Plotting II, Data types II, Pathing and IO

PNI Summer Internship 2020

Mai Nguyen

Course updates

- I’m starting a new job 7/13, not super flexible
- Working on getting guest lecturers on advanced topics
    - Zaid Zada (research engineer and incoming grad student in Hasson Lab) - Natural Language Processing
    - Sam Nastase (postdoc in Hasson lab) - reproducibility, open science, GitHub
    - Elise Piazza and Cătălin Iordan (postdocs in Hasson lab and Cohen lab) - Classifiers
- Shift some class to new time

Outline

- **HW #3 review - hard coding values, plotting redux, legends, colon operator**
- Plotting II
- Data types II: cell arrays, structs
- Pathing
- Input/Output (IO)

HW #3 Review: Hard coding values

Lots of people wrote code that looks like this:

```
% create data matrix of nans
data = NaN(15, 3, 190);

for i = 1:15
    % load subject data
    load(fullfile('tom_localizer', ['s' num2str(i) '_roidata.mat']));

    % org data
    data(i,:,:) = sub_data;
end
```

“Hard coded” values

- Less readable (e.g. what is 15? what is 3? what is 190?)
- Less flexible (e.g. can’t reuse for dataset with 20 subjects)

HW #3 Review: Hard coding values

Avoid hard coding values as much as possible

```
% Set data params
nSubs = 15;
nROIs = 3;
nTRs = 190;
```

HW #3 Review: Hard coding values

Avoid hard coding values as much as possible

```
% Set data params
nSubs = 15;
nROIs = 3;
nTRs = 190;

% Create data matrix of nans
data = NaN(nSubs, nROIs, nTRs);
```

HW #3 Review: Hard coding values

Avoid hard coding values as much as possible

```
% Set data params
nSubs = 15;
nROIs = 3;
nTRs = 190;

% Create data matrix of nans
data = NaN(nSubs, nROIs, nTRs);

for i = 1:nSubs
    % load subject data
    load(fullfile('tom_localizer', ['s' num2str(i) '_roidata.mat']));

    % org data
    data(i,:,:) = sub_data;
end
```

HW #3 Review: Colon operator

Another example from HW using colon operators:

```
for i = 1:nSubs
    % load subject data
    load(fullfile('tom_localizer', ['s' num2str(i) '_roidata.mat']));

    % org data
    data(i,:,:) = sub_data(:,:);
end
```

- sub_data is a 2D matrix: 1 x 190
- Using colon operator here is unnecessary - selects all the data in both dim1 and dim2
- ONLY need to use colon operator when selecting a **subset** of data

HW #3 Review: Colon operator

Better (cleaner):

```
for i = 1:nSubs
    % load subject data
    load(fullfile('tom_localizer', ['s' num2str(i) '_roidata.mat']));

    % org data
    data(i,:,:) = sub_data;
end
```

- sub_data is a 2D matrix: 1 x 190
- Using colon operator here is unnecessary - selects all the data in both dim1 and dim2
- ONLY need to use colon operator when selecting a **subset** of data

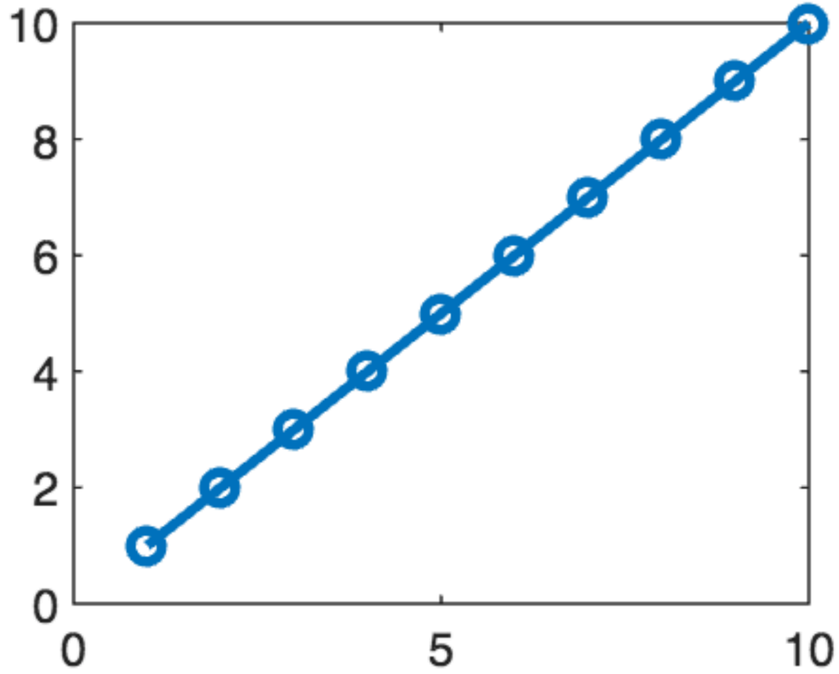
HW #3 Review: Plotting

Another example from HW plotting subject data:

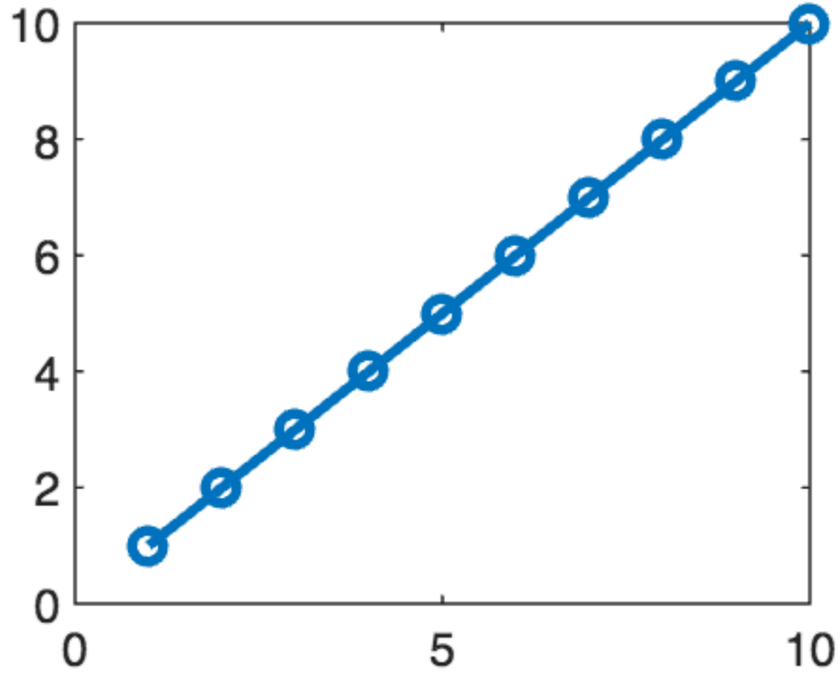
```
% sub_data: nSubs x nROIs x nTRs
for roiN = 1:nROIs

    % open figure
    figure('color', 'w')

    % plot each subject in this ROI
    for subN = 1:nSubs
        plot(1:190, squeeze(sub_data(subN, roiN, :)))
    end
end
```

- When you're plotting a timeseries, you can omit the x values
- MATLAB automatically plots y-values against index number

HW #3 Review: Plotting




```
figure('color', 'w')
plot(1:10, 1:10, '-o', 'linewidth', 2)
```




```
figure('color', 'w')
plot(1:10, '-o', 'linewidth', 2)
```

HW #3 Review: Plotting

Another example from HW plotting subject data:

```
% sub_data: nSubs x nROIs x nTRs
for roiN = 1:nROIs

    % open figure
    figure('color', 'w')

    % plot each subject in this ROI
    for subN = 1:nSubs
        plot(1:190, squeeze(sub_data(subN, roiN, :)))
    end
end
```

- Don’t need for loop
- If Y is a matrix, plots each row as a separate line

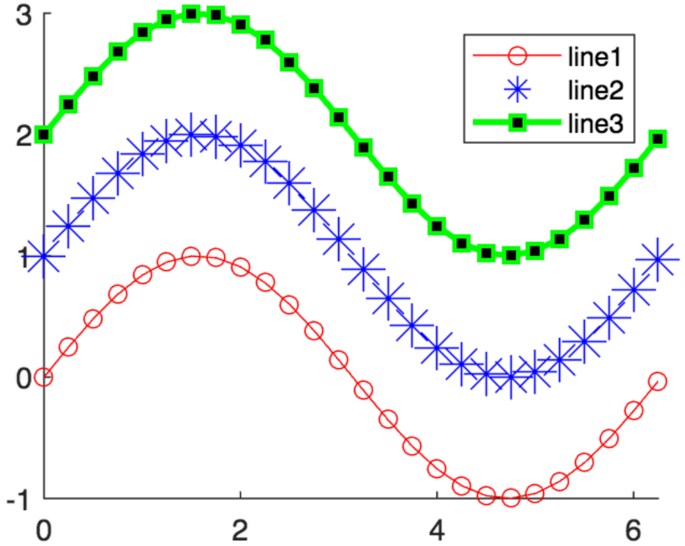
HW #3 Review: Plotting

Shorter, more efficient:

```
for roiN = 1:nROIs
    % get subj data for this roi
    roi_data = squeeze(data(:,i,:));

    % open figure
    figure('color', 'w')

    % plot each subject in this ROI
    plot(roi_data', 'color', [.5 .5 .5]);
end
```

HW #3 Review: Plotting

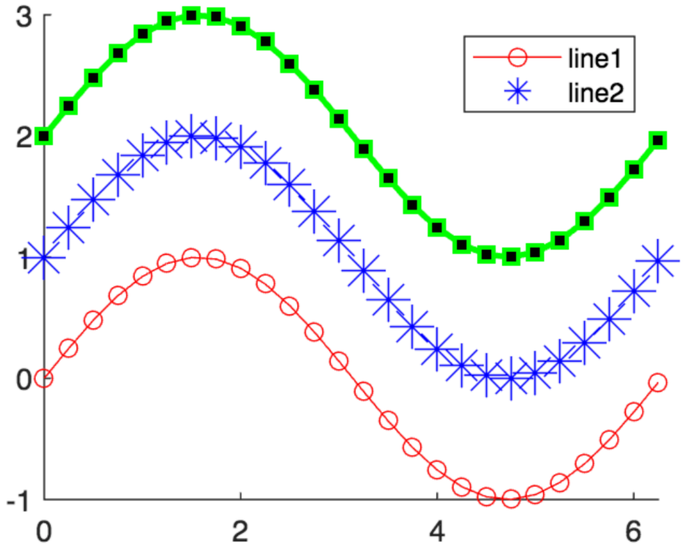
```
figure('color', 'w'); hold on;

x = 0:.25:6.3
plot(x, sin(x), 'ro-') % line 1
plot(x, sin(x) + 1, 'b*', 'markersize', 15) % line 2
plot(x, sin(x) + 2, 'g-s', 'markerfacecolor', [0 0 0], 'linewidth',2) % line 3

legend({'line1', 'line2', 'line3'})
```



