Introduction
## Roadmap

<table>
<thead>
<tr>
<th>Category</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS-INTRO</td>
<td></td>
</tr>
<tr>
<td>SS-LOGIC</td>
<td></td>
</tr>
<tr>
<td>SS-DATA</td>
<td></td>
</tr>
<tr>
<td>PY-INTRO</td>
<td></td>
</tr>
<tr>
<td>PY-BLOCK</td>
<td></td>
</tr>
<tr>
<td>PY-LOGIC</td>
<td></td>
</tr>
<tr>
<td>PY-LISTS</td>
<td></td>
</tr>
<tr>
<td>PY-LOOPS</td>
<td></td>
</tr>
<tr>
<td>PY-MEM</td>
<td></td>
</tr>
<tr>
<td>PY-FILES</td>
<td></td>
</tr>
<tr>
<td>PY-DICT</td>
<td></td>
</tr>
<tr>
<td>PY-IO</td>
<td></td>
</tr>
<tr>
<td>PY-NUMPY</td>
<td></td>
</tr>
<tr>
<td>PY- PLOT</td>
<td></td>
</tr>
<tr>
<td>PY-STATE</td>
<td></td>
</tr>
<tr>
<td>PY-RAND</td>
<td></td>
</tr>
<tr>
<td>PY-ERROR</td>
<td></td>
</tr>
<tr>
<td>PY-DEBUG</td>
<td></td>
</tr>
<tr>
<td>PY-EQN</td>
<td></td>
</tr>
<tr>
<td>PY-BRUTE</td>
<td></td>
</tr>
<tr>
<td>ML-INTRO</td>
<td></td>
</tr>
<tr>
<td>ML-IO</td>
<td></td>
</tr>
<tr>
<td>ML-POLY</td>
<td></td>
</tr>
<tr>
<td>ML-STATS</td>
<td></td>
</tr>
<tr>
<td>ML-FIT</td>
<td></td>
</tr>
<tr>
<td>WRAP-UP</td>
<td></td>
</tr>
</tbody>
</table>
Interface
Start MATLAB either at the command line, `matlab`, or by clicking the icon.
Why MATLAB?

- Designed for engineering.
- Excellent documentation and toolboxes.
- Strong areas of application:
  - Linear algebra
  - Control dynamics
  - Numerical analysis
  - Image processing
Why MATLAB?

- Can you do anything with it that you can’t do in Python?
Why MATLAB?

- Can you do anything with it that you can’t do in Python?
- All programming languages can be made “equivalent”—so it depends on the libraries and applications, and the culture of your working group.
What is MATLAB?

- Programming language + environment.
- Proprietary, owned and maintained by MathWorks.
- Dates from late 1970s, under active development.
- Was an influence on NumPy/MPL, so will have familiar syntax.
Basics

- Literals, variables, operators
  \[ 4 ^ 3 \]

- Expressions
  \[ \begin{align*}
     a &= 3 \times 2 \\
     b &= 1 + a \\
     \text{Semicolon suppresses output (mutes): } &; \\
     b &= b + 2;
  \end{align*} \]
  - `ans` is default result.
  \[ a / 4 \]

- `fprintf` displays the value only.
  \[ \text{fprintf}( \text{ans} ); \]
MATLAB implements:
- integers
- floating-point numbers
- complex numbers

in 8-, 16-, 32-, and 64-bit versions (like NumPy).

`whos` shows type, value of all variables in workspace.
Arrays are the fundamental type in MATLAB:

```matlab
a = [ 1 2 3 ];
```

Arrays are indexed using parentheses:

```matlab
b = a( 1 );
```
Arrays are the fundamental type in MATLAB:

```matlab
a = [ 1 2 3 ];
b = a( 1 );
```

- MATLAB indexes from one, not zero!
More dimensional arrays use semicolons to separate rows:

\[ A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} ; \]

Arrays are indexed using parentheses and commas:

\[ a = A(1,2) ; \]

Helper functions are available:

\[ B = \text{ones}(3,3) + \text{eye}(3,3) + \text{zeros}(3,3) ; \]
Which of the following will produce this array?

