

Athinoula A.

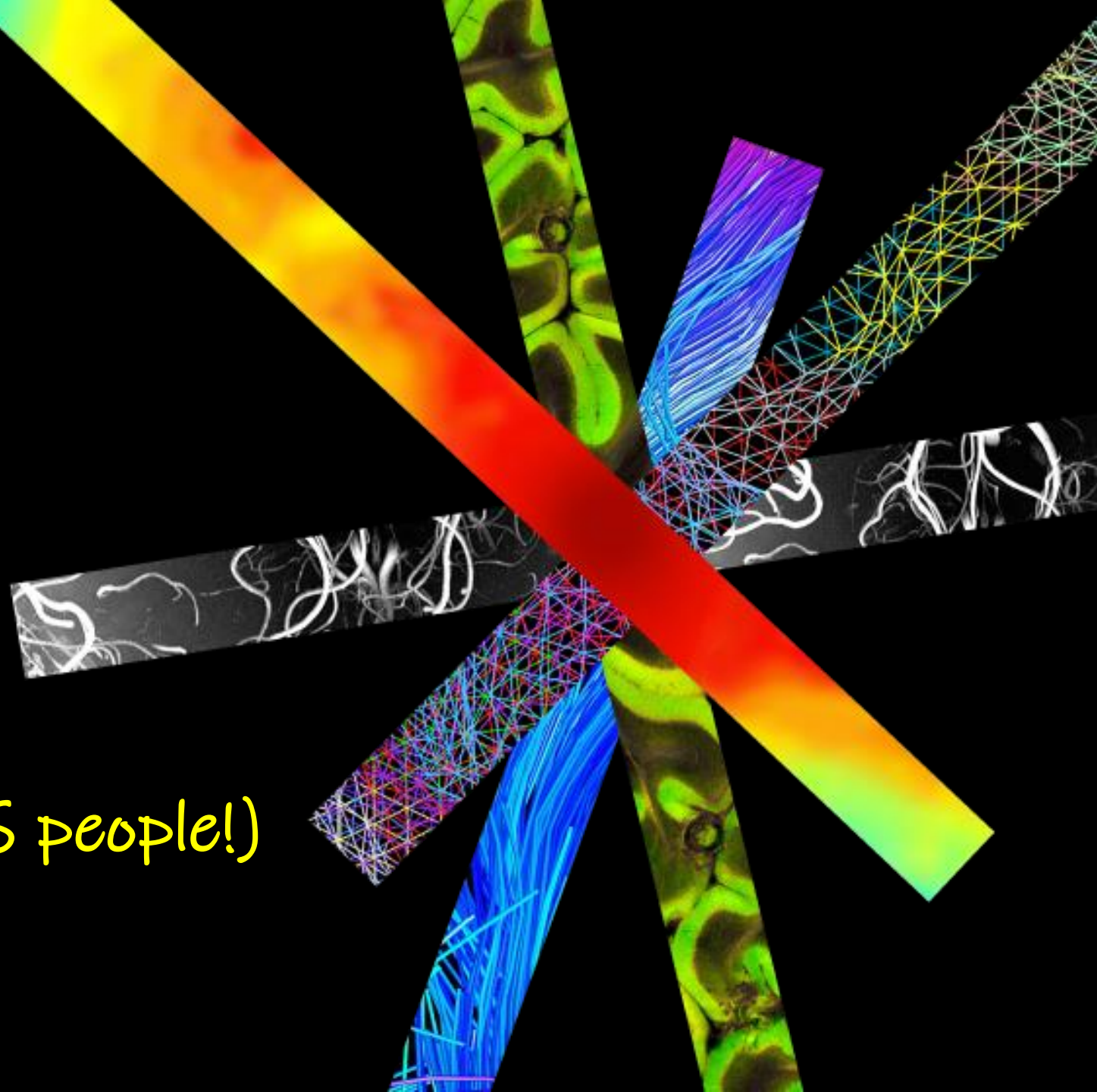
# Martinos Center

For Biomedical Imaging

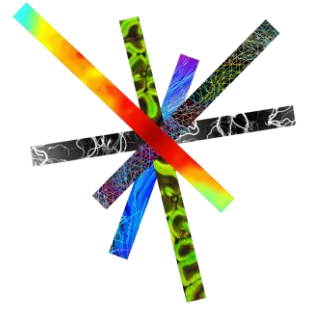
# Introduction to MATLAB<sup>®</sup> *(for non-CS people!)*

*Michele Scipioni, PhD*

3/12/2020



# Who am I?



Michele Scipioni  
Biomedical Engineer

**MSc + PhD @ University of Pisa, Italy**

**#PET**

**#ImageReconstruction**

**#KineticModeling**

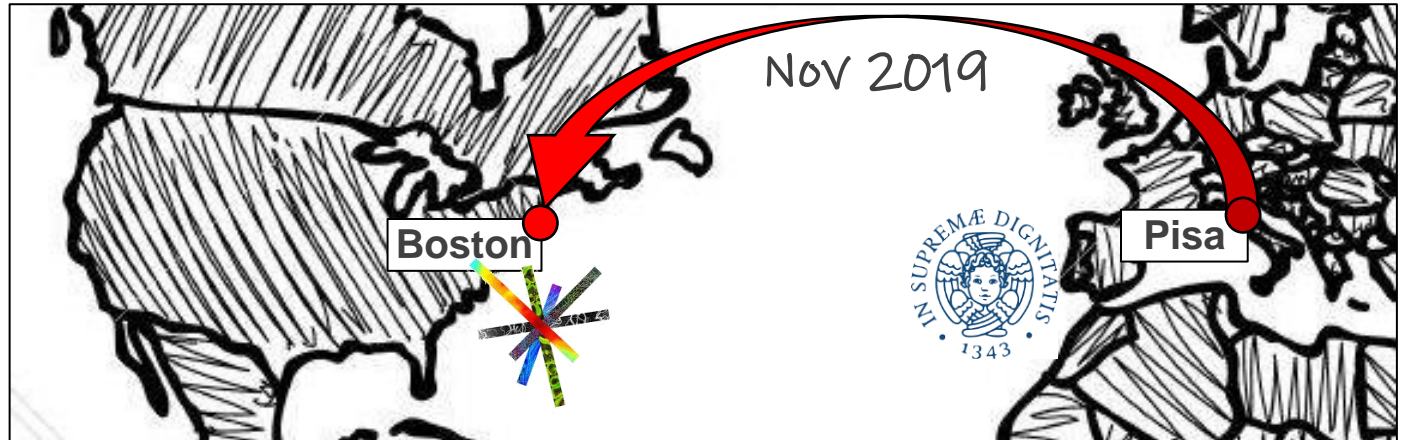
**Postdoc @ Martinos Center**

**#PET + PET/MR**

**#ImageReconstruction**

**#KineticModeling**

**#PET/MR tech design and development**

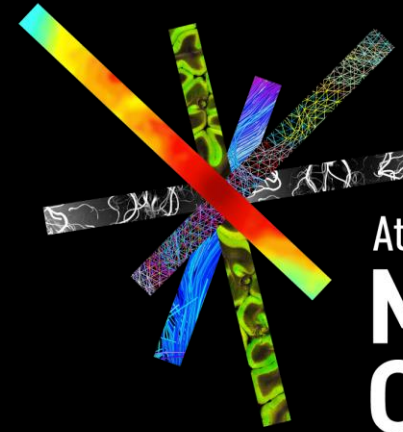


Ciprian Catana's  
PET-MR lab



# Overview

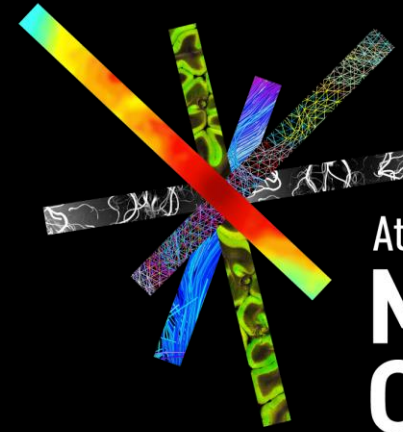
- **WHAT?**
- **WHY?**
- **HOW?**
  - GETTING STARTED
  - SCRIPTS, FUNCTIONS, AND THE EDITOR
  - VISUALIZATION TOOLS
- **BUT ... CAN I STILL USE IT, IF I DON'T KNOW HOW TO CODE?**



Athinoula A.  
**Martinos**  
**Center**  
For Biomedical Imaging

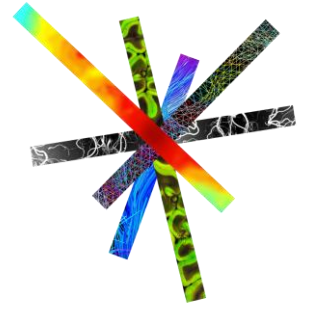
# Overview

- **WHAT?**
- **WHY?**
- **HOW?**
  - GETTING STARTED
  - SCRIPTS, FUNCTIONS, AND THE EDITOR
  - VISUALIZATION TOOLS
- **BUT ... CAN I STILL USE IT, IF I DON'T KNOW HOW TO CODE?**



Athinoula A.  
**Martinos**  
**Center**  
For Biomedical Imaging

# General background: what are we talking about?



## What is MATLAB?

**MATLAB** = **MAT**rix **LAB**oratory

- High-level *scripting* language
- Interactive *visualization* tool
- Interactive *computation* tool

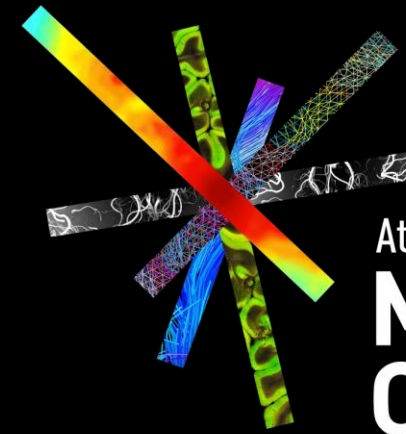
## What can I do with MATLAB?