- MATLAB makes legends in the order of lines you plot

HW #3 Review: Plotting

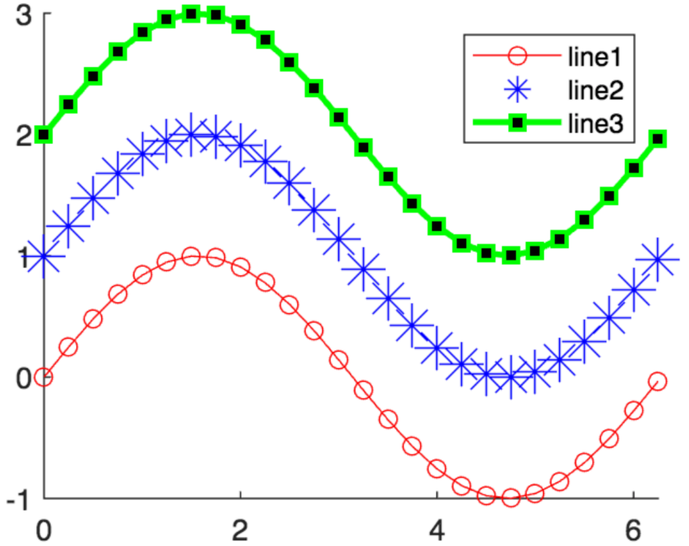
```
figure('color', 'w'); hold on;

x = 0:.25:6.3
plot(x, sin(x), 'ro-') % line 1
plot(x, sin(x) + 1, 'b*', 'markersize', 15) % line 2
plot(x, sin(x) + 2, 'g-s', 'markerfacecolor', [0 0 0], 'linewidth',2) % line 3

legend({'line1', 'line2'})
```



- If you have more lines than legend options, the legend just omits other lines

HW #3 Review: Plotting

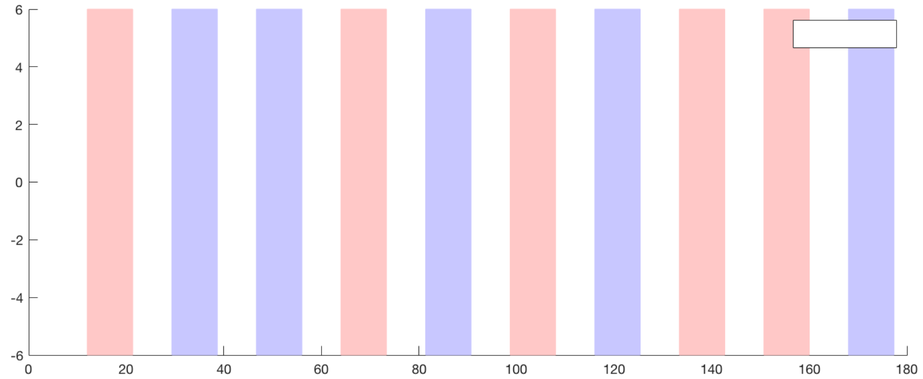
```
figure('color', 'w'); hold on;

x = 0:.25:6.3
plot(x, sin(x), 'ro-') % line 1
plot(x, sin(x) + 1, 'b*', 'markersize', 15) % line 2
plot(x, sin(x) + 2, 'g-s', 'markerfacecolor', [0 0 0], 'linewidth',2) % line 3

legend({'line1', 'line2', 'line3', 'line4'})
```



- If you have more legend options than lines, MATLAB just ignores the extra legend options and gives you a warning

HW #3 Review: Plotting

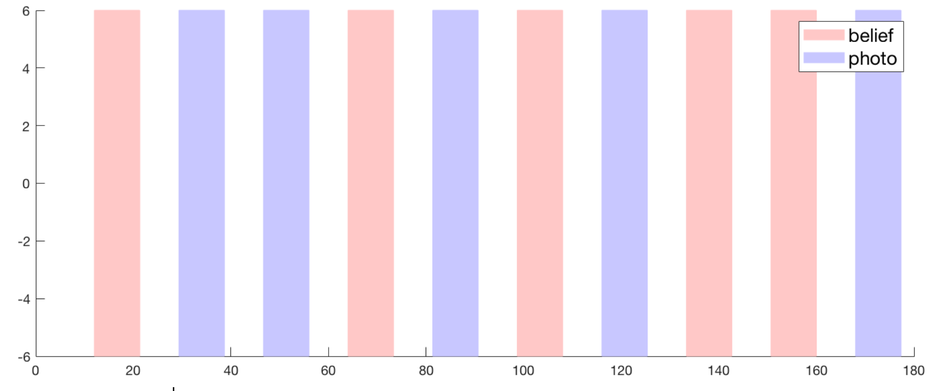
```
figure('color', 'w'); hold on;

for j = 1:length(belief)
    rectangle('position', [belief_lag(j), - 6, blockLength, 12],...
        'edgecolor', [1 .8 .8], 'facecolor', [1 .8 .8]);
end
for j = 1:length(photo)
    rectangle('position', [photo_lag(j), - 6, blockLength, 12], ...
        'edgecolor', [.8 .8 1], 'facecolor', [.8 .8 1]);
end

legend({'belief', 'photo'})
```



- MATLAB doesn’t “count” rectangles as lines, so the legend function doesn’t work

HW #3 Review: Plotting

```
figure('color', 'w'); hold on;

for j = 1:length(belief)
    rectangle('position', [belief_lag(j), - 6, blockLength, 12],...
        'edgecolor', [1 .8 .8], 'facecolor', [1 .8 .8]);
end
for j = 1:length(photo)
    rectangle('position', [photo_lag(j), - 6, blockLength, 12], ...
        'edgecolor', [.8 .8 1], 'facecolor', [.8 .8 1]);
end
plot(belief_lag(j), -6, '-', 'color', [1 .8 .8], 'linewidth', 8);
plot(photo_lag(j), -6, '-', 'color', [.8 .8 1], 'linewidth', 8);

legend({'belief', 'photo'})
```



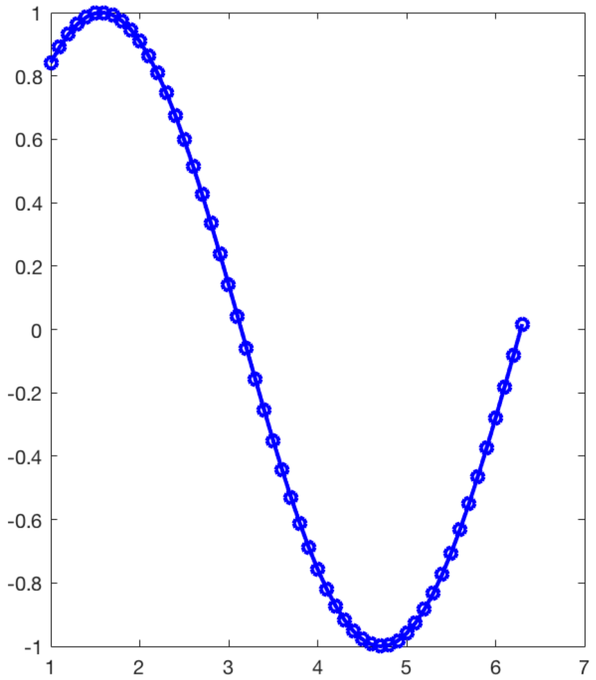
- Work around: plot lines to mimic rectangles

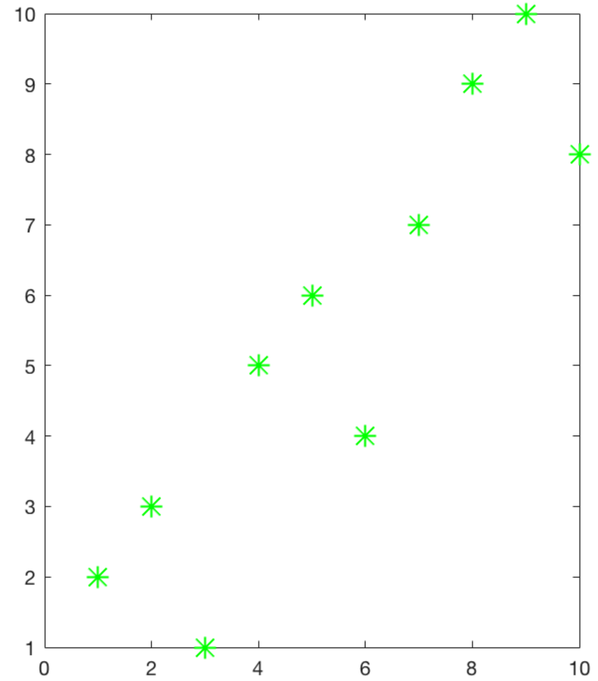
Outline

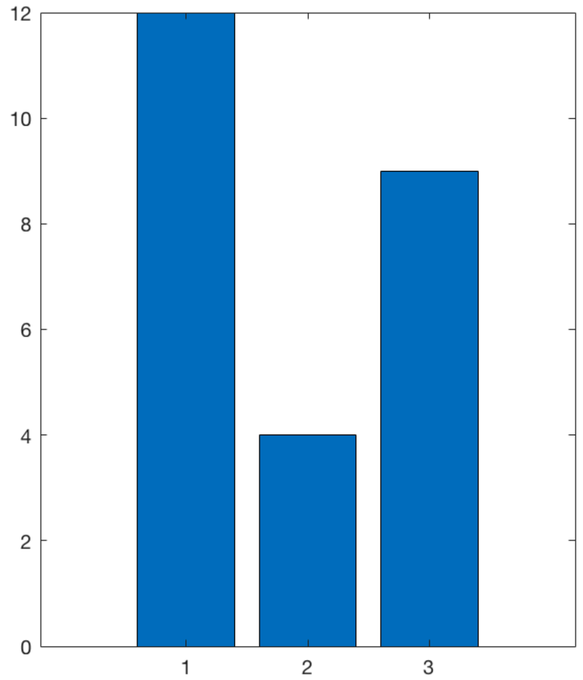
- HW #3 review - hard coding values, plotting redux, legends, colon operator
- **Plotting II**
- Data types II: cell arrays, structs
- Pathing
- Input/Output (IO)

