MATLAB: Introduction
Part 1

Bruno Abreu Calfa

Last Update: August 9, 2011
Outline

What is MATLAB?
MATLAB Windows
MATLAB as a Calculator
MATLAB Classes
Scripts and Functions
   Writing MATLAB Programs
   Code Cells and Publishing
Outline

What is MATLAB?
MATLAB Windows
MATLAB as a Calculator
MATLAB Classes
Scripts and Functions
Writing MATLAB Programs
Code Cells and Publishing
MATLAB: Introduction

What is MATLAB?

A powerful tool!

- MATLAB stands for *Matrix Laboratory*
- Enhanced by *toolboxes* (specific routines for an area of application)
  - Optimization
  - Statistics
  - Control System
  - Bioinformatics
  - ... 
- Excellent for numerical computations
- Commonly regarded as a ‘Rapid Prototyping Tool’
- Used in industry and academia
Help with MATLAB?

- MATLAB’s Help
- Google
- A book about MATLAB
Outline

What is MATLAB?

MATLAB Windows

MATLAB as a Calculator

MATLAB Classes

Scripts and Functions
  Writing MATLAB Programs
  Code Cells and Publishing
Main Window I

- Command Window (prompt >>)
- Current Directory
- Workspace (contains variables stored in memory)
- Help Menu
Main Window II
Editor Window I

- Window Menu (Tile)
- Debug Menu (Run, Step, Step In, Step Out...)
- Cell Menu (Cell Mode)
Editor Window II
Outline

What is MATLAB?

MATLAB Windows

MATLAB as a Calculator

MATLAB Classes

Scripts and Functions
  Writing MATLAB Programs
  Code Cells and Publishing
### MATLAB as a Calculator

#### Basic Operators

- MATLAB supports the following mathematical operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>−</td>
<td>Subtraction</td>
</tr>
<tr>
<td>×</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td>^</td>
<td>Exponentiation</td>
</tr>
</tbody>
</table>

- Some examples:
  - `1 + 2`
  - `2*3 + 4`
  - `4/3 - 3/4 + 2^3`
Basic Operators

- MATLAB supports the following mathematical operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>−</td>
<td>Subtraction</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td>^</td>
<td>Exponentiation</td>
</tr>
</tbody>
</table>

- Some examples:
  - `>> 1 + 2`
  - `>> 2*3 + 4`
MATLAB: Introduction

MATLAB as a Calculator

Basic Operators

- MATLAB supports the following mathematical operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>−</td>
<td>Subtraction</td>
</tr>
<tr>
<td>×</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td>^</td>
<td>Exponentiation</td>
</tr>
</tbody>
</table>

- Some examples:
  - `>> 1 + 2`
  - `>> 2*3 + 4`
  - `>> 4/3 - 3/4 + 2^3`
Basic Operators

- Beware of operator *precedence* rules!
  - $\gg 2 \times 3 + 4$
Basic Operators

- Beware of operator *precedence* rules!
  - >> 2 * 3 + 4
  - >> 2 * (3 + 4)
Basic Operators

- Beware of operator *precedence* rules!
  - `2*3 + 4`
  - `2*(3 + 4)`
  - `4.2/3 + 1.2`
  - `4.2/(3 + 1.2)`
  - `15/(2 + 3)*(4 - 1)`
  - `15/((2 + 3)*(4 - 1))`
  - `2^3/2`
  - `2^(3/2)`

Use parentheses to enforce the desired order.
Basic Operators

- Beware of operator *precedence* rules!
  - ```matlab
g 2*3 + 4
```  
  - ```matlab
g 2*(3 + 4)
```  
  - ```matlab
g 4.2/3 + 1.2
```  
  - ```matlab
g 4.2/(3 + 1.2)
```
Basic Operators

▶ Beware of operator *precedence* rules!

▶ >> 2*3 + 4
▶ >> 2*(3 + 4)
▶ >> 4.2/3 + 1.2
▶ >> 4.2/(3 + 1.2)
▶ >> 15/(2 + 3)*(4 - 1)
Basic Operators

- Beware of operator *precedence* rules!
  - >> 2*3 + 4
  - >> 2*(3 + 4)
  - >> 4.2/3 + 1.2
  - >> 4.2/(3 + 1.2)
  - >> 15/(2 + 3)*(4 - 1)
  - >> 15/((2 + 3)*(4 - 1))
Basic Operators

- Beware of operator *precedence* rules!
  - `2*3 + 4`
  - `2*(3 + 4)`
  - `4.2/3 + 1.2`
  - `4.2/(3 + 1.2)`
  - `15/(2 + 3)*(4 - 1)`
  - `15/((2 + 3)*(4 - 1))`
  - `2^3/2`
MATLAB: Introduction

MATLAB as a Calculator

Basic Operators

- Beware of operator *precedence* rules!
  - `>> 2*3 + 4`
  - `>> 2*(3 + 4)`
  - `>> 4.2/3 + 1.2`
  - `>> 4.2/(3 + 1.2)`
  - `>> 15/(2 + 3)*(4 - 1)`
  - `>> 15/((2 + 3)*(4 - 1))`
  - `>> 2^3/2`
  - `>> 2^(3/2)`

- Use parentheses to enforce the desired order
MATLAB: Introduction

MATLAB Classes

Outline

What is MATLAB?

MATLAB Windows

MATLAB as a Calculator

MATLAB Classes

Scripts and Functions
  Writing MATLAB Programs
  Code Cells and Publishing
All Matrices!

