Introduction to MATLAB

Johnathan Metz and Emily Maddox
Dept. of Atmospheric Sciences, University of North Dakota, Grand Forks, ND
Disclaimer

• MATLAB® and the MATLAB logo are registered trademarks of The MathWorks, Inc.

• The authors of this presentation are not affiliated with The MathWorks, Inc.
What is MATLAB?

- “MATLAB® is a high-level language and interactive environment for numerical computation, visualization, and programming.” (The MathWorks, 2014)

- Developed by The MathWorks, Inc.

- Designed to make scientific computations and visualizations easier
History of MATLAB

• Development began in the late 1970s by Cleve Moler

• Originally designed to give access to LINPACK and EISPACK (linear algebra and eigenvalue libraries) without needing to learn Fortran
MATLAB Under the Hood

- MATLAB interpreter takes instructions written in the MATLAB language and executes them
- Interpreter, GUI, and some base functions written in C
- Most functions written in MATLAB
The MATLAB Environment

- Command Window
- Workspace/Variable Browser
- Command History
- Current Working Directory
- File Browser
- Toolstrip
The MATLAB Language

• While some functions can be performed point-and-click in the GUI, the true power of MATLAB lies in its powerful language

• The language (also called MATLAB) is a dynamically typed, matrix-based language

• OK, cool. But what does that mean?
MATLAB is dynamically typed

• Dynamic typing: type check performed at runtime

• The program cannot catch errors like using a string as a number until the statement is executed.

• This sounds bad, but it gives us flexibility.
MATLAB is dynamically typed

• Dynamic type checking results in no need to declare variables!

• MATLAB will assign the type automatically (typically double for numeric data and a char array for strings)
MATLAB is matrix-based

• This is the main reason MATLAB is so powerful!

• Everything in MATLAB is a matrix (or array).

• Even a single number is just a 1x1 matrix.

• Why?
MATLAB is matrix-based

- Allows us to perform computations on arrays without for loops...

- Example (multiply all elements in array by 2):

```c
int i, j;
for(i = 0; i < NUM_ROWS; i++) {
  for(j = 0; j < NUM_COLS; j++) {
    array[i][j] = array[i][j] * 2;
  }
}
```

MATLAB:

```matlab
array = array .* 2;
```
MATLAB Disadvantages

• Before we start working with MATLAB, remember that it has disadvantages, too.

• Execution time: MATLAB is fast, but nothing is ever as fast as compiled code.

• This is why the atmospheric models are written in Fortran!
MATLAB Disadvantages

• Lack of fine control: The abstraction of MATLAB restricts precise control over things like memory management (i.e. pointers).

• We typically like this (pointers are a headache!), but for some applications (like memory-intensive programs or hardware drivers) that control is important.

• Remember: MATLAB is a great tool, but it isn’t necessarily the best tool for every computing task!
Basic MATLAB Syntax

• Operators in MATLAB are similar to C:
  • +, -, *, / (Add, Subtract, Multiply, Divide)

• Some differences/new things:
  • Comments: % instead of // or /* */
  • Exponentiation: ^
  • “Element-by-element”: .
  • Strings use single quotes (‘’) instead of double (“”)

Intro to MATLAB

March 9, 2014
Element by Element

• The dot is needed to differentiate between matrix multiplication and element by element multiplication.

• These are two different ways to multiply matrices.

• Needed because MATLAB is matrix-based!
Matrix vs. Element by Element

- **Matrix multiplication (•):**

  \[
  \begin{bmatrix}
  A_1 & A_2 & A_3 \\
  B_1 & B_2 & B_3 \\
  C_1 & C_2 & C_3
  \end{bmatrix}
  \times
  \begin{bmatrix}
  D_4 & D_5 & D_6 \\
  E_4 & E_5 & E_6 \\
  F_4 & F_5 & F_6
  \end{bmatrix}
  =
  \begin{bmatrix}
  \text{row}(A) \cdot \text{col}(4) & \text{row}(A) \cdot \text{col}(5) & \text{row}(A) \cdot \text{col}(6) \\
  \text{row}(B) \cdot \text{col}(4) & \text{row}(B) \cdot \text{col}(5) & \text{row}(B) \cdot \text{col}(6) \\
  \text{row}(C) \cdot \text{col}(4) & \text{row}(C) \cdot \text{col}(5) & \text{row}(C) \cdot \text{col}(6)
  \end{bmatrix}
  \]