Plotting II

- Previously: lines, scatter, and bar plots







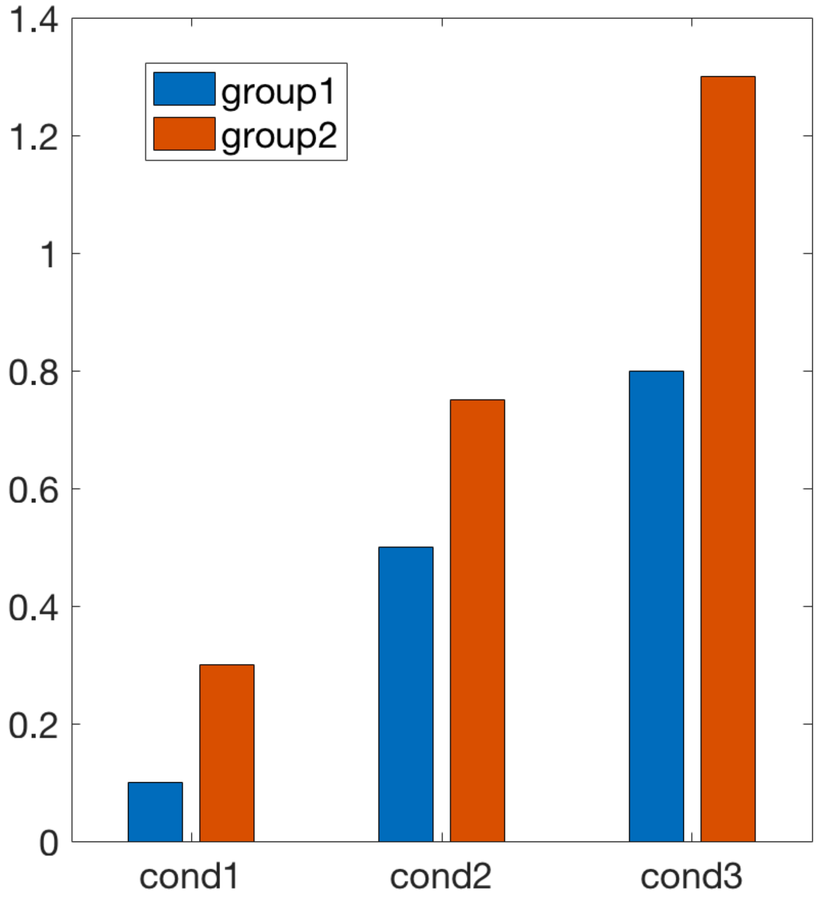

```
subplot(1,3,1);
plot(1:.1:6.3, sin(1:.1:6.3), 'bo-', 'linewidth', 2);
```

```
subplot(1,3,2);
plot([1:10], [2 3 1 5 6 4 7 9 10 8], 'g*', 'markersize', 10);
```

```
subplot(1,3,3);
bar([1:3], [12 4 9]);
```

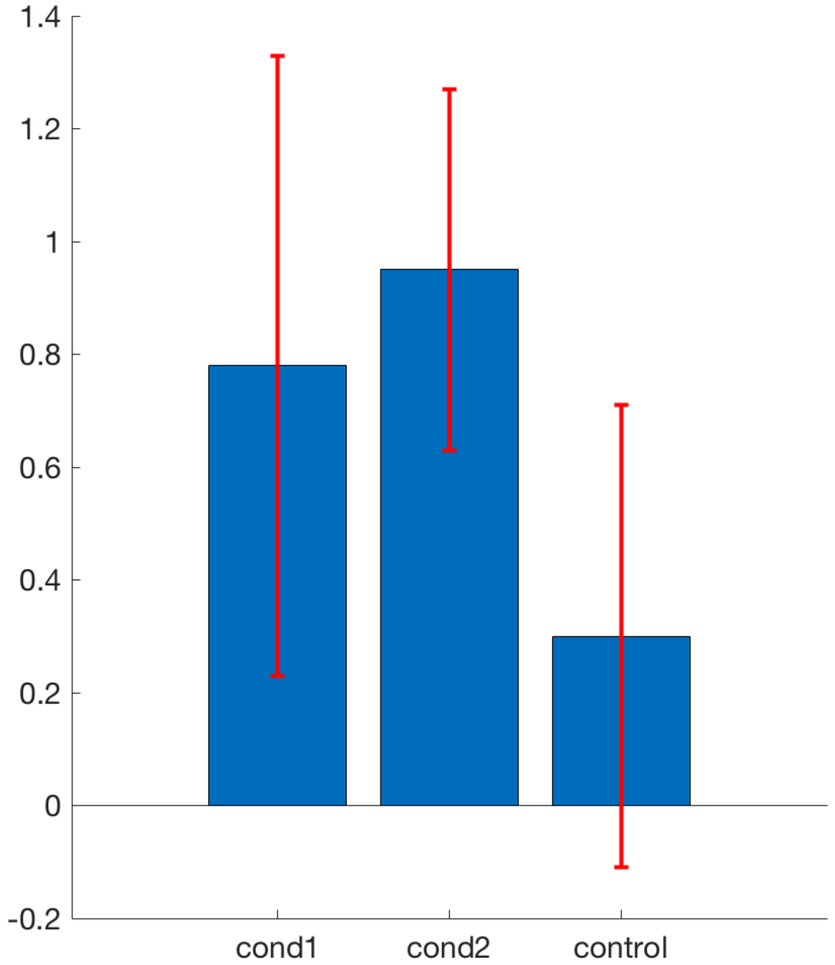
Bar plots: grouping bars

```
data = [.1 .3; .5 .75; .8 1.3];

figure('color', 'w'); hold on;
bar(data,.75,'grouped')

legend({'group1', 'group2'}, 'fontsize', 18);
set(gca, 'xticklabel', {'cond1', 'cond2', ...
    'cond3'}, 'fontsize', 18);
```



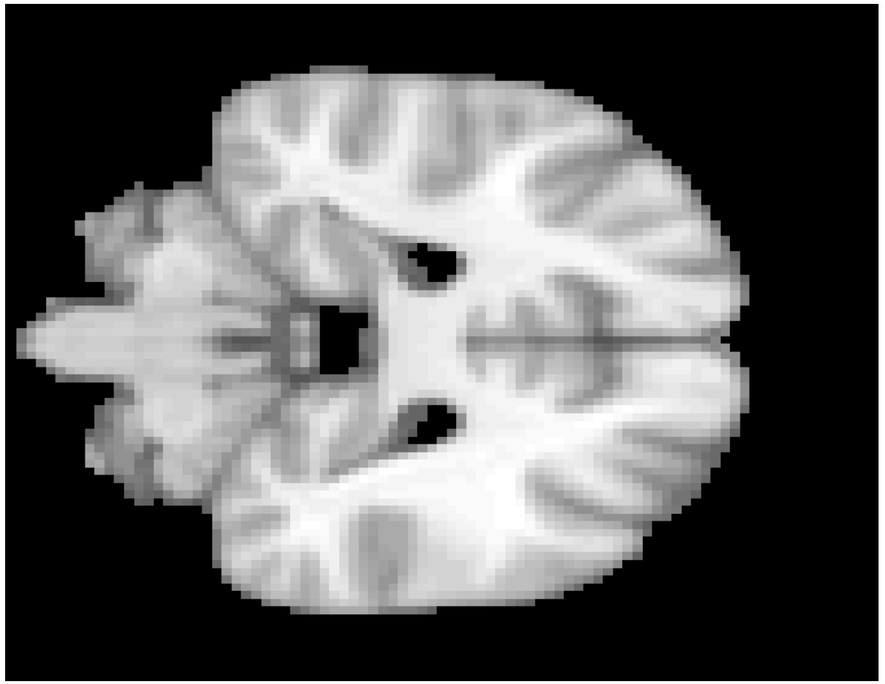
Adding error bars

```
x = 1:3;
y = [.78 .95 .3];
err =  [.55 .32 .41];

figure('color', 'w'); hold on;
bar(x, y);
errorbar(x, y, err, 'linestyle', 'none', ...
    'linewidth', 2, 'color', [1 0 0]);
set(gca, 'xtick', [1:3], 'xticklabel', ...
    {'cond1', 'cond2', 'control'}, ...
    'fontsize', 15);
```



Visualize a matrix with colors

```
brain = load('MNI152_T1_2mm_brain.mat');
slice = squeeze(brain.nifti_img(:,45,:));

figure('color', 'w'); colormap gray;
imagesc(slice);

axis off;
```



Visualize a matrix with colors

```
% load data
data = load('carbig.mat');
```

Command Window

```
>> data

data =

  struct with fields:

          Model: [406×36 char]
         Origin: [406×7 char]
            MPG: [406×1 double]
      Cylinders: [406×1 double]
   Displacement: [406×1 double]
     Horsepower: [406×1 double]
         Weight: [406×1 double]
   Acceleration: [406×1 double]
     Model_Year: [406×1 double]
           cyl4: [406×5 char]
            org: [406×7 char]
           when: [406×5 char]
            Mfg: [406×13 char]
```

