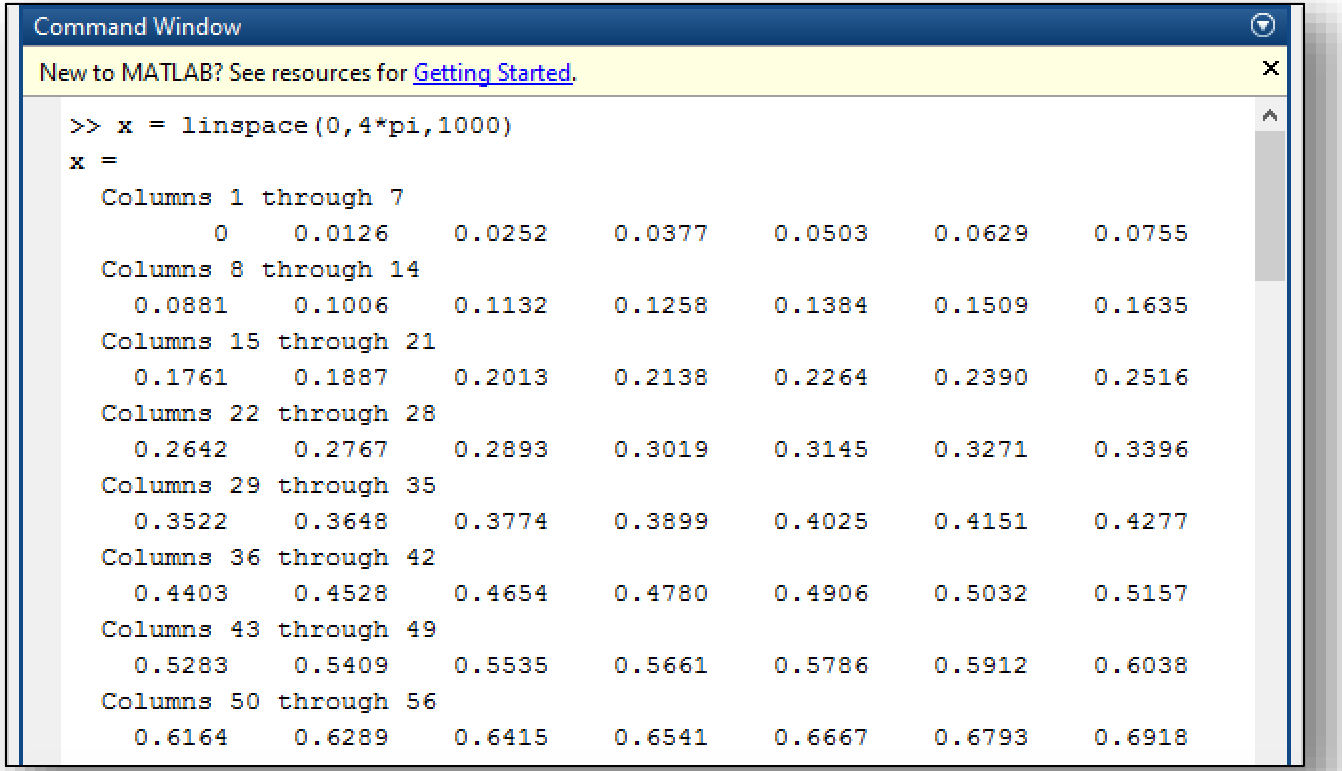


plot(x,y)

Simple Plotting: Increasing Resolution

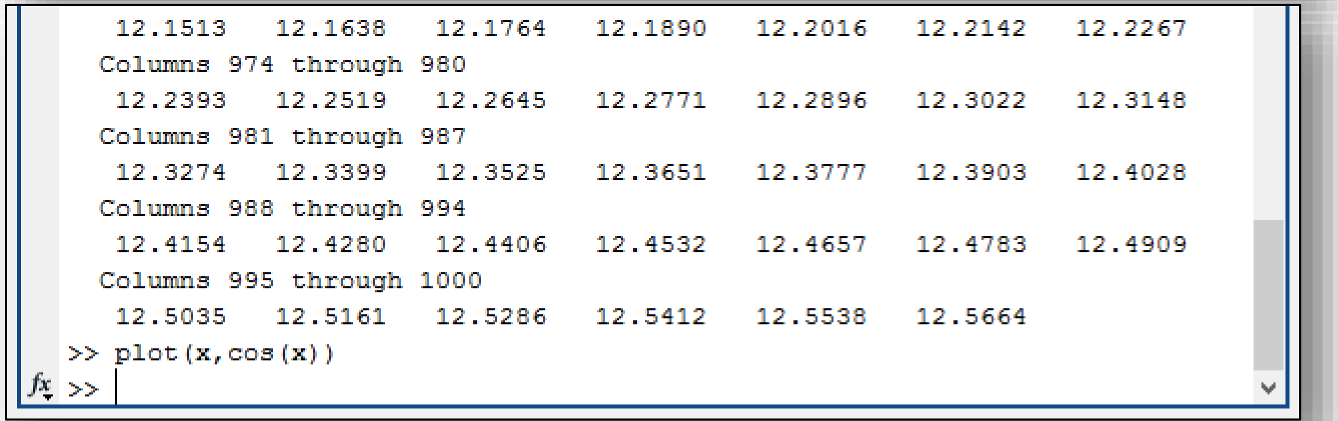
- Example
  - ✓ » `x = linspace(0,4*pi,1000)`
  - ✓ » `y = cos(x)`
- Plot values against their index
  - ✓ » `plot(x, cos(x))`





New to MATLAB? See resources for Getting Started.

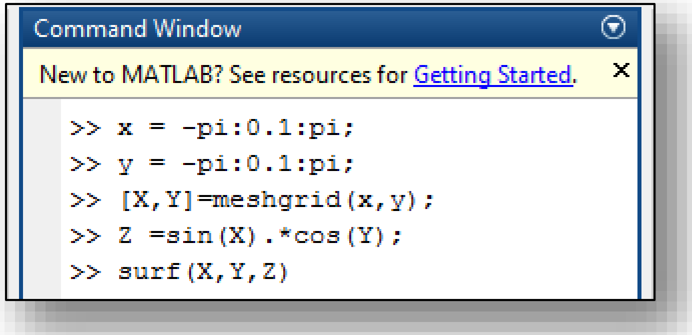
```
>> x = linspace(0,4*pi,1000)
x =
  Columns 1 through 7
          0    0.0126    0.0252    0.0377    0.0503    0.0629    0.0755
  Columns 8 through 14
    0.0881    0.1006    0.1132    0.1258    0.1384    0.1509    0.1635
  Columns 15 through 21
    0.1761    0.1887    0.2013    0.2138    0.2264    0.2390    0.2516
  Columns 22 through 28
    0.2642    0.2767    0.2893    0.3019    0.3145    0.3271    0.3396
  Columns 29 through 35
    0.3522    0.3648    0.3774    0.3899    0.4025    0.4151    0.4277
  Columns 36 through 42
    0.4403    0.4528    0.4654    0.4780    0.4906    0.5032    0.5157
  Columns 43 through 49
    0.5283    0.5409    0.5535    0.5661    0.5786    0.5912    0.6038
  Columns 50 through 56
    0.6164    0.6289    0.6415    0.6541    0.6667    0.6793    0.6918
```



```
   12.1513   12.1638   12.1764   12.1890   12.2016   12.2142   12.2267
  Columns 974 through 980
   12.2393   12.2519   12.2645   12.2771   12.2896   12.3022   12.3148
  Columns 981 through 987
   12.3274   12.3399   12.3525   12.3651   12.3777   12.3903   12.4028
  Columns 988 through 994
   12.4154   12.4280   12.4406   12.4532   12.4657   12.4783   12.4909
  Columns 995 through 1000
   12.5035   12.5161   12.5286   12.5412   12.5538   12.5664
>> plot(x,cos(x))
>> 
```

