Gettin' Funky: Outline

• Review of last class solutions
• Getting missing data right: iterations
• Iteration vs. matrix operations
• Laundry list of loops and checks
• Dealing with wind data
• Your first function
• Don't forget the documentation
• What the bleep is wrong?
Review of last class solutions

• We will continue with data from the last class today, so might as well use the solutions to the last class to get everything loaded.

• Take a look at the solutions and run them:
  » addpath z:\Pkg\WFCB\class3
  » edit class3_exercises_solutions
  » class3_exercises_solutions  % to run it

• This will step you through the figures and create most of the variables needed for rest of class (click to advance through figures).

• The addpath line adds that directory to the list of directories matlab uses to find scripts and functions (i.e. m-files).

• Use the path command to look at the current path.
Fixing the missing data correctly

- Last class we replaced missing data with NaN, but we weren't very careful.
- This time, we will be more careful when replacing the missing data from each column.
- We are going to do this three different ways, all of which yield the correct answer, but use different techniques.
- The first, and perhaps most obvious, is using iteration (aka for loops).
- On the next slide I introduce for loops.
Iteration

• Iteration or looping is simply repeating an operation several times, usually with slight differences each time.

• In matlab, iteration is often not necessary because matrix math can be used instead.

• The most useful iteration command in matlab is for:

```matlab
for k = 1:5
    k2 = k.^2;
    disp(['=> k^2 = ' int2str(k2)])
end
```

• This simply assigns to a variable k each of the values in the right-hand side of the equals and then performs all of the commands up to the `end`.

• `disp` displays variable without printing variable name.
• Iteration is often used with the indices of a matrix:

```matlab
» l = (-10:10)';
» l234 = [ l.^2, l.^3, l.^4 ];
» for k = 1:size(l234,2)
    plot( l, l234(:,k) )
    title('Click to continue')
    waitforbuttonpress
end
```

• `for` can also loop over the columns of a matrix:

```matlab
» for k = l234
    plot(l,k) % k becomes columns of l234
    title('Click to continue')
    waitforbuttonpress
end
```
Using iteration to fix missing data

● The raw data for the class should be in the structure `a`.
● The data with `NaN` for “missing data” from last class is in the variable `data2`.

**Exercise:** I want you to use iteration to more carefully replace missing data from the data columns (i.e. not the date columns) of the data.

**Hint:** Take a look at:

```python
» max(a.data)
```

**Exercise:** Is there any data that was incorrectly marked as missing using the simple technique from the last class?
Three solutions to the problem

- I have solved the last exercises three different ways to demonstrate matrix versus iterative approaches.
- Take a look at each:
  » addpath z:\Pkg\WFCB\class4
  » edit class4_iterations_exercises
  » class4_iterations_exercises
- If I were writing this code, I would use the last, array-based method to solve the problem.
- Array-based methods can be much easier to program and use less code.
- Usually, array operations are quicker than for loops that do the same thing, but not in this case. Oh well.
- But sometimes, must use a loop to solve a problem.
Other loops

- Matlab has several ways of looping. `for` is the most useful, but `while` is also sometimes used:

```matlab
ii = 1; k = 1;
while ii
    plot(-10:10, (-10:10).^k)
    title({[int2str(k) '^{th} power'], ...
        'Mouse=continue, Enter=stop'})
    waitforbuttonpress
    k = k+1;
end
```
### if statements

- **if** is used to check if something is true:
  » `n = rand; % Random num between 0 and 1`
  » `if n > 0.5`
    » `disp( 'Above average!' );`
  `end`

- **if** can also be used in combination with **else**:
  » `n = rand; % Random num between 0 and 1`
  » `if n <= 0.25`
    » `disp( 'Bottom quartile' )`
  `elseif n <= 0.5`
    » `disp( 'Second quartile' )`
  `elseif n <= 0.75`
    » `disp( 'Third quartile' )`
  `else`
    » `disp( 'Top quartile' )`
  `end`
Working with wind data

- Our buoy data also has wind direction and speed (columns 5 and 6, respectively).
- **Exercise:** I want you to plot the wind vectors using the `compass` command.
- **Note:** Wind direction is measured with respect to where the wind is coming FROM, but I want you to plot where it is going TO.
- **Note:** Wind direction is measured CLOCKWISE from NORTH in DEGREES. North should be up in your plot.
- **Hint:** The `pol2cart` function will be useful.
- **Hint:** `radians = degrees * pi / 180`
- **Exercise:** Add the mean wind vector on top in a different color.
- **Hints:** Always take the mean of the components, not the magnitude and direction. Watch out for NaN.
I have made the mean vector thicker so it is easier to see.
Making Functions

• Computing the wind components from speed and direction is a useful thing. We are going to create a function that does this for us easily.

• I am providing an example function as a template for how to write your own functions:
  » addpath z:\Pkg\WFCB\class4
  » edit class4_example_func
  » help class4_example_func

• The first line is critical: gives names for input and output variables.

• Notice how the help documentation works.

• Functions can only be used if they are on the current path or in the current working directory (pwd).
Wind components function

• **Exercise:** Make a function that takes in wind direction and speed, as measured by the buoy, and returns the components of the wind vector. Include documentation for the function.

• **Exercise:** Check to make sure that your function returns the same components as you plotted previously.

• **Hint:** Read carefully the NOTES in class4_example_func.m

• **Hint:** If you are having trouble, look at my debugging suggestions in the next slide.

• After you are done, compare with my solution:
  ```
  » edit class4_wind_comp_func
  ```
What to do when it doesn't work!

- A lot of programs won't work the first time. Matlab makes it relatively easy to figure out why.

- Suggestions:
  - Matlab gives good errors. If the commands are in a script or function, you get a line number and a column number.
  - The `keyboard` command can be very useful for stopping a function at some point and looking at what is going on. Use `return` to continue execution.
  - You can also use `dbstop`, etc. or the matlab-gui to set break points (points where the execution stops).
  - The `profile` command can be very useful for finding out why your code is taking so long.