Visualize a matrix with colors

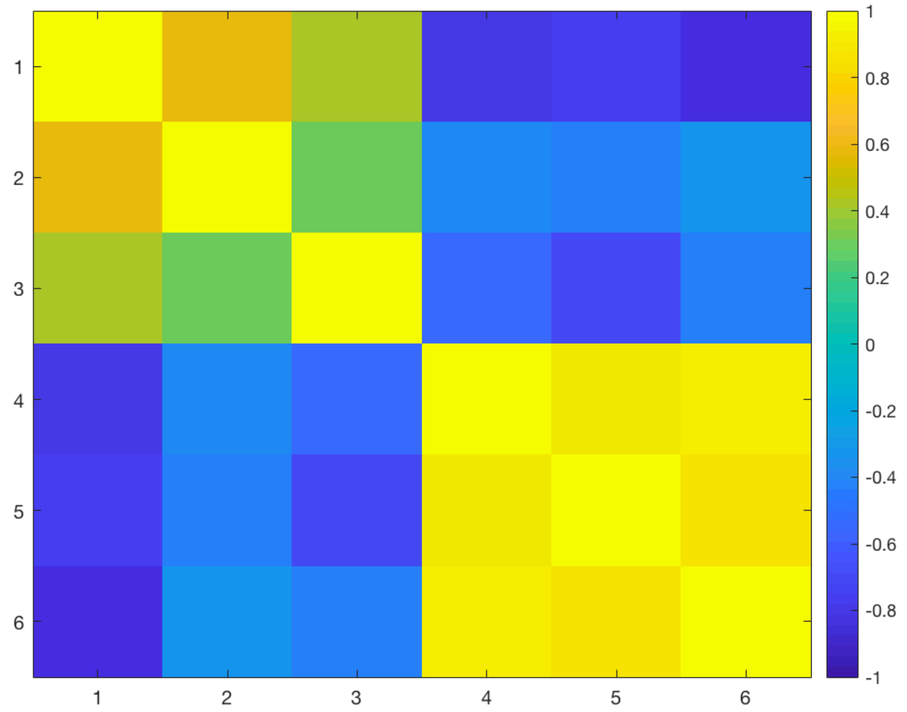
```
% load data
data = load('carbig.mat');

% build correlation matrix
data_mat = [MPG,  Model_Year, Acceleration, ...
    Displacement, Horsepower, Weight];
cor_mat = corr(data_mat, 'rows', 'pairwise');
```

Command Window

```
>> data

data =

  struct with fields:

          Model: [406×36 char]
         Origin: [406×7 char]
            MPG: [406×1 double]
      Cylinders: [406×1 double]
   Displacement: [406×1 double]
     Horsepower: [406×1 double]
         Weight: [406×1 double]
   Acceleration: [406×1 double]
     Model_Year: [406×1 double]
           cyl4: [406×5 char]
            org: [406×7 char]
           when: [406×5 char]
            Mfg: [406×13 char]
```

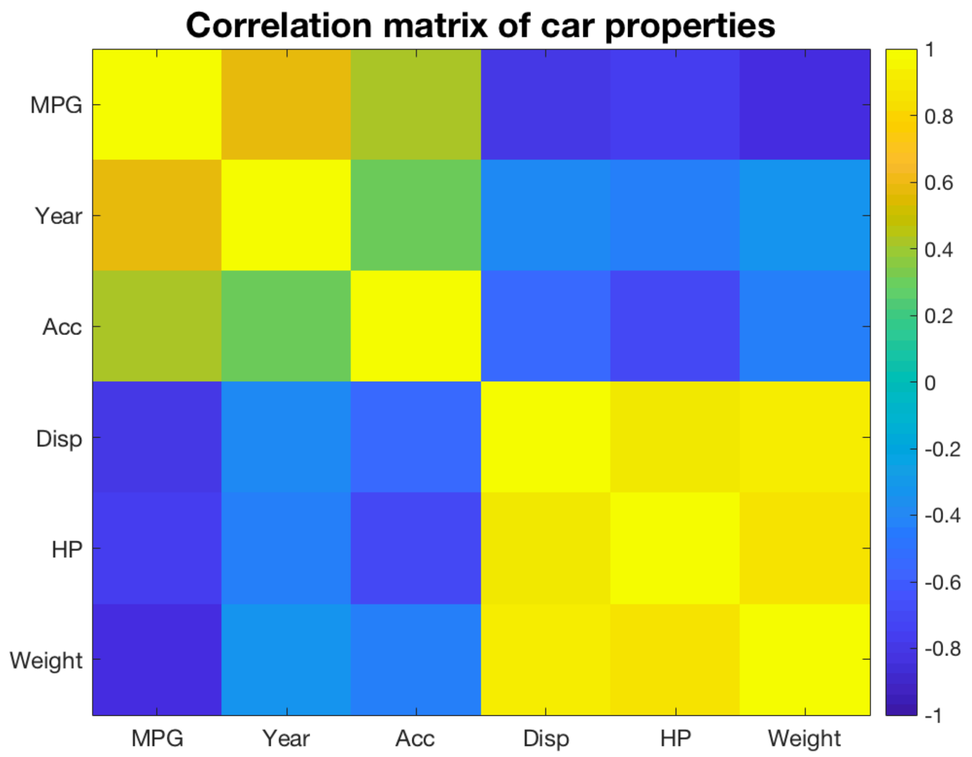
Visualize a matrix with colors

```
% load data
data = load('carbig.mat');

% build correlation matrix
data_mat = [MPG,  Model_Year, Acceleration, ...
    Displacement, Horsepower, Weight];
cor_mat = corr(data_mat, 'rows', 'pairwise');

% visualize
figure('color', 'w');
imagesc(cor_mat, [-1 1]);
colorbar;
```



Visualize a matrix with colors

```
% load data
data = load('carbig.mat');

% build correlation matrix
data_mat = [MPG,  Model_Year, Acceleration, ...
    Displacement, Horsepower, Weight];
cor_mat = corr(data_mat, 'rows', 'pairwise');

% visualize
figure('color', 'w');
imagesc(cor_mat, [-1 1]);
colorbar;

var_names = {'MPG', 'Year', 'Acc', 'Disp', ...
    'HP', 'Weight'};
set(gca, 'xticklabel', var_names, 'yticklabel', ...
    var_names, 'fontsize', 12);
title('Correlation matrix of car properties', ...
    'fontsize', 18);
```



tentative: more on clustering in a few weeks

Outline

- HW #3 review - hard coding values, plotting redux, legends, colon operator
- Plotting II
- **Data types II: cell arrays, structs**
- Pathing
- Input/Output (IO)

Data types so far:

- **Strings/chars**: contain text
- **Vectors, matrices**: contain numbers
- **Cell arrays:** collection of cells containing any data type (including cell arrays)
- **Structs:** organization of any data type to a field

Cell arrays

- **Vectors and matrices**
  - Can only contain numbers (and only numbers of one type)
  - Defined using square brackets [ ], select with parentheses ( )

```
my_vec = [1 2 3 4]
my_other_vec = [.1 .9 .13 .4]
```

Cell arrays

- **Cell arrays**
    - Can only contain contains any combination of data types
    - Defined using curly brackets { }, select using curly brackets { }

Cell arrays

- **Cell arrays**
  - Can only contain contains any combination of data types
  - Defined using curly brackets { }, select using curly brackets { }

untitled *

```
1  my_cell = {'hello!', [1:4], {'a', 'b', 'c'}};
2  disp(my_cell{1})
3  disp(my_cell{2})
4  disp(my_cell{3})
5
```

COMMAND WINDOW

```
hello!
     1     2     3     4

    {'a'}    {'b'}    {'c'}
```

Cell arrays

- Especially useful for storing a series of strings to iterate through later

untitled *

```
1  sub_ids = {'s01_061020', 's02_061120', ...
2      's03_062020', 's04_062320'};
3
4  for i = 1:length(sub_ids)
5      fprintf(['Analyze data for ' sub_ids{i} '\n'])
6  end
```

COMMAND WINDOW

```
Analyze data for s01_061020
Analyze data for s02_061120
Analyze data for s03_062020
Analyze data for s04_062320
>>
```

Cell arrays: Exercises

- **Exercise 1.1:**
    - Create a 2x3 cell array. The cells in the first row should contain the strings ‘a’, ‘b’, and ‘c’. The cells in the second row should contain the numbers 1, 2, and 3.
    - Print all the values in row 1. Print all the values in column 2.
    - Replace the values in row 2 with the numbers written out (‘one’, ‘two’, ‘three’)
- **Exercise 1.2:**
    - Create a cell array containing the 3 ToM ROI names, rTPJ, lTPJ, and precuneous. Write a for loop that iterates through the ROIs and prints them.
- **Exercise 1.3: on your own**
    - Create a cell array containing subj ids: s01, s02, s03, s04. Using a for loop, open a new figure for each subject. Name each figure by the corresponding subject id (you don't need to actually plot anything). Bonus: repeat but make a 2x2 subplot instead of a new figure for each subject

Python aside

- Python **lists** can contain any data type already (define with square brackets)
- python **dictionaries** are defined with curly brackets
- **numpy arrays** can only contain a single data type (define with `np.array()`)
- **numpy structured arrays** can contain multiple data types and are roughly analogous to MATLAB cell arrays (but not widely used, I think) (define similarly with `np.array()`, but with datatype argument)

Structs

- Structs are organize other data types with **field - value** pairs
- Define a structs fields and value with period; get field values with period

Structs

```
my_struct.name = 'mai';
my_struct.age = 30;
my_struct.bday = [06, 01, 1990];
my_struct.current_city = 'Princeton';
my_struct.current_state = 'NJ';
my_struct.cities = {'Olympia', 'Palo Alto', ...
    'Cambridge', 'NYC', 'Princeton'};

my_struct

disp(my_struct.name)
disp(my_struct.current_city)

disp(my_struct.bday(1))
disp(my_struct.cities{1})
```

COMMAND WINDOW

```
my_struct =

  struct with fields:

            name: 'mai'
             age: 30
            bday: [6 1 1990]
    current_city: 'Princeton'
   current_state: 'NJ'
          cities: {'Olympia'  'Palo Alto'  'Cambridge'  'NYC'  'Princeton'}

mai
Princeton
     6

Olympia
```

Structs: Exercises

- **Exercise 2.1**
    - Create a struct with fields for your name, university, class year, graduation year, and major(s). Display your struct. Display your class year and graduation year
    - Update your class year for next year. If you will have graduated next year, replace it with ‘graduated'

Python aside

- **Python dictionaries** with key-value pairs are roughly analogous to **MATLAB structs** with field-value pairs

- Python uses **curly brackets** to create and select dictionaries

Outline

- HW #3 review - hard coding values, plotting redux, legends, colon operator
- Plotting II
- Data types II: cell arrays, structs
- **Pathing**
- Input/Output (IO)

