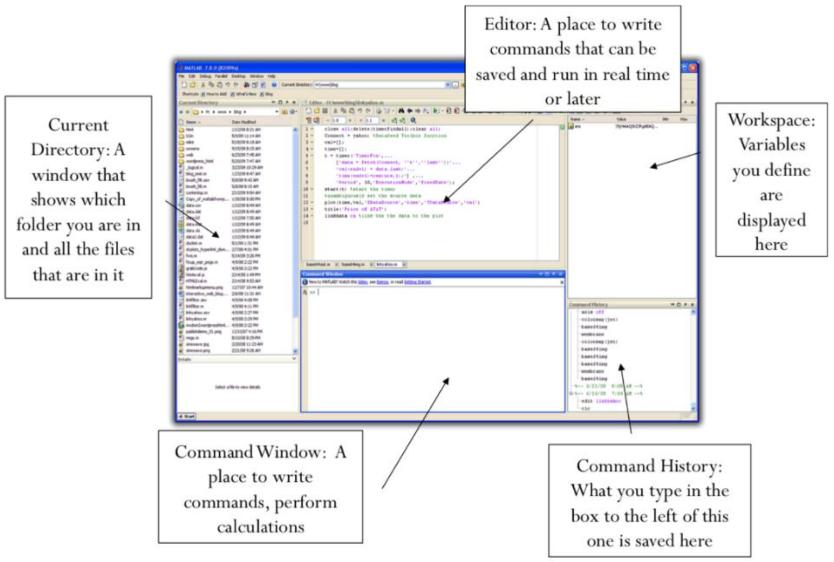
MATLAB Introduction

Accessing Matlab

- Available on all EWS machines
- Access through Citrix Reciever (https://it.engineering.illinois.edu/ews/lab-information/remote-connections/connecting-citrix)
- Purchase through Mathworks: Base cost: $49, Suite: $99 (https://www.mathworks.com/store/link/products/student/new?s_tid=ac_buy_sv_cta)

Matlab Interface

When you open Matlab you will see the following (note that the windows within Matlab maybe arranged slightly differently)


Editor: A place to write commands that can be saved and run in real time or later
Current Directory: A window that shows which folder you are in and all the files that are in it
Workspace: Variables you define are displayed here
Command Window: A place to write commands, perform calculations
Command History: What you type in the box to the left of this one is saved here

Calculations can be done either in the "Command window" or the "Editor". The command window is a temporary place to see your commands, the commands themselves are saved in the "Command history", but not the answers. The solution to the calculation is displayed in the "Command Window" and stored in the "Workspace". If you do not assign a variable name to the calculation, the answer is stored in the default variable name `ans`

The Basics

MATLAB is a high-level interpreted language, and uses a “read-evaluate-print” loop: it *reads* your command, *evaluates* it, then *prints* the answer. This means it works a lot like a calculator:

```
>> 1+2
ans =
    3
```

Here, it read the command 1+2, evaluated it to 3 and printed that. It also stores the answer from that in the variable `ans` so that you can refer to it in the next command if you want.

```
>> 1.5^2 + 2.5^2
ans =
    8.5000
>> sqrt(ans)
ans =
    2.9155
```

Be careful: after executing the second line, `ans` now has a new value. This also shows the syntax for a power (a^b is evaluated using `a^b`) and square root using the `sqrt` function. Much of the syntax follows mathematical syntax that you would expect: `+, -, *, /, ^, sqrt, etc`.

Variable Definition and Statement Suppression

You can create your own variables, and assign them values using =

```
>> a = 1 + 2
a =
    3
```

Once declared, the variable can be used in other statements:

```
>> b = a - 5
b =
    -2
```

