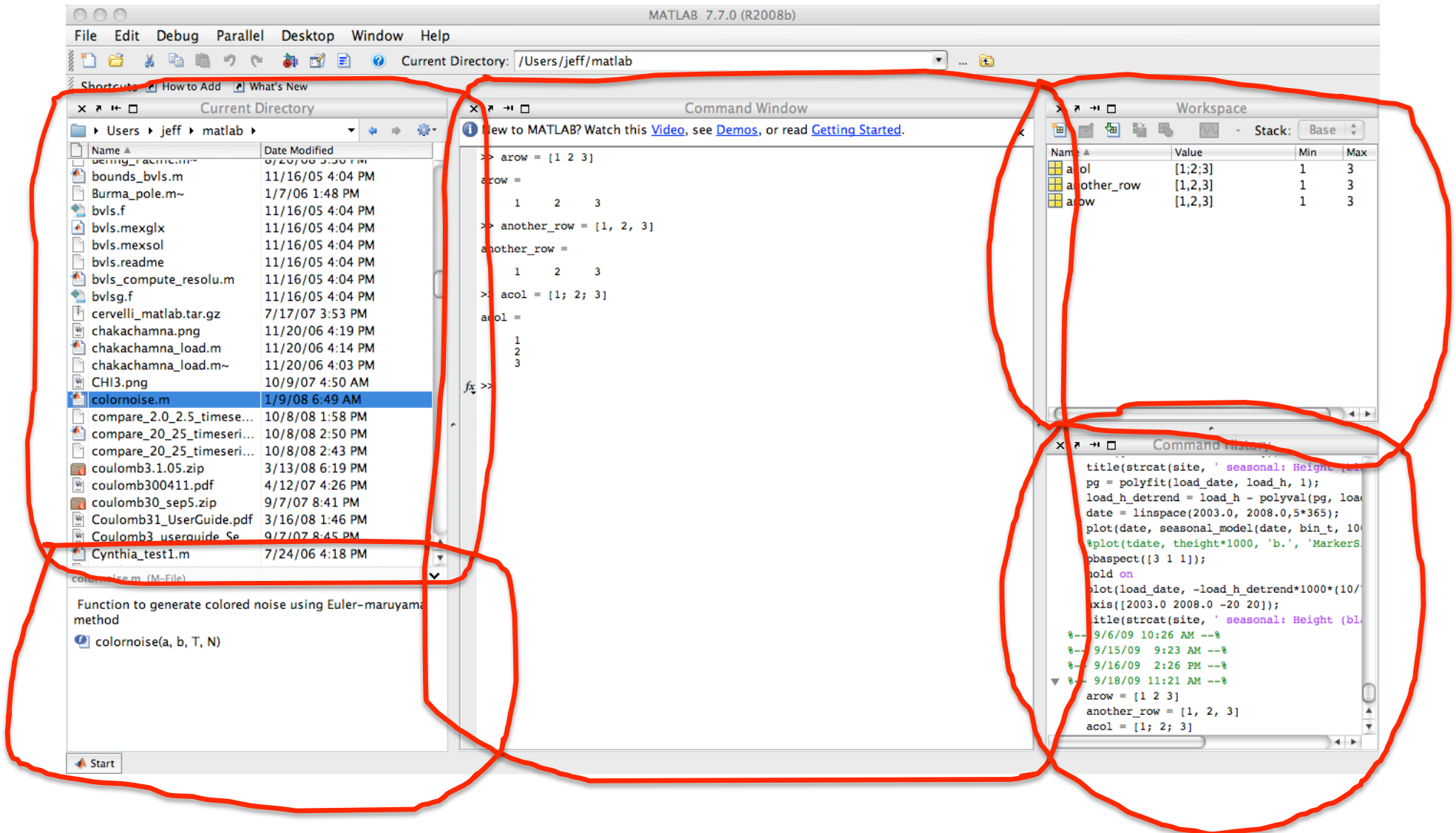


MATLAB 1

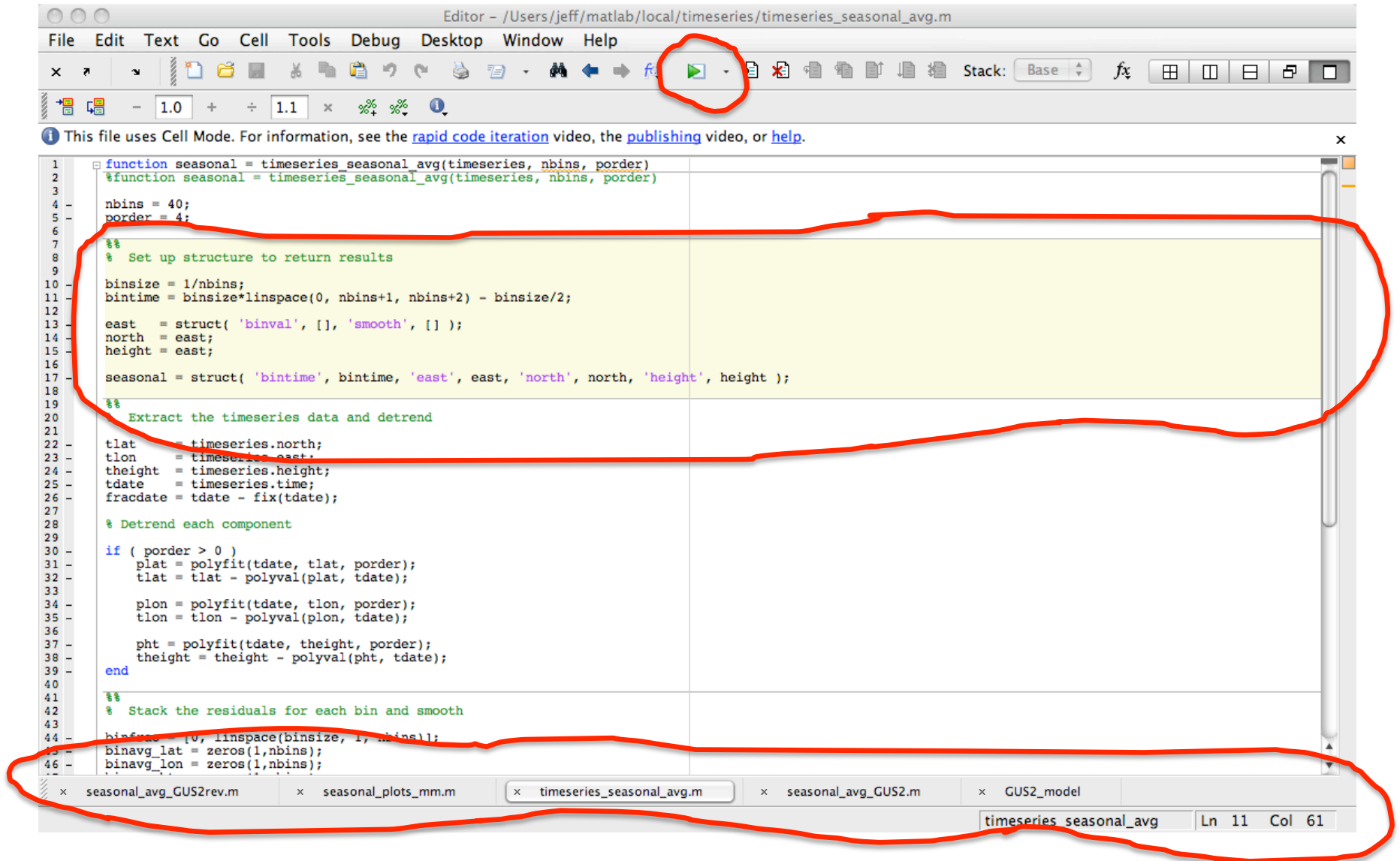
Jeff Freymueller

September 24, 2009

MATLAB IDE

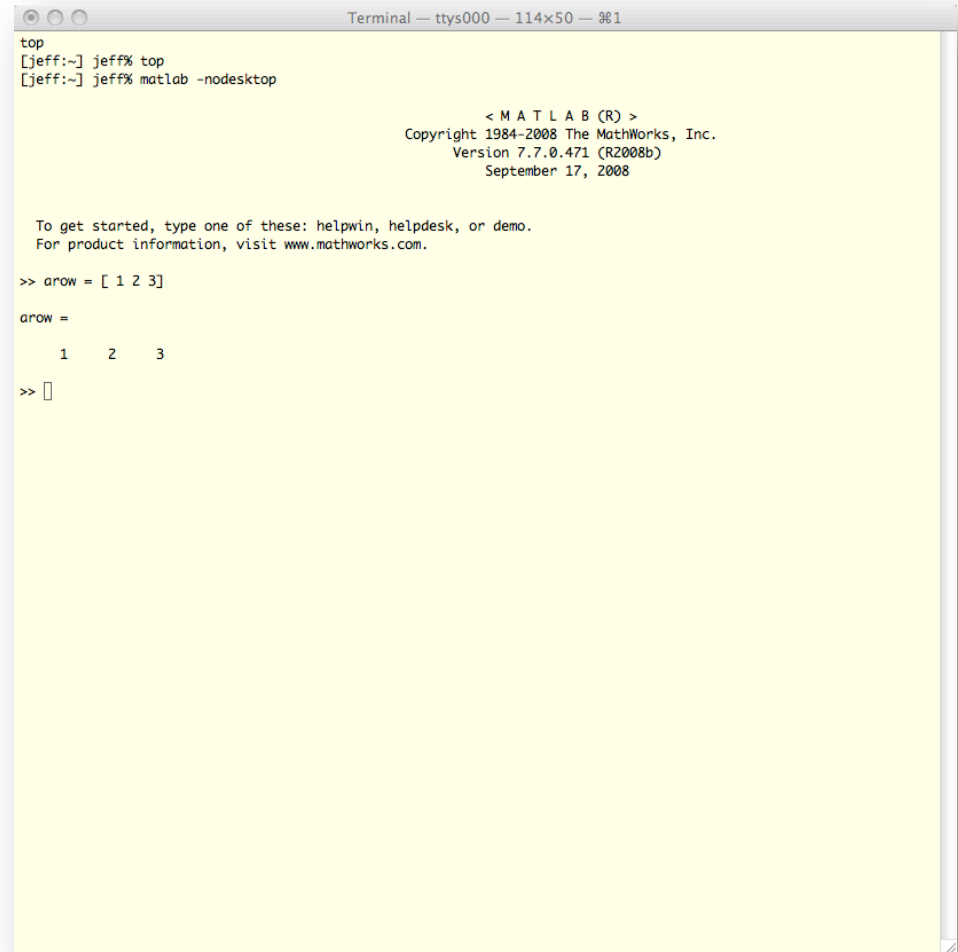


MATLAB Editing Window



We don't need no steenkin' GUI

- You can also use MATLAB without the fancy user interface, just a command window.
- Why?
 - You can run MATLAB on a remote machine (fast)
 - You can run a specific MATLAB program from a command line, or even automatically from a script
 - Some people like it better that way.

A terminal window titled "Terminal - ttys000 - 114x50 - 81" showing the execution of MATLAB. The user runs "top" and "jeff% top", then "jeff% matlab -nodesktop". The terminal displays the MATLAB startup screen, including the version "7.7.0.471 (R2008b)" and the date "September 17, 2008". The user enters the command ">> arow = [1 2 3]" and the terminal outputs "arow = 1 2 3". The prompt ">> []" is shown at the bottom.

```
top
[jeff:~] jeff% top
[jeff:~] jeff% matlab -nodesktop

< M A T L A B (R) >
Copyright 1984-2008 The MathWorks, Inc.
Version 7.7.0.471 (R2008b)
September 17, 2008

To get started, type one of these: helpwin, helpdesk, or demo.
For product information, visit www.mathworks.com.

>> arow = [ 1 2 3 ]
arow =
     1     2     3
>> []
```

