A Brief Introduction to MATLAB

MATLAB

- A *high-level* programming language
  - Graphics functions
  - Common operations are available as functions
  - Toolboxes for specific applications
- A development environment
  - Editor
  - Debugger
  - Variable browser
  - Profiler for optimization
  - GUI editor
  - On-line documentation with examples

MATLAB programs

- Programs are *interpreted*
  - Program development is fast
  - Program execution is slow (relative to C++)
  - Interaction with data is easy
  - Variables exist in a *workspace*
- Two kinds of programs
  - Scripts
    - Operate on variables in the base workspace
  - Functions
    - Input and output arguments--private workspace

Variables

- Floating point (double precision) by default

  a = 3.14

- Strings

  class = ‘QFI’

- Names can begin in upper- or lower-case letters (and are case-sensitive)

Operators

- Arithmetic operators: +, -, *, /, ^
- Comparison operators: <, >, ==
- Type help func to get help on the function or operator "func" (or use the help browser)

Expressions

- Combinations of variables and operators

  a + pi*b/5

- Interpreter will display the value of the expression unless the line ends with a semicolon (;)
- Comments begin in %

Arrays

- Scalars: a = 3;
- One dimensional arrays:
  - Row array: b = [1, 2, 3]; $$b = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$$
  - Column array: c = [1; 2; 3]; $$c = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$
- Two dimensional arrays:
  - Matrix: d = [1, 2, 3; 4, 5, 6]; $$d = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$
- Higher dimensional arrays

Naming conventions

- To remember dimensionality, I usually add a suffix:

  c_1d, d_2d, f_3d, etc

- Or more commonly

  c_v for a vector (1D array)

  d_m for a matrix (2D array)

Array indexing

- Accessing elements:

  b_1d(3) is the third element

$$b\_1d = \begin{pmatrix} 5 & 7 & 2 \end{pmatrix}$$

d_2d(1, 3) is the element in row 1, column 3

$$d\_2d = \begin{pmatrix} 1 & 2 & 4 \\ 6 & 9 & 0 \end{pmatrix}$$

Creating arrays

- Colon operator:

```
a = 1:5;
```

is the same as

```
a = [1, 2, 3, 4, 5];
```

- Create an array of zeros (2 rows, 3 columns) with

```
b = zeros(2, 3);
```

Higher dimensional arrays

- Elements of 3D arrays can be specified using three indices (row, column, and page):

  ```
  image_3d(i, j, k) = 5;
  ```

  For an image dataset, the 3^{rd} dimension could represent slice number or time index
- Create 3D arrays with predefined arrays or concatenation:

  ```
  image_3d = zeros(128, 128, 64);
  image_3d = cat(3, image1_m, image2_m);
  ```
- Any number of dimensions is possible (given sufficient memory)

Array operations

- Element-by-element operations
- Usually much faster than using for loops
- Multiply with .*

$$c\_m = a\_m.*b\_m$$

$$= \begin{pmatrix} 1 & 4 \\ 2 & 0 \end{pmatrix} .* \begin{pmatrix} 3 & 1 \\ 4 & 5 \end{pmatrix}$$

$$= \begin{pmatrix} 3 & 4 \\ 8 & 0 \end{pmatrix}$$

- Array Division with ./

$$c\_m = a\_m./b\_m$$

$$= \begin{pmatrix} 10 & 4 \\ 6 & 0 \end{pmatrix} ./ \begin{pmatrix} 5 & 1 \\ 2 & 5 \end{pmatrix}$$

$$= \begin{pmatrix} 2 & 4 \\ 3 & 0 \end{pmatrix}$$

Program control

- Loops:

```
for j = 1:5
   a = a+5;
end
```

For loop

```
j = 1;
while (j<6)
   a = a+5;
   j = j + 1;
end
```

While loop

- Conditional execution:

```
if (a == 0)
   b = 1;
end
```

If block

```
switch c
   case 1
         d = 26;
   case 2
         d = 77;
   case 3
         d = 103;
end
```

Switch statement

Using functions

- Functions take input arguments and return output variables

  a = sin(x);

- Functions are usually defined in an *M-file*
  - A text file named <function name>.m, for example

    myFunc.m

  - The file begins with the line

    function [output1, output2] = myFunc(input1, input2)

  and can have any number of input and output arguments
- *Subfunctions* are functions defined in another function's M-file
  - Can only be used within the defining M-file

Anonymous functions

- A quick way to define a one-line function:

  fHandle = @(arg list) expr

  for example

  myMean = @(x,y,z) (x+y+z)/3
- The *function handle* fHandle is a variable that can be passed to another function

Model fitting

- Fit a line (first order polynomial) to the data in x_v and y_v:

```
coeff_v = polyfit(x_v, y_v, 1);
```

where coeff_v = [slope, intercept].