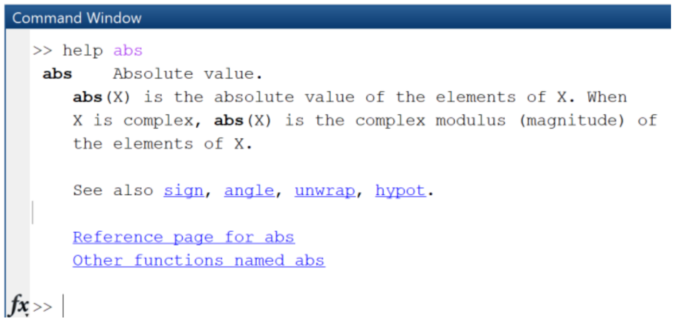
Statements can be suppressed with the `;` symbol so that the result is not printed to the command window. The result can be viewed by calling the variable name or by opening the variable from the workspace variables window.

```
>> c = 1.3^2 - 1;
>>
```

Keyboard Shortcuts

Help

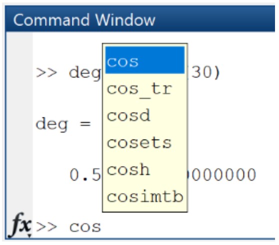
If you're not sure what a command does, type help command name. If you can't remember if cos uses radians or degrees, then help cos will tell you. You can also search the documentation in the upper right hand corner.



```
>> help abs
 abs    Absolute value.
    abs(X) is the absolute value of the elements of X. When
    X is complex, abs(X) is the complex modulus (magnitude) of
    the elements of X.

    See also sign, angle, unwrap, hypot.

    Reference page for abs
    Other functions named abs

fx >>
```

Tab completion

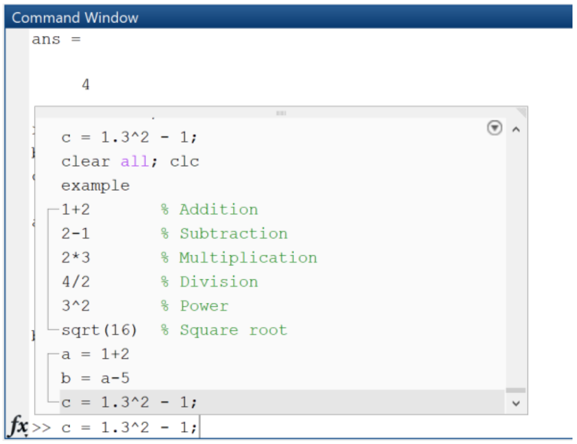
If you're typing a command like `cos`, when you hit the TAB key, it will give you a list of commands that start with the letters `cos`.



```
>> deg   30)

deg =

   0.5   000000

fx >> cos
```

Command history

You can use the up and down arrows to move through previous commands that you've entered. You can then press ENTER to rerun that command exactly, or move the cursor left and right in the line and make edits (e.g., if you made a mistake you need to correct). This is useful if you've made an error with a variable value and need to reevaluate an expression.



```
ans =

     4

c = 1.3^2 - 1;
clear all; clc
example
1+2        % Addition
2-1        % Subtraction
2*3        % Multiplication
4/2        % Division
3^2        % Power
sqrt(16)   % Square root
a = 1+2
b = a-5
c = 1.3^2 - 1;
fx >> c = 1.3^2 - 1;
```

More Common Functions

There are a variety of functions in Matlab, including absolute value, `abs`, and most trigonometric functions such as: `sin, cos, tan`. You can also use `pi` instead of typing out an approximation such as 3.14, but *be careful*: it can be overwritten, do not use it as a variable.

It is also important to note that default trigonometric functions are in radians, adding a `d` at the end of the expression will ensure it is evaluated in degrees.

```
>> rad = sin(30)
rad =
  -0.988031624092862
>> deg = sind(30)
deg =
   0.500000000000000
```

If you want to switch between radians and degrees before you perform evaluations you can use `rad2deg` or `deg2rad.` You can also find the inverse of trigonometric functions: `asin, acos,` and `atan`. These can be evaluated in degrees in the same way as before.

In TAM 212, it may be useful to know the function `atan2`, use `help` function to learn more.

