Cell Arrays, Struct Arrays, Matlab Path

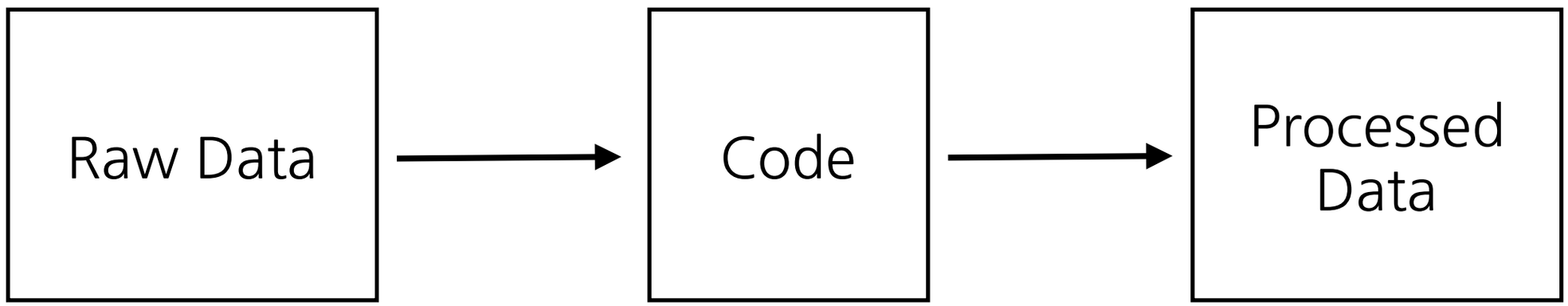
NENS 230: Analysis Techniques in Neuroscience

Fall 2015

Outline

1. **Cell arrays**
    1. Creating and concatenating cell arrays
    2. {} Indexing vs. () Indexing
2. **Struct arrays**
    1. Associating related data
    2. Structs, struct arrays, and indexing
    3. Accessing fields, aggregating data across a struct array
3. **The Matlab PATH**
    1. What is the path?
    2. How can I set the path from code?
    3. Common errors involving the path

Why are data types so important?



Examples:

- Voltage clamp traces
  - Current/signal, organized by channel, time
- Image file
  - Intensity, organized by channel, x pos, y pos, z pos

Processed form:

- Opsin tracking by frequency
  - Spikes evoked, organized by pulse frequency
- Cell positions
  - List of x,y,z coordinates for each cell detected

Other data structures

In addition to numeric arrays, there are common data types

- Character arrays or strings
- Cell arrays
- Structures

And some less commonly used ones

- Map containers
- Tables
- Time Series

Lists of strings?

What if we had a list of several strings? How would we keep track of all of them?

```
channelName1 = ‘gfp’;
channelName2 = ‘dapi’;
channelName3 = ‘neun’;
```

.

.

.

This is not a good idea, because any code that operates on each of the channels necessarily has to repeat itself N times.

Lists of strings?

Why can’t / shouldn’t we do something like this?

```
channelNames = [‘gfp’; ...
                ‘dapi’; ...
                ‘neun’]
```

You can have multidimensional arrays of characters, but this doesn’t work if the strings have different lengths! All rows must have the same number of columns. You could pad with spaces to make all rows the same length, but there’s a better way…

Collections of unlike items

Or what if we wanted to store lists of spike times?

The spike times on a given trial would be an array of class **`double`**

But, we probably don't have the same number of spikes on every trial, so we again can't (shouldn't) use a multidimensional array where each row is a trial and each column is a spike number.

```
spikesTrial1 = [3.2 5.8 9.1];
spikesTrial2 = [1.3 5.3 9.3 10.1];
spikesTrial3 = [0.3 2.1];
```
(please don't ever do this)

Cell arrays

Cell arrays work similarly to numeric arrays or character arrays, except inside each element you can store whatever you want:

- Numerical arrays
- Strings (character arrays)
- Other cell arrays
- etc.

Creating cell arrays

Use the cell() function just like you would zeros() or ones(), except it returns an array of empty cells.

```
channelNames = cell(5,1);
```

`channelNames` evaluates to

```
[]
[]
[]
[]
[]
```

5 rows, 1 column

Each of these represents the empty content stored in each cell of the array.