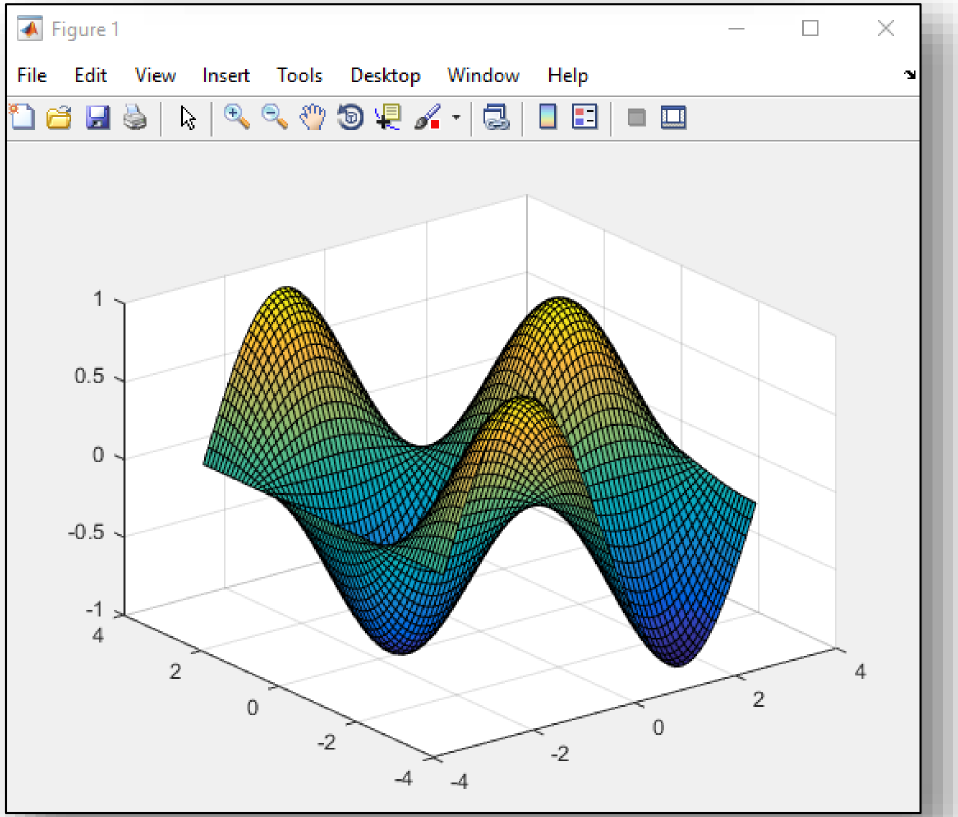
Surface Plots

- “surf” puts vertices at specified points in space x,y,z, and connects all the vertices to make a surface
- Eample: make the x and y vectors
  - ✓ » `x = -pi:0.1:pi;`
  - ✓ » `y = -pi:0.1:pi;`
- Use meshgrid to make matrices (this is the same as loop)
  - ✓ » `[X,Y] = meshgrid(x,y);`
- To get function values, evaluate the matrices
  - ✓ » `Z =sin(X).*cos(Y);`
- Plot the surface
  - ✓ » `surf(X,Y,Z) or surf(x,y,Z);`



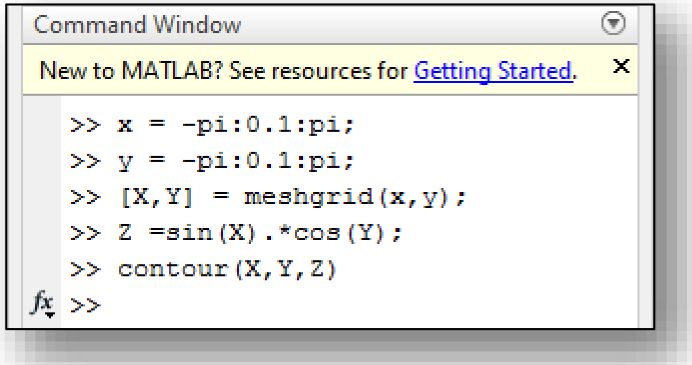
New to MATLAB? See resources for Getting Started.

```
>> x = -pi:0.1:pi;
>> y = -pi:0.1:pi;
>> [X,Y]=meshgrid(x,y);
>> Z =sin(X).*cos(Y);
>> surf(X,Y,Z)
```



Contour Plots

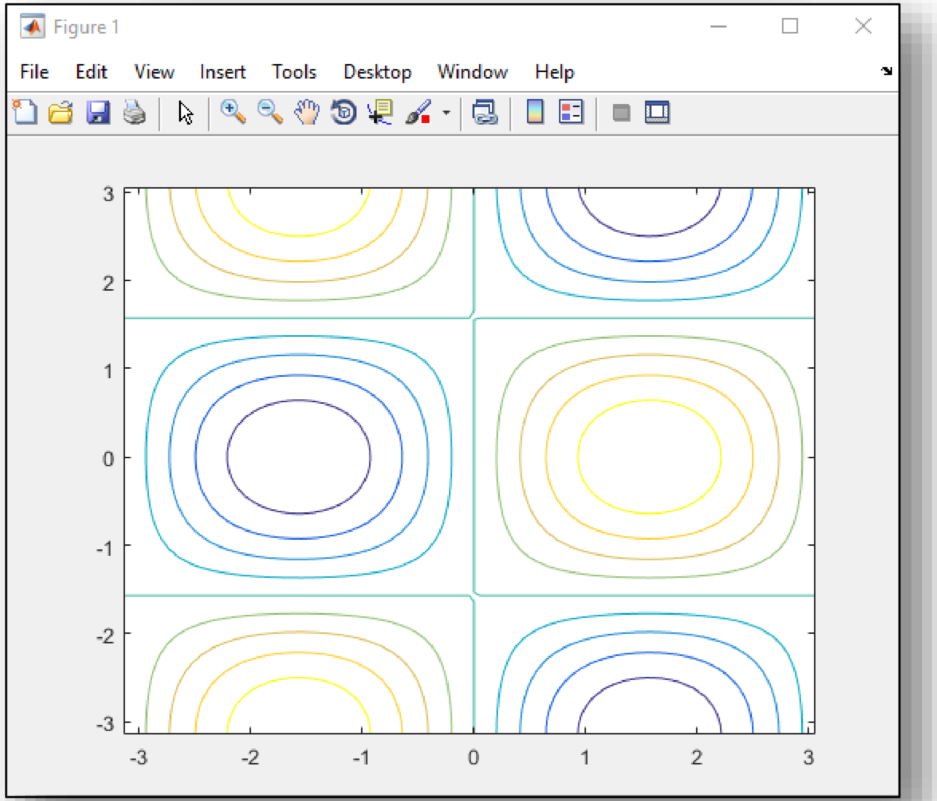
- Contour plots make surfaces two-dimensional
  - ✓ » `contour(X,Y,Z)`
  - ✓ Same arguments as surf
  - ✓ Color indicates height
- Example: make the x and y vectors
  - ✓ » `x = -pi:0.1:pi;`
  - ✓ » `y = -pi:0.1:pi;`
- Use meshgrid to make matrices (this is the same as loop)
  - ✓ » `[X,Y] = meshgrid(x,y);`
- To get function values, evaluate the matrices
  - ✓ » `Z =sin(X).*cos(Y);`
- Plot the surface
  - ✓ » `contour(X,Y,Z)`