- **Element-by-element (•):**

  \[
  \begin{bmatrix}
  A_1 & A_2 & A_3 \\
  B_1 & B_2 & B_3 \\
  C_1 & C_2 & C_3
  \end{bmatrix}
  \cdot
  \begin{bmatrix}
  D_4 & D_5 & D_6 \\
  E_4 & E_5 & E_6 \\
  F_4 & F_5 & F_6
  \end{bmatrix}
  =
  \begin{bmatrix}
  A_1D_4 & A_2D_5 & A_3D_6 \\
  B_1E_4 & B_2E_5 & B_3E_6 \\
  C_1F_4 & C_2F_5 & C_3F_6
  \end{bmatrix}
  \]

- **My advice:** Unless you know for sure that you want matrix multiply, you probably need element by element.
clear

- Since MATLAB is array-based, *every variable* is an array.

- MATLAB uses parenthesis to access arrays:

  ```matlab
  1 % Set the fifth element of array to 25.
  2 array(5) = 25;
  ```

- **IMPORTANT:** In MATLAB, arrays start at 1, NOT 0!
Colon Operator

• Use to specify ranges.

• Example:

  • a(1:5) = First five elements of a
  • b(24:27) = Elements 24 - 27 of b
  • c(:) = All elements of c
Colon Operator

• Can also be used to define arrays

• Example:
  
  - a = 1:7; \%a = [1 2 3 4 5 6 7]

• Also used in the form start:increment:end

  - Example: b = 1:3:10; \%b = [1 4 7 10]
The Semicolon

• Unlike in C, the semicolon is **not** required in MATLAB.

• However, it serves a different purpose: the semicolon *suppresses* output.
The Semicolon

With Semicolon:  

\[
\begin{align*}
\text{>> } a &= 1:3:10; \\
\text{>> } & \\
\text{>> } &
\end{align*}
\]

Without Semicolon:

\[
\begin{align*}
\text{>> } a &= 1:3:10 \\
a &= \\
\begin{array}{cccc}
1 & 4 & 7 & 10 \\
\end{array}
\text{>> }
\end{align*}
\]

• Some commands can generate a lot of output, so be careful!
Conditional Statements (if)

• Has this form (no curly braces!):

```matlab
if condition
    do this
elseif condition
    do something else
else
    fine, do this
end
```
Conditional Statements (if)

• Example:

```matlab
1 if value < 10
2    thestr = 'value<10';
3 elseif value == 10
4    thestr = 'value==10';
5 else
6    thestr = 'value>10';
7 end
```
Looping Statements (for)

- Looping statements are much less common than in low level languages.
- They exist in MATLAB, but should be avoided as much as possible.
- MATLAB is not optimized for looping operations (read: they execute slowly)!
- Most of the time, there are alternatives to using loops in MATLAB (see slide 10 for one example).
Looping Statements (for)

• Has this form:

```matlab
for loopvariable=setofvalues
do something
end
```
Looping Statements (for)

• Example:

```matlab
1  %Create a staggered grid (common in numerical modeling).
2  %The for loop is needed if dz is unknown.
3  for k = 1:length(z_full) - 1
4      z_half(k) = (z_full(k) + z_full(k + 1)) ./ 2;
5  end
```

Courtesy: Dr. Gretchen Mullendore
Unnecessary Loops

• If performing an operation on all elements of an array, typically a loop is not required (slide 10).

• Also unnecessary if multiplying/dividing/exponentiating each element of an array by another (slide 15)

• Typically necessary only if doing operations with mismatched indices or if the operation requires multiple indices to compute one value (see previous slide)
Putting it Together: Exercise 1

• Approximate the saturation vapor pressure for water vapor from 40°F to 100°F in 10°F increments.