- **Automate** complex data processing streams.
- **Analyze** data.
- **Develop** algorithms.
- **Create** models and applications.
- **Write your own** data analysis/computation tools.

MATLAB  
is *complete package* made  
of a programming language,  
computing environment, IDE,  
and many toolboxes for data  
processing and plotting.

# Overview

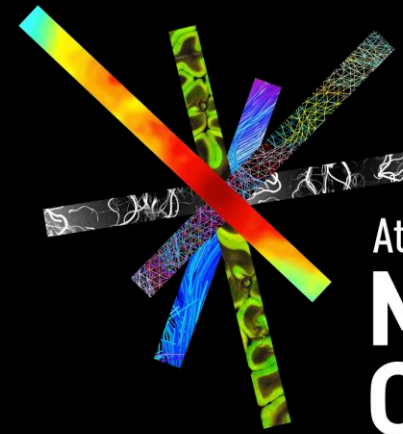
- **WHAT?**
- **WHY?**
- **HOW?**
  - GETTING STARTED
  - SCRIPTS, FUNCTIONS, AND THE EDITOR
  - VISUALIZATION TOOLS
- **BUT ... CAN I STILL USE IT, IF I DON'T KNOW HOW TO CODE?**



Athinoula A.  
**Martinos**  
**Center**  
For Biomedical Imaging

# Overview

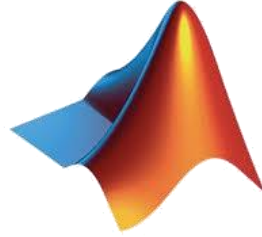
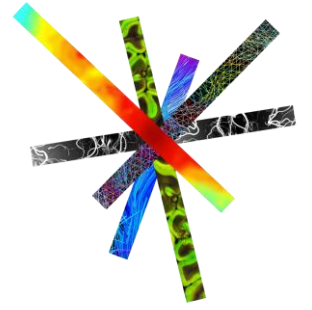
- **WHAT?**
- **WHY?**
- **HOW?**
  - GETTING STARTED
  - SCRIPTS, FUNCTIONS, AND THE EDITOR
  - VISUALIZATION TOOLS
- **BUT ... CAN I STILL USE IT, IF I DON'T KNOW HOW TO CODE?**



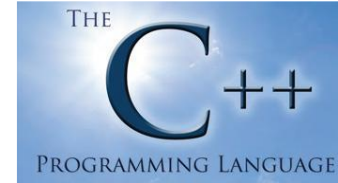
Athinoula A.  
**Martinos**  
**Center**  
For Biomedical Imaging



# Matlab vs C / C++ / Fortran



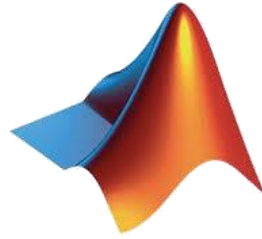
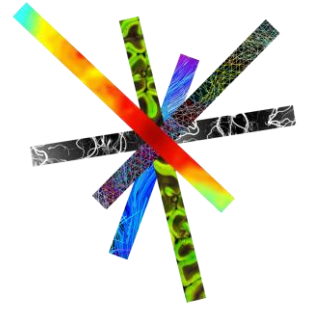
**high level** language  
**easy to learn**  
professionally developed **tools**  
**and built-in functions**  
user-friendly **GUI**  
(very expensive) **commercial product**



**compiled** language  
**(significantly) faster**  
**general-purpose**



# Matlab vs Python

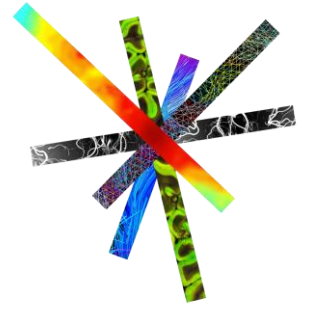
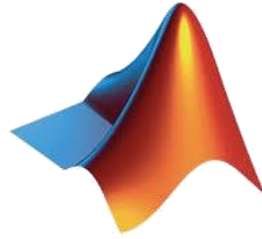


**interpreted** languages  
easy multi-OS **portability**  
sub-optimal **performance** (wrt C/C++)

**high level** language  
**easy to learn**  
professionally developed **tools**  
**and built-in functions**  
user-friendly **GUI**  
(very expensive) **commercial product**

**general-purpose**  
**open and free**  
**open source libraries**  
go-to language for **machine**  
**learning and data science** (at the  
moment)

# Matlab vs R



**faster!**

**easy to learn** and intuitive

professionally developed **tools and built-in functions**

user-friendly **GUI**

**'can' do statistics and ML**, but also much more

(very expensive) **commercial product**

**syntax** closer to conventional languages

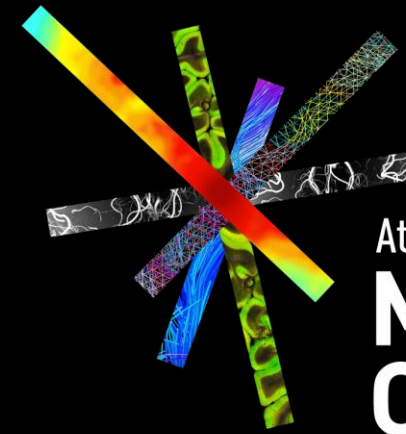
**open and free**

**open source libraries**

go-to language **data analysis and statistics**  
(at the moment)

# Overview

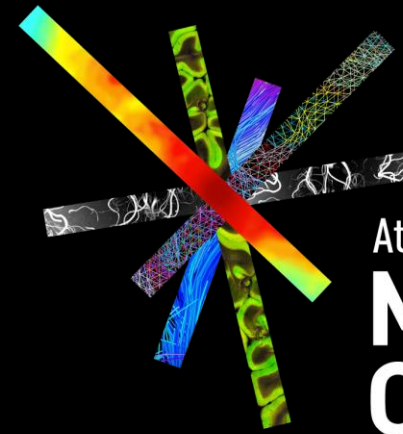
- **WHAT?**
- **WHY?**
- **HOW?**
  - GETTING STARTED
  - SCRIPTS, FUNCTIONS, AND THE EDITOR
  - VISUALIZATION TOOLS
- **BUT ... CAN I STILL USE IT, IF I DON'T KNOW HOW TO CODE?**



Athinoula A.  
**Martinos  
Center**  
For Biomedical Imaging

# Overview

- **WHAT?**
- **WHY?**
- **HOW?**
  - GETTING STARTED
  - SCRIPTS, FUNCTIONS, AND THE EDITOR
  - VISUALIZATION TOOLS
- **BUT ... CAN I STILL USE IT, IF I DON'T KNOW HOW TO CODE?**

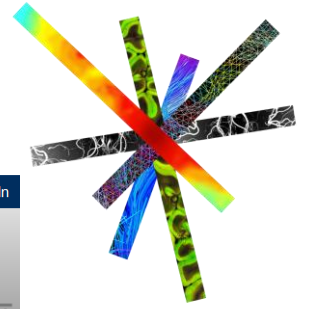


Athinoula A.