Command Window

New to MATLAB? See resources for Getting Started.

```
>> x = -pi:0.1:pi;
>> y = -pi:0.1:pi;
>> [X,Y] = meshgrid(x,y);
>> Z =sin(X).*cos(Y);
>> contour(X,Y,Z)
>>
```

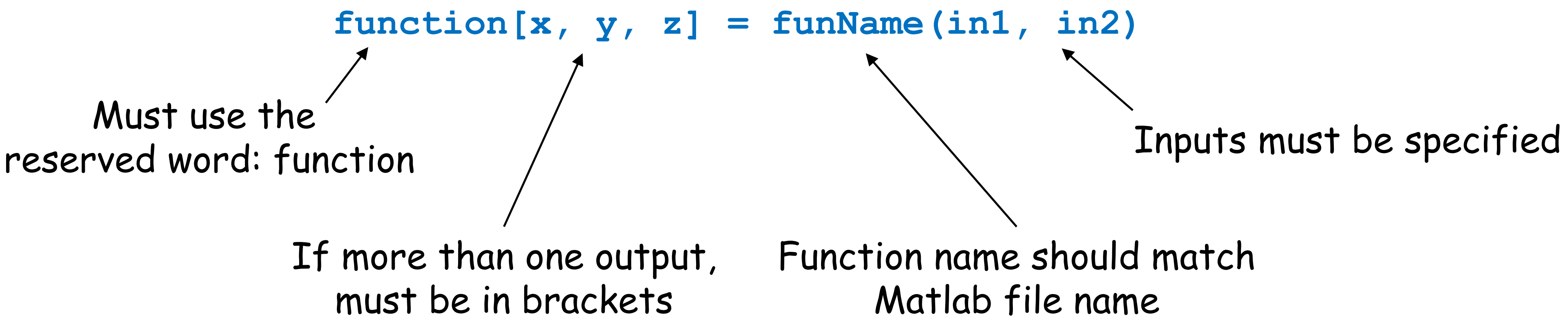


Course Introduction

- Desktop
- Variables
- Scripts
- Operations
- Visualization
- ➡ Programming
- Data Structures

User-defined Functions

- The function declaration is given by



```
function[x, y, z] = funName(in1, in2)
```

Must use the reserved word: function

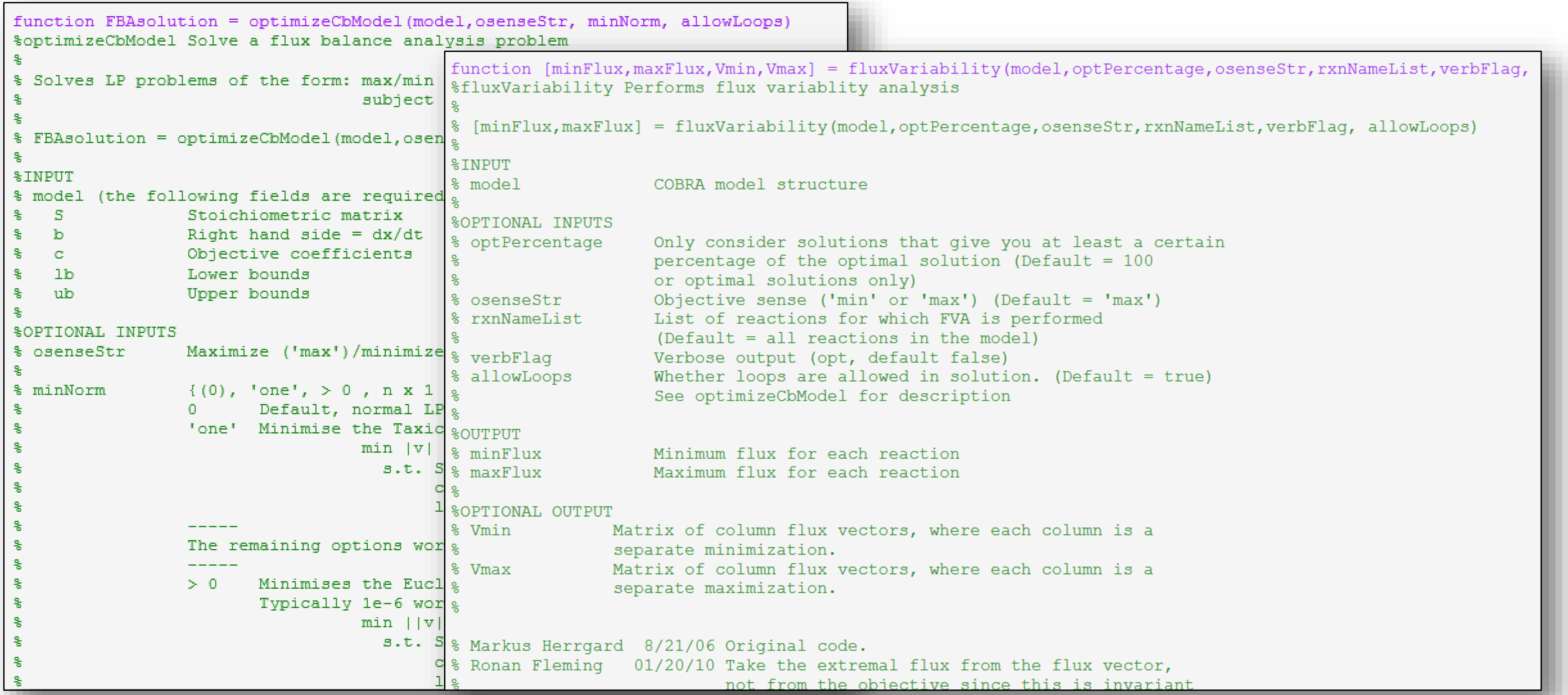
If more than one output, must be in brackets

Function name should match Matlab file name

Inputs must be specified

- MATLAB 'returns' the variables whose names match those in the function declaration
- Any variables created within the function but not returned disappear after the function stops running