• From Bolton (1980): \( e_s(T) = 6.112e^{\left(\frac{17.67T}{T+243.5}\right)} \)

• T is in °C, saturation vapor pressure is in mb
Exercise 1

- Step 1: Set up independent variable (°F). (Use colon operator!)

- Step 2: Convert °F to °C (use element-by-element multiplication and vector addition).

- Step 3: Calculate saturation vapor pressure. Hint: In Matlab, use exp(x) to get e^x.
Exercise 1 Solution

1 %Set up independent variable.
2 T_F = 40:10:100;
3
4 %Convert to °C.
5 T_C = (T_F - 32) .* (5./9);
6
7 %Calculate saturation vapor pressure.
8 e_s = 6.112 .* exp((17.67.*T_C)./(T_C+243.5))


e_s =

8.3842  12.2602  17.6378  24.9888  34.8992  48.0870  65.4234
Multidimensional arrays (matrices)

• Multiple dimensions are separated by commas

• Example:

1 %Retrieve the element located at the third row, second column.
2 element = array(3, 2);

• Note: the first index is the row, the second is the column

This is opposite of normal (x,y) coordinates!
Scripts and Functions

• Typing commands into the interpreter one by one is tedious.

• We also can’t re-execute the code very easily.

• Solution: place the code into a script!

• File extension: .m (Note: same as Objective-C)
Scripts and Functions

- MATLAB has a built-in text editor, but any other editor can be used.

- This includes Vim (yes!), Emacs (no), gedit (I suppose), TextEdit (okay), and Notepad (seriously?).
MATLAB Editor

- Click “New Script” or type `edit` into the command line.
Scripts and Functions

• To execute a script, type its name (minus the “.m”) into the command line.

• Make sure that MATLAB is working in the correct directory!

Change if needed!
Scripts and Functions

- Just like in C, MATLAB functions take in some value (or values) and return anywhere from zero to multiple elements.

- They have this form:

  ```matlab
  function [out1 out2 ...] = name_of_function(in1, in2, ...)
  do something
  end
  ```

- Must be saved in .m files with same name as function
Example Function

```matlab
function [e_s] = escalc(temp, units)

    % Convert units if needed.
    if strcmp(units, 'F') || strcmp(units, 'f')
        temp = (temp - 32) .* 5./9;
    elseif strcmp(units, 'C') || strcmp(units, 'c')
        % Do nothing.
    elseif strcmp(units, 'K') || strcmp(units, 'k')
        temp = temp - 273.15;
    else
        error('Error: Incorrect unit specification');
    end

    e_s = 6.1094 .* exp((17.625.*temp)./(temp+243.04));

end
```
help and doc

- MATLAB has a great amount of built in documentation.
- Type `doc` to open the documentation.
- Type `doc some_command` to get help with `some_command`.
- `help` does the same thing, except it pastes the output into the command window.
Plotting Functions

• One of the primary reasons that MATLAB is so useful for scientific computing is its versatile and powerful plotting capabilities.

• MATLAB can produce numerous different plots; see the documentation for the full list.
Line Plot

• One of the most useful plots is the basic line plot.

• In MATLAB, it is created using the plot function.

```matlab
1 %Define x values.
2 x = 0:0.01:2.*pi;
3
4 %Calculate y.
5 y = sin(x);
6
7 %Plot.
8 plot(x, y);
```
Color and Line Customization

• Can customize using string placed after x and y.

plot(x, y, ‘format’);

• Consists of three parts:
  • Color
  • Line Style
  • Marker
Color and Line Customization

- The elements can appear in any order or be omitted
- Some values (see documentation for full list):

<table>
<thead>
<tr>
<th>Colors</th>
<th>Line Styles</th>
<th>Markers</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘r’     Red</td>
<td>‘-’ Solid</td>
<td>‘+’ Plus</td>
</tr>
<tr>
<td>‘b’     Blue</td>
<td>‘--’ Dashed</td>
<td>‘*’ Asterisk</td>
</tr>
<tr>
<td>‘g’     Green</td>
<td>‘.’ Dotted</td>
<td>‘x’ Cross</td>
</tr>
<tr>
<td>‘k’     Black</td>
<td>‘.-.’ Dash-dot</td>
<td>‘s’ Square</td>
</tr>
</tbody>
</table>
Color and Line Customization