**Martinos  
Center**

For Biomedical Imaging

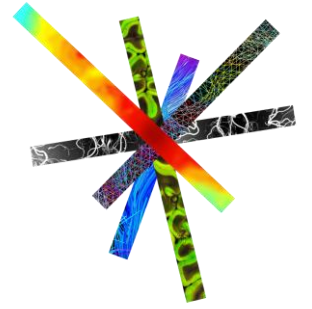
# MATLAB Graphical User Interface (GUI)



The screenshot shows the MATLAB R2018b GUI with the following components and callouts:

- 1. Command window:** The central area where code is executed. It shows the command `>> a = [1 2 ; 3 4]` and the resulting matrix `a = [1 2; 3 4]`.
- 2. Command history:** A window at the bottom right showing the history of executed commands, including `title('biqaa');`, `subplot(2,3,2), boxplot(score.piqe_mat, 'No... title('piqe');`, `subplot(2,3,3), boxplot(score.brisque, 'No... title('brisque');`, `subplot(2,3,3), boxplot(score.niqe_mat, 'No... title('niqe');`, and `main`.
- 3. Workspace:** A window at the top right showing the current workspace variables. It lists variable `a` with value `[1,2;3,4]`.
- 4. File explorer:** A window on the left showing the current folder structure, including `BISHARP_code` and `matlabPyrTools-master`.
- 5. Toolbar:** A horizontal bar at the top containing various icons for file operations, code execution, and preferences.

# MATLAB syntax – Looking for help!

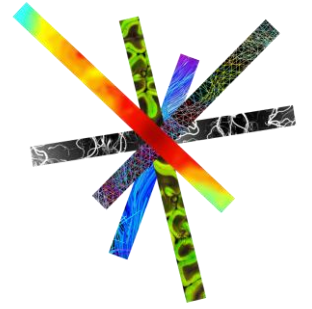


**Don't be scared to ask for help!**

In many case, the documentation texts are quite informative and educational.

```
>> help           % lists available packages/toolboxes on system.  
>> help elfun    % lists functions in elementary functions package  
>> help sin      % instructions on the sine function  
>> lookfor sine  % if you don't know the function name ...  
>> doc sin       % for full details o ffunction
```

# MATLAB syntax – Looking for help!



**Don't be scared to ask for help!**

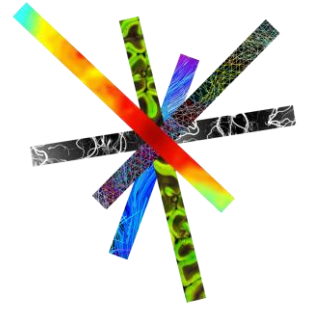
In many case, the documentation texts are quite informative and educational.

```
>> help  
>> help elfun  
>> help sin  
>> lookfor sine  
>> doc sin
```

```
>> help sin  
sin    Sine of argument in radians.  
    sin(X) is the sine of the elements of X.  
  
See also asin, sind, sinpi.  
  
Reference page for sin  
Other functions named sin  
  
>>
```



# MATLAB syntax – Looking for help!



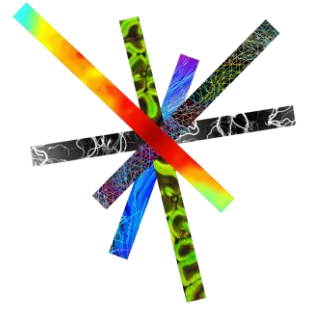
**Don't be scared to ask for help!**

In many case, the documentation texts are quite informative and educational.

```
>> help  
>> help elfun  
>> help sin  
>> lookfor sine  
>> doc sin
```

The screenshot shows the MATLAB Help browser interface. The main content area displays the documentation for the `sin` function. The title is `sin` (R2018b), with the subtitle "Sine of argument in radians". The **Syntax** section shows `Y = sin(X)`. The **Description** section explains that `sin(X)` returns the sine of the elements of `X`, operating element-wise on arrays. The **Examples** section includes a "Plot Sine Function" example with the code `x = -pi:0.01:pi; plot(x,sin(x)), grid on` and a corresponding plot of the sine wave. The left sidebar shows a "CONTENTS" menu with options like "Documentation Home", "MATLAB", "Mathematics", "Elementary Math", and "Trigonometry".

# MATLAB syntax – Matrices, vectors, arrays ...



## Scalar

```
>> s = 5; % no need to specify data type (default is double)
```

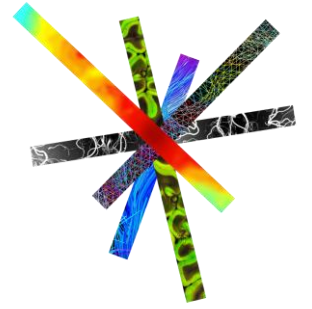
## Vector

```
>> a = [1, 2, 3]; % row vector  
>> b = [4; 5; 6]; % column vector
```

## Matrix

```
>> A = [1, 2, 3 ; 4, 5, 6 ; 7, 8, 9]; % 3 x 3 matrix
```

# MATLAB syntax – Matrices, vectors, arrays ...



## Scalar

```
>> s = 5; % no need to specify data type (default is double)
```

## Vector

```
>> a = [1, 2, 3]; % row vector  
>> b = [4; 5; 6]; % column vector
```

```
>> a = [1, 2, 3]
```

```
a =  
  
     1     2     3
```

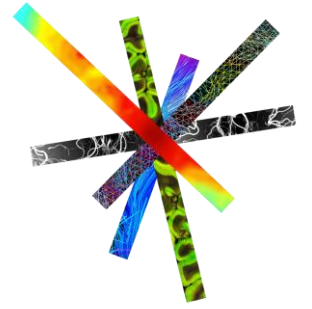
```
>> size(a)
```

```
ans =  
  
     1     3
```

## Matrix

```
>> A = [1, 2, 3 ; 4, 5, 6 ; 7, 8, 9]; % 3 x 3 matrix
```

# MATLAB syntax – Matrices, vectors, arrays ...



## Scalar

```
>> s = 5; % no need to specify data type (default is double)
```

## Vector

```
>> a = [1, 2, 3]; % row vector  
>> b = [4; 5; 6]; % column vector
```

```
>> b = [4; 5; 6]
```

```
>> size(b)
```

```
b =
```

```
ans =
```

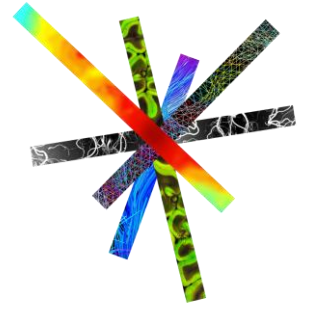
```
4  
5  
6
```

```
3 1
```

## Matrix

```
>> A = [1, 2, 3 ; 4, 5, 6 ; 7, 8, 9]; % 3 x 3 matrix
```

# MATLAB syntax – Matrices, vectors, arrays ...



## Scalar

```
>> s = 5; % no need to specify data type (default is double)
```

## Vector

```
>> a = [1, 2, 3]; % row vector
```

```
>> b = [4; 5; 6]; % column vector
```

A =

1	2	3
4	5	6
7	8	9

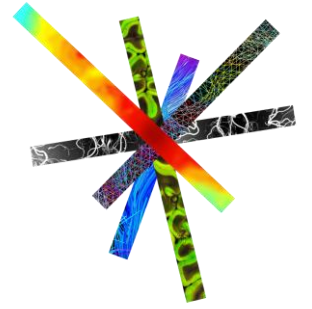
## Matrix

```
>> A = [1, 2, 3 ; 4, 5, 6 ; 7, 8, 9]; % 3 x 3 matrix
```

Use percent (%) sign to start a comment (everything after it **IS NOT** code)

Suppress (interactive console) output by adding a **semicolon (;)** at the end of each line

# MATLAB syntax – Matrices, vectors, arrays ...



## Scalar

```
>> s = 5; % no need to specify data type (default is double)
```

## Vector

```
>> a = [1, 2, 3]; % row vector  
>> b = [4; 5; 6]; % column vector
```

```
>> A = [1, 2, 3 ; 4, 5, 6 ; 7, 8, 9];  
>> A = [1, 2, 3 ; 4, 5, 6 ; 7, 8, 9]
```

A =

1	2	3
4	5	6
7	8	9

## Matrix

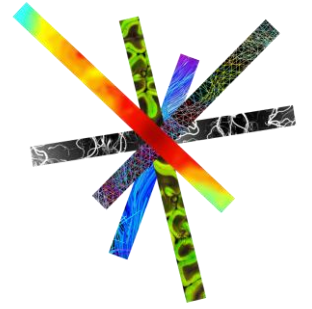
```
>> A = [1, 2, 3 ; 4, 5, 6 ; 7, 8, 9]; % 3 x 3 matrix
```

Use percent (%) sign to start a comment (everything after it **IS NOT** code)

Suppress (interactive console) output by adding a **semicolon (;)** at the end of each line

# MATLAB syntax – Matrices, vectors, arrays ...

## FUNCTIONS TO CREATE MATRICES

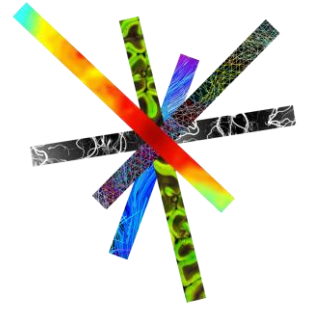


```
>> zeros(5,3); % All zeros
>> ones(8,5); % All ones
>> eye(5); % Identity matrix
>> rand(3,9); % Uniformly distributed random numbers (between 0 and 1)
>> randn(10,5); % Normally distributed random numbers (mean 0 and var 1)
```