Vectors and Matrices

MATLAB (MATrix LABoratory) is optimized for working with vectors and matrices. As such, it has a nice syntax for making vectors and matrices easily, using the `[]` syntax

```
>> A = [1 2]
A =
    1     2
>> B = [3,4]
B =
    3     4
>> M = [5 6 ; 7 8]
M =
    5     6
    7     8
```

You can separate entries in a vector using a space or a comma (and can mix and match: `[1 2,3]`),and you separate the rows in a matrix using a semicolon. You can then access the values inside a vector (v_i) or matrix (M_{ij}) with `()`

```
>> B(1)
ans =
    3
>> M(1,1)
ans =
    5
```

The indices follow row-column order, so that M_{ij} is `M(i,j)`, and the indices begin at 1. In addition to accessing entries, you can also assign values.

```
>> M(2,1) = 10
M =
     5      6
     10      8
```

If you want a row or column vector out of a matrix, you use the `:` operator `;` then `M(1,:)` gives you the row M_{1j}, while `M(:,1)` gives you the column M_{i1}.

```
>> M(1,:)
ans =
     5      6
>> M(:,1)
ans =
     5
     10
```

You can do things like get the dot product of $\vec{a}$ and $\vec{b}$ with `dot(a,b);` you can get the cross product $\vec{a}\times\vec{b}$ with `cross(a,b)`. You can get the transpose of a vector or matrix with the `'` operator

```
>> B'
ans =
     3
     4
>> M'
ans =
     5    10
     6     8
>> M*B'
ans =
     39
     62
```

Note: the transpose of a *row vector* (like `[1 2]`) is a *column vector* (like `[1;2]`). To right-multiply a vector times a matrix (like $M \cdot \vec{v}$), the vector needs to be a column vector. You can also use this to take dot-products if you want: if `A` and `B` are row vectors, then `dot(A,B)` is the same as `A*B'`.

For a matrix, you can access the determinant with `det(M)` and the trace (sum along the diagonal) with `trace(M)`

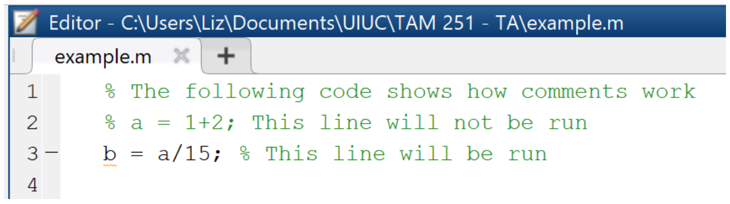
```
>> det(M)
ans =
     -20
>> trace(M)
ans =
     13
```

Creating a Script

Scripting is a valuable tool in Matlab. Instead of entering each command line-by-line in the command window, a script allows all lines to be saved and run multiple times without having to scroll through the "command history".

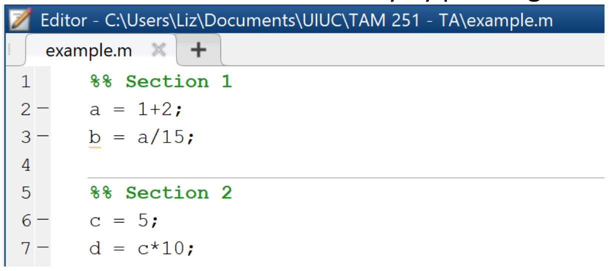
Commenting

If you want to keep track of what you are doing or do not want a line of code to run, you can use the % to comment out a line. This can be done to an entire line, or just a portion of the line:



```
% The following code shows how comments work
% a = 1+2; This line will not be run
b = a/15; % This line will be run

```

Two `%%` symbols indicates a section header. This is useful for separating and organizing your code. These sections can also be run individually by pressing the “Run Section”.