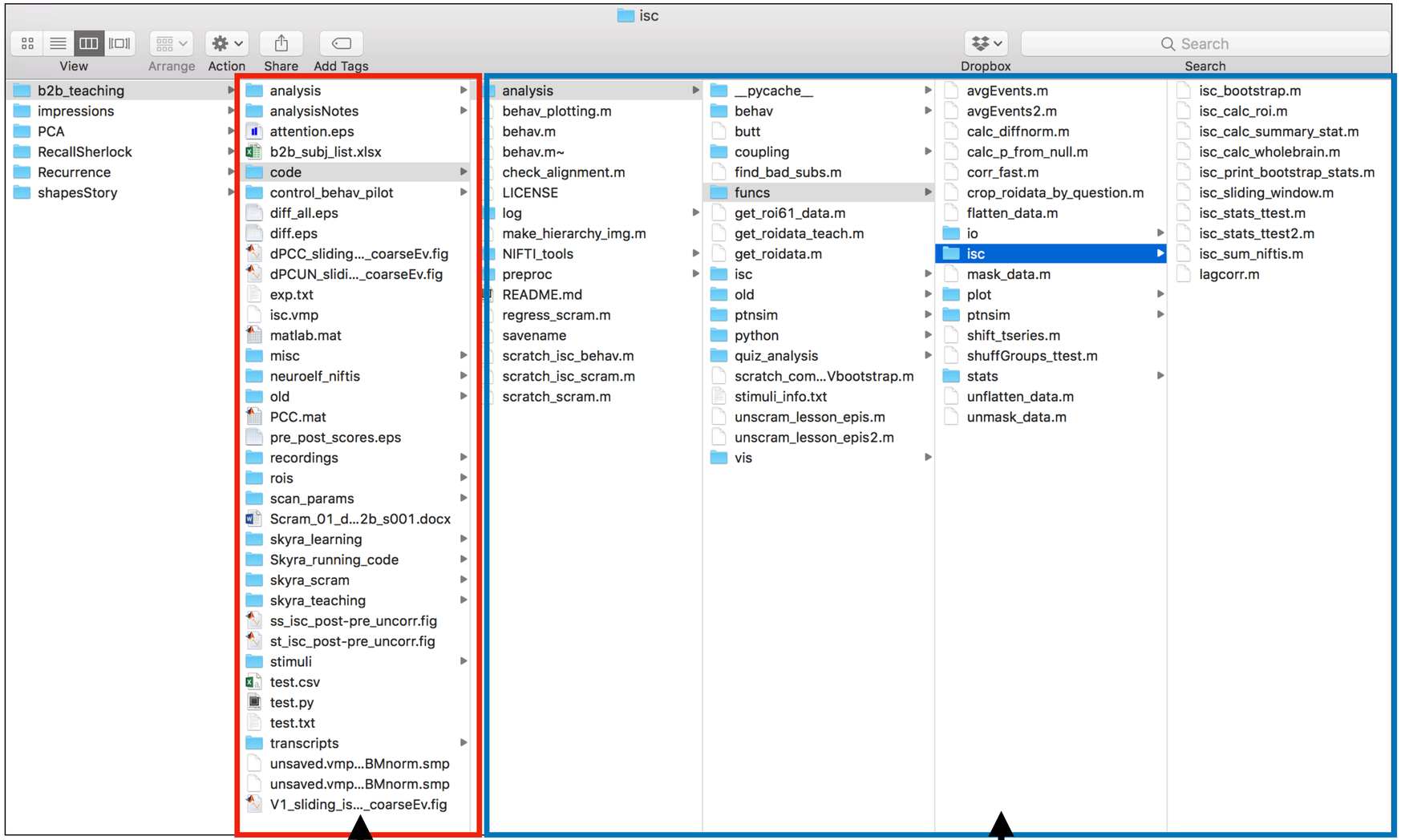
Pathing

- Previously
  - Working directory, cd, and ls
  - Keep all your files (data, functions, scripts) in working directory
  - Super inconvenient (especially as projects get more complicated)

```
>> pwd

ans =

    '/Users/Mai/Projects/teaching/matlab/week4'
```

Pathing

incidentally this project directory is a nightmare and this is very much an example of WHAT NOT TO DO



set my current directory here

still want to access this stuff

Pathing: access functions not in your wd

COMMAND WINDOW

```
>> pwd

ans =

    '/MATLAB Drive/week4'
```

Pathing: access functions not in your wd

COMMAND WINDOW

```
>> pwd

ans =

    '/MATLAB Drive/week4'

>> ls code
hide_and_seek.m
```

Pathing: access functions not in your wd

COMMAND WINDOW

```
>> pwd

ans =

    '/MATLAB Drive/week4'

>> ls code
hide_and_seek.m

>> hide_and_seek
'hide_and_seek' is not found in the current folder or on the MATLAB path, but exists in:
    /MATLAB Drive/week4/code

Change the MATLAB current folder or add its folder to the MATLAB path.
```

Pathing: access functions not in your wd

COMMAND WINDOW

```
>> pwd

ans =

    '/MATLAB Drive/week4'

>> ls code
hide_and_seek.m

>> hide_and_seek
'hide_and_seek' is not found in the current folder or on the MATLAB path, but exists in:
    /MATLAB Drive/week4/code

Change the MATLAB current folder or add its folder to the MATLAB path.

>> addpath('code')
```

addpath(dir) tells MATLAB to look for functions in this directory also

Pathing: access functions not in your wd

COMMAND WINDOW

```
>> pwd

ans =

    '/MATLAB Drive/week4'

>> ls code
hide_and_seek.m

>> hide_and_seek
'hide_and_seek' is not found in the current folder or on the MATLAB path, but exists in:
    /MATLAB Drive/week4/code

Change the MATLAB current folder or add its folder to the MATLAB path.

>> addpath('code')
>> hide_and_seek
you found me!
>>
```

Pathing: access functions not in your wd

- **`addpath(dir)`**: look for functions in this directory also, in addition to function in working directory

- **`addpath(genpath(dir))`**: look for functions in this directory and its subdirectories

Pathing: why can I always access built-in functions?

- When MATLAB starts up, it automatically adds paths to all the functions that come with MATLAB and its libraries/toolboxes (built-ins)

- You can see all the paths that MATLAB searches in for functions by using `path()`:

```
>> path

                MATLABPATH

        /Users/Mai/Documents/MATLAB
        /private/var/folders/2z/y9db62xj357_3zxmq8ff4_lh0000gn/T/Editor
        /Applications/MATLAB_R2019a.app/toolbox/matlab/images
        /Applications/MATLAB_R2019a.app/toolbox/matlab/networklib
        /Applications/MATLAB_R2019a.app/toolbox/matlab/parallel
        /Applications/MATLAB_R2019a.app/toolbox/matlab/codetools
        /Applications/MATLAB_R2019a.app/toolbox/matlab/codetools/embeddedoutputs
        /Applications/MATLAB_R2019a.app/toolbox/matlab/testframework/measurement/ext
        /Applications/MATLAB_R2019a.app/toolbox/matlab/datatools/importtool/matlab/server
        /Applications/MATLAB_R2019a.app/toolbox/matlab/testframework/unittest/core
        /Applications/MATLAB_R2019a.app/toolbox/matlab/testframework/obsolete
        /Applications/MATLAB_R2019a.app/toolbox/matlab/mex
        /Applications/MATLAB_R2019a.app/toolbox/matlab/cefclient
        /Applications/MATLAB_R2019a.app/toolbox/matlab/connector2/interpreter
        /Applications/MATLAB_R2019a.app/toolbox/matlab/connector2/configuration
        /Applications/MATLAB_R2019a.app/toolbox/matlab/project
        /Applications/MATLAB_R2019a.app/toolbox/matlab/project/example
        /Applications/MATLAB_R2019a.app/toolbox/matlab/project/toolstrip
        /Applications/MATLAB_R2019a.app/toolbox/matlab/parquetio
        /Applications/MATLAB_R2019a.app/toolbox/matlab/graphfun
        /Applications/MATLAB_R2019a.app/toolbox/matlab/timefun
        /Applications/MATLAB_R2019a.app/toolbox/matlab/webcam
        /Applications/MATLAB_R2019a.app/toolbox/matlab/datamanager
        /Applications/MATLAB_R2019a.app/toolbox/matlab/connector2/shadowfiles
        /Applications/MATLAB_R2019a.app/toolbox/matlab/external/interfaces/webservices/wsdl
        /Applications/MATLAB_R2019a.app/toolbox/matlab/plottools/inspector
        /Applications/MATLAB_R2019a.app/toolbox/matlab/datatools/datatoolsservices/matlab
        /Applications/MATLAB_R2019a.app/toolbox/matlab/guide

                                                ...
```

Pathing: building paths to stuff (usually data)

COMMAND WINDOW

```
>> pwd

ans =

    '/MATLAB Drive/week4'
```

Pathing: building paths to stuff (usually data)

COMMAND WINDOW

```
>> pwd

ans =

    '/MATLAB Drive/week4'

>> ls
code  data  results
```

Pathing: building paths to stuff (usually data)

COMMAND WINDOW

```
>> pwd

ans =

    '/MATLAB Drive/week4'

>> ls
code  data  results

>> ls data
data1.mat  data2.mat  data3.mat  data4.mat
```

Pathing: building paths to stuff (usually data)

COMMAND WINDOW

```
>> pwd

ans =

    '/MATLAB Drive/week4'

>> ls
code  data  results

>> ls data
data1.mat  data2.mat  data3.mat  data4.mat

>> data_pth = fullfile('data', 'data1.mat')

data_pth =

    'data/data1.mat'
```

`fullfile(part1, part2, …)`

- Creates a path for your OS

Pathing: building paths to stuff (usually data)

COMMAND WINDOW

```
>> pwd

ans =

    '/MATLAB Drive/week4'

>> ls
code  data  results

>> ls data
data1.mat  data2.mat  data3.mat  data4.mat

>> data_pth = fullfile('data', 'data1.mat')

data_pth =

    'data/data1.mat'

>> load(data_pth)
```

`fullfile(part1, part2, …)`

- Creates a path for your OS

Pathing: building paths to stuff (usually data)

COMMAND WINDOW

```
>> data_dir = 'learning_data';
>> subs = {'s01', 's02', 's03', 's04'};
```

`fullfile(part1, part2, …)`

- Creates a path for your OS
- Can use variables to build paths
- More flexible

Pathing: building paths to stuff (usually data)

COMMAND WINDOW

```
>> data_dir = 'learning_data';
>> subs = {'s01', 's02', 's03', 's04'};
>> for i = 1:length(subs)
sub_data_pth = fullfile(data_dir, subs{i});
disp(sub_data_pth);
end
```

`fullfile(part1, part2, …)`

- Creates a path for your OS
- Can use variables to build paths
- More flexible

Pathing: building paths to stuff (usually data)