`size(channelNames)` evaluates to `[5 1]`

Creating cell arrays

Multidimensional cell arrays work too. Say you have 5 subjects, 3 conditions. Create a cell array where each subject is a row, each condition is a column.

```
dataByCondition = cell(5,3);
```

`dataByCondition` evaluates to

```
[]  []  []
[]  []  []
[]  []  []
[]  []  []
[]  []  []
```

`size(dataByCondition)` evaluates to `[5 3]`

Cell array indexing

There are two different ways to index into a cell array. They have different uses and different syntaxes.

The most common case is when you want to extract or store something into a particular cell of the array. For this you use `{ }` (curly brackets, braces).

Cell array indexing

```
dataByCondition{1,3} = [5 10 102];
```

`dataByCondition{1,3}` evaluates to `[5 10 102]`

```
dataByCondition{3,2} = [192 10];
```

`dataByCondition` evaluates to

```
[]  []          [5 10 102]
[]  []          []
[]  [192 10]    []
[]  []          []
[]  []          []
```

Cell array indexing

```
nChannels = 3;
channelNames = cell(nChannels,1);
channelNames{1} = ‘dapi’;
channelNames{2} = ‘gfp’;
channelNames{3} = ‘neun’;
channelNames
```

evaluates to

```
‘dapi’
‘gfp’
‘neun’
```

Cell array indexing

There are two different ways to index into a cell array.

1) The most common case: extract or store something into a particular cell use: `{ }` (curly brackets, braces).

2) Less common case: select part of a cell array and **keep it as a cell array, without extracting the contents**. For this you can use `( )`.

Cell array indexing

dataByCondition: each subject is a row, each condition is a column.

What if we want to select all the data for a given **subject**? But keep the filtered data in a cell array.

```
dataSubj1 = dataByCondition(1,:);
```

`dataSubj1` evaluates to `[] [] [5 10 102]`

`size(dataSubj1)` evaluates to `[1 3]`

`dataSubj1{3}` evaluates to `[5 10 102]`

Note that `dataSubj1` is still a cell array, so use `{ }` to extract the contents.

Cell array indexing

dataByCondition: each subject is a row, each condition is a column.

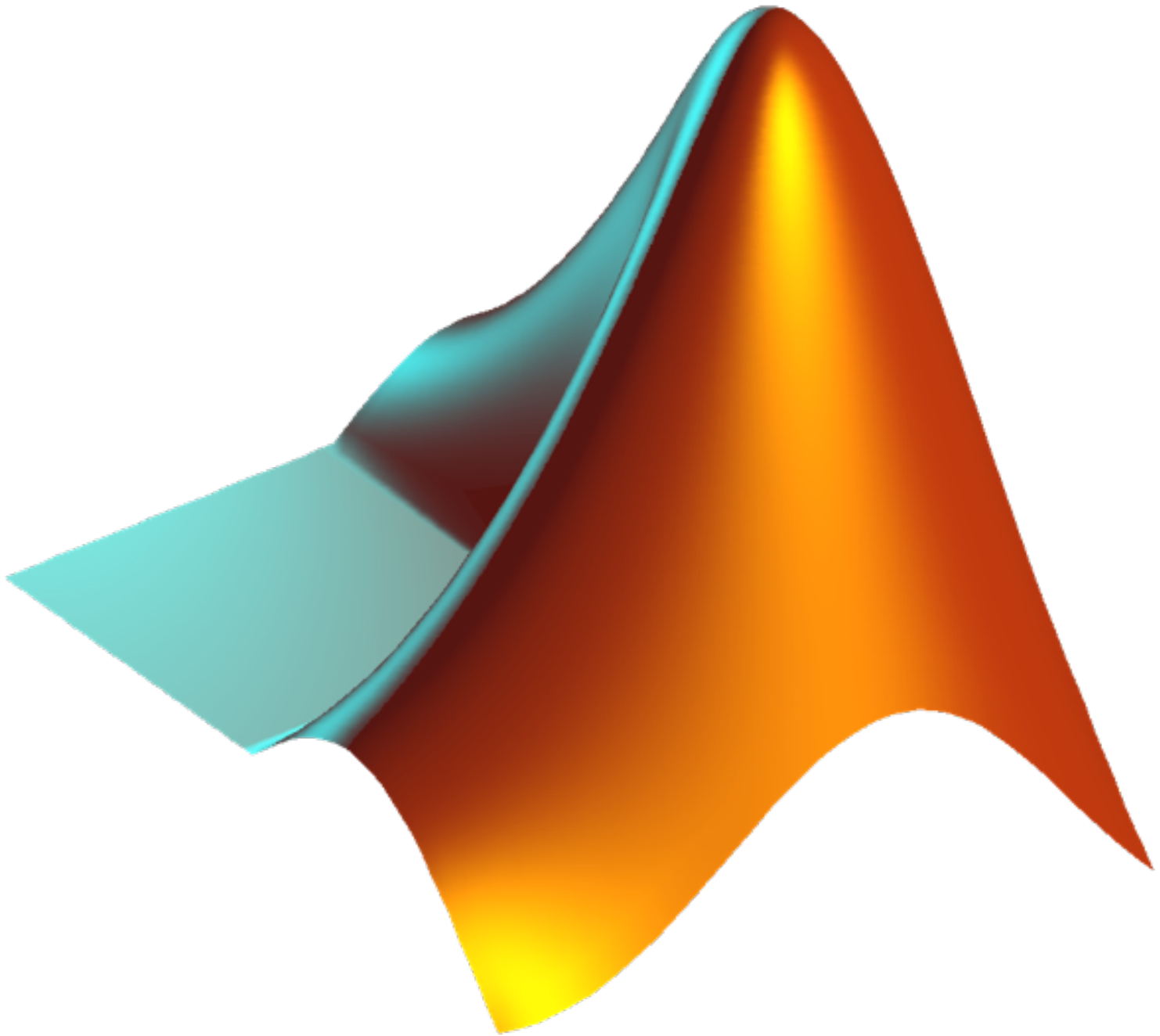
What if we want to select all the data for a given **condition**? But keep the filtered data in a cell array.

```
dataCond2 = dataByCondition(:,2);
```