A. \[
\begin{pmatrix}
1 & 1 & 1 \\
2 & 2 & 2
\end{pmatrix}
\]
B. \[
\begin{pmatrix}
1 & 1 & 1 \\
2 & 2 & 2
\end{pmatrix}
\]
C. \[
\begin{pmatrix}
1 & 2 \\
1 & 2 \\
1 & 2
\end{pmatrix}
\]
D. \[
\begin{pmatrix}
1 & 2 \\
1 & 2 \\
1 & 2
\end{pmatrix}
\]
E. \[
\begin{pmatrix}
\begin{pmatrix}1 & 1 & 1\end{pmatrix}, \begin{pmatrix}2 & 2 & 2\end{pmatrix}
\end{pmatrix}
\]
Which of the following will produce this array?

A \[ [1 1 1] ; [2 2 2] \]
B \[ [1 1 1 ; 2 2 2] \]
C \[ [1 2] ; [1 2] ; [1 2] \]
D \[ [1 2 ; 1 2 ; 1 2] \]
E \[ [ [1 1 1] , [2 2 2] ] \]

\[
\begin{pmatrix}
1 & 1 & 1 \\
2 & 2 & 2 \\
\end{pmatrix}
\]
Which of the following will access 4 in this array?

A) A(1,0)
B) A[2,1]
C) A(2,1)
D) A(1)(0)
Which of the following will access 4 in this array?

A  A( 1,0 )
B  A[ 2,1 ]
C  A( 2,1 )
D  A( 1 )( 0 )
% basic mathematics:
A = ( ones( 3,3 ) + 1 ) / 2
B = sin( ones( 3,3 ) * pi )
C = B'  % transpose with '

% matrix multiplication:
D = eye( 3,4 ) * ones( 4,5 ) * pi
\[
\begin{pmatrix}
2 & 1 \\
1 & 2
\end{pmatrix}
\]

Which of the following will produce this array?

A \(3 \times \text{ones}(2,2) - 2 \times \text{eye}(2,2)\)

B \(2 \times \text{ones}(2,2) + \text{eye}(2,2)\)

C \(3 \times \text{ones}(2,2) - \text{eye}(2,2)\)

D \(\text{ones}(2,2) + \text{eye}(2,2)\)
Which of the following will produce this array?

A 3*ones( 2,2 ) - 2*eye( 2,2 )
B 2*ones( 2,2 ) + eye( 2,2 )
C 3*ones( 2,2 ) - eye( 2,2 )
D ones( 2,2 ) + eye( 2,2 )*
% concatenating arrays
A = [ eye(3,4), eye(3,5); ones(2,4), ones(2,5) ]
How can we produce this array?

A \[ \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix} \]

B \[ \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \]

C \[ \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix} \]

D \[ \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \]
How can we produce this array?

A \[
\begin{bmatrix}
1 & 3 & 5 \\
2 & 4 & 6 \\
\end{bmatrix}
\]

B \[
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
5 & 6 \\
\end{bmatrix}
\]

C \[
\begin{bmatrix}
1 & 3 & 5 \\
2 & 4 & 6 \\
\end{bmatrix}
\]

D \[
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
5 & 6 \\
\end{bmatrix}
\]

\star
MATLAB uses .m files for two purposes: scripts and functions.

Comments are indicated as follows:

```matlab
% this is a comment
%{  
   this is a block comment
}%
```
Scripting

- Use the built-in editor to create these.
- Make sure you have the correct working directory.
- Scripts contain regular commands in order of execution.
Functions

- Functions must be located in a file of the same name as the function.

```matlab
function [ output ] = function_name( input )
    % ...
end
```

- No explicit `return` statements—rely on values in output variable list.
Functions

$$T_F = \frac{180}{100} T_c + 32$$

File TempC2F.m:

```matlab
function [ Tf ] = TempC2F( Tc )
    Tf = Tc * ( 180/100 ) + 32;
end
```
Strings

- Indicated with single quotes (only!).
  
  \[ s = 'XFEM'; \]
  
  - Print formatted strings with `fprintf`:
    
    ```c
    fprintf( '%f %f', sin(pi/3), cos(pi/4) );
    ```
“Matrix dimensions must agree.”

It is necessary to distinguish *elementwise* operations and *matrix* operations.