• Example:

1 %Define x values.
2 x = 0:0.1:2.*pi;
3
4 %Calculate y.
5 y = sin(x);
6
7 %Plot.
8 plot(x, y, 'sr-.');
Logarithmic Axes

• To plot on a log plot use of these functions instead of plot:
  
  • semilogx() - Logarithmic x-axis, normal y-axis
  
  • semilogy() - Normal x-axis, logarithmic y-axis
  
  • loglog() - Logarithmic x-axis and y-axis
Plots with Multiple Curves

- By default, MATLAB erases a plot before plotting a new one.

- To prevent this (and thus plot more than one curve), use the `hold on` command.

- Use `hold all` to force MATLAB to keep switching colors.

- Use `hold off` to return to normal behavior.
Labeling and Titling Plots

• Use these functions:
  • xlabel(‘Insert x-axis label’)
  • ylabel(‘Insert y-axis label’)
  • title(‘Insert title’)
  • legend(‘Title for line 1’, ‘Title for line 2’, …)
Labeling and Titling Plots

• Example:

```matlab
1 %Define x values.
2 x = 0:0.01:2.*pi;
3
4 %Calculate y.
5 y = sin(x);
6
7 %Plot.
8 plot(x, y);
9
10 %Label and title.
11 xlabel('x');
12 ylabel('sin(x)');
13 title('Plot of sin(x) from 0 to 2*pi');
```
Adjusting Label Font Size

• Adjust the FontSize property of the label

• For example:

```matlab
10 %Label and title.
11 xlabel('x', 'FontSize', 12);
12 ylabel('sin(x)', 'FontSize', 12);
13 title('Plot of sin(x) from 0 to 2*pi', 'FontSize', 14);
```
Multiple Plot Windows

• Use *figure* to open another plot window.

• MATLAB will plot to the *active* window (i.e. the one most recently clicked on).
Scatterplots

- MATLAB can also create scatterplots using the `scatter` function.

- Example:

```matlab
1  %Generate two sets of 100 random numbers.
2  ran_numbers = rand(100, 2);
3
4  %Create scatterplot.
5  scatter(ran_numbers(:, 1), ran_numbers(:, 2));
```
Exercise 2

• Plot $\sin(x)$ and $\cos(x)$ on the same plot for $x=[0, 2\pi]$. Add a legend and labels!

• Step 1: Set up independent variable.

• Step 2: Compute functions.

• Step 3: Plot.

• Step 4: Add legend/labels.
Exercise 2 Solution

1 %Set up axes.
2 \textit{x} = 0:0.1:2.*\textit{pi};
3
4 %Plot
5 \texttt{plot(x, \textit{sin}(x), 'r')};
6 \texttt{hold on;}
7 \texttt{plot(x, \textit{cos}(x), 'b')};
8
9 %Add title/labels/legend.
10 \texttt{title('Plots of cos(x) and sin(x)' )};
11 \texttt{xlabel('x')};
12 \texttt{ylabel('f(x)')};
13 \texttt{legend('sin(x)', 'cos(x)')};}
Plots of Multivariable Functions

- MATLAB can create multiple plots suitable for multivariable functions.

Surfaces

Contours
Surfaces

• Create surfaces using the *surf* function.

\[
\text{surf}(x, y, \text{data})
\]

• Where:

  • \( x \) and \( y \) define the axes for the plotted data.
    \[
    \text{size(data)} == [y, x]
    \]

  • \( \text{data} \) is the 2D array to be plotted (Note: \( \text{data} \) contains no info as to which values of \( x \) and \( y \) correspond to each data point. That is why \( x \) and \( y \) are required.)
Surfaces

• Why is \texttt{size(data)} == [y, x] and not [x, y]?

• Recall that array indices are in the order (row, column).

\begin{center}
\begin{tabular}{ccc}
(1,1) & (1,2) & (1,3) \\
(2,1) & (2,2) & (2,3) \\
(3,1) & (3,2) & (3,3) \\
\end{tabular}
\end{center}

This is opposite of normal (x,y) coordinates!