# MATLAB syntax – Matrices, vectors, arrays ...

## MATRIX INDEXING / SLICING



- In MATLAB matrix and vector **indexing start from 1 (not from 0)**.
- It uses a **column-major** convention (it affects reshaping and transpositions)

A =

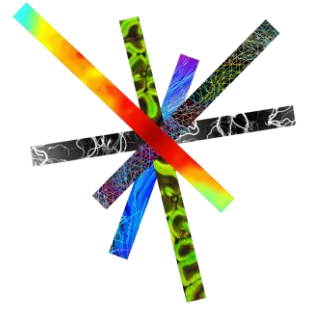
68	59	96	82	48	76	78
66	23	55	25	36	76	94
17	76	14	93	84	39	13
12	26	15	35	59	57	57
50	51	26	20	55	8	47
96	70	85	26	92	6	2
35	90	26	62	29	54	34

```
>> A(3,2)           % Access a single element (3rd row, 2nd col)
```

```
>> disp(A(3,2))  
76
```

# MATLAB syntax – Matrices, vectors, arrays ...

## MATRIX INDEXING / SLICING



- In MATLAB matrix and vector **indexing start from 1 (not from 0)**.
- It uses a **column-major** convention (it affects reshaping and transpositions)

A =

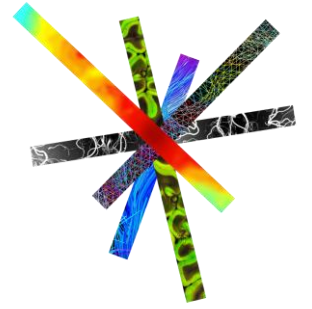
68	59	96	82	48	76	78
66	23	55	25	36	76	94
17	76	14	93	84	39	13
12	26	15	35	59	57	57
50	51	26	20	55	8	47
96	70	85	26	92	6	2
35	90	26	62	29	54	34

```
>> A(3,2)           % Access a single element (3rd row, 2nd col)
>> A(:,1)          % Select the whole 1° column
```

```
>> disp(A(:,1))
68
66
17
12
50
96
35
```

# MATLAB syntax – Matrices, vectors, arrays ...

## MATRIX INDEXING / SLICING



- In MATLAB matrix and vector **indexing start from 1 (not from 0)**.
- It uses a **column-major** convention (it affects reshaping and transpositions)

A =

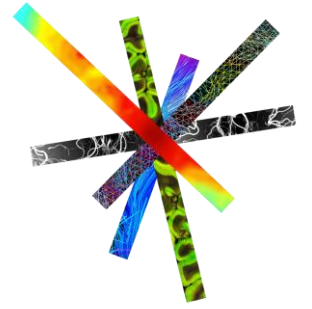
68	59	96	82	48	76	78
66	23	55	25	36	76	94
17	76	14	93	84	39	13
12	26	15	35	59	57	57
50	51	26	20	55	8	47
96	70	85	26	92	6	2
35	90	26	62	29	54	34

```
>> A(3,2)           % Access a single element (3rd row, 2nd col)
>> A(:,1)           % Select the whole 1° column
>> A(2,2:5)         % Select a subset of 2° row
```

```
>> disp(A(2,2:5))
    23    55    25    36
```

# MATLAB syntax – Matrices, vectors, arrays ...

## MATRIX INDEXING / SLICING



- In MATLAB matrix and vector **indexing start from 1 (not from 0)**.
- It uses a **column-major** convention (it affects reshaping and transpositions)

A =

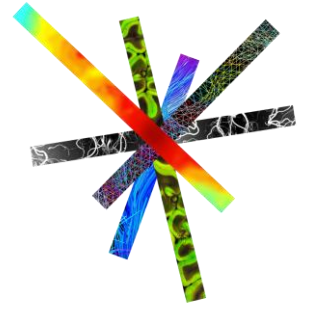
68	59	96	82	48	76	78
66	23	55	25	36	76	94
17	76	14	93	84	39	13
12	26	15	35	59	57	57
50	51	26	20	55	8	47
96	70	85	26	92	6	2
35	90	26	62	29	54	34

```
>> A(3,2)           % Access a single element (3rd row, 2nd col)
>> A(:,1)           % Select the whole 1° column
>> A(2,2:5)         % Select a subset of 2° row
>> sum(A(2,:))      % Sum all elements of 2° row
```

```
>> disp(sum(A(2,:)))
375
```

# MATLAB syntax – Matrices, vectors, arrays ...

## MATRIX INDEXING / SLICING



- In MATLAB matrix and vector **indexing start from 1 (not from 0)**.
- It uses a **column-major** convention (it affects reshaping and transpositions)

A =

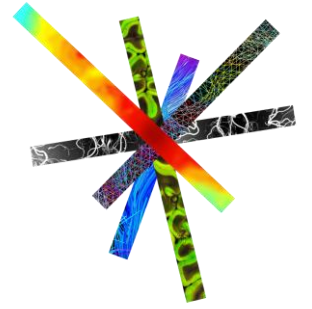
68	59	96	82	48	76	78
66	23	55	25	36	76	94
17	76	14	93	84	39	13
12	26	15	35	59	57	57
50	51	26	20	55	8	47
96	70	85	26	92	6	2
35	90	26	62	29	54	34

```
>> A(3,2)           % Access a single element (3rd row, 2nd col)
>> A(:,1)           % Select the whole 1° column
>> A(2,2:5)         % Select a subset of 2° row
>> sum(A(2,:))      % Sum all elements of 2° row
>> max(A(:,3))      % Max value of 3° column
```

```
>> disp(max(A(:,3)))
96
```

# MATLAB syntax – Matrices, vectors, arrays ...

## MATRIX INDEXING / SLICING



- In MATLAB matrix and vector **indexing start from 1 (not from 0)**.
- It uses a **column-major** convention (it affects reshaping and transpositions)

A =

68	59	96	82	48	76	78
66	23	55	25	36	76	94
17	76	14	93	84	39	13
12	26	15	35	59	57	57
50	51	26	20	55	8	47
96	70	85	26	92	6	2
35	90	26	62	29	54	34

```
>> A(3,2)           % Access a single element (3rd row, 2nd col)
>> A(:,1)           % Select the whole 1° column
>> A(2,2:5)         % Select a subset of 2° row
>> sum(A(2,:))      % Sum all elements of 2° row
>> max(A(:,3))      % Max value of 3° column
>> find(isprime(A)) % Index of prime numbers among all elements
```

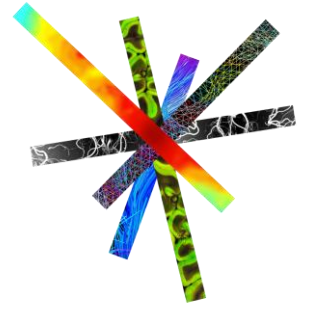
```
>> disp(isprime(A))
0 1 0 0 0 0 0
0 1 0 0 0 0 0
1 0 0 0 0 0 1
0 0 0 0 1 0 0
0 0 0 0 0 0 1
0 0 0 0 0 0 1
0 0 0 0 1 0 0
```

```
>> disp(find(isprime(A))')
3 8 9 32 35 45 47 48
```

column-major indexes!

# MATLAB syntax – Matrices, vectors, arrays ...

## VECTOR OPERATIONS



```
>> a + 3    % Add a scalar to a vector
>> a * 3    % Multiply a scalar and a vector
>> pinv(a)  % Moore-Penrose pseudoinverse
>> norm(b)  % norm of a vector
>> a'       % transpose
```

### Elementwise ops [ MUST BE same size! ]

```
>> a + b    % vector addition
>> a - b    % vector subtraction
>> a .* b   % vector multiplication
>> a ./ b   % vector division
```

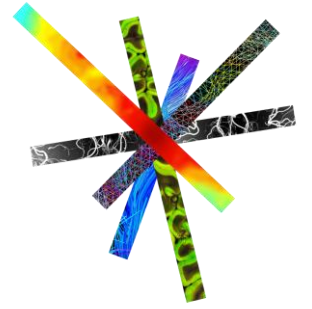
### MUST BE of compatible size!

```
>> a * c    % dot product
>> dot(a,c) % dot product
>> a / b    % equiv to a*pinv(b)
```



# MATLAB syntax – Matrices, vectors, arrays ...

## VECTOR OPERATIONS



```
>> a + 3    % Add a scalar to a vector
>> a * 3    % Multiply a scalar and a vector
>> pinv(a)  % Moore-Penrose pseudoinverse
>> norm(b)  % norm of a vector
>> a'       % transpose
```

```
a = [1,2,3]
b = [4,5,6]
c = [7;8;9]
```

```
>> disp(a')
    1
    2
    3
```