User-defined Function Example



```
function FBAsolution = optimizeCbModel(model,osenseStr, minNorm, allowLoops)
%optimizeCbModel Solve a flux balance analysis problem
%
% Solves LP problems of the form: max/min
%                                 subject
%
% FBAsolution = optimizeCbModel(model,osen
%
%INPUT
% model (the following fields are required
%   S            Stoichiometric matrix
%   b            Right hand side = dx/dt
%   c            Objective coefficients
%   lb           Lower bounds
%   ub           Upper bounds
%
%OPTIONAL INPUTS
% osenseStr      Maximize ('max')/minimize
%
% minNorm        {(0), 'one', > 0 , n x 1
%                0      Default, normal LP
%                'one'  Minimise the Taxic
%                                 min |v|
%                                   s.t. S
%                                        c
%                                        l
%                -----
%                The remaining options wor
%                -----
%                > 0    Minimises the Eucl
%                       Typically 1e-6 wor
%                                 min ||v|
%                                   s.t. S
%                                        c
%                                        l
```

```
function [minFlux,maxFlux,Vmin,Vmax] = fluxVariability(model,optPercentage,osenseStr,rxnNameList,verbFlag,
%fluxVariability Performs flux variablity analysis
%
% [minFlux,maxFlux] = fluxVariability(model,optPercentage,osenseStr,rxnNameList,verbFlag, allowLoops)
%
%INPUT
% model              COBRA model structure
%
%OPTIONAL INPUTS
% optPercentage      Only consider solutions that give you at least a certain
%                    percentage of the optimal solution (Default = 100
%                    or optimal solutions only)
% osenseStr          Objective sense ('min' or 'max') (Default = 'max')
% rxnNameList        List of reactions for which FVA is performed
%                    (Default = all reactions in the model)
% verbFlag           Verbose output (opt, default false)
% allowLoops         Whether loops are allowed in solution. (Default = true)
%                    See optimizeCbModel for description
%
%OUTPUT
% minFlux            Minimum flux for each reaction
% maxFlux            Maximum flux for each reaction
%
%OPTIONAL OUTPUT
% Vmin           Matrix of column flux vectors, where each column is a
%                separate minimization.
% Vmax           Matrix of column flux vectors, where each column is a
%                separate maximization.
%

% Markus Herrgard  8/21/06 Original code.
% Ronan Fleming   01/20/10 Take the extremal flux from the flux vector,
%                          not from the objective since this is invariant
```

Relational & Logical Operators

- MATLAB uses mostly standard relational operators

✓ equal	==
✓ not equal	~=
✓ greater than	>
✓ less than	<
✓ greater or equal	>=
✓ less or equal	<=

• Logical operators (scalars)	elementwise	short-circuit
✓ And	&	&&
✓ Or	|	||
✓ Not	~	
✓ Xor	xor	
✓ All true	all	
✓ Any true	any	

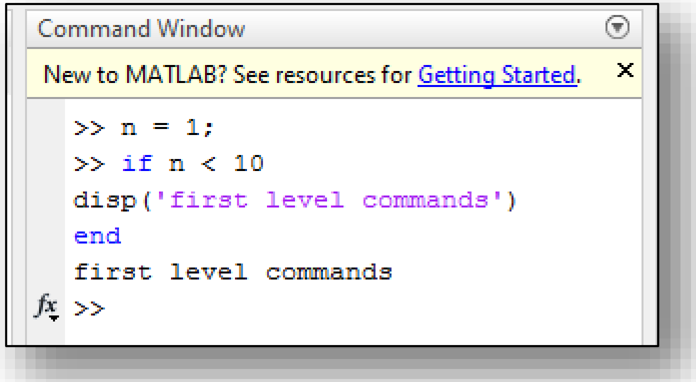
- Boolean values: zero is false, nonzero is true

if/else/elseif

- Basic flow-control, common to all languages
- MATLAB syntax is somewhat unique
- No need for parentheses: since command blocks are between reserved words

IF

```
if cond
   commands
end
```

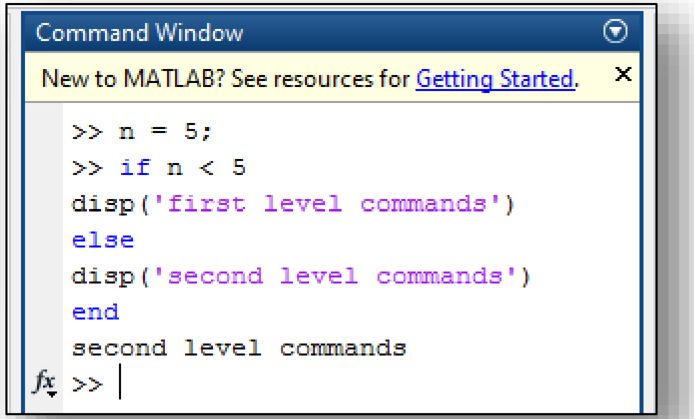


New to MATLAB? See resources for Getting Started.

```
>> n = 1;
>> if n < 10
disp('first level commands')
end
first level commands
>>
```

ELSE

```
if cond
   commands1
else
   commands2
end
```

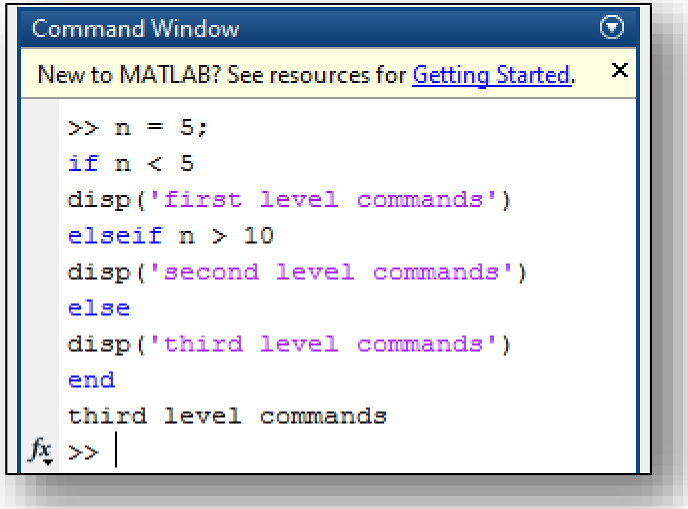


New to MATLAB? See resources for Getting Started.

```
>> n = 5;
>> if n < 5
disp('first level commands')
else
disp('second level commands')
end
second level commands
>>
```

ELSEIF

```
if cond1
   commands1
elseif cond2
   commands2
else
   commands3
end
```



New to MATLAB? See resources for Getting Started.