▶ “Everything” in MATLAB is a matrix
  ▶ A scalar is a 1-by-1 matrix
  ▶ A 1D array of $n$ elements can be a $n$-by-1 (row vector) or a 1-by-$n$ (column vector) matrix
  ▶ A string of $n$ characters is a 1-by-$n$ matrix
  ▶ …

▶ Some MATLAB classes:
  ▶ double (Double-precision floating-point number array) (default)
  ▶ single (Single-precision floating-point number array)
  ▶ char (Character array)
  ▶ cell (Cell array)
  ▶ struct (Structure array)
  ▶ function_handle (Array of values for calling functions indirectly)
Scalar Variables: 1-by-1 Matrices!

- Use the ‘=’ sign for assignment
  - >>> a = 1 % The scalar variable ‘a’ stores the value 1
Scalar Variables: 1-by-1 Matrices!

- Use the ‘=’ sign for *assignment*
  - `a = 1` % The scalar variable ‘a’ stores the value 1
  - `>> % This is a comment and is ignored by the interpreter`
Scalar Variables: 1-by-1 Matrices!

- Use the ‘=’ sign for assignment
  - `>> a = 1` % The scalar variable ‘a’ stores the value 1
  - `>> % This is a comment and is ignored by the interpreter`
  - `>> sin(a)` % Sine of ‘a’ = 0.8415
Scalar Variables: 1-by-1 Matrices!

Use the ‘=’ sign for assignment

- `>> a = 1` % The scalar variable ‘a’ stores the value 1
- `>> % This is a comment and is ignored by the interpreter`
- `>> sin(a) % Sine of ‘a’ = 0.8415`
- `>> sin(a); % ‘;’ avoids displaying the result of the command`
Scalar Variables: 1-by-1 Matrices!

- Use the ‘=’ sign for assignment
  - `>> a = 1` % The scalar variable ‘a’ stores the value 1
  - `>> % This is a comment and is ignored by the interpreter`
  - `>> sin(a) % Sine of ‘a’ = 0.8415`
  - `>> sin(a); % ‘;’ avoids displaying the result of the command`
  - `>> size(a) % = [1,1], i.e. 1-by-1 matrix`
Scalar Variables: 1-by-1 Matrices!

- Use the ‘=’ sign for assignment
  - `a = 1` % The scalar variable ‘a’ stores the value 1
  - `sin(a)` % Sine of ‘a’ = 0.8415
  - `sin(a);` % ‘;’ avoids displaying the result of the command
  - `size(a)` % = [1, 1], i.e. 1-by-1 matrix
  - `b = a + 2` % `b = 3`
Scalar Variables: 1-by-1 Matrices!

- Use the ‘=’ sign for assignment
  - `>> a = 1` % The scalar variable ‘a’ stores the value 1
  - `>> % This is a comment and is ignored by the interpreter`
  - `>> sin(a) % Sine of ‘a’ = 0.8415`
  - `>> sin(a); % ‘;’ avoids displaying the result of the command`
  - `>> size(a) % = [1, 1], i.e. 1-by-1 matrix`
  - `>> b = a + 2 % b = 3`
  - `>> c = cos(b*pi/.2) % ‘pi’ is the builtin constant π`
Use the ‘=’ sign for assignment

- `>> a = 1` % The scalar variable ‘a’ stores the value 1
- `>> % This is a comment and is ignored by the interpreter`
- `>> sin(a)` % Sine of ‘a’ = 0.8415
- `>> sin(a);` % ‘;’ avoids displaying the result of the command
- `>> size(a)` % = [1,1], i.e. 1-by-1 matrix
- `>> b = a + 2` % b = 3
- `>> c = cos(b*pi/.2)` % ‘pi’ is the builtin constant π
- `>> d = rand` % A random scalar
Scalar Variables: 1-by-1 Matrices!

- Use the ‘=’ sign for assignment
  - `>> a = 1` % The scalar variable ‘a’ stores the value 1
  - `>> % This is a comment and is ignored by the interpreter`
  - `>> sin(a)` % Sine of ‘a’ = 0.8415
  - `>> sin(a);` % ‘;’ avoids displaying the result of the command
  - `>> size(a)` % = [1,1], i.e. 1-by-1 matrix
  - `>> b = a + 2` % b = 3
  - `>> c = cos(b*pi/.2)` % ‘pi’ is the builtin constant π
  - `>> d = rand` % A random scalar

- Use the commands `who` or `whos` to list the variables defined in the Workspace
Scalar Variables: 1-by-1 Matrices!

- Use the ‘=’ sign for assignment
  - `>> a = 1` % The scalar variable ‘a’ stores the value 1
  - `>> % This is a comment and is ignored by the interpreter`
  - `>> sin(a)` % Sine of ‘a’ = 0.8415
  - `>> sin(a);` % ‘;’ avoids displaying the result of the command

- Use the commands `who` or `whos` to list the variables defined in the Workspace

- Other common functions are available: exp, tan, sinh, acos, ...
1D Arrays: Real Vectors (or Matrices!)

- Use `[...,...]` or `[... ...]` for *horizontal stacking* and `[...;...]` for *vertical stacking*
- `>> v1 = [1 2 3]` % Row vector, same as `v1 = [1,2,3]`
1D Arrays: Real Vectors (or Matrices!)

- Use `[...,...]` or `[...;...]` for horizontal stacking and `[...;...]` for vertical stacking.
- `v1 = [1 2 3]` % Row vector, same as `v1 = [1,2,3]`
- `v2 = [4;5;6]` % Column vector
1D Arrays: Real Vectors (or Matrices!)