MATLAB help

- MATLAB has a lot of help functions.
 - First: “help matlab”
 - General: “help command”
- There is also a graphical help browser. Access it from the Help menu or using “doc command”
- When you write your own functions, you can add comments that will appear when you type “help my_function”

```
>> help if
IF Conditionally execute statements.
The general form of the IF statement is
```

```
IF expression
statements
ELSEIF expression
statements
ELSE
statements
END
```

The statements are executed if the real part of the expression has all non-zero elements. The ELSE and ELSEIF parts are optional. Zero or more ELSEIF parts can be used as well as nested IF's. The expression is usually of the form `expr rop expr` where `rop` is `==`, `<`, `>`, `<=`, `>=`, or `~=`.

```
Example
if I == J
    A(I,J) = 2;
elseif abs(I-J) == 1
    A(I,J) = -1;
else
    A(I,J) = 0;
end
```

See also RELOP, ELSE, ELSEIF, END, FOR, WHILE, SWITCH.

Reference page in Help browser
`doc if`

Some Good Habits

- You can copy lines from the command window or command history and paste them into an editing window to save them.
 - If you do anything important by typing into the command window, save your commands as a script for future reference and future use!
- You can write several lines of code in the editing window, and then copy and paste them into the command window to try them out (just hit return after the paste)
 - Test out small, manageable pieces of your code!
- The run button in the editing window is really handy for testing the whole script or program.
 - The editing window has some great debugging features as well. After you have mastered the basics, learn how to use the debugger.

Variables in MATLAB

- MATLAB treats all variables as arrays (vectors or matrices). Program's roots are in linear algebra.
 - Scalars are just 0-dimensional arrays (single values)
 - Values assigned using = : "a_row = [1 2 3]"
 - MATLAB displays the answer after every command, unless you end the command with a semi-colon.
- Most of the time, you don't need to worry about the variable type, MATLAB handles it invisibly.
- But you do have to remember that there is a difference between a row vector and a column vector.

Matrix, Row Vector, Column Vector

- A matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- A row vector

$$[1 \ 2 \ 3]$$

- A column vector

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

$$[1 \ 2 \ 3] \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = 1 + 4 + 9 = 14$$

– They are not the same:

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} [1 \ 2 \ 3] = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{bmatrix}$$

Bracketology

- MATLAB uses three different kinds of brackets, parentheses, braces, all meaning different things
- [] Square brackets
 - Vectors, arrays and matrices are contained inside
- () Parentheses
 - Access a particular element of an array by putting the indices inside parentheses
- { } Braces or Curly brackets
 - Like parentheses, except for cell arrays

Array vs. Cell array

- Cell arrays behave a bit differently than regular arrays.
 - Every element of a regular array must be the same kind. Not so for cell arrays. Each element of a cell array is a container that can hold any one thing.
 - Cell arrays are really useful for strings, and also are returned by some functions that read files.
 - You can do numerical operations across regular arrays, but not cell arrays.
- You'll see more examples as we go along.

Special “Numbers”