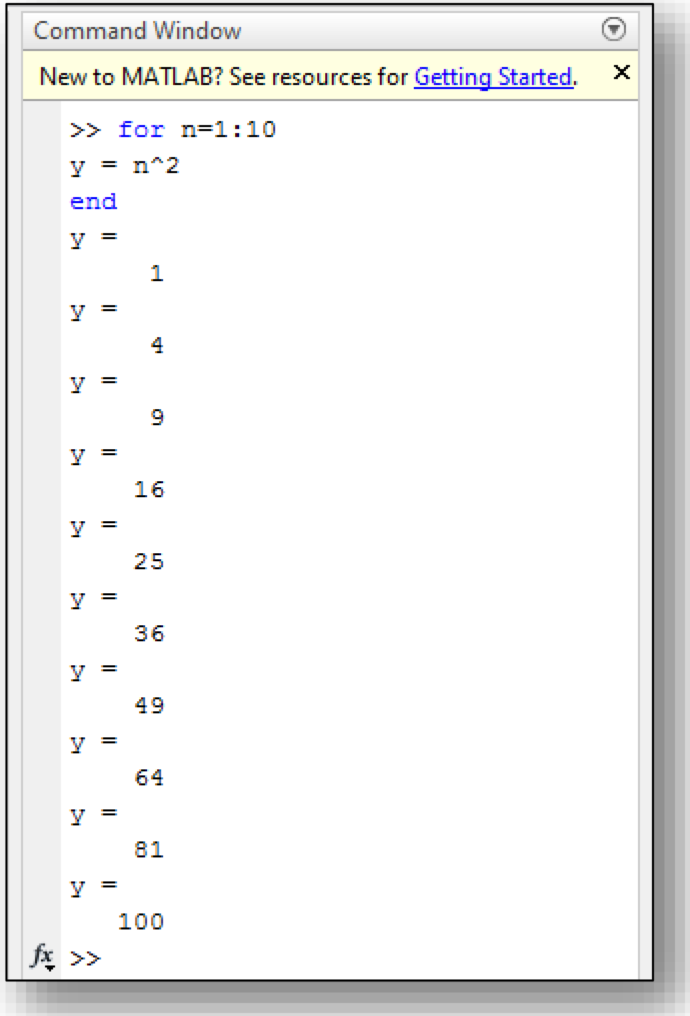
```
>> n = 5;
if n < 5
disp('first level commands')
elseif n > 10
disp('second level commands')
else
disp('third level commands')
end
third level commands
>>
```

For Loops

- for loops: use for a known number of iterations
- MATLAB syntax:

```
for n=1:100
      commands
end
```

- The loop variable (n)
  - ✓ Is defined as a vector
  - ✓ Is a scalar within the command block
- The command block
  - ✓ Anything between the for line and the end



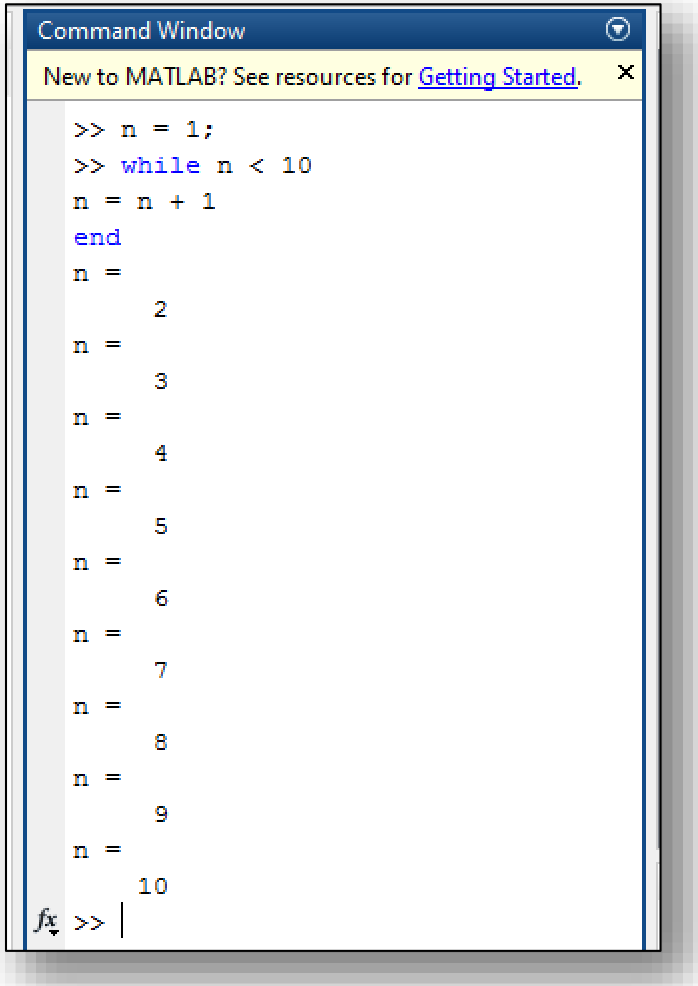
```
>> for n=1:10
y = n^2
end
y =
     1
y =
     4
y =
     9
y =
    16
y =
    25
y =
    36
y =
    49
y =
    64
y =
    81
y =
   100
>>
```

While Statement

- The **while** is like a more general for loop that doesn't require the need to know the number of iterations

```
while cond
        commands
end
```

- The command block will execute while the conditional expression is true
- Beware of infinite loops!



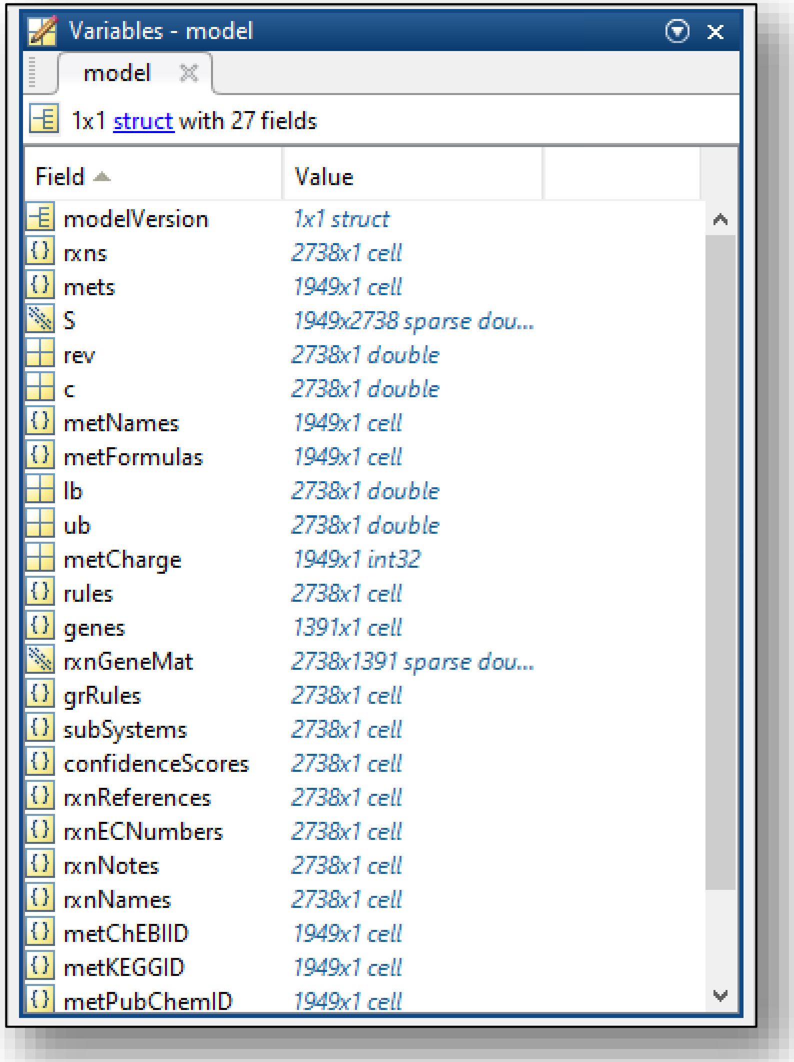
```
>> n = 1;
>> while n < 10
n = n + 1
end
n =
     2
n =
     3
n =
     4
n =
     5
n =
     6
n =
     7
n =
     8
n =
     9
n =
    10
>> 
```