- Use \([\ldots, \ldots]\) or \([\ldots \ldots]\) for horizontal stacking and \([\ldots; \ldots]\) for vertical stacking

- \(>> v1 = [1 2 3]\) % Row vector, same as \(v1 = [1, 2, 3]\)

- \(>> v2 = [4; 5; 6]\) % Column vector

- \(>> v3 = v2 - v1\) % Error! Incompatible matrix dimensions
1D Arrays: Real Vectors (or Matrices!)

- Use \([\ldots,\ldots]\) or \([\ldots\ldots\ldots]\) for horizontal stacking and \([\ldots;\ldots]\) for vertical stacking

- \(\gg v1 = [1\ 2\ 3]\) \% Row vector, same as \(v1 = [1,2,3]\)
- \(\gg v2 = [4;5;6]\) \% Column vector
- \(\gg v3 = v2 - v1\) \% Error! Incompatible matrix dimensions
- \(\gg v3 = v2 - v1.'\) \% Transpose a real matrix with \(.'\)
1D Arrays: Real Vectors (or Matrices!)

- Use \([\ldots,\ldots]\) or \([\ldots;\ldots]\) for horizontal stacking and \([\ldots;\ldots]\) for vertical stacking

- >> v1 = [1 2 3] \% Row vector, same as \(v1 = [1,2,3]\)
- >> v2 = [4;5;6] \% Column vector
- >> v3 = v2 - v1 \% Error! Incompatible matrix dimensions
- >> v3 = v2 - v1.' \% Transpose a real matrix with .'
- >> v4 = v1*v2 \% Dot product, also \texttt{dot(v1,v2)}
1D Arrays: Real Vectors (or Matrices!)

- Use \([...,...]\) or \([...;...]\) for horizontal stacking and \([...;...]\) for vertical stacking

- \(>> v1 = [1 \ 2 \ 3]\) \% Row vector, same as \(v1 = [1,2,3]\)
- \(>> v2 = [4;5;6]\) \% Column vector
- \(>> v3 = v2 - v1\) \% Error! Incompatible matrix dimensions
- \(>> v3 = v2 - v1.'\) \% Transpose a real matrix with .'
- \(>> v4 = v1*v2\) \% Dot product, also \(\text{dot}(v1,v2)\)
- \(>> v7 = .1*v4\) \% Scalar-vector multiplication
1D Arrays: Real Vectors (or *Matrices*)

- Use `[...,...]` or `[... ...]` for *horizontal stacking* and `[...;...]` for *vertical stacking*
  - `>> v1 = [1 2 3]` % Row vector, same as `v1 = [1,2,3]`
  - `>> v2 = [4;5;6]` % Column vector
  - `>> v3 = v2 - v1` % Error! Imcompatible *matrix* dimensions
  - `>> v3 = v2 - v1.'` % Transpose a real *matrix* with `.,'
  - `>> v4 = v1*v2` % Dot product, also `dot(v1,v2)`
  - `>> v7 = .1*v4` % Scalar-vector multiplication
  - `>> v7(1)` % First element of array `v7`
1D Arrays: Real Vectors (or Matrices!)

- Use \[ ...,... \] or \[ ... \ldots \] for horizontal stacking and \[ ...; ... \] for vertical stacking

- \texttt{v1} = \[1,2,3\] % Row vector, same as \texttt{v1} = \[1 2 3\]
- \texttt{v2} = \[4;5;6\] % Column vector
- \texttt{v3} = \texttt{v2} - \texttt{v1} % Error! Imcompatible matrix dimensions
- \texttt{v3} = \texttt{v2} - \texttt{v1}'. % Transpose a real matrix with .'
- \texttt{v4} = \texttt{v1} \ast \texttt{v2} % Dot product, also \texttt{dot(v1,v2)}
- \texttt{v7} = .1 \ast \texttt{v4} % Scalar-vector multiplication
- \texttt{v7}(1) % First element of array ‘\texttt{v7}’
- \texttt{v8} = \exp(\texttt{v7}) % Element-wise operation
1D Arrays: Real Vectors (or Matrices!)

- Use \([...,...]\) or \([...;...]\) for horizontal stacking and \([...;...]\) for vertical stacking
- \(\texttt{v1 = [1 2 3]}\) % Row vector, same as \(\texttt{v1 = [1,2,3]}\)
- \(\texttt{v2 = [4;5;6]}\) % Column vector
- \(\texttt{v3 = v2 - v1}\) % Error! Imcompatible matrix dimensions
- \(\texttt{v3 = v2 - v1.'}\) % Transpose a real matrix with .'
- \(\texttt{v4 = v1*v2}\) % Dot product, also \texttt{dot(v1,v2)}
- \(\texttt{v7 = .1*v4}\) % Scalar-vector multiplication
- \(\texttt{v7(1)}\) % First element of array 'v7'
- \(\texttt{v8 = exp(v7)}\) % Element-wise operation
- \(\texttt{sz8 = size(v8)}\) % = \([1 3]\)
1D Arrays: Real Vectors (or Matrices!)