### Elementwise ops [ MUST BE same size! ]

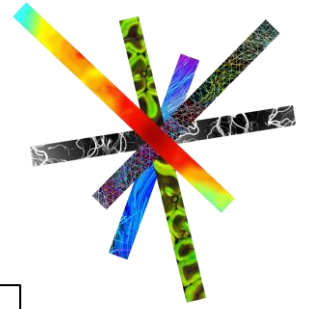
```
>> a + b    % vector addition
>> a - b    % vector subtraction
>> a .* b   % vector multiplication
>> a ./ b   % vector division
```

### MUST BE of compatible size!

```
>> a * c    % dot product
>> dot(a,c) % dot product
>> a / b    % equiv to a*pinv(b)
```

# MATLAB syntax – Matrices, vectors, arrays ...

## VECTOR OPERATIONS



```
>> a + 3    % Add a scalar to a vector
>> a * 3    % Multiply a scalar and a vector
>> pinv(a)  % Moore-Penrose pseudoinverse
>> norm(b)  % norm of a vector
>> a'       % transpose
```

```
a = [1,2,3]
b = [4,5,6]
c = [7;8;9]
```

```
>> a .* b

ans =

     4     10     18
```

### Elementwise ops [ MUST BE same shape! ]

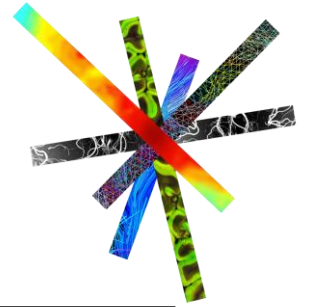
```
>> a + b    % vector addition
>> a - b    % vector subtraction
>> a .* b   % vector multiplication
>> a ./ b   % vector division
```

### MUST BE of compatible shape!

```
>> a * c    % dot product
>> dot(a,c) % dot product
>> a / b    % equiv to a*pinv(b)
```

# MATLAB syntax – Matrices, vectors, arrays ...

## VECTOR OPERATIONS



```
>> a + 3    % Add a scalar to a vector
>> a * 3    % Multiply a scalar and a vector
>> pinv(a)  % Moore-Penrose pseudoinverse
>> norm(b)  % norm of a vector
>> a'       % transpose
```

```
a = [1,2,3]
b = [4,5,6]
c = [7;8;9]
```

```
>> a ./ b

ans =

    0.2500    0.4000    0.5000
```

### Elementwise ops [ MUST BE same shape! ]

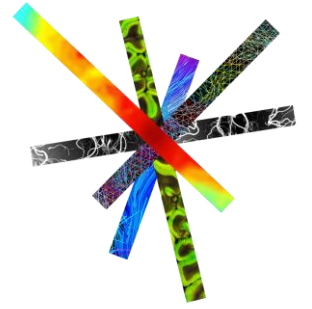
```
>> a + b    % vector addition
>> a - b    % vector subtraction
>> a .* b   % vector multiplication
>> a ./ b   % vector division
```

### MUST BE of compatible shape!

```
>> a * c    % dot product
>> dot(a,c) % dot product
>> a / b    % equiv to a*pinv(b)
```

# MATLAB syntax – Matrices, vectors, arrays ...

## VECTOR OPERATIONS



```
>> a + 3    % Add a scalar to a vector
>> a * 3    % Multiply a scalar and a vector
>> pinv(a)  % Moore-Penrose pseudoinverse
>> norm(b)  % norm of a vector
>> a'       % transpose
```

```
a = [1,2,3]
b = [4,5,6]
c = [7;8;9]
```

```
>> a*c

ans =

    50
```

### Elementwise ops [ MUST BE same shape! ]

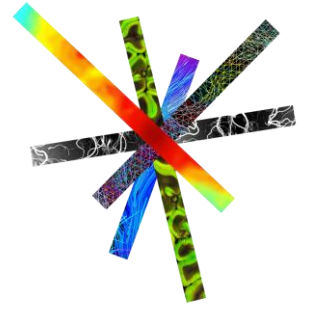
```
>> a + b    % vector addition
>> a - b    % vector subtraction
>> a .* b   % vector multiplication
>> a ./ b   % vector division
```

### MUST BE of compatible shape!

```
>> a * c    % dot product
>> dot(a,c) % dot product
>> a / b    % equiv to a*pinv(b)
```

# MATLAB syntax – Matrices, vectors, arrays ...

## VECTOR OPERATIONS



```
>> a + 3    % Add a scalar to a vector
>> a * 3    % Multiply a scalar and a vector
>> pinv(a)  % Moore-Penrose pseudoinverse
>> norm(b)  % norm of a vector
>> a'       % transpose
```

```
a = [1,2,3]
b = [4,5,6]
c = [7;8;9]
```

```
>> a/b

ans =

    0.4156
```

### Elementwise ops [ MUST BE same shape! ]

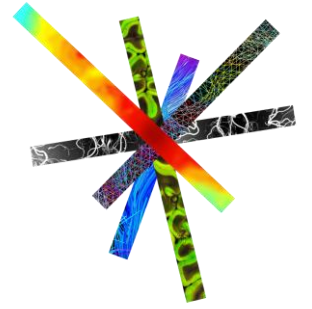
```
>> a + b    % vector addition
>> a - b    % vector subtraction
>> a .* b   % vector multiplication
>> a ./ b   % vector division
```

### MUST BE of compatible shape!

```
>> a * c    % dot product
>> dot(a,c) % dot product
>> a / b    % equiv to a*pinv(b)
```

# MATLAB syntax – Matrices, vectors, arrays ...

## MATRIX OPERATIONS



```
>> A + 3    % Add a scalar
>> A * 3    % Multiply a scalar
>> sin(A)   % Elementwise sine
>> exp(A)   % Elementwise exponential
>> inv(A)   % Inverse of a matrix
```

```
>> pinv(A)  % Pseudoinverse of a matrix
>> det(A)   % Determinant of a matrix
>> A .^ 3   % Elementwise power
>> A'       % Transpose
```

### Elementwise ops [ MUST BE same shape! ]

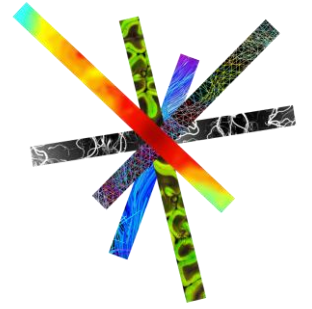
```
>> A + B    % Matrices addition
>> A .* B   % Matrices multiplication
>> A ./ B   % Matrices division
```

### MUST BE of compatible shape!

```
>> A * C    % Matrix multiplication
>> A * a    % Matrix-vector product
>> A / B    % A*inv(B)
>> A \ B    % inv(A)*B
```

# MATLAB syntax – Matrices, vectors, arrays ...

## MATRIX OPERATIONS



```
>> A + 3    % Add a scalar
>> A * 3    % Multiply a scalar
>> sin(A)   % Elementwise sine
>> exp(A)   % Elementwise exponential
>> inv(A)   % Inverse of a matrix
```

```
>> pinv(A)  % Pseudoinverse of a matrix
>> det(A)   % Determinant of a matrix
>> A .^ 3   % Elementwise power
>> A'       % Transpose
```

### Elementwise ops [ MUST BE same shape! ]

```
>> A + B    % Matrices addition
>> A .* B   % Matrices multiplication
>> A ./ B   % Matrices division
```

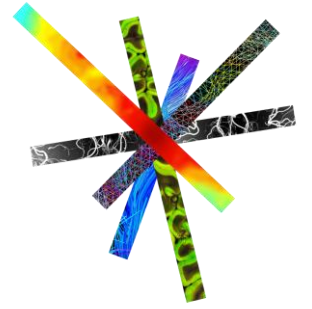
### MUST BE of compatible shape!

```
>> A * C    % Matrix multiplication
>> A * a    % Matrix-vector product
>> A / B    % A*inv(B)
>> A \ B    % inv(A)*B
```



# MATLAB syntax – ‘Unusual’ data structures

## WHAT IS A ‘STRUCT’ IN MATLAB?



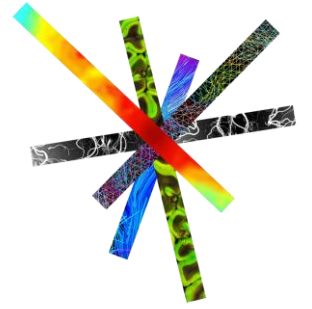
- A **structure array** is a data type that groups related data using data containers called **fields**.
- Each **field** can contain **any type of data**.
- Access data in a field using **dot notation** of the form ***structName.fieldName***

```
data.x = linspace(0,2*pi);  
data.y = sin(data.x);  
data.title = 'y = sin(x)'
```

```
data = struct with fields:  
    x: [1x100 double]  
    y: [1x100 double]  
    title: 'y = sin(x)'
```