• Therefore, the order must be swapped.
Surfaces

• But this poses a problem!

• Often, when we input data to MATLAB, we treat arrays as if the order of indices is \((x, y)\) and not \((y, x)\).

• In this case, the **data** must first be transposed before plotting.
Transpositions

• MATLAB provides a transpose operator for this purpose: ' .

• Simply add the prime after an array to transpose it

```matlab
1 array_transposed = array';
```

• This reverses the dimensions so the array can now be plotted.
Contours

• Contours are plotted in the same way as surfaces; they simply use a different keyword.
  
  • For regular contours: \textit{contour}
  
  • For filled contours: \textit{contourf}
Color Bars

• Adding color bars to your plot is very simple.

• Just use the *colorbar* command!
meshgrid and ndgrid

• meshgrid/ndgrid generates a \textit{mesh} to allow calculations of multivariable functions.

• \textit{meshgrid} transposes the axes for you; therefore, you don’t have to transpose the array afterward.

• \textit{ndgrid} does not transpose the axes. Use this if you want your output data to have dimensions ordered \((x, y)\).
meshgrid vs. ndgrid

\[
\begin{align*}
\text{>> } x &= 1:3; \\
\text{>> } y &= 4:6; \\
\text{>> } [X \ Y] &= \text{meshgrid}(x, y) \\
X &=
\begin{pmatrix}
1 & 2 & 3 \\
1 & 2 & 3 \\
1 & 2 & 3
\end{pmatrix} \\
Y &=
\begin{pmatrix}
4 & 4 & 4 \\
5 & 5 & 5 \\
6 & 6 & 6
\end{pmatrix} \\
\text{>> } [X \ Y] &= \text{ndgrid}(x, y) \\
X &=
\begin{pmatrix}
1 & 1 & 1 \\
2 & 2 & 2 \\
3 & 3 & 3
\end{pmatrix} \\
Y &=
\begin{pmatrix}
4 & 5 & 6 \\
4 & 5 & 6 \\
4 & 5 & 6
\end{pmatrix}
\end{align*}
\]
meshgrid vs. ndgrid

- Typically, we use meshgrid if we are just generating data to be plotted.

- We typically use ndgrid if we have pre-existing data or if we are performing other calculations on the data first.

- This is not necessarily the case! The function to use depends on how the data is stored.
meshgrid and ndgrid

• We don’t always need to use meshgrid or ndgrid.

• The functions only need to be used if we are calculating a function.

• If we already have data to contour, then we only need to define the axes.
Example

• Plot \( \sin(xy) \) from \( x,y = [0, 2\pi] \)

1 \%Set up axes.
2 \( x = 0:0.1:2.*\pi; \)
3 \( y = 0:0.1:2.*\pi; \)
4
5 \%Use meshgrid to generate the mesh.
6 \([X Y] = \text{meshgrid}(x, y); \)
7
8 \%Generate the surface.
9 \text{surf}(X, Y, \sin(X .* Y));
Graphics Handles

- Graphics handles provide a means to change properties of the figure and the axes.
- For example, figure height/width or axes labels.
- `gca` - Get current axes
- `gcf` - Get current figure
Graphics Handles

- Use `get(gcf)` or `get(gca)` to get a list of the properties of the current figure or current axes, respectively.

- Use `set(gcf_or_gca, 'property', new_value)` to change values.

- Example: `set(gca, 'XLim', [0 10]);`

- Sets the x-axis to an interval from 0 to 10
Graphics Handles

• Handles can be saved to variables for easy reuse.

• Example:

```matlab
1 ax = gca;
2 get(ax);
3 set(ax, 'XLim', [0 10]);
```
Exercise 3

- Plot the function \( z = x^2 + y^2 \) as a surface and filled contour for \( x, y = [-5, 5] \) with \( dx, dy = 0.1 \).