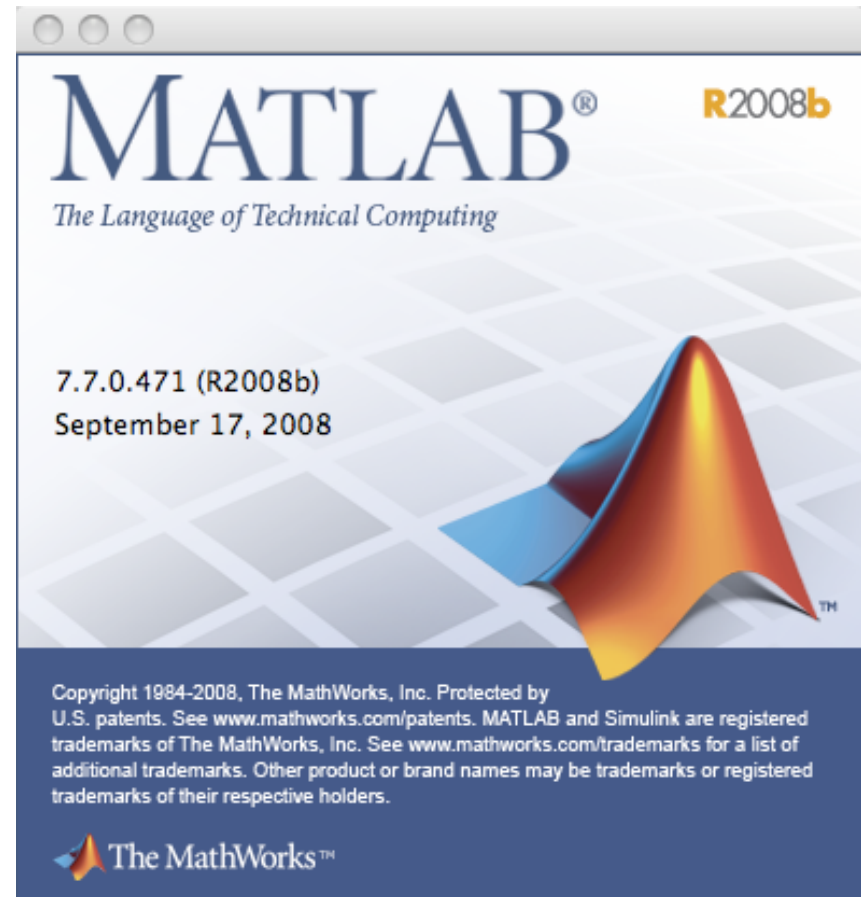
- MATLAB handles complex numbers seamlessly
 - $5 + \text{sqrt}(-1) = 5.0000 + 1.0000i$
 - If you do not define a variable “i”, MATLAB will use that symbol for $\text{sqrt}(-1)$.
- Not a Number (NaN) is a very handy “number”
 - Use NaN to represent missing values
 - DO NOT use “9999” for missing values!
 - Any arithmetic operation with NaN produces NaN
 - The function “isnan” finds all the NaNs in its argument
 - `idx = isnan(has_a_nan)`

Functions

- What is a function?
 - A set of mathematical operations that take some input values (“variables”) and produce an output value.
 - Remember every value can be a scalar or a vector or matrix
 - A little black box of code that takes some inputs and produces one or more outputs
- Examples: `sqrt`, `isnan`, `find`, `eye`, `zeros`, `ones`, `size`, `inv`

Let's Explore MATLAB for a while

- Assigning variables
- Help
- Features of the IDE
- Some basic functions
- MATLAB is vectorized!
- Load and save. See also:
 - xlsread, xlswrite
 - csvread, csvwrite



Defining your own functions

- Save your function in its own .m file
- Start with a function declaration
 - `output = function my_func(input1, input2)`
 - `[out1, out2] = function two_out(in1, in2, in3)`
- End with “return” (not required, but good habit)
- You can use any number of inputs and any number of outputs
 - If you change the values of the input variables inside the function, those changes are lost when you exit
 - The arguments to the function are ***passed by value***
 - Some languages (like fortran) pass arguments by reference, so you can change any variable passed to a subroutine/function
- Any comment lines immediately afterward are used by help

Scope of variables

- The function exists in its own separate little space. It interacts with its calling routine only through the arguments it is passed and the values it returns.
 - It can only modify the values it returns, and not the arguments.
- Variables used inside the function are created when the function is called, and thrown out when it is done.
- You can re-use variable names inside a function that are also used somewhere outside.

More of your own functions

- As long as MATLAB knows where to find it, your function becomes just as much a part of the language as the built-in functions
 - Build up a set of your own useful functions!
 - MATLAB will always find functions in the current directory
 - Use `addpath` to specify other directories where it should look for your functions
 - Assemble small pieces into bigger tools!
- Be sure to use sensible names!