# MATLAB syntax – ‘Unusual’ data structures

## WHAT IS A ‘STRUCT’ IN MATLAB?



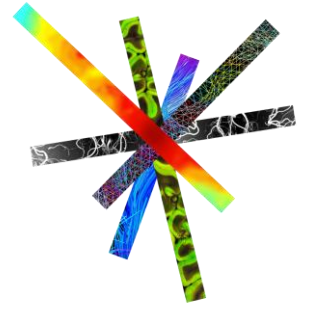
- A **structure array** is a data type that groups related data using data containers called **fields**.
- Each **field** can contain **any type of data**.
- Access data in a field using **dot notation** of the form ***structName.fieldName***

```
field1 = 'f1'; value1 = zeros(1,10);  
field2 = 'f2'; value2 = {'a', 'b'};  
field3 = 'f3'; value3 = {pi, pi.^2};  
field4 = 'f4'; value4 = {'fourth'};  
  
s = struct(field1,value1,field2,value2,field3,value3,field4,value4)
```

→ s=2x4 struct  
f1  
f2  
f3  
f4

# MATLAB syntax – ‘Unusual’ data structures

## WHAT IS A ‘STRUCT’ IN MATLAB?



- A **structure array** is a data type that groups related data using data containers called **fields**.
- Each **field** can contain **any type of data**.
- Access data in a field using **dot notation** of the form ***structName.fieldName***

```
field1 = 'f1'; value1 = zeros(1,10);  
field2 = 'f2'; value2 = {'a', 'b'};  
field3 = 'f3'; value3 = {pi, pi.^2};  
field4 = 'f4'; value4 = {'fourth'};  
  
s = struct(field1,value1,field2,value2,field
```

s(1) ←

```
ans = struct with fields:  
f1: [0 0 0 0 0 0 0 0 0 0]  
f2: 'a'  
f3: 3.1416  
f4: 'fourth'
```

→ s=2x4 struct  
f1  
f2  
f3  
f4

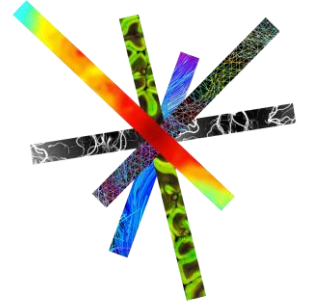
s(2) ←

```
ans = struct with fields:  
f1: [0 0 0 0 0 0 0 0 0 0]  
f2: 'b'  
f3: 9.8696  
f4: 'fourth'
```



# MATLAB syntax – ‘Unusual’ data structures

## WHAT IS A ‘STRUCT’ IN MATLAB?



- A **structure array** is a data type that groups related data using data containers called **fields**.
- Each **field** can contain **any type of data**.
- Access data in a field using **dot notation** of the form **structName.fieldName**

```
field1 = 'f1'; value1 = zeros(1,10);
field2 = 'f2'; value2 = {'a', 'b'};
field3 = 'f3'; value3 = {pi, pi.^2};
field4 = 'f4'; value4 = {'fourth'};

s = struct(field1,value1,field2,value2,field
```

s(1) ←

```
ans = struct with fields:
  f1: [0 0 0 0 0 0 0 0 0 0]
  f2: 'a'
  f3: 3.1416
  f4: 'fourth'
```

→ s=2x4 struct

```
  f1
  f2
  f3
  f4
```

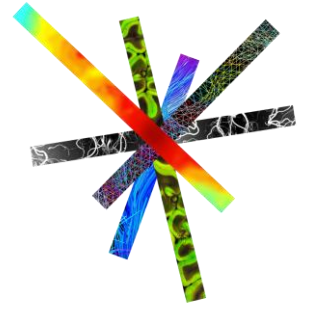
s(2) ←

```
ans = struct with fields:
  f1: [0 0 0 0 0 0 0 0 0 0]
  f2: 'b'
  f3: 9.8696
  f4: 'fourth'
```



# MATLAB syntax – ‘Unusual’ data structures

## WHAT ABOUT ‘CELL ARRAYS’?



- A **cell array** is a data type with *indexed data containers* called **cells**
- Each **cell** can contain **any type of data**.

## Creation

When you have data to put into a cell array, create the array using the cell array **construction operator**, `{}`.

```
C = {'2017-08-16',[56 67 78]}
```

```
C=1x2 cell  
    {'2017-08-16'}    {1x3 double}
```

```
C(2,:) = {'2017-08-17',[58 69 79]};  
C(3,:) = {'2017-08-18',[60 68 81]}
```

```
C=3x2 cell  
    {'2017-08-16'}    {1x3 double}  
    {'2017-08-17'}    {1x3 double}  
    {'2017-08-18'}    {1x3 double}
```

## Indexing

When you index with **smooth parentheses**, `()`, the result is a cell array that is a subset of the cell

```
C(1,:)
```

```
ans=1x2 cell  
    {'2017-08-16'}    {1x3 double}
```

When you index with **curly braces**, `{}`, the result is the data that is contained in the specified cell.

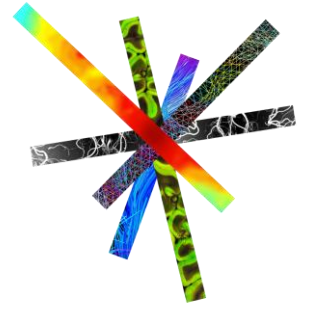
```
C{1,2}
```

```
ans = 1x3
```

```
56    67    78
```

# MATLAB syntax – Control flow

## 'IF - ELSE' CONDITION

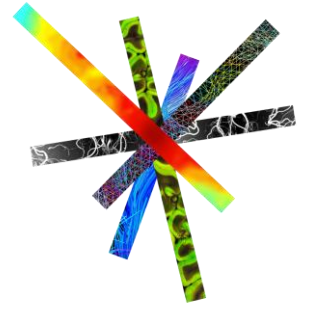


Use an if-else **condition to check the value** of some variable within the code:

```
a = randi(100,1);  
if a < 30  
    fprintf('%d is smaller than 30. \n', a)  
elseif a > 80  
    fprintf('%d is larger than 80. \n', a)  
else  
    X = [num2str(a), ' is between 30 and 80.'];  
    disp(X)  
end
```

# MATLAB syntax – Control flow

## 'IF - ELSE' CONDITION

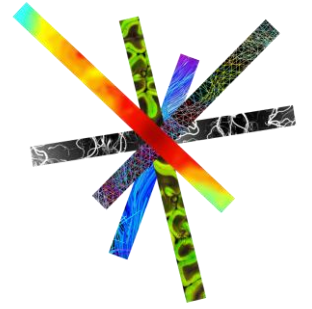


Use an if-else **condition to check the value** of some variable within the code:

```
a = randi(100,1);  
if a < 30  
    fprintf('%d is smaller than 30. \n', a)  
elseif a > 80  
    fprintf('%d is larger than 80. \n', a)  
else  
    X = [num2str(a), ' is between 30 and 80.'];  
    disp(X)  
end
```

# MATLAB syntax – Control flow

## 'IF - ELSE' CONDITION



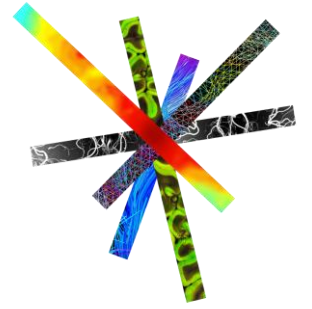
Use an if-else **condition to check the value** of some variable within the code:

```
a = randi(100,1);  
if a < 30  
    fprintf('%d is smaller than 30. \n', a)  
elseif a > 80  
    fprintf('%d is larger than 80. \n', a)  
else  
    X = [num2str(a), ' is between 30 and 80.'];  
    disp(X)  
end
```



# MATLAB syntax – Control flow

## 'FOR' LOOPS

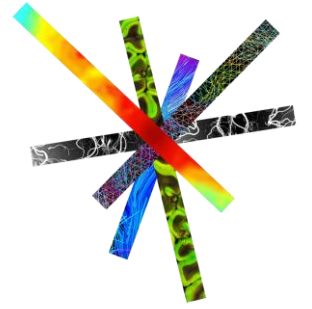


Use for-loops to execute iterations with a **known, and fixed number of repetitions**

```
for i=1:5                                % row index  
    for j=1:3                            % col index  
        A(i, j) = i + j ; % use loop iterable to index a matrix  
    end  
end                                       % close each loop with an 'end'
```

# MATLAB syntax – Control flow

## 'WHILE - BREAK' LOOPS



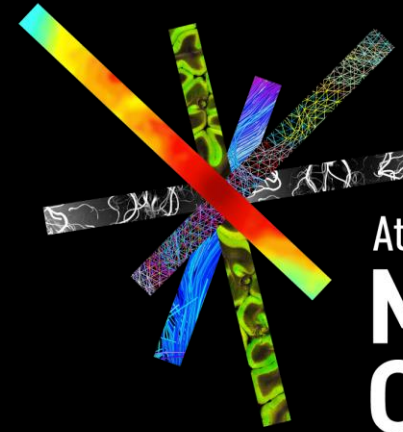
Use **while-loops** to execute iterations with **unknown number of repetitions**.