A = 2 * ones( 2,2 )
B = A .* eye( 2,2 )
C = A * eye( 2,2 )

These are distinguished by a dot . in front of the operator.
%% loop through time steps
for i = 1:2:10
    fprintf( 'The number is %i.' , i );
end
The for loop ranges over a set of possible values.
The for loop ranges over a set of possible values.

This is not as flexible as Python’s in syntax—think of always having to loop over the index rather than the item.

Ranges are straightforward: 1:10, 1:2:10, 0.1:0.1:0.5. Also have linspace available.
We create a for loop as follows:

- statement for var = range, where you create var and provide range
- one or more statements
- closing statement end

Also have continue and break available.
function [ y ] = absolute( x )
    y = 0;
    if x >= 0
        y = x;
    else
        y = -x;
    end %function
We create an if/else statement as follows:
- the keyword if
- a logical comparison (more on these!)
- a **block** of code
We create an if/else statement as follows:
- the keyword if
- a logical comparison (more on these!)
- a block of code
- the keyword elseif (note this!)
- a new logical comparison
- a different block of code
We create an `if/else` statement as follows:

- the keyword `if`
- a logical comparison (more on these!)
- a **block** of code
- the keyword `elseif` (note this!)
- a new logical comparison
- a different **block** of code
- the keyword `else`
- a different **block** of code
We create an \texttt{if}/\texttt{else} statement as follows:
- the keyword \texttt{if}
- a logical comparison (more on these!)
- a \textbf{block} of code
- the keyword \texttt{elseif} (note this!)
- a new logical comparison
- a different \textbf{block} of code
- the keyword \texttt{else}
- a different \textbf{block} of code
- the keyword \texttt{end}
MATLAB does not have a bool data type.
MATLAB does not have a bool data type.
Instead of True/False, MATLAB uses integers:
- 0 means False
- 1 means True
MATLAB does *not* have a `bool` data type.

Instead of `True/False`, MATLAB uses integers:
- 0 means False
- 1 means True

Available logical operators include:
- `<, >, <=, >=, ==, ~=`
- `&&` for ‘and’, `||` for ‘or’
- `ismember` checks equality of elements in arrays.
- Also, logical operators work as indices!
MATLAB does *not* have a `bool` data type.

Instead of True/False, MATLAB uses integers:
- 0 means False
- 1 means True

Available logical operators include:
- `<`, `>`, `<=`, `>=`, `==`, `~=`
- `&&` for ‘and’, `||` for ‘or’
- `ismember` checks equality of elements in arrays.

Also, logical operators work as indices!
- `A( A<0 )`
while loop

%% loop until condition is met
i = 0;
while i < 10
    i = i + 1;
    fprintf( 'The number is %i.' , i );
end
MATLAB supports many varieties of RNG:
- \texttt{rand}, uniform distribution \([0, 1)\)
- \texttt{randn}, normal distribution
- \texttt{randi}, random integers \([0, n)\)

Note that the interfaces for these are quite different from Python!
rand(5); % generate 5x5 matrix
rand(5,1); % generate 5x1 column vector
10 * rand(3); % 3x3 matrix from [0,10)
randi( 5 ); % generate number from [1,5]
randi( 5,2 ); % generate 2x2 matrix
randi( [ -5,5 ],10,1 ); % from [-5,5] in 10x1
randn(); % single normal number
randn( 5 ); % generate 5x5 matrix
randn( 5,2 ); % generate 5x2 matrix
Example: Seed

```matlab
randn( 'seed',1 );
x = linspace( 0,2*pi,101 );
y = sin( x/50 ) ./ x + .002 * randn( 101,1 );
clf
plot( x,y,'.' );
```
Dates and times can usefully be stored as values:

```matlab
t = datetime( Y,M,D,H,MI,S );
t = datetime( 'now','TimeZone','local','Format','d-MMM-y HH:mm:ss Z' );
t = datetime( '2017-12-01','InputFormat','yyyy-MM-dd' );
fprintf( t );
```
Like NumPy, but no imports (anywhere).

Remember to change: parentheses, indexing from 1, end keywords.

Hard to do dict-like things, easy to do numpy-like operations.