`dataCond2` evaluates to

```
[]
[]
[192 10]
[]
[]
```

Demo: Cell array indexing



Cell array creation

Remember that you can create numeric arrays by placing a list of numbers in square brackets [ ] separated by spaces/commas (horizontal) or semicolons (vertical).

You can build cell arrays by listing the items inside {} brackets.

```
channelNames = {‘dapi’, ‘gfp’, ‘neun’};
```

`channelNames` evaluates to

```
‘dapi’ ‘gfp’ ‘neun’
```

`size(channelNames)` evaluates to `[1 3]`

Cell array creation

```
channelNames = {‘dapi’; ‘gfp’; ‘neun’};
```

`channelNames` evaluates to

```
‘dapi’
‘gfp’
‘neun’
```

`size(channelNames)` evaluates to `[3 1]`

Cell array creation

If you have two cell arrays that you wish to concatenate (join) together, enclose them in `[]` brackets, separated by a space/comma (horizontal) or semicolon (vertical).

If you accidentally enclose them in `{}`, you'll end up with the two cell arrays nested inside an outer cell array.

Cell array creation

```
channelNames = {‘dapi’; ‘gfp’; ‘neun’};
moreChannelNames = {‘vglut’; ‘syn’};
allChannels = [channelNames; moreChannelNames]
```

evaluates to

```
‘dapi’
‘gfp’
‘neun’
‘vglut’
‘syn’
```

`size(allChannels)` evaluates to `[5 1]`

Outline

1. **Cell arrays**
    1. Creating and concatenating cell arrays
    2. {} Indexing vs. () Indexing
2. **Struct arrays**
    1. Associating related data
    2. Structs, struct arrays, and indexing
    3. Accessing fields, aggregating data across a struct array
3. **The Matlab PATH**
    1. What is the path?
    2. How can I set the path from code?
    3. Common errors involving the path

Associating related data

Suppose we have many cells that we've patched, each with some **metadata** (like the date and the opsin it's expressing) alongside some **data** (spikes evoked vs. frequency).

How could we organize this in MATLAB?