Course Introduction

- Desktop
- Variables
- Scripts
- Operations
- Visualization
- Programming
- ➡ Data Structures

Data Structures

- Matrices
  - ✓Can create n-dimensional matrices
  - ✓All elements must be the same type (integers, double, character, …)
  - ✓Matrices are space-efficient and convenient for calculations
- More complex data structures are also possible in Matlab
  - ✓Cell arrays – Like an array but the elements don't have to have to be the same type
  - ✓Structs – Can be used to bundle variable names and values into one structure

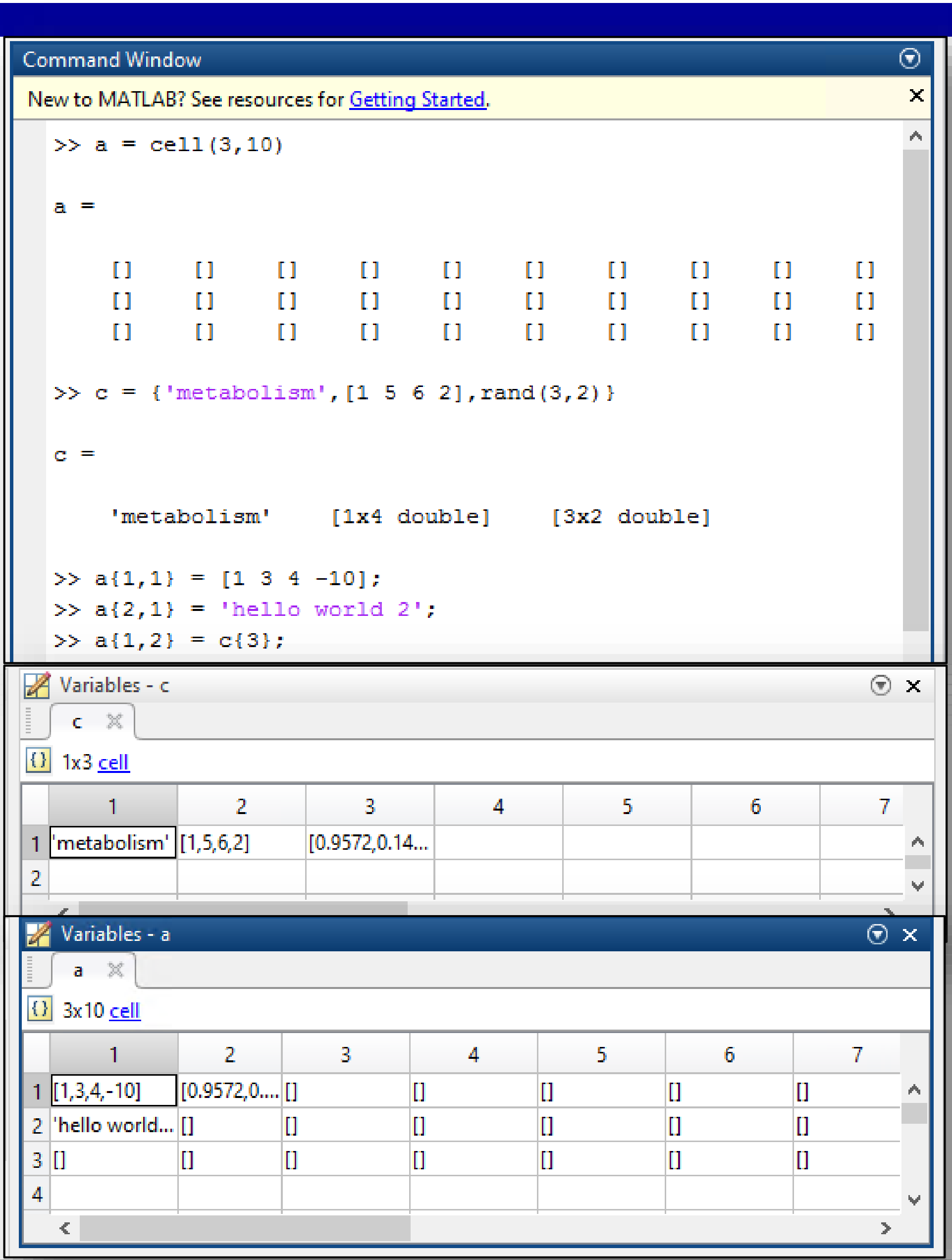


1x1 struct with 27 fields

Field	Value
modelVersion	1x1 struct
rxns	2738x1 cell
mets	1949x1 cell
S	1949x2738 sparse dou...
rev	2738x1 double
c	2738x1 double
metNames	1949x1 cell
metFormulas	1949x1 cell
lb	2738x1 double
ub	2738x1 double
metCharge	1949x1 int32
rules	2738x1 cell
genes	1391x1 cell
rxnGeneMat	2738x1391 sparse dou...
grRules	2738x1 cell
subSystems	2738x1 cell
confidenceScores	2738x1 cell
rxnReferences	2738x1 cell
rxnECNumbers	2738x1 cell
rxnNotes	2738x1 cell
rxnNames	2738x1 cell
metChEBIID	1949x1 cell
metKEGGID	1949x1 cell
metPubChemID	1949x1 cell

Cells

- A cell is just like a matrix, but each field can contain anything (even other matrices):
- To initialize a cell, specify the size
  - ✓ » `a = cell(3,10);`
    - will create a cell with 3 rows and 10 columns
- Create a cell manually with curly braces {}
  - ✓ » `c = {'metabolism',[1 5 6 2],rand(3,2)};`
    - c is a cell with 1 row and 3 columns
- Each element of a cell can be anything
- To access a cell element, use curly braces {}
  - ✓ » `a{1,1} = [1 3 4 -10];`
  - ✓ » `a{2,1} = 'hello world 2';`
  - ✓ » `a{1,2} = c{3};`



New to MATLAB? See resources for Getting Started.

```
>> a = cell(3,10)

a =

    []    []    []    []    []    []    []    []    []    []
    []    []    []    []    []    []    []    []    []    []
    []    []    []    []    []    []    []    []    []    []

>> c = {'metabolism',[1 5 6 2],rand(3,2)}

c =

    'metabolism'    [1x4 double]    [3x2 double]

>> a{1,1} = [1 3 4 -10];
>> a{2,1} = 'hello world 2';
>> a{1,2} = c{3};
```