COMMAND WINDOW

```
>> data_dir = 'learning_data';
>> subs = {'s01', 's02', 's03', 's04'};
>> for i = 1:length(subs)
sub_data_pth = fullfile(data_dir, subs{i});
disp(sub_data_pth);
end
learning_data/s01
learning_data/s02
learning_data/s03
learning_data/s04
```

`fullfile(part1, part2, …)`

- Creates a path for your OS
- Can use variables to build paths
- More flexible

Python aside

- `os.path.join()` is roughly analogous to MATLAB's `fullfile()`
- Note that you'll have to import the os.path module into python to use it

Outline

- HW #3 review - hard coding values, plotting redux, legends, colon operator
- Data types II: cell arrays, structs
- Pathing
- **Input/Output (IO)**

IO: loading things (usually data)

- `load(pth_to_mat_file)`: load a .mat file, which is MATLAB's file type

Load variables directly to workspace

```
>> data_file = fullfile('hw', 'behav_data');
>> load(data_file)
>> whos
  Name                 Size            Bytes  Class     Attributes

  attention           42x2                672  double
  data_file            1x13                26  char
  intact_data         22x25              4400  double
  scrambled_data      20x25              4000  double
```

Load variables to a struct (better)

```
>> data_file = fullfile('hw', 'behav_data');
>> data = load(data_file);
>> data

data =

  struct with fields:

        attention: [42×2 double]
      intact_data: [22×25 double]
   scrambled_data: [20×25 double]
```

IO: loading things (usually data)

- `csvread(pth_to_csv_file)`: load a CSV file
    - Can only read numbers from CSVs! (unambiguous example of MATLAB being bad)
    - If you have column headers, use second argument to specify start row

```
>> data = csvread('vis_hierarchy_eyetracking.csv',1);
>> size(data)

ans =

    63     5
```

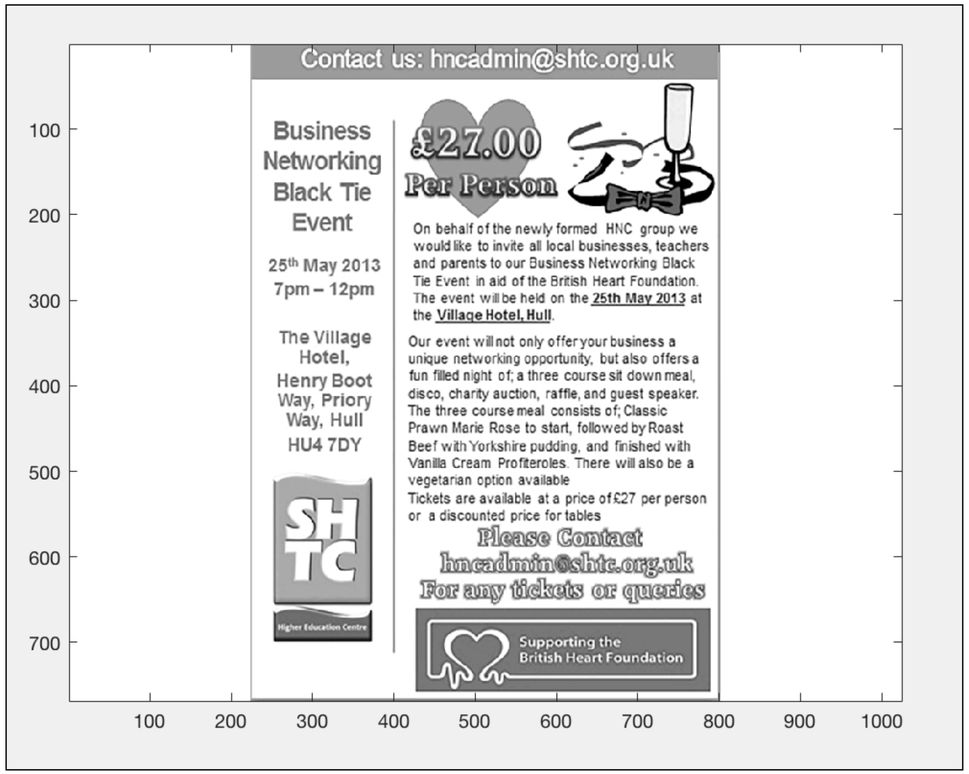
IO: loading things (usually data)

- `xlsread(pth_to_excel_file)`: load an excel file
    - Can only read excel sheets on Windows (who knows why? another unambiguous example of MATLAB being bad)
- `xlsread(pth_to_col_file)`:

IO: loading things (usually data)

- `imread(pth_to_image)`: load an image into a matrix

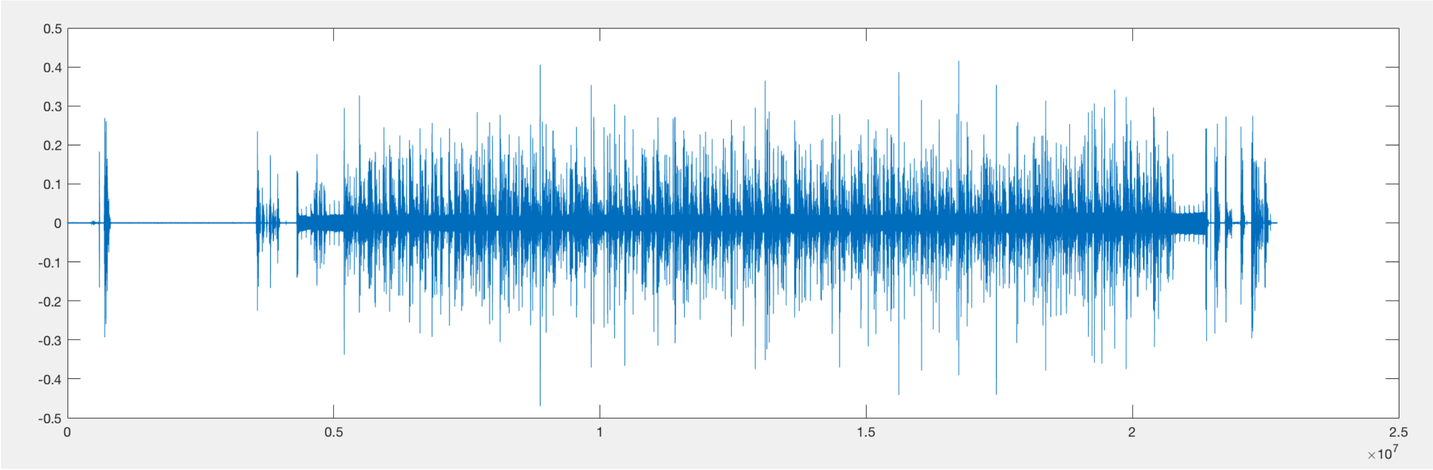
```
>> image = imread('img1.jpg');
>> size(image)

ans =

         768        1024

>> figure; imagesc(image)
```



IO: loading things (usually data)

- `audioread(pth_to_audio)`: load an audio as a vector (timeseries)

```
>> audio_file = 'recordings/review_run1_022618/audios/review_run1_022618.wav';
>> audio = audioread(audio_file);

>> size(audio)

ans =

    22716416           1

>> figure; plot(audio)
```



IO: saving things

- `save(pth_to_savefile, 'var1', 'var2'…)`: save variables to a .mat file

```
>> name = 'mai';
>> age = 30;
>> city = 'princeton';
>> state = 'nj';
>> save(fullfile('examples', 'mai'), 'name', 'age', 'city', 'state')
>> data = load(fullfile('examples', 'mai'))

data =

  struct with fields:

      age: 30
     city: 'princeton'
     name: 'mai'
    state: 'nj'
```

*** 9/10 this is what I do**

IO: saving things

- `csvwrite(pth_to_savefile, matrix)`: save a matrix to a csv file
- `writetable()` also works

```
>> my_mat = rand(5,5);
>> csvwrite('examples/my_matrix.csv', my_mat)
```

IO: saving things

- Writing to a plain text file (slightly complicated)

```
% open a text file in write mode
fileID = fopen('my_text.txt', 'w');
```

IO: saving things

- Writing to a plain text file (slightly complicated)

```
% open a text file in write mode
fileID = fopen('my_text.txt', 'w');

% write a couple lines
fprintf(fileID, 'I wrote a line!\n');
fprintf(fileID, 'I wrote a second line!\n');
fprintf(fileID, 'One last line!\n');
```

IO: saving things

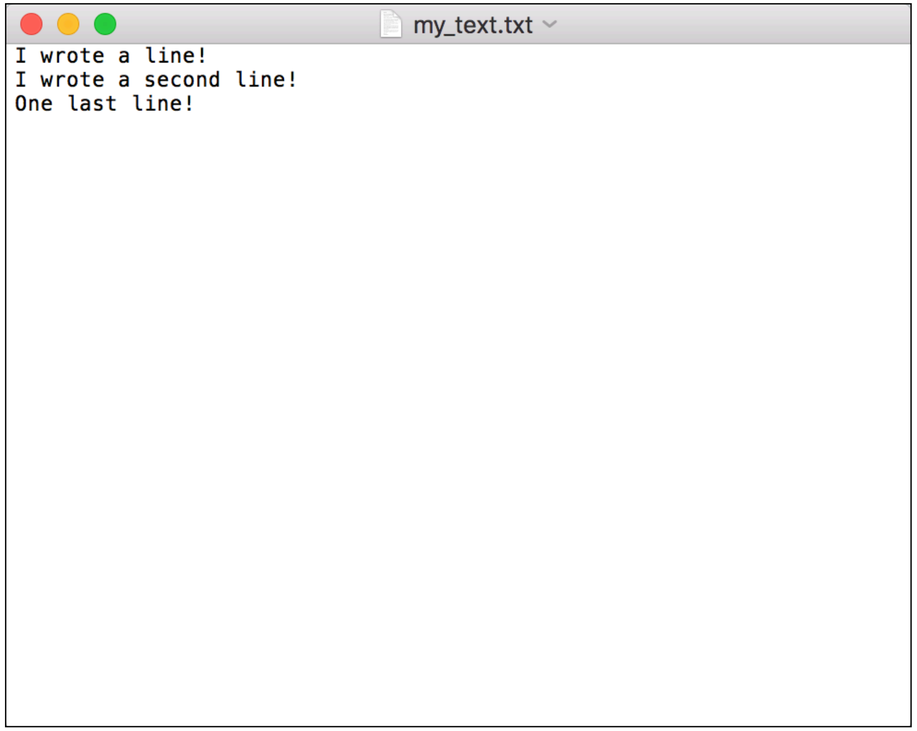
- Writing to a plain text file (slightly complicated)

```
% open a text file in write mode
fileID = fopen('my_text.txt', 'w');

% write a couple lines
fprintf(fileID, 'I wrote a line!\n');
fprintf(fileID, 'I wrote a second line!\n');
fprintf(fileID, 'One last line!\n');

% close text file
fclose(fileID);
```