Associating related data

Approach 1: use separate variables to hold each type of data.

```
nPatched = 3;

recordingDates = cell(nPatched,1);
constructName = cell(nPatched,1);
frequencyTracking = cell(nPatched,1);
for iNeuron = 1:nPatched
     frequencyTracking{iNeuron} = ...
          zeros(nFreq, 1);
end
```

Associating related data

Approach 1: separate variables for each type of data.

Pros:

- Easy to access all patched cells' data at once, i.e. get a list of dates on which the cells were recorded

Cons:

- Requires you to keep track of many different variables
- When passing this information to a utility function, you have to pass several different variables for a particular patched cell, rather than just one.
- If you want to select particular patched cells or remove some invalid ones, you have to do this for each of the variables you're using.

Associating related data

Approach 2: Use one big cell array where each column stores a particular type of information and each row stores a particular.

```
data = cell(nPatched, 3);
data{1,1} = ‘2011-07-09’;
data{1,2} = ‘ChR2’;
data{1,3} = zeros(nFreq,1);
data{2,1} = ‘2011-07-10’;
```

This is clunky, difficult to read, and inelegant!

Associating related data

Approach 2: Use one big cell array where each column stores a particular type of information and each row stores a particular.

Pros:

- Only one variable to keep track of
- You can filter for particular patched cells by indexing whole rows

Cons:

- You need to keep track of what each column means
- Difficult for other people to read the code

Associating related data

Approach 3: Use a structure array.

What's a structure?

Take note! Struct arrays are one of the most useful tools for organizing experimental data and one of MATLAB's key advantages over other languages! Many of us use struct arrays as the bread and butter of organizing data.

Structures

A `struct` is a collection of **fields** and their **values** that are grouped into one variable. Access fields using the 'dot' notation

```
patchData.date = '2011-07-09';
patchData.opsin = 'ChR2';
patchData.freqTracking = zeros(nFreq,1);
```

`patchData` evaluates to

```
        date: '2011-07-09'
       opsin: 'ChR2'
freqTracking: [7x1 double]
```

`patchData.date` evaluates to `'2011-07-09'`

Struct arrays

You can also create an array of structs, known as a struct array. **Each struct in the array must have the same set of fields as the others**, but the values stored in each struct can be completely different.

```
patchData(1).date = '2011-07-09';
patchData(1).opsin = 'ChR2';
patchData(1).freqTracking = zeros(nFreq,1);
patchData(2).date = '2011-07-10';
patchData(2).opsin = 'ChETA';
patchData(2).freqTracking = zeros(nFreq,1);
```

Struct arrays

You can also create an array of structs, known as a struct array. Each struct in the array must have the same set of fields as the others, but the values stored in each struct can be completely different.

`patchData` evaluates to

```
1x2 struct array with fields:
    date
    opsin
    freqTracking
```

Struct array indexing

Indexing on struct arrays works the same as indexing on numeric arrays, except you get the whole structure (or array of structures) that you've selected.

`patchData(1)` evaluates to

```
          date: '2011-07-09'
         opsin: 'ChR2'
    freqTracking: [7x1 double]
```

`patchData(1).date` evaluates to `'2011-07-09'`

Assigning into a struct array

You can either assign into the struct array one field at a time:

```
patchData(3).date = ‘2011-08-11’;

patchData(3).opsin = ‘C1V1’;

patchData(3).freqTracking = zeros(nFreq,1);
```

Or you can build a struct with identical fields in the same order and assign it in or concatenate it.

```
newData.date = ‘2011-08-11’;

newData.opsin = ‘ChETA’;

newData.freqTracking = zeros(nFreq,1);

patchData(3) = newData;
```

Grabbing all values in a certain field

What if we want to know the list of all the opsins we've used? Or all the dates we've patched on?

The idea is to grab the field from entire the struct array (without indexing first)