Exercises

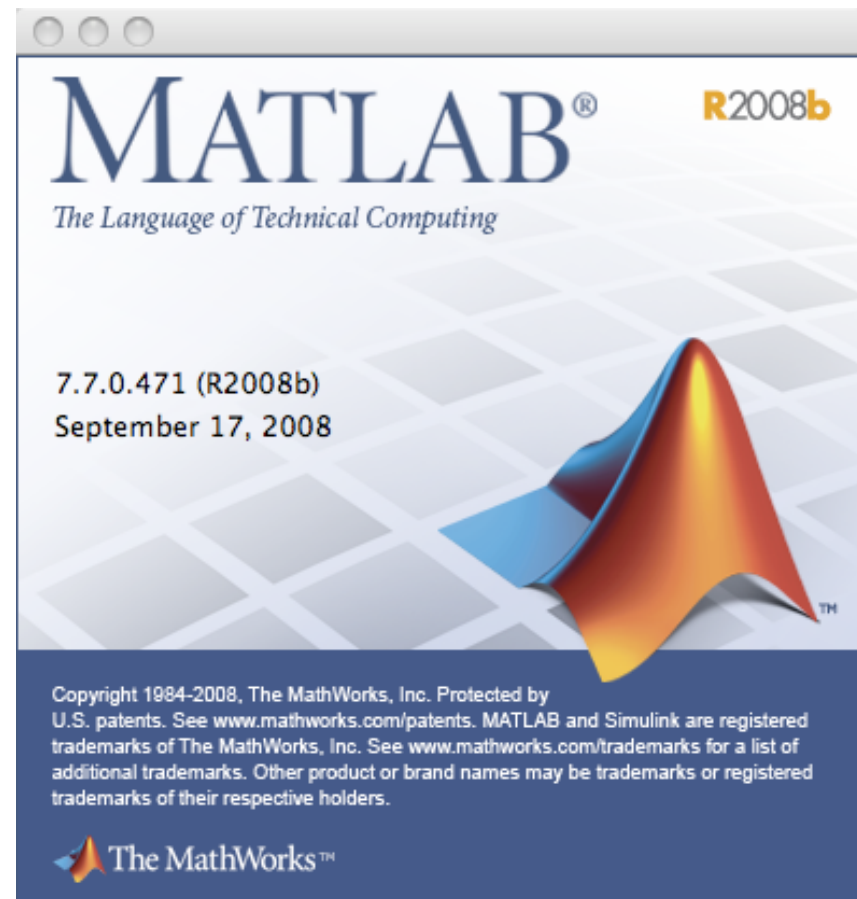
- 3.1. Explore the help function, and start up the help browser from the Help menu, and using “doc”
- 3.2. Make a version of “class_hasbugs.m” that runs without errors.
 - Save it as “class_nobugs_yourname.m” and email it to me.
- 3.3. Modify the function class_new_function. Define $x = \text{linspace}(1, 10, 10)$. Then make a plot:
 - `plot(x, class_new_function(x))`
- 3.4. Define a new function of your own, in its own source file. Make it have reasonable values for a range of inputs. Plot it.

Structures

- Suppose you have a set of variables that go together. A structure (or struct) lets you package them together, making it easy to keep track of things.
- A ***struct*** has one or more ***fields***, which are named, and each can store a value (or vector, or array, or a struct, or ...)
- You define a struct by naming its fields and assigning a value to each (or use [] for an empty value).
- Access a field like this: weather.year, weather.temp(5)
 - Or getfield(weather, “year”) or getfield(weather, name) where name is a variable.

Let's Explore MATLAB for a while

- Structs
 - Let's look at the "timeseries" struct
 - This is an example of packaging a variety of related information into one "container"
 - If I want to store multiple timeseries, I could make a cell array and store each timeseries in one element.