- Use [ . . . , . . . ] or [ . . . . . . ] for horizontal stacking and [ . . . ; . . . ] for vertical stacking

\[
\begin{align*}
\text{v1} &= \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \quad \text{(Row vector, same as } v1 = [1,2,3]) \\
\text{v2} &= \begin{bmatrix} 4 ; 5 ; 6 \end{bmatrix} \quad \text{(Column vector)} \\
\text{v3} &= \text{v2} - \text{v1} \quad \text{Error! Imcompatible matrix dimensions} \\
\text{v3} &= \text{v2} - \text{v1}. \quad \text{(Transpose a real matrix with .')} \\
\text{v4} &= \text{v1} * \text{v2} \quad \text{(Dot product, also } \text{dot}(\text{v1}, \text{v2}) \\
\text{v7} &= 0.1 * \text{v4} \quad \text{(Scalar-vector multiplication)} \\
\text{v7}(1) &= \text{First element of array 'v7'} \\
\text{v8} &= \exp(\text{v7}) \quad \text{(Element-wise operation)} \\
\text{sz8} &= \text{size}(\text{v8}) \quad = [1 3] \\
\text{v9} &= \text{rand}(1,5) \quad \text{(Random 1-by-5 array)}
\end{align*}
\]
1D Arrays: Real Vectors (or *Matrices*)

- Use `[...,...]` or `[... ...]` for *horizontal stacking* and `[...;...]` for *vertical stacking*
  - `>> v1 = [1 2 3]` % Row vector, same as `v1 = [1,2,3]`
  - `>> v2 = [4;5;6]` % Column vector
  - `>> v3 = v2 - v1` % Error! Imcompatible matrix dimensions
  - `>> v3 = v2 - v1.'` % Transpose a real matrix with `.'`
  - `>> v4 = v1*v2` % Dot product, also `dot(v1,v2)`
  - `>> v7 = .1*v4` % Scalar-vector multiplication
  - `>> v7(1)` % First element of array ‘v7’
  - `>> v8 = exp(v7)` % Element-wise operation
  - `>> sz8 = size(v8)` % = `[1 3]`
  - `>> v9 = rand(1,5)` % Random 1-by-5 array
  - `>> p = prod(v1)` % Product of elements = 6
Use horizontal stacking and vertical stacking likewise.

```matlab
m1 = [1 2 3; 4 5 6] \% 2-by-3
```

```matlab
m2 = rand(2,3) \% Random 2-by-3 matrix
```

```matlab
m3 = m1 + m2 \% Matrix addition
```

```matlab
m4 = m1 * m2 \% Error! Dimensions don't agree
```

```matlab
m4 = m1 * m2.' \% OK! Transpose a real matrix with .'
```

```matlab
m4(1,2) \% Element in row 1 and column 2 of m4
```

```matlab
len4 = length(m4) \% Size of longest dimension
```

```matlab
m5 = m3 / 2 \% Element-wise division
```

```matlab
m6 = tan(m5) \% Element-wise operation
```
2D Arrays: Real Matrices

- Use *horizontal stacking* and *vertical stacking* likewise
  - `>> m1 = [1 2 3; 4 5 6] % 2-by-3`
  - `>> m1p = [1,2,3; 4,5,6] % 2-by-3, same as m1`
Use **horizontal stacking** and **vertical stacking** likewise

- `m1 = [1 2 3; 4 5 6]` % 2-by-3
- `m1p = [1,2,3; 4,5,6]` % 2-by-3, same as `m1`
- `m2 = rand(2,3)` % Random 2-by-3 matrix
2D Arrays: Real Matrices

- Use *horizontal stacking* and *vertical stacking* likewise
  - `>> m1 = [1 2 3; 4 5 6] % 2-by-3`
  - `>> m1p = [1,2,3; 4,5,6] % 2-by-3, same as m1`
  - `>> m2 = rand(2,3) % Random 2-by-3 matrix`
  - `>> m3 = m1 + m2 % Matrix addition`
2D Arrays: Real Matrices

- Use *horizontal stacking* and *vertical stacking* likewise

```matlab
% MATLAB: Intro
m1 = [1 2 3; 4 5 6] % 2-by-3
m1p = [1, 2, 3; 4, 5, 6] % 2-by-3, same as m1
m2 = rand(2,3) % Random 2-by-3 matrix
m3 = m1 + m2 % Matrix addition
m4 = m1.*m2 % Error! Dimensions don't agree
```
2D Arrays: Real Matrices

- Use *horizontal stacking* and *vertical stacking* likewise
  
  ```matlab
  >> m1 = [1 2 3; 4 5 6] % 2-by-3
  >> m1p = [1,2,3; 4,5,6] % 2-by-3, same as m1
  >> m2 = rand(2,3) % Random 2-by-3 matrix
  >> m3 = m1 + m2 % Matrix addition
  >> m4 = m1*m2 % Error! Dimensions don’t agree
  >> m4 = m1*m2.' % OK! Transpose a real matrix with .'
  ```
2D Arrays: Real Matrices

- **Use horizontal stacking and vertical stacking likewise**
  ```matlab
  >> m1 = [1 2 3; 4 5 6] % 2-by-3
  >> m1p = [1,2,3; 4,5,6] % 2-by-3, same as m1
  >> m2 = rand(2,3) % Random 2-by-3 matrix
  >> m3 = m1 + m2 % Matrix addition
  >> m4 = m1*m2 % Error! Dimensions don’t agree
  >> m4 = m1*m2.' % OK! Transpose a real matrix with .'
  >> m4(1,2) % Element in row 1 and column 2 of ‘m4’
  ```
2D Arrays: Real Matrices

- Use *horizontal stacking* and *vertical stacking* likewise
  
  ```
  >> m1 = [1 2 3; 4 5 6]  % 2-by-3
  >> m1p = [1,2,3; 4,5,6]  % 2-by-3, same as m1
  >> m2 = rand(2,3)  % Random 2-by-3 matrix
  >> m3 = m1 + m2  % Matrix addition
  >> m4 = m1*m2  % Error! Dimensions don't agree
  >> m4 = m1*m2.'  % OK! Transpose a real matrix with .'
  >> m4(1,2)  % Element in row 1 and column 2 of 'm4'
  >> len4 = length(m4)  % Size of longest dimension
  ```
2D Arrays: Real Matrices