- Fit the parameters p_v in more complicated models to the data in data_v:

```
pFit_v = fminsearch(@(p_v) …
        sum((myFunc(p_v, const_v) – data_v).^2));
```

where

model_v = myFunc(p_v, const_v)

fminsearch uses an unconstrained search algorithm to find the value of p_v that minimizes the sum of squared errors between myFunc and data_v.

Plotting

- Open a new window for graphics with

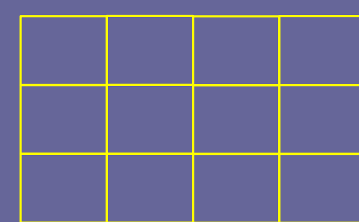
    figure

- To plot elements of y_1d versus those of x_1d use

    plot(x_1d, y_1d)

- To put more than one plot in a figure, define subplots. For 3 rows and 4 columns of subplots (with the first as the current subplot), use

    subplot(3, 4, 1)



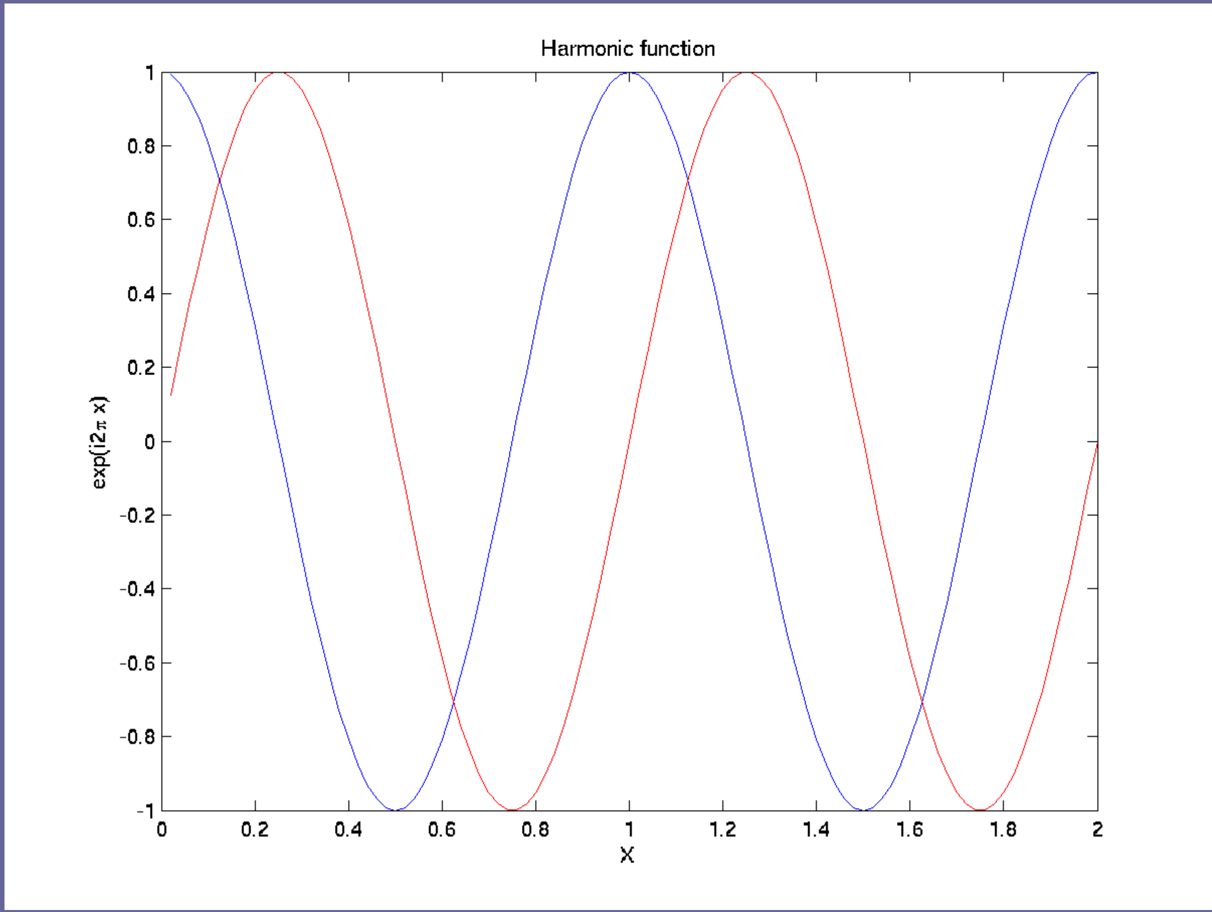
- Add the label 'time' to the x axis:

    xlabel('time')

ylabel and title add labels to the y axis and plot box

Plotting example: $e^{i2\pi x}$

```
dx = 0.02;
x_v = (1:100)*dx;
y_v = exp(1i*2*pi*x_v);
plot(x_v, real(y_v), 'b', …
    x_v, imag(y_v), 'r')
title('Harmonic function')
xlabel('X')
ylabel('exp(i2\pi x)')
```



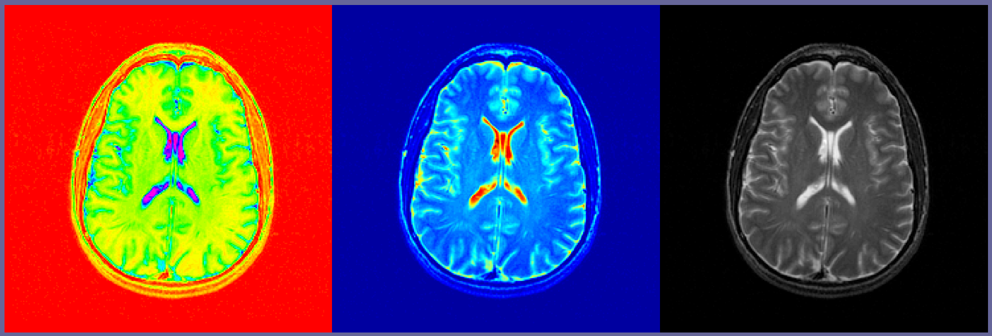
Summary

- MATLAB is a high-level programming environment
  - Optimized for array computation
  - Interaction with data
- Many similarities (but also some differences) to other languages
- Best way to learn MATLAB is to use it
- Next video:
  - A Brief Introduction to Image Analysis in MATLAB

A Brief Introduction to Image Analysis in MATLAB

Image display

- Display grayscale images with imagesc
- View in color by changing color map:
  - colormap(hsv)
  - colormap(jet)
  - colormap(gray)



- Show color image with image(color_3d)
  - Page 1: red values
  - Page 2: green values
  - Page 3: blue values

Finding image coordinates

- Define a region of interest (ROI):

  [mask_m, x_v, y_v] = roipoly;

  line(x_v, y_v, 'Color', 'y')

- Find coordinates of pixels in a mask with

  [row_v, col_v] = find(mask_m);

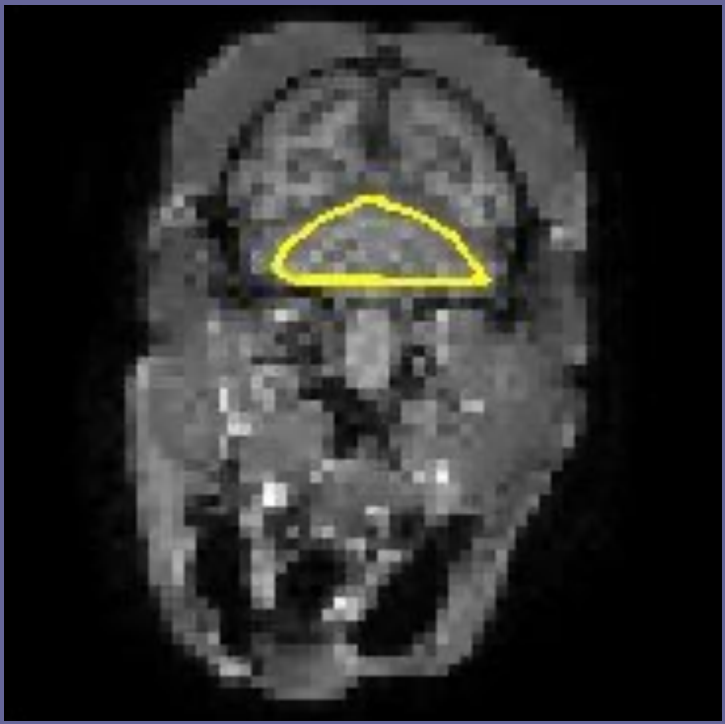
- Get coordinates of points selected with mouse clicks:

  [x_v, y_v] = ginput(n);

- Find coordinates of all pixels:

  [x_m, y_m] = meshgrid(1:nx, 1:ny);

  The rows of x_m are all 1:nx, the columns of y_m are all 1:ny.