Struct Example 2

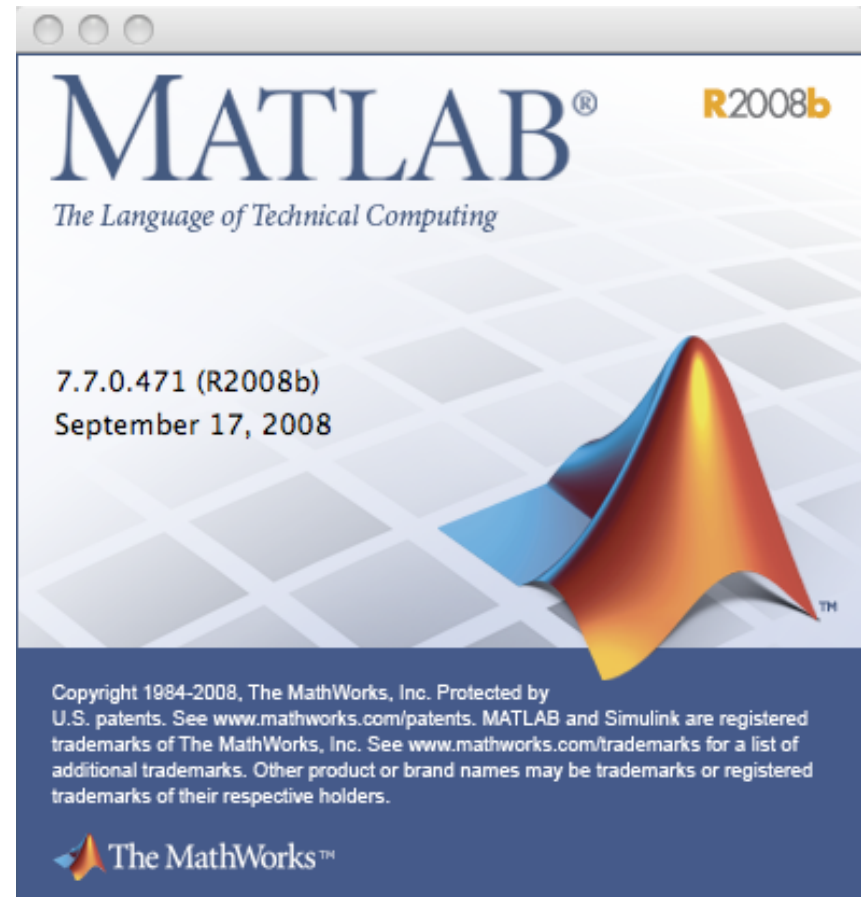
- Suppose you have a data set that has a number of arrays of the same size. There might also be some other information.
 - For each day of year, you have temperature, pressure and humidity readings
- What are the fields?
 - Year (scalar)
 - Day_of_year (array)
 - temperature, pressure, humidity (arrays)
- In this case, it makes sense to make an array of structs, which will allow you to do some numerical operations on the elements, like finding the minimum, maximum or mean.

Example 2

- The easy way to make an array of structs is to put all the numerical values into cell arrays and use these to define the struct.
 - Note: some input routines give you the data in cell arrays already!
 - You'll get an error message if the cell arrays are different sizes.
- `weather = struct('year', 2009, 'day_of_year', {225, 230, 235}, 'temp', {64, 69, 58}, 'pressure', {30.1, 30.5, 29.5}, 'humidity', {0.64, 0.34, 0.88});`
- This produces a 3 element array of structs. You can access the values in different ways:
 - `weather(2).temp`
 - `median([weather.pressure])`

Let's Explore MATLAB for a while

- Structs
 - Let's look at the "weather" struct
 - This is an example of packaging several related variables, like a set of multi-variable observations



Summary

- MATLAB has several different types of variables
 - Scalars, vectors matrices
 - Strings
 - Structs
 - Cell arrays
- MATLAB provides a handy environment for writing code.
- MATLAB is actually very fast, especially for vectorized operations.