- Step 1: Define axes.
- Step 2: Use meshgrid/ndgrid.
- Step 3: Calculate the function.
- Step 4: Plot.
Exercise 3 Solution

1 %Define axes.
2 x = -5:0.1:5;
3 y = -5:0.1:5;
4
5 [X Y] = meshgrid(x, y);
6
7 f1 = figure;
8
9 surf(X, Y, X.^2 + Y.^2);
10
11 f2 = figure;
12
13 contourf(X, Y, X.^2 + Y.^2);
File I/O

- MATLAB has many different functions for file I/O.
- Some are modeled from C and some from Fortran.
- See the documentation for details on all of these.
- The simplest function, however, is `importdata`. 
importdata

• If you have data in a text file, and it is uniformly arranged (i.e. no headers or other text), *importdata* is the way to go.

• Example:

```
1, 3, 5,
7, 9, 11,
13, 15, 17,
```

```
>> data = importdata('indatatest.txt')

data =

1   3   5
7   9  11
13  15  17
```
Saving MATLAB Workspaces

• To save all current variables to disk:
  • `save insert_output_filename`

• To load variables from disk:
  • `load insert_input_filename`
Saving MATLAB Plots

• To save MATLAB plots, there are two options.

• The GUI has a save function, or…

• Use the *print* command.

• Example:

  • *print -dpdf output_file.pdf*
Other File Output Methods

- MATLAB also supports C-style `fprintf` and `fgets`.
- They look very similar to their C counterparts, but they have a few differences due to the lack of pointers in MATLAB.
- They use file handles instead.
- See documentation for details.
NetCDF I/O

• NetCDF is one of the most commonly used data formats in the atmospheric sciences.

• Used for WRF output.

• Also used for some sounding data, and many other datasets.
NetCDF I/O Functions

• $fid = \text{netcdf.open('filename')}$ - Open file and assign it to handle $fid$.

• $\text{varid} = \text{netcdf.inqVarID}(fid, 'var\_name')$ - Get variable ID.

• $\text{var} = \text{netcdf.getVar}(fid, \text{varid})$ - Get variable from file.

• $\text{netcdf.close}(fid)$ - Close file.
Example

P contains the data!

```matlab
1 %Open NetCDF file.
2 fid = netcdf.open('wrfout.nc');
3
4 %Get perturbation pressure.
5 P_id = netcdf.inqVarID(fid, 'P');
6 P = netcdf.getVar(fid, P_id);
7
8 %Close file.
9 netcdf.close(fid);
```
Exercise 4

• Create a contour of the pressure at the lowest model level at the initial time (Note 4-D data).

• Step 1: Get P and PB from NetCDF.

• Step 2: Sum them (perturbation + base).

• Step 3: Contour (You don’t need meshgrid/ndgrid, but you will need to transpose!)
1 %Open NetCDF file.
2 fid = netcdf.open('wrfout_d01_2001-01-09_00:00:00');
3
4 %Get perturbation pressure.
5 P_id = netcdf.inqVarID(fid, 'P');
6 P = netcdf.getVar(fid, P_id);
7
8 %Get base state pressure.
9 PB_id = netcdf.inqVarID(fid, 'PB');
10 PB = netcdf.getVar(fid, PB_id);
11
12 %Close file.
13 netcdf.close(fid);
14
15
16 P_tot = P + PB;
17
18 contourf(P_tot(:, :, 1, 1)');
19 colorbar;
Optimizations

• How do we speed up code?

• It’s actually a very complicated question, and it has a whole field of computer science devoted to it.

• However, there is one simple thing that will make some code run faster in MATLAB.
Resizing Variables in Loops

• Avoid resizing variables in loops!

• Example:

```matlab
1 for i=1:15
2    array(i) = 2;
3 end
```

• If array is not previously defined, this continues increasing the size of array in the loop.

• Instead, preallocate some memory.
Preallocating Memory

• Use either *zeros* or *NaN* to preallocate memory.

• Example:

```matlab
1 array = NaN([15 1]);
2 for i=1:15
3    array(i) = 2;
4 end
```

• Why does this matter?
Back Under the Hood

• Because MATLAB is written in C!

• Every time an array’s size is increased, MATLAB must allocate a new array, copy the data, and free the old array.

• This takes time!

• Preallocating prevents this constant shuffle of memory.
Acknowledgements

- Dr. Gretchen Mullendore
- Dr. Dave Delene