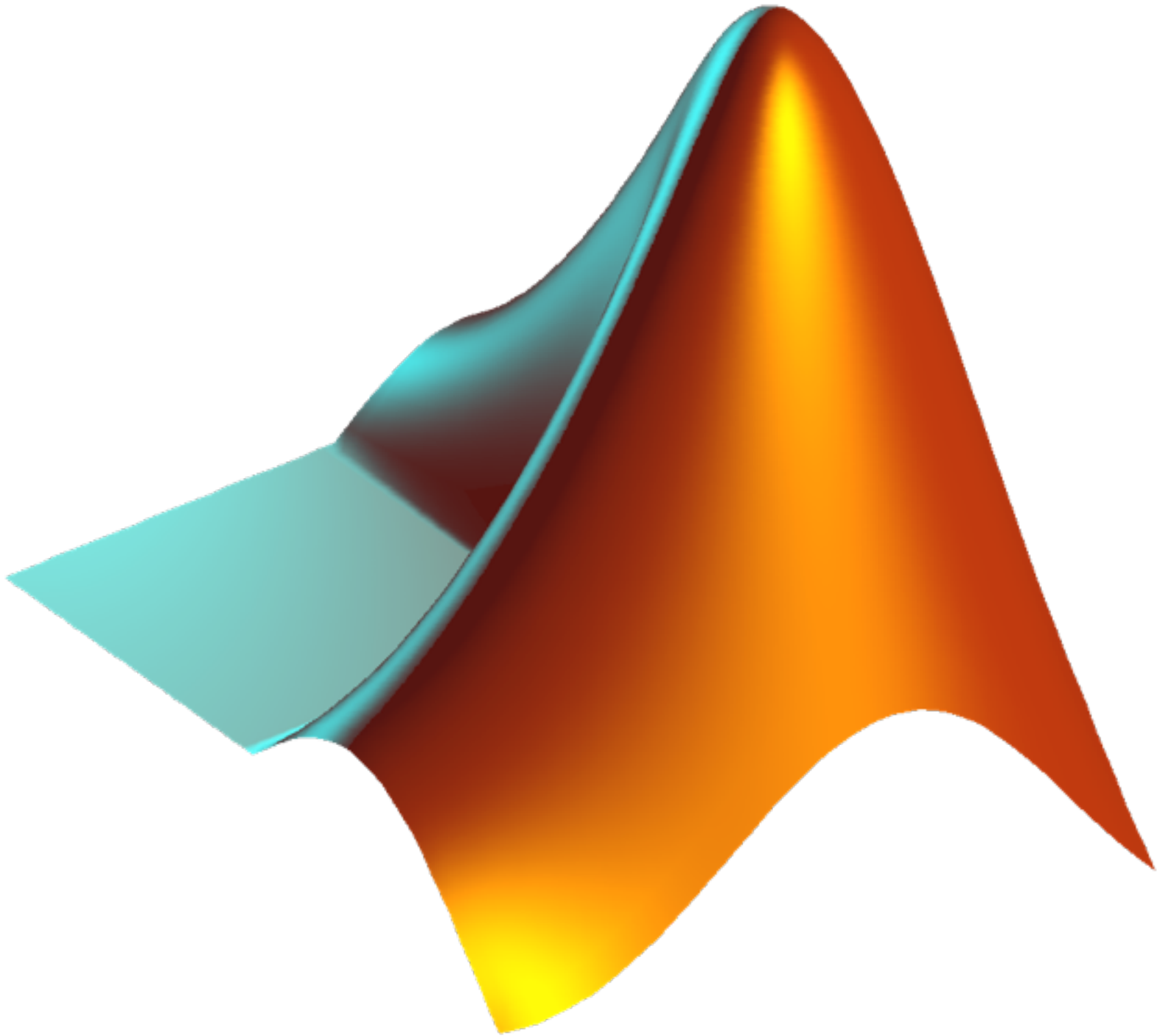
Then to "**capture**" the results in an array using `[]` or a cell array using `{}` depending on whether the contents are numeric or strings

```
opsinNamesByCell = {patchData.opsin}
```

evaluates to

```
'ChR2'
'ChETA'
'ChR2'
```

Demo: Structs and Struct Arrays



Outline

1. **Cell arrays**
    1. Creating and concatenating cell arrays
    2. {} Indexing vs. () Indexing
2. **Struct arrays**
    1. Associating related data
    2. Structs, struct arrays, and indexing
    3. Accessing fields, aggregating data across a struct array
3. **The Matlab PATH**
    1. What is the path?
    2. How can I set the path from code?
    3. Common errors involving the path

Matlab Path

- Matlab uses something called the path to find data, scripts, functions, and folders.
- Anytime you call any function, it has to be located somewhere on the path.
- There are several ways to add new folders to the path:

1. Right click in the folder browser and select “Add to path”
2. pathtool (GUI based tool for adding and saving path)
3. Command line: >> addpath()

Matlab Path

- One particularly handy code pattern:

```
addpath(genpath('/path/to/some/folder'))
```

- Can also use:

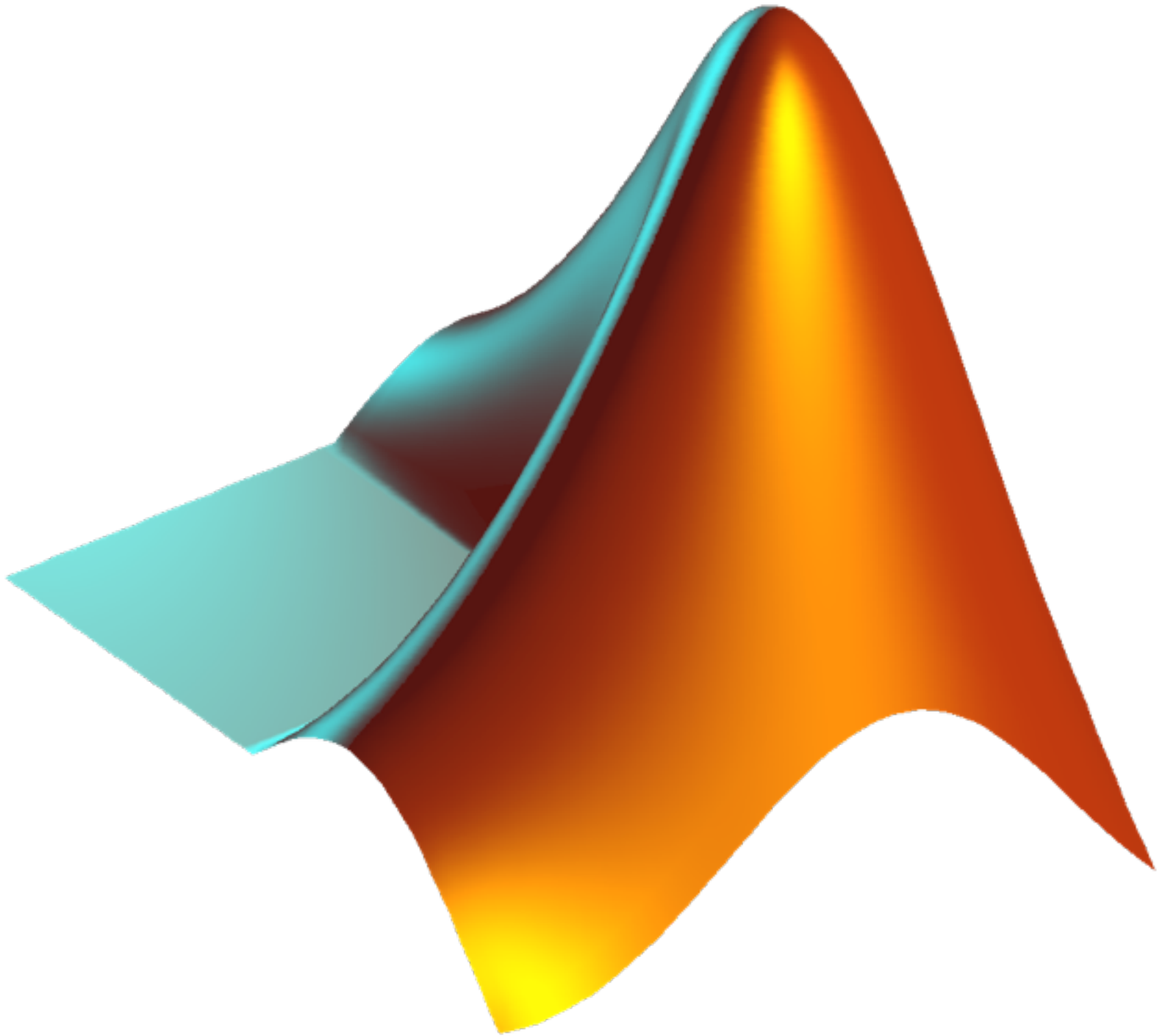
```
addpath(genpath(pwd))
```

- pwd: print working directory (returns string containing current working directory)