IO: saving things

- Writing to a plain text file (slightly complicated)

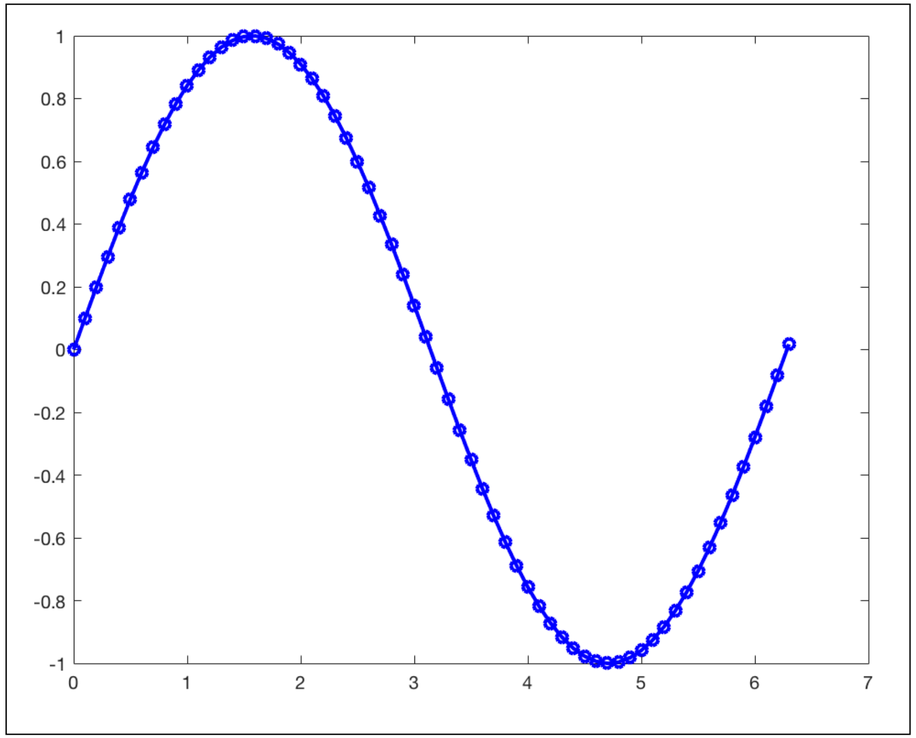
```
% open a text file in write mode
fileID = fopen('my_text.txt', 'w');

% write a couple lines
fprintf(fileID, 'I wrote a line!\n');
fprintf(fileID, 'I wrote a second line!\n');
fprintf(fileID, 'One last line!\n');

% close text file
fclose(fileID);
```



IO: saving things

- `saveas(gcf, pth_to_save, 'filetype')`: save a figure. By default saves to .fig

```
>> x = 0:.1:6.3;
>> figure('color', 'w');
>> plot(x, sin(x), 'b-o', 'linewidth', 2);
>> saveas(gcf, 'examples/sine_plot', 'fig')
```



Review

Plotting
- X-axis as index vals
- Legends
- Grouped bars
- `errorbar()`
- `imagesc()`

Data types II
- Cell array
- Structs

Pathing
- `fullfile()`
- `addpath()`
- `genpath()`

IO
- `load()`
- `csvread()`
- `xlsread()`
- `imread()`
- `audioread()`
- `save()`
- `csvwrite()`
- `writetable()`
- `fopen(), fprintf(), fclose()`
- `saveas()`

A final note

- You are fully equipped now with the basic MATLAB tools for doing basically any analysis

- More complicated stuff = larger data sets, using more complicated functions, but not necessarily more complicated code

A final note

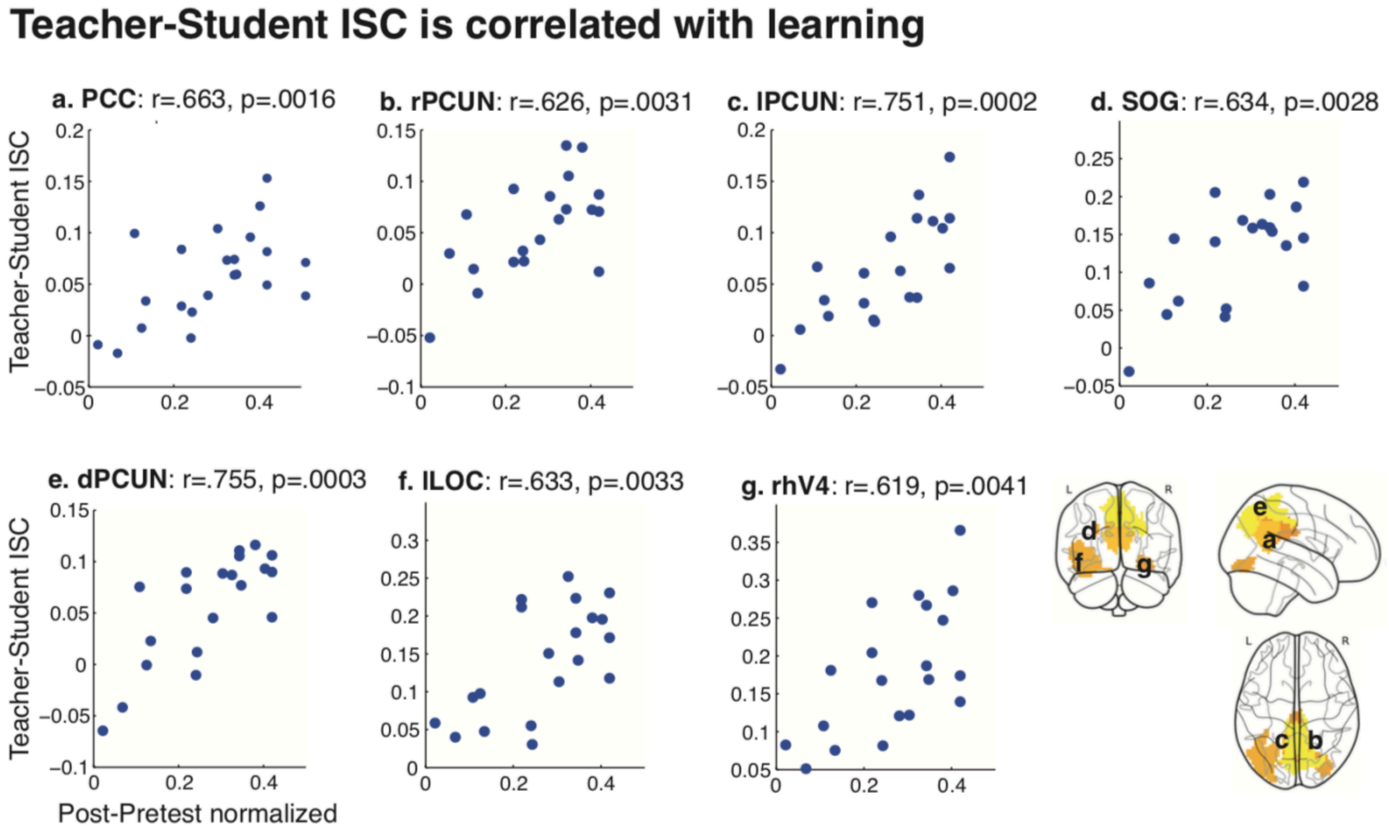


Figure 5. ISC correlation with behavior. Teacher-student ISC is correlated with normalized improvement in quiz score (Post-Pretest / (1/avg(Post+Pretest)) in five ROIs, primarily in posterior medical cortex (q<.05, FDR corrected). PCC = posterior cingulate cortex, rPCUN = right precuneous, lPCUN = left precuneous, SOG = superior occipital gyurs, dPCUN = dorsal precuneous, lLOC = left lateral occipital complex, rhV4 = right human V4. ROIs from 61 ROI parcellation (Regev et al., 2018).

A final note

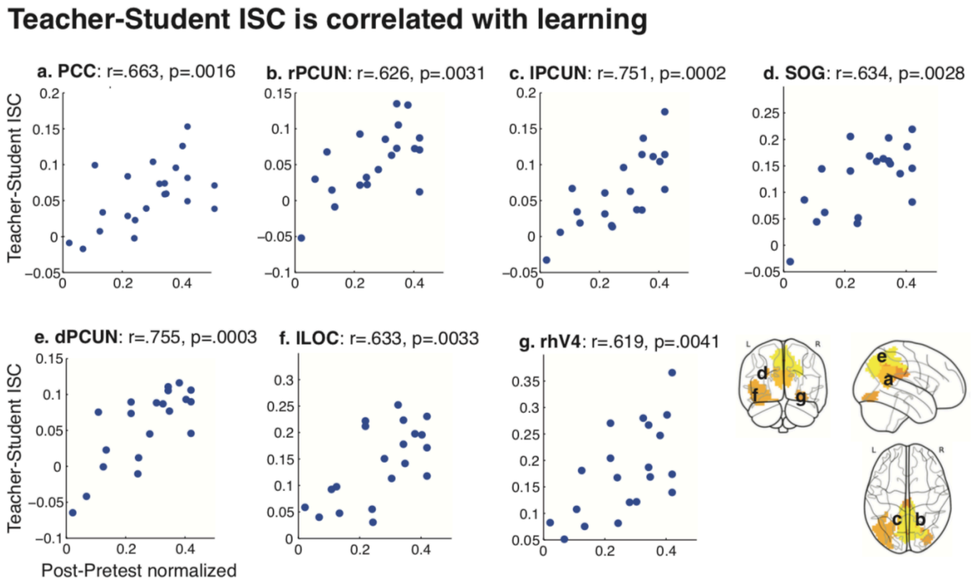


Figure 5. ISC correlation with behavior. Teacher-student ISC is correlated with normalized improvement in quiz score (Post-Pretest / (1/avg(Post+Pretest)) in five ROIs, primarily in posterior medical cortex (q<.05, FDR corrected). PCC = posterior cingulate cortex, rPCUN = right precuneous, lPCUN = left precuneous, SOG = superior occipital gyurs, dPCUN = dorsal precuneous, lLOC = left lateral occipital complex, rhV4 = right human V4. ROIs from 61 ROI parcellation (Regev et al., 2018).