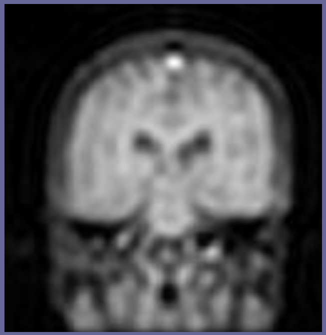
Operations on images

- Get the maximum or minimum value in an image array:

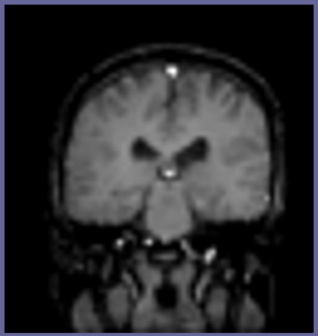
```
imageMax = max(image_m(:));
imageMin = min(image_m(:));
```

- Convolve an image with a blurring array:

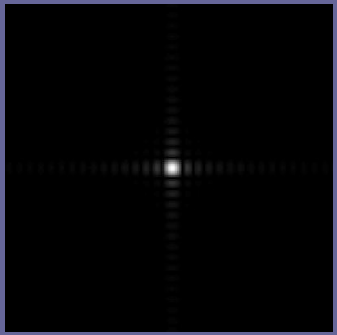
```
im2_m = conv2(im1_m, psf_m, 'same');
```



=



*



Array indexing

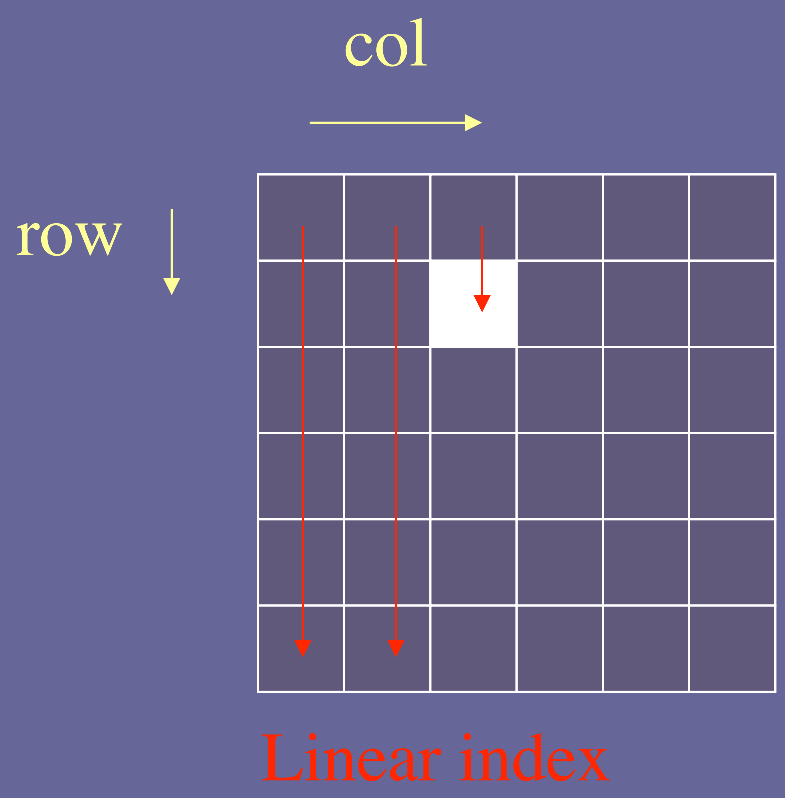
- Two ways to index a pixel in an image
  - Row, column indices
    - [row, col] = [2, 3]
  - Linear index
    - Index = 14
- Translate between these:

  [row, col] = ind2sub(siz, index)

  index = sub2ind(siz, row, col)
- Eliminate 'singleton' dimensions:

  image_3d = zeros(64,64,1);

  image_m = squeeze(image_3d);



Making movies

- Assemble images into a movie:

```
for j = 1:n
    <display image>
    F(j) = getframe;
end
movie(F)
```