1x3 cell

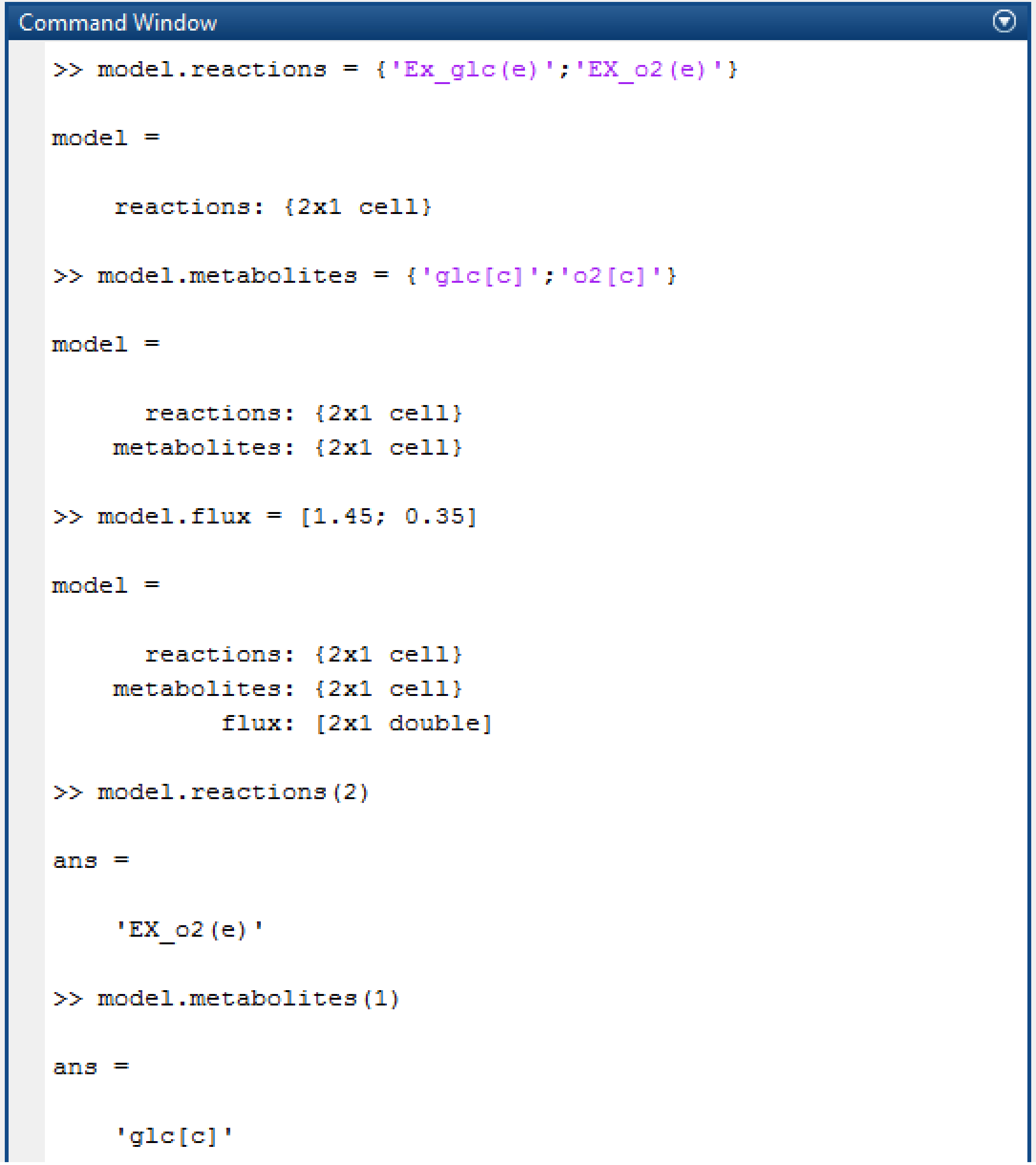
	1	2	3	4	5	6	7
1	'metabolism'	[1,5,6,2]	[0.9572,0.14...				
2							

3x10 cell

	1	2	3	4	5	6	7
1	[1,3,4,-10]	[0.9572,0....	[]	[]	[]	[]	[]
2	'hello world...	[]	[]	[]	[]	[]	[]
3	[]	[]	[]	[]	[]	[]	[]
4							

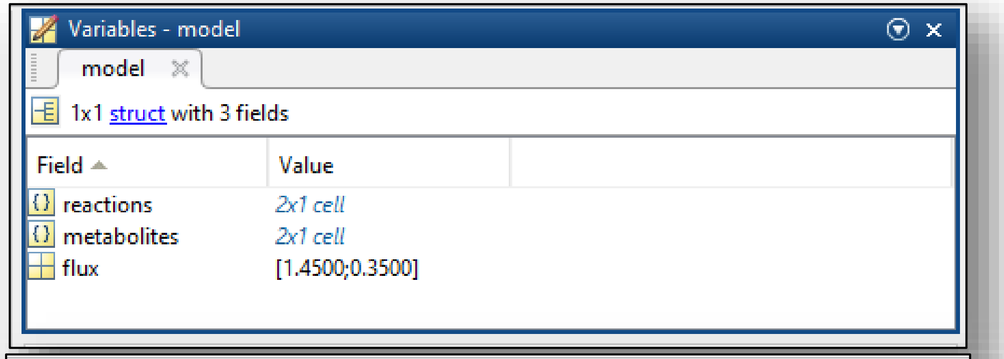
Structs

- Structs allow you to name and bundle relevant variables
  - ✓ Like C-structs, which are objects with fields
- To add fields
  - ✓ Fields can be anything: matrix, cell, even struct
  - ✓ Useful for keeping variables together
  - ✓ » `model.reactions = {'Ex_glc(e)';'EX_o2(e)'}`
  - ✓ » `model.metabolites = {'glc[c]';'o2[c]'}`
  - ✓ » `model.flux = [1.45; 0.35]`
- Accessing values from the struct
  - ✓ » `model.reactions(2)`
  - ✓ » `model.metabolites(1)`
  - ✓ » `model.flux(2)`



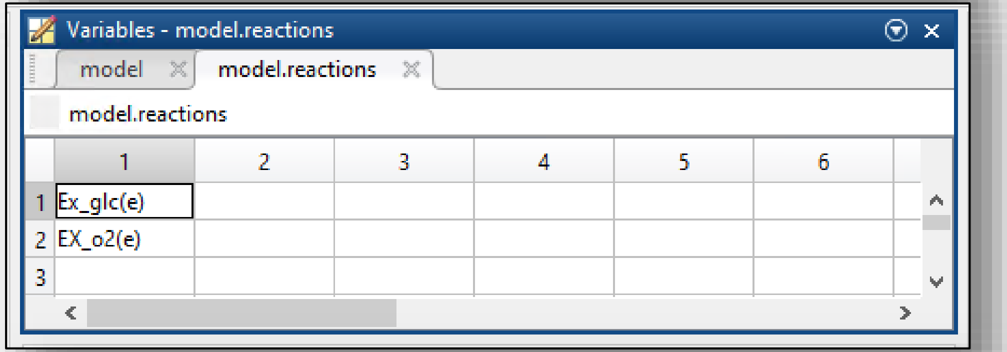
```
>> model.reactions = {'Ex_glc(e)';'EX_o2(e)'}

model =

    reactions: {2x1 cell}

>> model.metabolites = {'glc[c]';'o2[c]'}

model =

      reactions: {2x1 cell}
    metabolites: {2x1 cell}

>> model.flux = [1.45; 0.35]

model =

      reactions: {2x1 cell}
    metabolites: {2x1 cell}
           flux: [2x1 double]

>> model.reactions(2)

ans =

    'EX_o2(e)'

>> model.metabolites(1)

ans =

    'glc[c]'
```