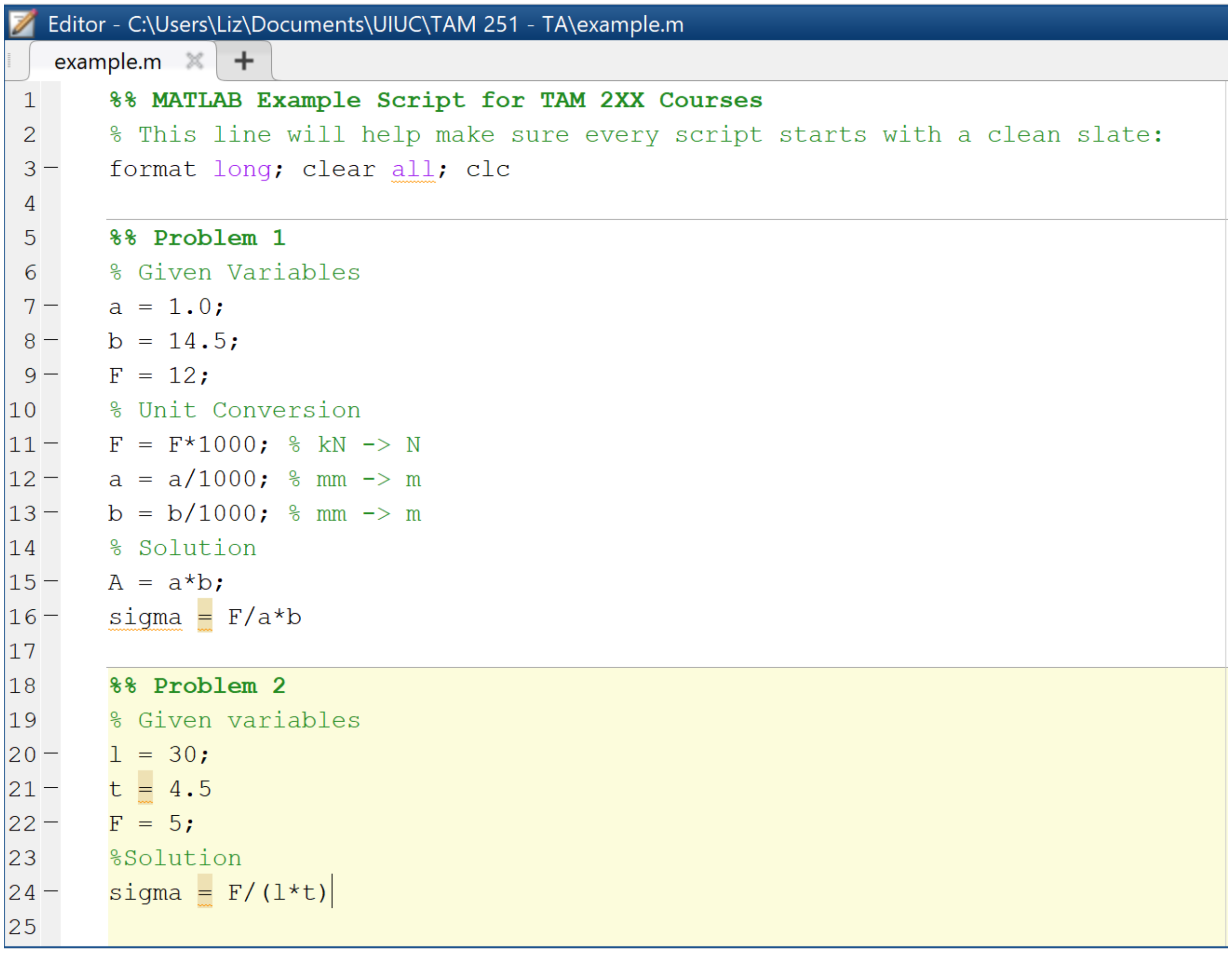
```
%% Section 1
a = 1+2;
b = a/15;

%% Section 2
c = 5;
d = c*10;
```

TAM 2XX Specific Tips

Generate a script for each homework assignment or quiz. Separate each problem with two `%%` symbols and copy and paste your given variables underneath the problem header. If a problem requires a unit conversion, you should do this conversion after pasting the given variables. This will allow you to flip through multiple variations of the problem without having to change the given variables every time.