Example: field mapping in MRI

- The problem:
  - The magnetic field in the body is not uniform and this can distort an MRI image.
  - If the field variation is known, it can be corrected or the images can be corrected
- Program steps
  - Read image data
  - Loop through all pixels
    - Find change in magnetization orientation with time
  - Convert from frequency to magnetic field error

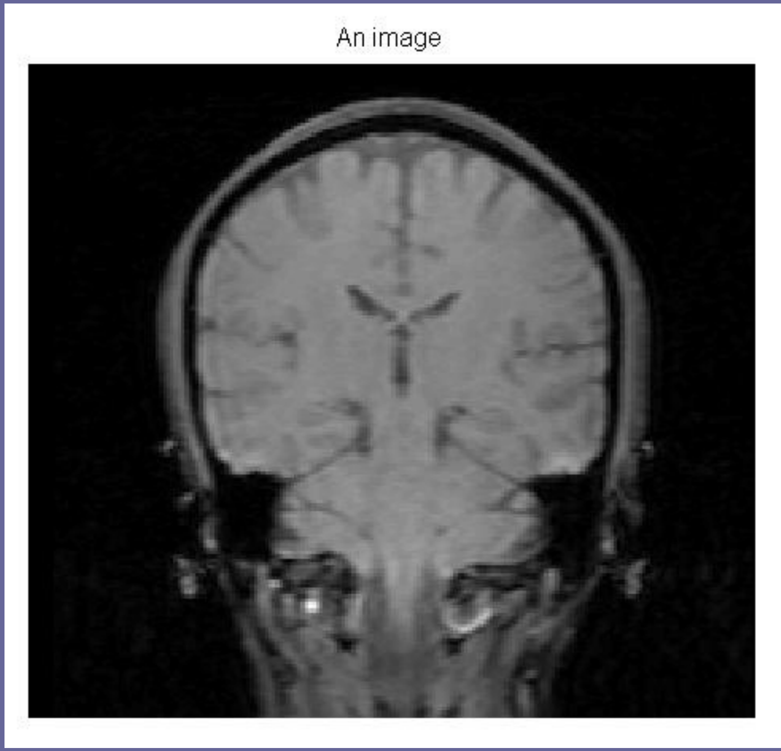
$$\delta B_z = -\omega/\gamma$$

  - Display

Read image data

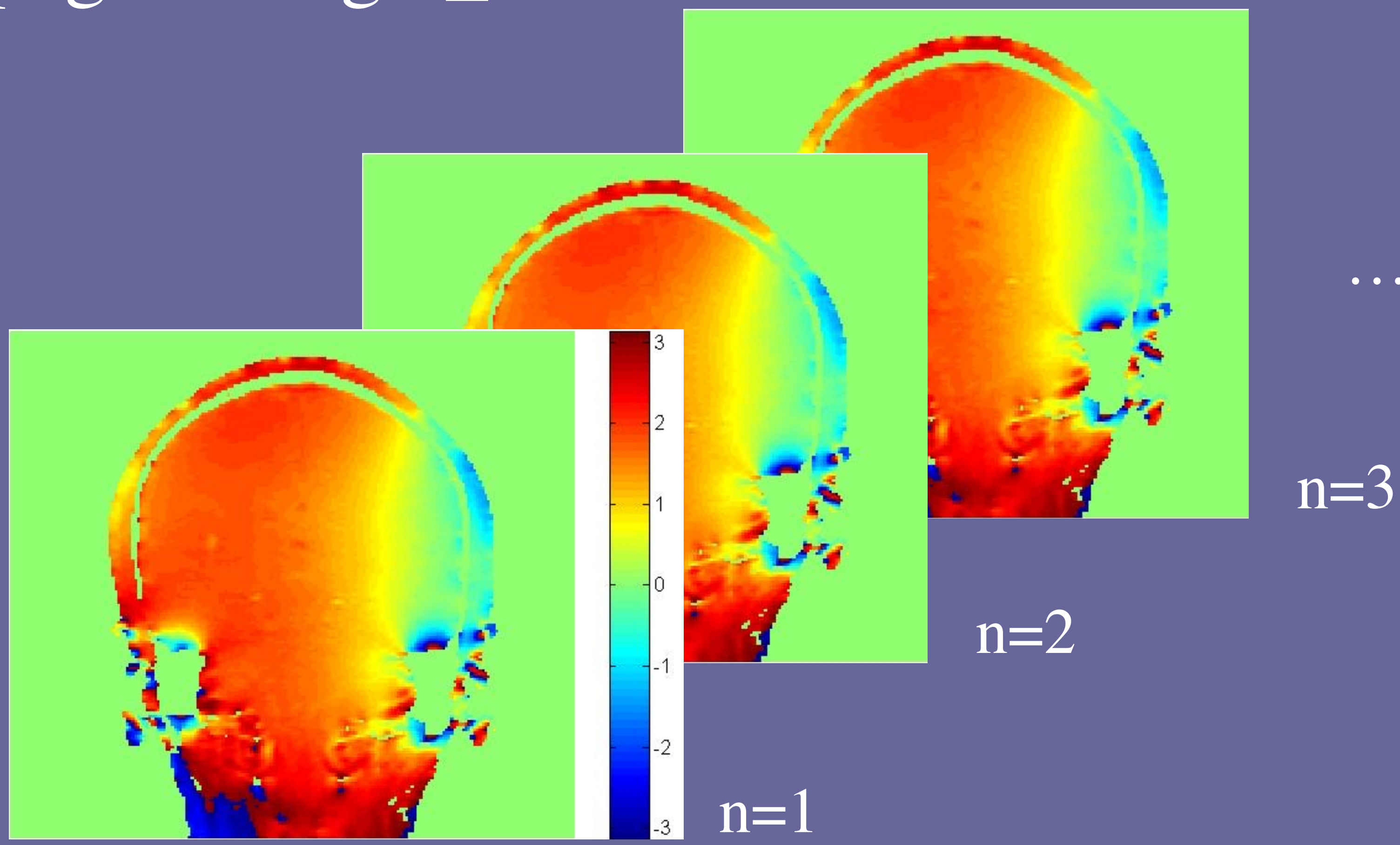
- Read and display the image matrix

```
load('myData.mat')
imagesc(image_m)
colormap(gray)
axis image
axis off
title('An image')
```