Structs (2 of 2)



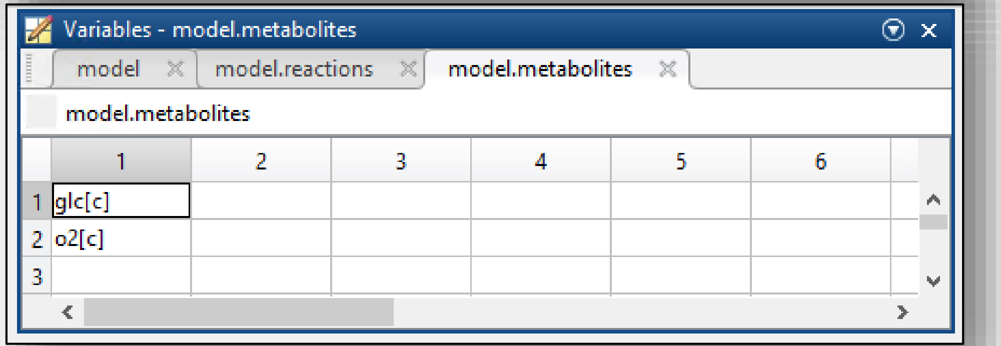
1x1 struct with 3 fields

Field	Value
reactions	2x1 cell
metabolites	2x1 cell
flux	[1.4500;0.3500]



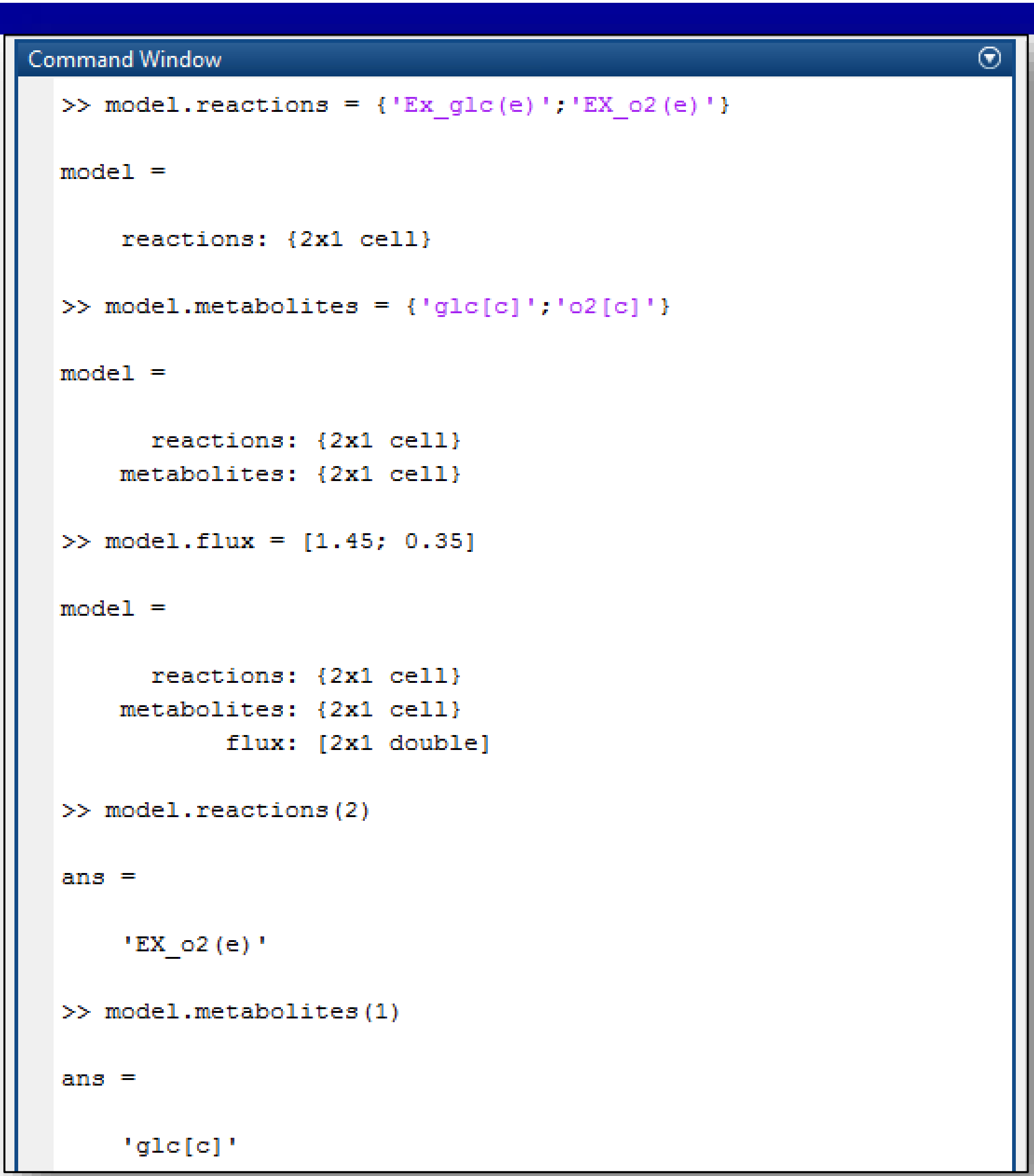
model.reactions

	1	2	3	4	5	6
1	Ex_glc(e)					
2	EX_o2(e)					
3						



model.metabolites

	1	2	3	4	5	6
1	glc[c]					
2	o2[c]					
3						



```
>> model.reactions = {'Ex_glc(e)';'EX_o2(e)'}

model = 

    reactions: {2x1 cell}

>> model.metabolites = {'glc[c]';'o2[c]'}

model = 

      reactions: {2x1 cell}
    metabolites: {2x1 cell}

>> model.flux = [1.45; 0.35]

model = 

      reactions: {2x1 cell}
    metabolites: {2x1 cell}
           flux: [2x1 double]

>> model.reactions(2)

ans = 

    'EX_o2(e)'

>> model.metabolites(1)

ans = 

    'glc[c]'
```

Course Introduction

- Desktop
- Variables
- Scripts
- Operations
- Visualization
- Programming
- Data Structures

Lecture Learning Objectives

Each student should be able to:

- Describe the Matlab desktop
- Explain the basic use of Matlab variables
- Explain the basic use of Matlab scripts
- Explain the basic mathematical operations in Matlab
- Explain the simple Matlab visualization techniques
- Explain simple Matlab programming
- Explain the basic data structures available in Matlab

References

- Matlab documentation
- MIT Opencourseware, introduction to Matlab by Danilo Šćepanović
- Matlab Academy - https://matlabacademy.mathworks.com/?s_tid=srchtitle
- Getting started with Matlab