- Use *horizontal stacking* and *vertical stacking* likewise
  - `>> m1 = [1 2 3; 4 5 6]`  % 2-by-3
  - `>> m1p = [1,2,3; 4,5,6]`  % 2-by-3, same as `m1`
  - `>> m2 = rand(2,3)`  % Random 2-by-3 matrix
  - `>> m3 = m1 + m2`  % Matrix addition
  - `>> m4 = m1*m2`  % Error! Dimensions don’t agree
  - `>> m4 = m1*m2.'`  % OK! Transpose a real matrix with `.‘`
  - `>> m4(1,2)`  % Element in row 1 and column 2 of `m4`  
  - `>> len4 = length(m4)`  % Size of longest dimension
  - `>> m5 = m3/2`  % Element-wise division
2D Arrays: Real Matrices

Use *horizontal stacking* and *vertical stacking* likewise

- \( \gg \ m1 = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \) \% 2-by-3
- \( \gg \ m1p = \begin{bmatrix} 1,2,3 \\ 4,5,6 \end{bmatrix} \) \% 2-by-3, same as \( m1 \)
- \( \gg \ m2 = \text{rand}(2,3) \) \% Random 2-by-3 matrix
- \( \gg \ m3 = m1 + m2 \) \% Matrix addition
- \( \gg \ m4 = m1 * m2 \) \% Error! Dimensions don’t agree
- \( \gg \ m4 = m1 * m2. ' \) \% OK! Transpose a real matrix with \( . ' \)
- \( \gg \ m4(1,2) \) \% Element in row 1 and column 2 of ‘\( m4 \)’
- \( \gg \ len4 = \text{length}(m4) \) \% Size of longest dimension
- \( \gg \ m5 = m3 / 2 \) \% Element-wise division
- \( \gg \ m6 = \tan(m5) \) \% Element-wise operation
Element-wise Operations

The following are element-wise mathematical operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>.*</td>
<td>Element-wise Multiplication</td>
</tr>
<tr>
<td>./</td>
<td>Element-wise Division</td>
</tr>
<tr>
<td>.^</td>
<td>Element-wise Exponentiation</td>
</tr>
</tbody>
</table>

More examples:

```matlab
>> v1 = [1 2 3] % 1-by-3
```
Element-wise Operations

The following are element-wise mathematical operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>.*</td>
<td>Element-wise Multiplication</td>
</tr>
<tr>
<td>./</td>
<td>Element-wise Division</td>
</tr>
<tr>
<td>.^</td>
<td>Element-wise Exponentiation</td>
</tr>
</tbody>
</table>

More examples:

- `v1 = [1 2 3]` % 1-by-3
- `v2 = [2 4 6]` % 1-by-3
### Element-wise Operations

The following are element-wise mathematical operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>.*</td>
<td>Element-wise Multiplication</td>
</tr>
<tr>
<td>./</td>
<td>Element-wise Division</td>
</tr>
<tr>
<td>.^</td>
<td>Element-wise Exponentiation</td>
</tr>
</tbody>
</table>

More examples:

- `v1 = [1 2 3] % 1-by-3`
- `v2 = [2 4 6] % 1-by-3`
- `v3 = v1.*v2 % = [2 8 18]`
Element-wise Operations

The following are element-wise mathematical operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>.*</td>
<td>Element-wise Multiplication</td>
</tr>
<tr>
<td>./</td>
<td>Element-wise Division</td>
</tr>
<tr>
<td>.^</td>
<td>Element-wise Exponentiation</td>
</tr>
</tbody>
</table>

More examples:

- `>> v1 = [1 2 3] % 1-by-3`
- `>> v2 = [2 4 6] % 1-by-3`
- `>> v3 = v1.*v2 % = [2 8 18]`
- `>> v4 = v2./v1 % = [2 2 2]`
Element-wise Operations

The following are element-wise mathematical operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>.*</td>
<td>Element-wise Multiplication</td>
</tr>
<tr>
<td>./</td>
<td>Element-wise Division</td>
</tr>
<tr>
<td>.^</td>
<td>Element-wise Exponentiation</td>
</tr>
</tbody>
</table>

More examples:

- >>> v1 = [1 2 3] % 1-by-3
- >>> v2 = [2 4 6] % 1-by-3
- >>> v3 = v1.*v2 % = [2 8 18]
- >>> v4 = v2./v1 % = [2 2 2]
- >>> v5 = v1.^v4 % = [1 4 9]
Element-wise Operations

The following are element-wise mathematical operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>.*</td>
<td>Element-wise Multiplication</td>
</tr>
<tr>
<td>./</td>
<td>Element-wise Division</td>
</tr>
<tr>
<td>.^</td>
<td>Element-wise Exponentiation</td>
</tr>
</tbody>
</table>

More examples:

- `>> v1 = [1 2 3]` % 1-by-3
- `>> v2 = [2 4 6]` % 1-by-3
- `>> v3 = v1.*v2` % = [2 8 18]
- `>> v4 = v2./v1` % = [2 2 2]
- `>> v5 = v1.^v4` % = [1 4 9]
- `>> m1 = [0 1; 1 0]` % 2-by-2
Element-wise Operations

The following are element-wise mathematical operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>.*</td>
<td>Element-wise Multiplication</td>
</tr>
<tr>
<td>./</td>
<td>Element-wise Division</td>
</tr>
<tr>
<td>.^</td>
<td>Element-wise Exponentiation</td>
</tr>
</tbody>
</table>

More examples:

- \( \gg \gg v1 = [1 \ 2 \ 3] \% 1\text{-by-}3 \)
- \( \gg \gg v2 = [2 \ 4 \ 6] \% 1\text{-by-}3 \)
- \( \gg \gg v3 = v1.*v2 \% = [2 \ 8 \ 18] \)
- \( \gg \gg v4 = v2./v1 \% = [2 \ 2 \ 2] \)
- \( \gg \gg v5 = v1.^v4 \% = [1 \ 4 \ 9] \)
- \( \gg \gg m1 = [0 \ 1; \ 1 \ 0] \% 2\text{-by-}2 \)
- \( \gg \gg m2 = [3 \ 5; \ 7 \ 2] \% 2\text{-by-}2 \)
Element-wise Operations

- The following are element-wise mathematical operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>.*</td>
<td>Element-wise Multiplication</td>
</tr>
<tr>
<td>./</td>
<td>Element-wise Division</td>
</tr>
<tr>
<td>.^</td>
<td>Element-wise Exponentiation</td>
</tr>
</tbody>
</table>

- More examples:
  - >> v1 = [1 2 3] \% 1-by-3
  - >> v2 = [2 4 6] \% 1-by-3
  - >> v3 = v1.*v2 \% = [2 8 18]
  - >> v4 = v2./v1 \% = [2 2 2]
  - >> v5 = v1.^v4 \% = [1 4 9]
  - >> m1 = [0 1; 1 0] \% 2-by-2
  - >> m2 = [3 5; 7 2] \% 2-by-2
  - >> m3 = m1.*m2 \% = [0 5; 7 0]
The Colon (:) Operator

- Use it extensively!
  - >> v1 = 1:10 % Same as v1 = [1, 2, 3, ..., 10]
  - >> v2 = 0:.1:1 % Same as v2 = [0,.1,.2,...,1]
  - >> m1 = rand(5) % Random 5-by-5 matrix
  - >> v3 = v1(5:end) % v3 = [5,6,7,8,9,10]
  - >> v4 = m1(:,3) % 'v4' has the elements in column 3 of 'm1'
  - >> v5 = m1(1,:) % 'v5' has the elements in row 1 of 'm1'
  - Do not forget linspace to generate linearly spaced vectors!
  - >> v6 = linspace(0,1,10) % = [0,0.1111,0.2222,...,1]
  - >> v7 = linspace(0,10,5) % = [0,2.5,5,7.5,10]
  - >> v8 = linspace(0,1,100) % = [0,0.0101,0.0202,...,1]
The Colon (:) Operator

- Use it extensively!
  - `v1 = 1:10` % Same as `v1 = [1, 2, 3, ..., 10]`
  - `v2 = 0:.1:1` % Same as `v2 = [0, .1, .2, ..., 1]`

- Do not forget `linspace` to generate linearly spaced vectors!
  - `v6 = linspace(0,1,10)` % = [0,0.1111,0.2222,...,1]
  - `v7 = linspace(0,10,5)` % = [0,2.5,5,7.5,10]
  - `v8 = linspace(0,1,100)` % = [0,0.0101,0.0202,...,1]
MATLAB: Introduction

MATLAB Classes

The Colon (:) Operator

▶ Use it extensively!
  ▶ >> v1 = 1:10 % Same as v1 = [1, 2, 3, ..., 10]
  ▶ >> v2 = 0:.1:1 % Same as v2 = [0, .1, .2, ..., 1]
  ▶ >> m1 = rand(5) % Random 5-by-5 matrix

▶ >> v3 = v1(5:end) % v3 = [5, 6, 7, 8, 9, 10]
▶ >> v4 = m1(:,3) % 'v4' has the elements in column 3 of 'm1'
▶ >> v5 = m1(1,:) % 'v5' has the elements in row 1 of 'm1'
▶ Do not forget linspace to generate linearly spaced vectors!
  ▶ >> v6 = linspace(0,1,10) % = [0, 0.1111, 0.2222, ..., 1]
  ▶ >> v7 = linspace(0,10,5) % = [0, 2.5, 5, 7.5, 10]
  ▶ >> v8 = linspace(0,1,100) % = [0, 0.0101, 0.0202, ..., 1]
The Colon (:) Operator

Use it extensively!

- >> v1 = 1:10 % Same as v1 = [1, 2, 3, ..., 10]
- >> v2 = 0:.1:1 % Same as v2 = [0, .1, .2, ..., 1]
- >> m1 = rand(5) % Random 5-by-5 matrix
- >> v3 = v1(5:end) % v3 = [5, 6, 7, 8, 9, 10]
The Colon (:) Operator

- Use it extensively!
  - `>> v1 = 1:10` % Same as `v1 = [1, 2, 3, ..., 10]`
  - `>> v2 = 0:.1:1` % Same as `v2 = [0, .1, .2, ..., 1]`
  - `>> m1 = rand(5)` % Random 5-by-5 matrix
  - `>> v3 = v1(5:end)` % `v3 = [5, 6, 7, 8, 9, 10]`
  - `>> v4 = m1(:,3)` % 'v4' has the elements in column 3 of 'm1'

- Do not forget `linspace` to generate linearly spaced vectors!
  - `>> v6 = linspace(0,1,10)` % = `[0, 0.1111, 0.2222, ..., 1]`
  - `>> v7 = linspace(0,10,5)` % = `[0, 2.5, 5, 7.5, 10]`
  - `>> v8 = linspace(0,1,100)` % = `[0, 0.0101, 0.0202, ..., 1]`
The Colon (:) Operator