Orientation of magnetization

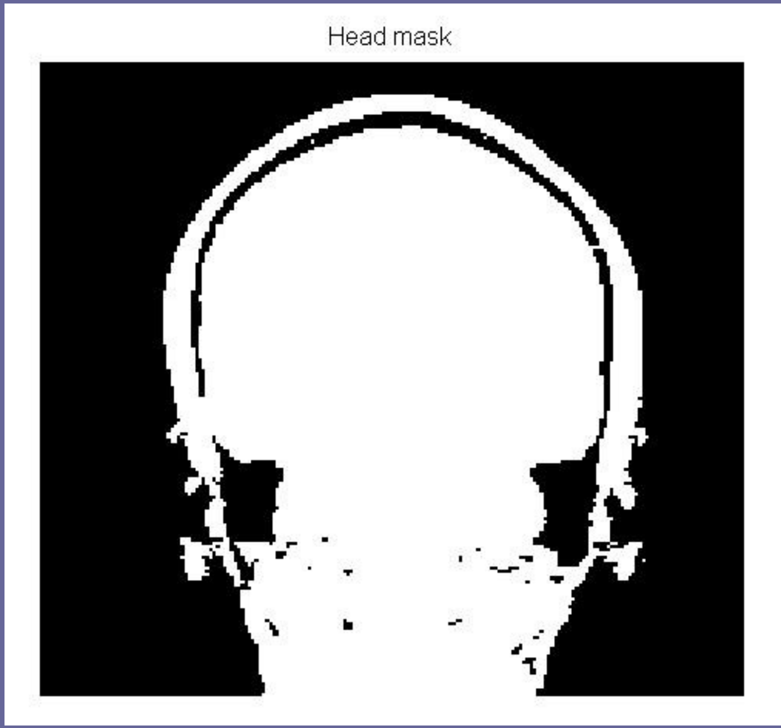
- Orientation angle at time *n* is stored in the n^{th} page of angle_3d



Ignore pixels outside the head

- Set intensity threshold at 10% of maximum:

```
mask_m = (image_m > 0.1*max(image_m(:)));
figure
imagesc(mask_m)
colormap(gray)
axis image
axis off
title('Head mask')
```