- *Parenthesis:* You should be obsessive about this**.** If you are raising something to a power, make sure the entire base is wrapped in parenthesis as well as the power. If you are dividing, make sure the numerator and denominator are both individually wrapped in parenthesis. Your code should look like you do not trust Matlab's built-in order of operations.
- *Units:* Do not trust the Matlab entries given in a problem. It is your responsibility to check the units in your problem. If everything is given in N and m, but the answers ask for kN and mm, then it is your responsibility to convert the answer. The Matlab input might also be missing a 10^{-6} or might give the units in m when the problem statement was in mm, so again, be careful about checking these things before submitting your answer.
- *Format long:* PrarieLearn grades homework using this setting. If you are not using a script, be sure to type this into the "Command Window" before solving your problem. If you do generate a script, as you should, put this at the top so that it runs every time.

An example script is provided on the following page:


Editor - C:\Users\Liz\Documents\UIUC\TAM 251 - TA\example.m

example.m

```
%% MATLAB Example Script for TAM 2XX Courses
% This line will help make sure every script starts with a clean slate:
format long; clear all; clc

%% Problem 1
% Given Variables
a = 1.0;
b = 14.5;
F = 12;
% Unit Conversion
F = F*1000; % kN -> N
a = a/1000; % mm -> m
b = b/1000; % mm -> m
% Solution
A = a*b;
sigma = F/a*b

%% Problem 2
% Given variables
l = 30;
t = 4.5
F = 5;
%Solution
sigma = F/(l*t)

```

Solving (Linear) Equations

We can use MATLAB to solve equations, including systems of equations. For our purposes, we will almost exclusively deal with linear equations. To start, we define a 3x3 matrix `A` and a 3x1 matrix `b`:

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

     1     2     1
     2     2     3
    -1    -3     0
>> b = [0 ;3; 2]
b =

     0
     3
     2
```

This is a linear problem in the matrix form $Ax = b$. We can solve it using the `linsolve` function:

```
>> x = linsolve(A,b)
x =
     1
    -1
     1
```

Alternatively, you can use the backslash operator:

```
>> x = A\b
x =
     1
    -1
     1
```

Mathworks Website

Anything not covered in this brief introduction can be found on the Mathworks website:

https://www.mathworks.com/

Things that may be useful in the future: breakpoints, differentiation, integrals, functions, plotting, for loops, while loops, timing, etc.

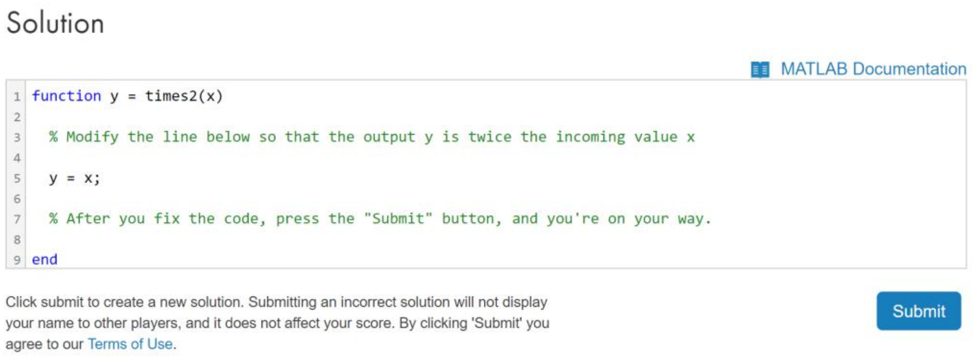
Cody

Mathworks provides a free training tool called Cody. It does not require a license for Matlab, so you can practice without going to an EWS machine, setting up Citrix, or paying yourself.

https://www.mathworks.com/matlabcentral/cody/

Got to the link above, sign up for an account, and look through the "Problems" tab. Sorting by "Solvers High-Low" will show you the problems with the most solutions, the easiest ones.

In each problem, they explain what you have to do and ask you to click the "Solve" button. This will open up a Matlab function in your browser as shown in the image below. It will provide you with code, however, you will have to change the lines in the function to solve the problem.



```
function y = times2(x)

  % Modify the line below so that the output y is twice the incoming value x

  y = x;

  % After you fix the code, press the "Submit" button, and you're on your way.

end
```

Click submit to create a new solution. Submitting an incorrect solution will not display your name to other players, and it does not affect your score. By clicking 'Submit' you agree to our Terms of Use.