- Use it extensively!
  - \( \gg v1 = 1:10 \) % Same as \( v1 = [1, 2, 3, \ldots, 10] \)
  - \( \gg v2 = 0:.1:1 \) % Same as \( v2 = [0, .1, .2, \ldots, 1] \)
  - \( \gg m1 = \text{rand}(5) \) % Random 5-by-5 matrix
  - \( \gg v3 = v1(5:\text{end}) \) % \( v3 = [5, 6, 7, 8, 9, 10] \)
  - \( \gg v4 = m1(:,3) \) % ‘\( v4 \)’ has the elements in column 3 of ‘\( m1 \)’
  - \( \gg v5 = m1(1,:) \) % ‘\( v5 \)’ has the elements in row 1 of ‘\( m1 \)’

- Do not forget \text{linspace} to generate linearly spaced vectors!
  - \( \gg v6 = \text{linspace}(0,1,10) \) % = \( [0, 0.1111, 0.2222, \ldots, 1] \)
  - \( \gg v7 = \text{linspace}(0,10,5) \) % = \( [0, 2.5, 5, 7.5, 10] \)
  - \( \gg v8 = \text{linspace}(0,1,100) \) % = \( [0, 0.0101, 0.0202, \ldots, 1] \)
The Colon (:) Operator

- Use it extensively!
  - `v1 = 1:10` % Same as `v1 = [1, 2, 3, ..., 10]`
  - `v2 = 0:.1:1` % Same as `v2 = [0, .1, .2, ..., 1]`
  - `m1 = rand(5)` % Random 5-by-5 matrix
  - `v3 = v1(5:end)` % `v3 = [5, 6, 7, 8, 9, 10]`
  - `v4 = m1(:,3)` % ‘v4’ has the elements in column 3 of ‘m1’
  - `v5 = m1(1,:)` % ‘v5’ has the elements in row 1 of ‘m1’

- Do not forget `linspace` to generate linearly spaced vectors!
  - `v6 = linspace(0,1,10)` % = `[0, 0.1111, 0.2222, ..., 1]`
The Colon (:) Operator

- Use it extensively!
  - >> v1 = 1:10 % Same as v1 = [1, 2, 3, ..., 10]
  - >> v2 = 0:.1:1 % Same as v2 = [0, .1, .2, ..., 1]
  - >> m1 = rand(5) % Random 5-by-5 matrix
  - >> v3 = v1(5:end) % v3 = [5, 6, 7, 8, 9, 10]
  - >> v4 = m1(:,3) % 'v4' has the elements in column 3 of 'm1'
  - >> v5 = m1(1,:)
     % 'v5' has the elements in row 1 of 'm1'
- Do not forget linspace to generate linearly spaced vectors!
  - >> v6 = linspace(0,1,10) % =
     [0, 0.1111, 0.2222, ..., 1]
  - >> v7 = linspace(0,10,5) % =
     [0, 2.5, 5, 7.5, 10]
The Colon (:) Operator

- Use it extensively!
  - >> v1 = 1:10 % Same as v1 = [1, 2, 3, ..., 10]
  - >> v2 = 0:.1:1 % Same as v2 = [0, .1, .2, ..., 1]
  - >> m1 = rand(5) % Random 5-by-5 matrix
  - >> v3 = v1(5:end) % v3 = [5, 6, 7, 8, 9, 10]
  - >> v4 = m1(:,3) % 'v4' has the elements in column 3 of 'm1'
  - >> v5 = m1(1,: % 'v5' has the elements in row 1 of 'm1'
- Do not forget linspace to generate linearly spaced vectors!
  - >> v6 = linspace(0,1,10) %=
    [0, 0.1111, 0.2222, ..., 1]
  - >> v7 = linspace(0,10,5) %=
    [0, 2.5, 5, 7.5, 10]
  - >> v8 = linspace(0,1,100) %=
    [0, 0.0101, 0.0202, ..., 1]
Strings: `char` Arrays