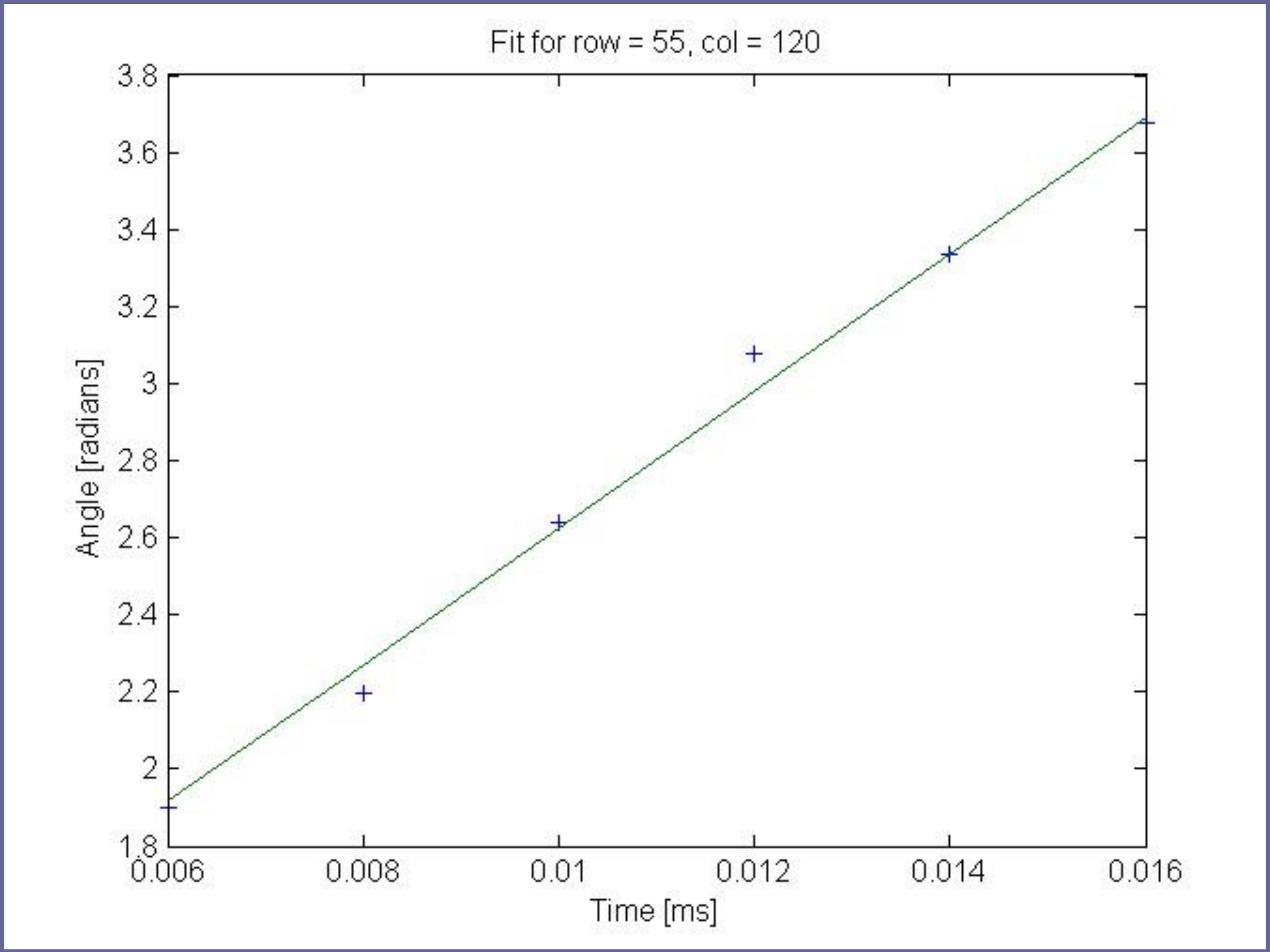
- mask_m has 1's in head

Loop through pixels in head

- Find the precession frequency for each pixel:

```
freq_m = zeros(nRows, nCols);
for row = 1:nRows
    for col = 1:nCols
        if (mask_m(row, col) > 0)
            angle_v = angle_3d(row, col, :);
            uwAngle_v = unwrap(angle_v);
            a_v = polyfit(time_v, uwAngle_v, 1);
            slope = a_v(1);
            intercept = a_v(2);
            % Store frequency:
            freq_m(row, col) = slope;
        end
    end
end
```