Use a **break** command to exit the while once a certain condition is met.

```
% find the root of the polynomial x3 - 2x - 5
a = 0; fa = -Inf;
b = 3; fb = Inf;
while b-a > eps*b
    x = (a+b) /2;    fx = x3-2*x-5;
    if fx == 0
        break % Already found the root, exit the loop
    elseif sign(fx) == sign(fa) % This method only works when the polynomial
        a = x; fa = fx; % is increasing in proximity of the root
    else
        b = x; fb = fx;
    end
end
```

# Overview

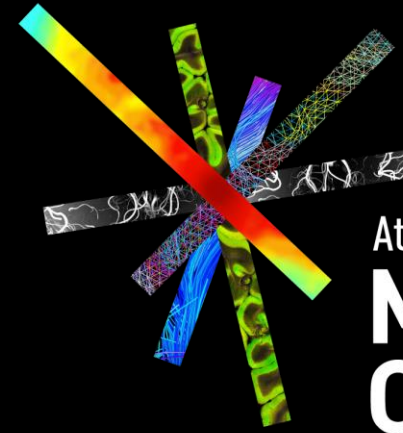
- **WHAT?**
- **WHY?**
- **HOW?**
  - GETTING STARTED
  - SCRIPTS, FUNCTIONS, AND THE EDITOR
  - VISUALIZATION TOOLS
- **BUT ... CAN I STILL USE IT, IF I DON'T KNOW HOW TO CODE?**



Athinoula A.  
**Martinos**  
**Center**  
For Biomedical Imaging

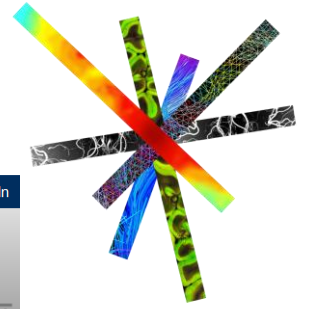
# Overview

- **WHAT?**
- **WHY?**
- **HOW?**
  - GETTING STARTED
  - SCRIPTS, FUNCTIONS, AND THE EDITOR
  - VISUALIZATION TOOLS
- **BUT ... CAN I STILL USE IT, IF I DON'T KNOW HOW TO CODE?**



Athinoula A.  
**Martinos**  
**Center**  
For Biomedical Imaging

# MATLAB's Graphical User Interface (GUI)



The screenshot displays the MATLAB R2018b graphical user interface. The top menu bar includes options like HOME, PLOTS, and APPS. A pink arrow points to the 'New Script' button in the HOME tab. The Command Window shows the following code and output:

```
>> a = [1 2 ; 3 4]

a =

     1     2
     3     4

fx >>
```

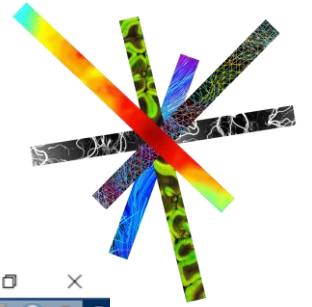
The Workspace window shows a variable 'a' with the value [1,2;3,4]. The Command History window shows the following code:

```
title('biqaa');
subplot(2,3,2), boxplot(score.piqe_mat', 'No...
title('piqe');
% subplot(2,3,3), boxplot(score.brisque', 'N...
% title('brisque');
subplot(2,3,3), boxplot(score.niqe_mat', 'No...
title('niqe');
%-- 23/01/2020 10:46 --%

- main
templateModel = load('templatemodel.mat');
templateModel = templateModel.templateModel;
mu_prisparam = templateModel{1};
cov_prisparam = templateModel{2};
meanOfSampleData = templateModel{3};
```

# MATLAB's EDITOR – M-file

## SCRIPT M-FILE



```
1 % find the root of the polynomial x3 - 2x - 5
2 a = 0;
3 fa = -Inf;
4 b = 3;
5 fb = Inf;
6
7 while b-a > eps*b
8     x = (a+b) / 2;
9     fx = x3-2*x-5;
10    if fx == 0
11        break % Already found the root, exit the loop
12    elseif sign(fx) == sign(fa)
13        a = x;
14        fa = fx;
15    else
16        b = x;
17        fb = fx;
18    end
19 end
20
21 disp (x)
```

- It is a simple text file where you can place MATLAB commands
- It will be executed **top-to-bottom**, one line at a time, as if you were typing the same commands in the console
- Run directly (even portion of it, up to a single line): **no need to be compiled**
- **Save** your works, and allows **reproducibility**

# MATLAB's EDITOR – M-file

## SCRIPT M-FILE



```
1 find the root of the polynomial x3 - 2x - 5
2 a = 0;
3 fa = -Inf;
4 b = 3;
5 while b-a > 1e-6
6     x = (a+b) / 2;
7     fx = x3 - 2*x - 5;
8     if sign(fx) == sign(fa)
9         a = x;
10        fa = fx;
11    else
12        b = x;
13        fb = fx;
14    end
15 end
16 disp(x)
```

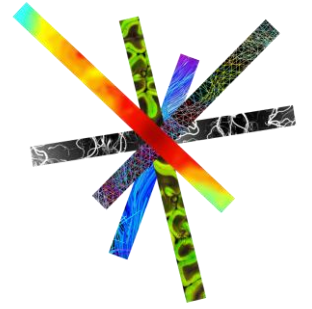
- ✓ A script shares same memory space from which it was invoked
- ✓ Script works as if sequentially inserting the commands in the m-file at the command line

*Be careful!*

- it is a simple text file where you can place MATLAB commands
- it will be executed *top-to-bottom*, one line at a time, as if you were typing the same commands in the console
- Save your work as often as possible to ensure reproducibility
- convenient for debugging
- Run directly (even portion of it, up to a single line): **no need to be compiled**

# MATLAB's EDITOR – M-file

## FUNCTION M-FILE



If you use a piece of code often, it is better to write it as a separate function.

The m-file begins with the keyword "function".

The output argument(s) are in brackets [ ].

The input argument(s) are in parentheses ( ).

```
1 function [ output_arg ] = compute_square( input_arg )
2
3 - output_arg=input_arg.^2;
4
5 - end
```

The file ends with the keyword "end".

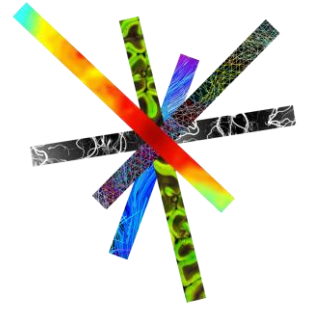
The name of the function and of the file should be the same!

Save this as m-file: `compute_square.m`



# MATLAB's EDITOR – M-file

## FUNCTION M-FILE



- ✓ Once we save the function m-file, it may be called from a script or another function:

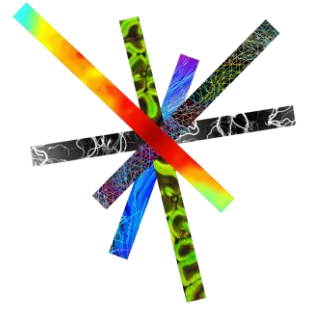
```
>> a = [1,2,3];  
>> b = compute_square(a)  
>> disp(b)  
1      4      9
```

- ✓ **All parameters** defined and used within a function **reside in function's own workspace** and are **deleted upon exiting the function**.

*Good to keep  
in mind!*

# MATLAB's EDITOR – M-file

## SCRIPT OR FUNCTION M-FILE?



### Scripts

#### Pros:

- **convenient**; script's variables are in same workspace as caller's

#### Cons:

- **slow**; script commands loaded and interpreted each time used
- risks of variable **name conflicts** inside & outside of script

### Functions

#### Pros:

- Scope of function's **variables** is **confined to within function**.
- **Easier debugging** of input and outputs
- Compiled the first time it is used; it **runs faster subsequent times**.
- Easily be **re-usable** in another project.
- **Auto cleaning** of temporary variables.

#### Cons:

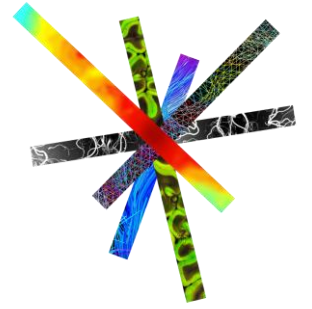
- **I/O are highly regulated**, if the function requires many pre-defined variables, it is cumbersome to pass in and out of the function – a script m-file is more convenient

Tip:

Use a **script** as your 'main' file, and refactor as much code as possible into **as small as possible functions**

# MATLAB's EDITOR – Standard editor

## TIPS AND TRICKS



### Automatic code checking and programming tips

You can view **warning and error messages about your code**, and modify your file based on the messages. The **messages update automatically and continuously** so you can see if your changes addressed the issues noted in the messages.