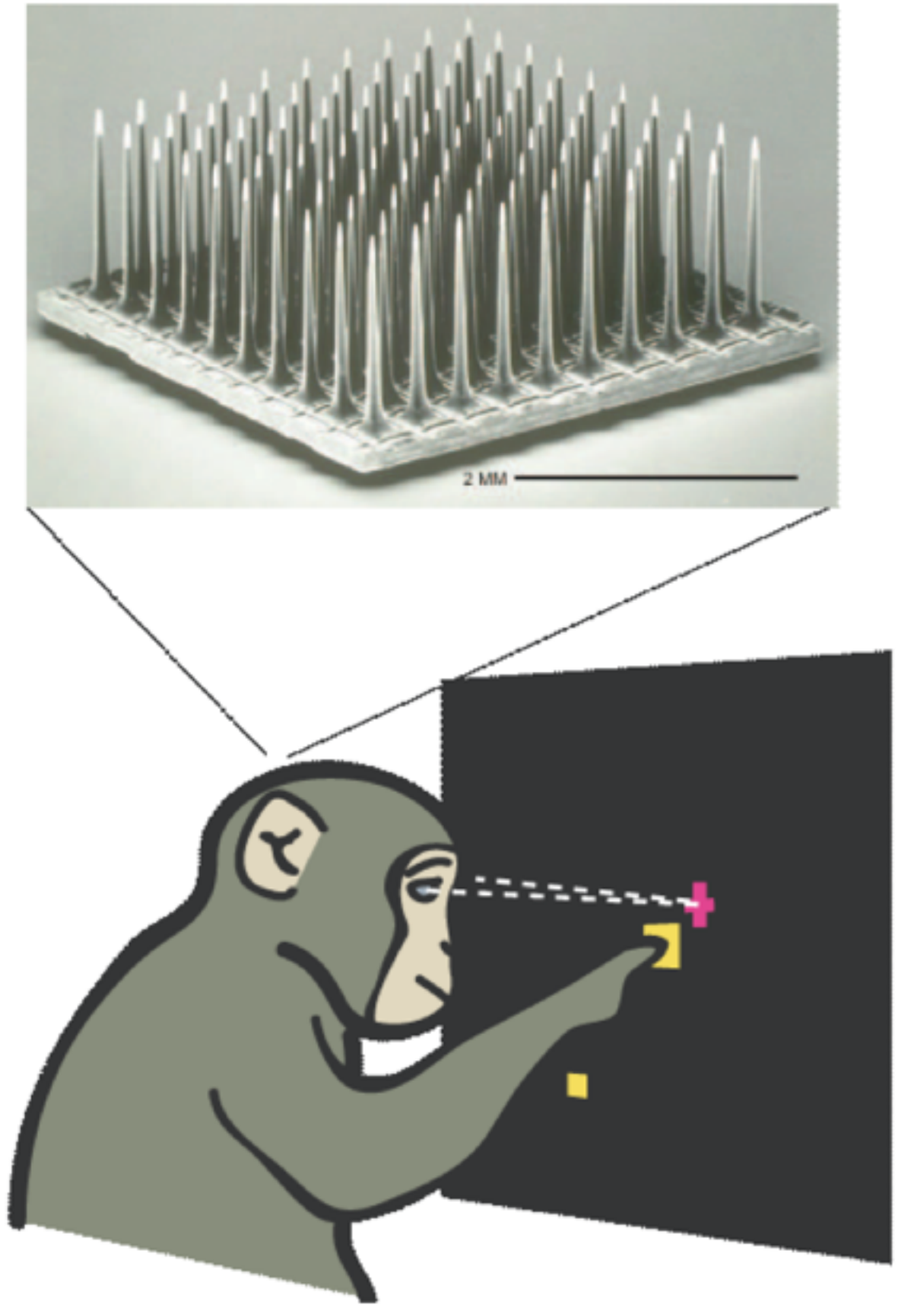
Matlab Path

- Common path errors
  1. Two functions with the same name both on the path. (this can easily happen if you download other people's code)
  2. Forgetting to add some toolbox or library folder to the path
  3. Not adding subfolders when you add a folder

Demo: Matlab Path and shortcuts



Assignment Overview



Assignment Overview