Angle versus time for one pixel

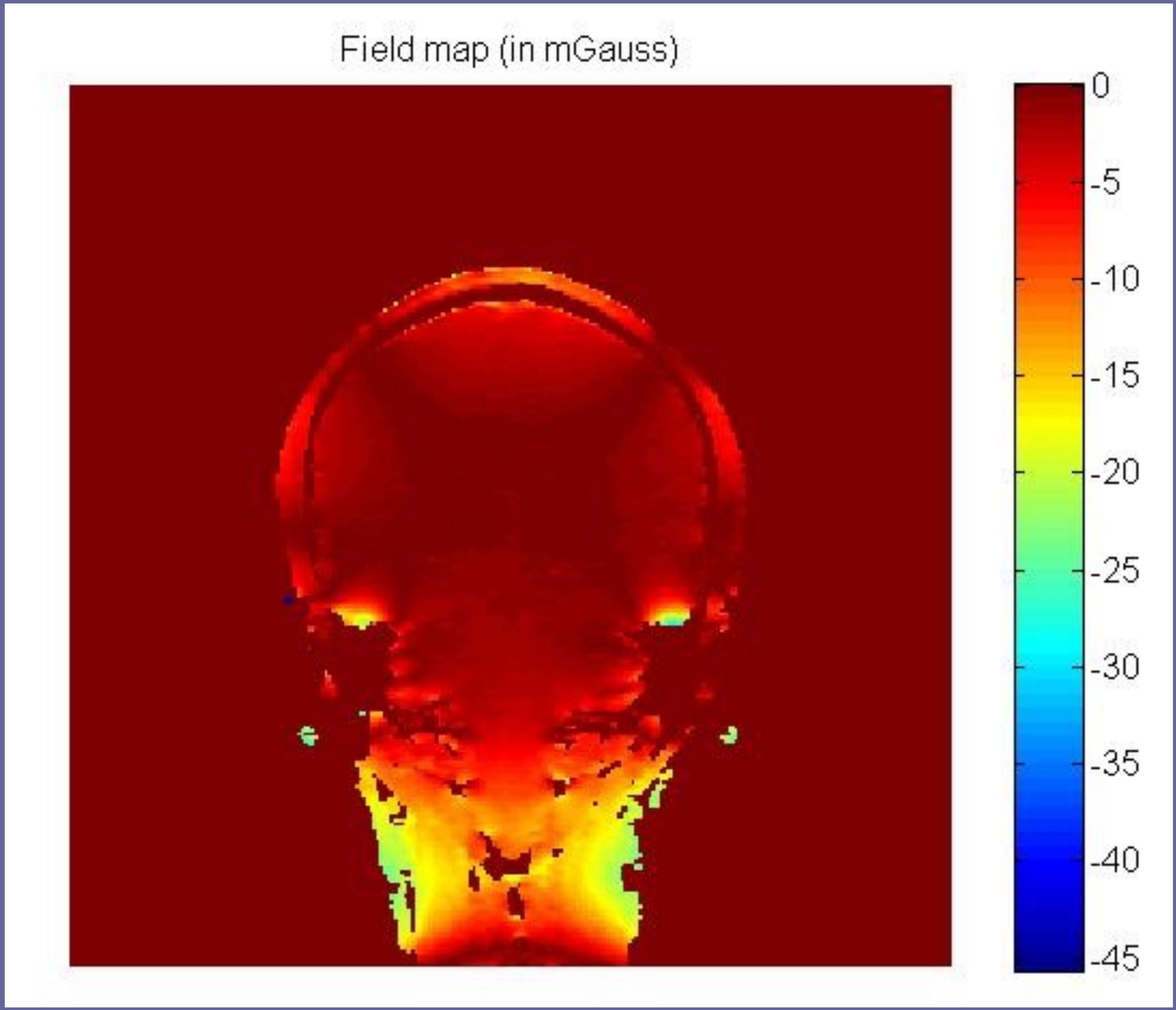


Convert frequencies to field errors

- Use Larmor relation: $\delta B_z = -\omega/\gamma$

```
field_m = -freq_m / (2*pi*4.26);
figure
imagesc(field_m)
axis image; axis off
colormap(jet)
colorbar
title('Field map (in mGauss)')
```

Map of magnetic field errors



Results of calculation

- Field errors, $B_z - B_0$, are small in the center of the brain, larger near the edges
  - Why?
- How could the errors be reduced?

Summary

- MATLAB is a high-level language that facilitates interacting with data
  - Easy to view intermediate results
  - Test algorithms
- Provides many useful functions for data analysis and modeling
  - Image display
  - ROI definition
  - Mask generation
  - Data fitting functions
- Will be our main tool for data analysis projects