- Remember that strings are also *matrices* in MATLAB!

  ```matlab
  >> str1 = 'Hello, world!' % A simple string
  >> sz1 = size(str1)
  % = 1-by-13
  >> a = rand; str2 = ['a = ' num2str(a)]
  % Horizontal stacking concatenates strings
  >> b = str2num('500')*rand
  % MATLAB has many handy functions!
  >> sprintf('Volume of reactor = %.2f', 10.23451)
  % Floating-point format with two decimal digits
  >> str3 = sprintf('A large number = %e', rand*10^5)
  % Exponential notation format
  >> sprintf('Another large number = %g', rand*10^5)
  % More compact format between %e and %f
Strings: `char` Arrays

- Remember that strings are also *matrices* in MATLAB!
  - `>> str1 = 'Hello, world!'` % A simple string
  - `>> sz1 = size(str1) % = 1-by-13`
Strings: `char` Arrays

- Remember that strings are also *matrices* in MATLAB!
  - `>> str1 = 'Hello, world!'` % A simple string
  - `>> sz1 = size(str1) % = 1-by-13`
  - `>> a = rand; str2 = ['a = ' num2str(a)] % Horizontal stacking concatenates strings`
Strings: char Arrays

- Remember that strings are also *matrices* in MATLAB!
  - `str1 = 'Hello, world!'` % A simple string
  - `sz1 = size(str1) % = 1-by-13`
  - `a = rand; str2 = ['a = ' num2str(a)]` % Horizontal stacking concatenates strings
  - `b = str2num('500')*rand` % MATLAB has many handy *2* functions!
Strings: char Arrays

- Remember that strings are also *matrices* in MATLAB!
  - `>> str1 = 'Hello, world!' % A simple string`
  - `>> sz1 = size(str1) % = 1-by-13`
  - `>> a = rand; str2 = ['a = ' num2str(a)] % Horizontal stacking concatenates strings`
  - `>> b = str2num('500')*rand % MATLAB has many handy *2* functions!`
- Format your strings with `sprintf`
  - `>> sprintf('Volume of reactor = %.2f', 10.23451) % Floating-point format with two decimal digits`
Strings: \texttt{char} Arrays

- Remember that strings are also \textit{matrices} in MATLAB!
  - \texttt{str1 = 'Hello, world!' } % A simple string
  - \texttt{sz1 = size(str1) } % = 1-by-13
  - \texttt{a = rand; str2 = ['a = ' num2str(a)] } % Horizontal stacking concatenates strings
  - \texttt{b = str2num('500')*rand } % MATLAB has many handy *2* functions!
- Format your strings with \texttt{sprintf}
  - \texttt{sprintf('Volume of reactor = %.2f', 10.23451) } % Floating-point format with two decimal digits
  - \texttt{str3 = sprintf('A large number = %e', rand*10^5) } % Exponential notation format
Strings: char Arrays

- Remember that strings are also matrices in MATLAB!
  - `str1 = 'Hello, world!'` % A simple string
  - `sz1 = size(str1)` % = 1-by-13
  - `a = rand; str2 = ['a = ' num2str(a)]` % Horizontal stacking concatenates strings
  - `b = str2num('500')*rand` % MATLAB has many handy *2* functions!
- Format your strings with `sprintf`
  - `sprintf('Volume of reactor = %.2f', 10.23451)` % Floating-point format with two decimal digits
  - `str3 = sprintf('A large number = %e', rand*10^5)` % Exponential notation format
  - `sprintf('Another large number = %g', rand*10^5)` % More compact format between %e and %f
**function_handle (@) Class**

- Used in calling functions indirectly
function_handle (@) Class

- Used in calling functions indirectly
  
  ```matlab
  >> Sin = @sin; % The variable ‘Sin’ points to the function ‘sin’
  ```
function_handle (@) Class

- Used in calling functions indirectly
  - >> Sin = @sin; % The variable ‘Sin’ points to the function ‘sin’
  - >> Sin(pi) % Evaluates the sine of \pi
function_handle (@) Class

- Used in calling functions indirectly
  - ```
    >> Sin = @sin; % The variable 'Sin' points to the function 'sin'
    >> Sin(pi) % Evaluates the sine of π
  ```

- Can be used to create ‘anonymous functions’
  - ```
    >> myfun = @(x) 1./(x.^3 + 3*x - 5) % Anonymous function
  ```
function_handle (@) Class

- Used in calling functions indirectly
  - `>> Sin = @sin;` % The variable ‘Sin’ points to the function ‘sin’
  - `>> Sin(pi)` % Evaluates the sine of π

- Can be used to create ‘anonymous functions’
  - `>> myfun = @(x) 1./(x.^3 + 3*x - 5)` % Anonymous function
  - `>> quad(myfun,0,1)` % Adaptive Simpson quadrature to integrate ‘myfun’
Outline

What is MATLAB?

MATLAB Windows

MATLAB as a Calculator

MATLAB Classes

Scripts and Functions
  Writing MATLAB Programs
  Code Cells and Publishing
M-Files

- The file with source code is called M-File (*.m)
- **Scripts**: No input and no output arguments. Contain a series of commands that may call other scripts and functions.
- **Functions**: Accept input and output arguments. Usually called program *routines* and have a special definition syntax.
- Inside scripts and functions you may use programming statements, such as *flow*, *loop*, and *error control*
- Open the Editor Window and start coding!
Function M-Files

- General form:

```matlab
function [out1, out2, ...] = funname(in1, in2, ...)
    statement
    ...
end % Optional
```

- Example:

```matlab
function Z = virialgen(P,Pc,T,Tc,omega)
Pr = P/Pc;
Tr = T/Tc;
[B0,B1] = virialB(Tr);
Z = 1 + Pr/Tr*(B0 + omega*B1);

function [B0,B1] = virialB(Tr)
B0 = 0.083 - 0.422/Tr^1.6;
B1 = 0.139 - 0.172/Tr^4.2;
```
Code Cells

- Allow you to divide your M-files into sections (cells)
- Enable you to execute cell by cell
- Foundations for *publishing* your M-file to HTML, PDF, and other formats
- To begin a code cell, type `%%` at the beginning of a line
- The first line after the `%%` is the **title** of the code cell
- The next lines starting with `%` are a description of the code cell
- Place your code in the next lines
- A new code cell starts at the next `%%` at the beginning of a line
Simple example:

```matlab
%% 99-999: Homework 1
% Bruno Abreu Calfa

%% Problem 1
x = linspace(0,1);
y = sin(x.^2).*exp(-x.*tan(x));
plot(x,y);

%% Problem 2
a = 0;
b = 1;
f = @(t) exp(-t.^2);
intf = quad(f,a,b);
sprintf('Integral of f from %g to %g = %g',a,b,intf)
```
Publishing your Code

- Saves output of your code to a specific file type
- Formats available:

<table>
<thead>
<tr>
<th>File Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc</td>
<td>Microsoft Word⁷</td>
</tr>
<tr>
<td>latex</td>
<td>LaTeX⁷</td>
</tr>
<tr>
<td>ppt</td>
<td>Microsoft Powerpoint⁷</td>
</tr>
<tr>
<td>xml</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>pdf</td>
<td>Portable Document Format</td>
</tr>
<tr>
<td>html</td>
<td>Hypertext Markup Language</td>
</tr>
</tbody>
</table>

- MATLAB evaluates your M-file and generates the output
- To publish your M-file, go to: File -> Publish

⁷Syntax highlighting not preserved