1. Load data from 20 subjects
    - Use special load function, load_nii()
    - Similar to csvread() or load()

A final note

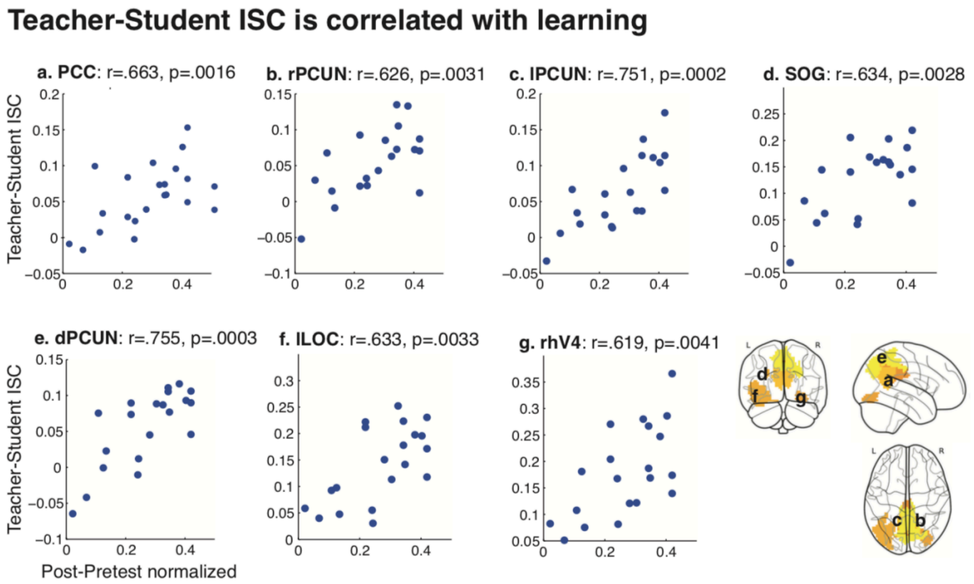


Figure 5. ISC correlation with behavior. Teacher-student ISC is correlated with normalized improvement in quiz score (Post-Pretest / (1/avg(Post+Pretest)) in five ROIs, primarily in posterior medical cortex (q<.05, FDR corrected). PCC = posterior cingulate cortex, rPCUN = right precuneous, lPCUN = left precuneous, SOG = superior occipital gyurs, dPCUN = dorsal precuneous, lLOC = left lateral occipital complex, rhV4 = right human V4. ROIs from 61 ROI parcellation (Regev et al., 2018).

1.Load data from 20 subjects

- Use special load function, load_nii()
- Similar to csvread() or load()

2. Get ROI data

- 64 ROIs from 61x73x61x956 data
- For loops, logical indexing, slicing ND matrix

A final note

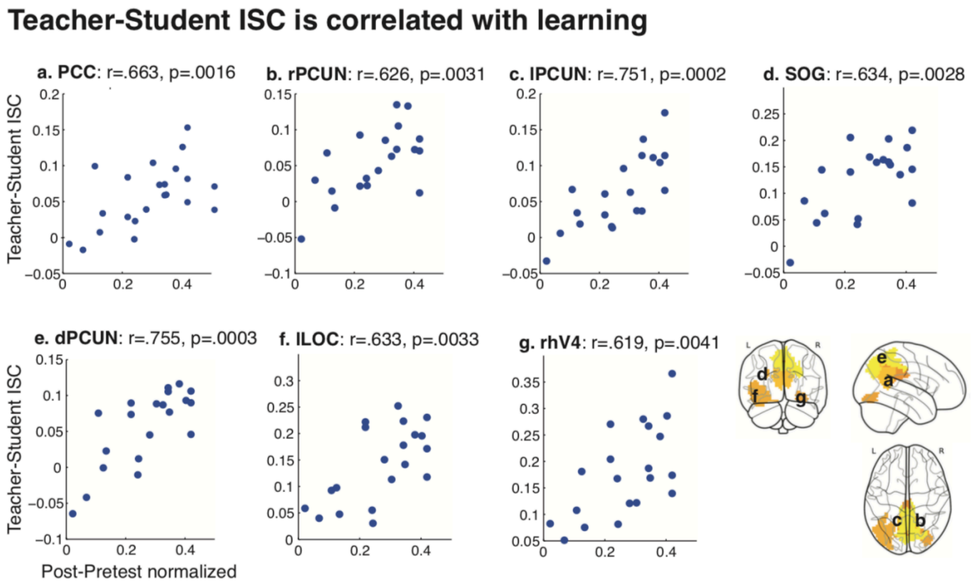


Figure 5. ISC correlation with behavior. Teacher-student ISC is correlated with normalized improvement in quiz score (Post-Pretest / (1/avg(Post+Pretest)) in five ROIs, primarily in posterior medical cortex (q<.05, FDR corrected). PCC = posterior cingulate cortex, rPCUN = right precuneous, lPCUN = left precuneous, SOG = superior occipital gyurs, dPCUN = dorsal precuneous, lLOC = left lateral occipital complex, rhV4 = right human V4. ROIs from 61 ROI parcellation (Regev et al., 2018).

3. Calculate ISC in each ROI

- For loop, filling a matrix, save .mat
- Slightly more difficult than running mean()

A final note

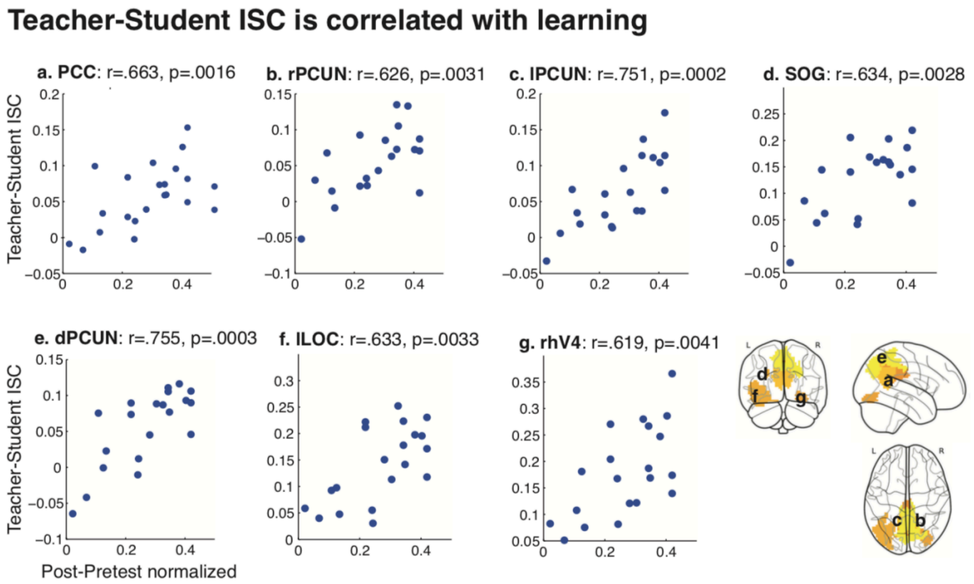


Figure 5. ISC correlation with behavior. Teacher-student ISC is correlated with normalized improvement in quiz score (Post-Pretest / (1/avg(Post+Pretest)) in five ROIs, primarily in posterior medical cortex (q<.05, FDR corrected). PCC = posterior cingulate cortex, rPCUN = right precuneous, lPCUN = left precuneous, SOG = superior occipital gyurs, dPCUN = dorsal precuneous, lLOC = left lateral occipital complex, rhV4 = right human V4. ROIs from 61 ROI parcellation (Regev et al., 2018).

3. Calculate ISC in each ROI

- For loop, filling a matrix, save .mat
- Slightly more difficult than running mean()

4. Correlate ISC with behavior, run bootstrapped statistics

- Again, only slight more difficult than mean()
- Basically involves repeatedly rearranging matrices (e.g. another for loop)

A final note

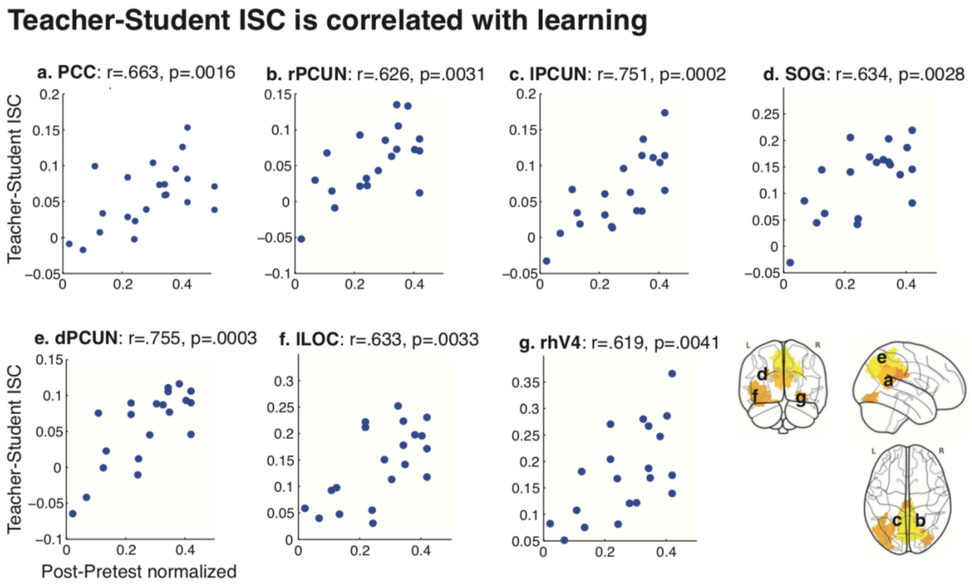


Figure 5. ISC correlation with behavior. Teacher-student ISC is correlated with normalized improvement in quiz score (Post-Pretest / (1/avg(Post+Pretest)) in five ROIs, primarily in posterior medical cortex (q<.05, FDR corrected). PCC = posterior cingulate cortex, rPCUN = right precuneous, lPCUN = left precuneous, SOG = superior occipital gyurs, dPCUN = dorsal precuneous, lLOC = left lateral occipital complex, rhV4 = right human V4. ROIs from 61 ROI parcellation (Regev et al., 2018).

5. Make a scatter plot

- Plot, subplot, etc