The image displays three overlapping MATLAB editor windows illustrating code checking and error handling.

- Left Window (Message Indicator):** Shows the documentation for the `lengthofline` function. The title is "Message Indicator". The code includes comments explaining the function's purpose and usage.
- Center Window (Code Editor):** Shows MATLAB code with a warning message. The code includes:

```
25 len = zeros(size(hline));  
26 for nl = 1:prod(size(hline))  
27     if ~notline(nl)  
28         flds = get(hline(nl));  
29         fdata = {'XData','YData','ZData'};  
30         for nd = 1:length(fdata)  
31             data(nd, nl) = ...  
32             ...  
33             ...
```

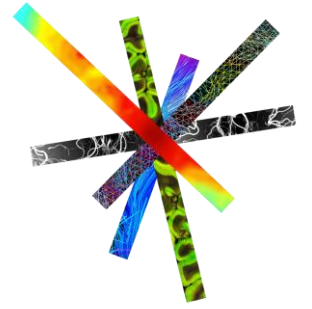
A warning message is displayed: "NUMEL(x) is usually faster than PROD(SIZE(x))." with a "Fix" button and the text "compute the length".
- Right Window (Code Editor):** Shows MATLAB code with error messages. The code includes:

```
21 nothandle = ~ishandle(hline);  
22 for nh = 1:prod(size(hline))  
23     notline(nh) = ~ishandle(hline(nh)) || ~strcmp('line',  
24     end
```

Three error messages are displayed:
  - Line 47: Invalid syntax at '\': Possibly, a \') is missing.
  - Line 47: Invalid syntax at '\': Possibly, a \') is missing.
  - Line 47: Parse error at '\': usage might be invalid MATLAB syntax.Each message has "Details" and "Fix" buttons.

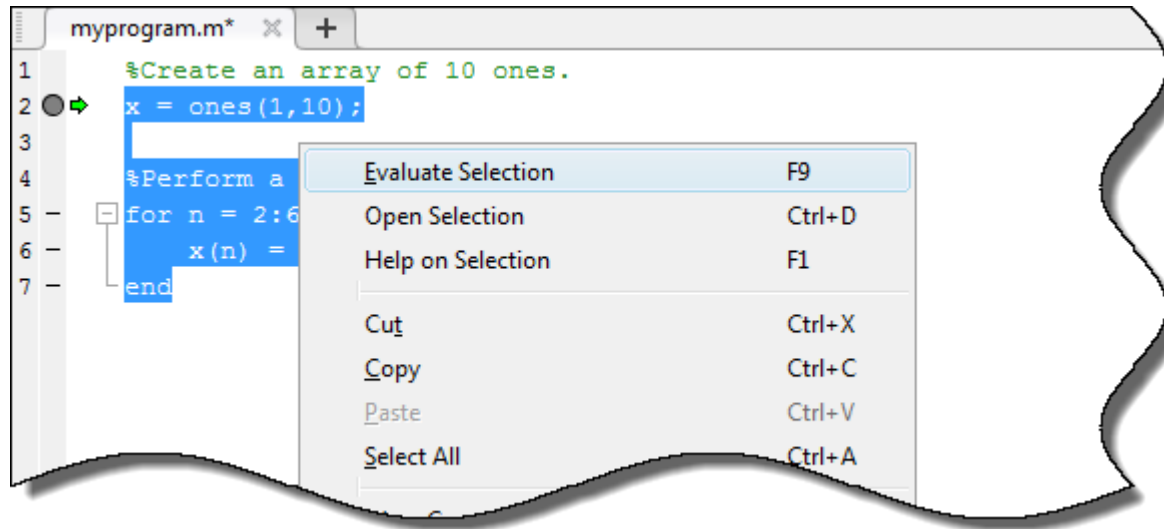
# MATLAB's EDITOR – Standard editor

## TIPS AND TRICKS



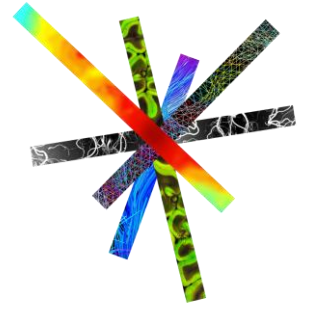
### Interactive debugging

To run piece of code: Highlight it & press F9:



# MATLAB's EDITOR – Standard editor

## TIPS AND TRICKS



### 'PROPER' debugging functionalities

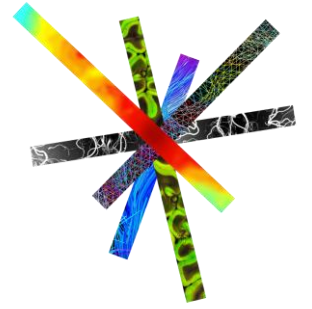
1. **Set breakpoints** to pause the execution of a MATLAB file so you can examine the value or variables where you think a problem could be.
2. **Run the file.**
3. MATLAB **pauses at the first breakpoint** in the program.
4. While your code is paused, you can **view or change the values of variables**, or you can **modify the code**.
5. Press **Continue** to run the next line of code.

```
myprogram.m x +
1      %Create an array of 10 ones.
2  ●   x = ones(1,10);
3
4      %Perform a calculation on items 2-6 in the array
5  -   for n = 2:6
6  -       x(n) = 2 * x(n-1);
7  -   end
```

```
1      % Create an array of 10 ones.
2  ● → x = ones(1,10);
```

# MATLAB's EDITOR – Standard editor

## TIPS AND TRICKS



### ‘Use “cell mode” to improve code readability!’

- Inserting **%%** at the beginning of a line creates a cell, which is a block of code, within a script or a function
- If you execute the whole file, cells will be ignored (they are NOT breakpoint)
- But you can decide to evaluate just a single cell, and then jump to the next one (like F9 to evaluate a single line, but on steroids!)

```
%%%%CREATE A PATCH OBJECT WITH DESIRED VERTICES & FACES

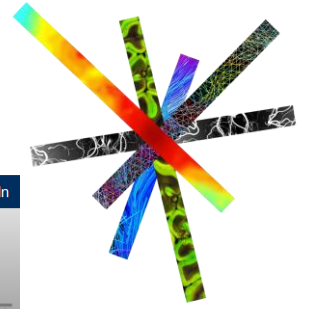
%%
clf; cameratoolbar; axis equal off;
P_lh=patch('Faces',faces_lh_red,'Vertices',vertices_lh_red);

set(P_lh,'EdgeColor','black','FaceColor','green');

set(P_lh,'Marker','*');

%%
```

# MATLAB's Graphical User Interface (GUI)



The screenshot displays the MATLAB R2018b graphical user interface. The top menu bar includes options like HOME, PLOTS, and APPS. Below it is a ribbon with various tool icons. The main workspace is divided into three panes:

- Current Folder:** Shows a file tree with folders like '000final' and 'BISHARP\_code'. The 'BISHARP\_code' folder is expanded, showing files such as 'BISHARP.m', 'BISHARP\_bibtex.txt', 'BISHARP\_example.m', and 'image.bmp'.
- Command Window:** Contains the following MATLAB code:

```
>> a = [1 2 ; 3 4]

a =

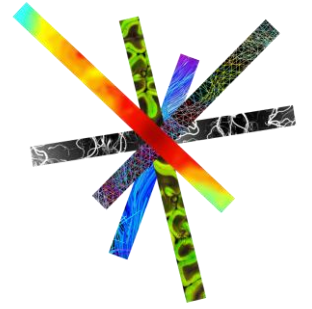
     1     2
     3     4

fx >>
```
- Workspace:** A table showing the current workspace variables:

Name	Value
a	[1,2;3,4]
- Command History:** Shows a list of executed commands, including:

```
title('biqaa');
subplot(2,3,2), boxplot(score.piqe_mat', 'No...
title('piqe');
% subplot(2,3,3), boxplot(score.brisque', 'N...
% title('brisque');
subplot(2,3,3), boxplot(score.niqe_mat', 'No...
title('niqe');
%-- 23/01/2020 10:46 --%
- main
templateModel = load('templatemodel.mat');
templateModel = templateModel.templateModel;
mu_prisparam = templateModel{1};
cov_prisparam = templateModel{2};
meanOfSampleData = templateModel{3};
```

# MATLAB's EDITOR – Live editor



MATLAB live scripts and live functions are **interactive documents** that combine MATLAB code with formatted text, equations, and images in a single environment called the Live Editor. In addition, live scripts **store and display output alongside the code that creates it.**

**Distribution of Fatalities**

We can use a bar chart to see the distribution of fatality rates among the states. There are 11 states that have a fatality rate greater than 0.02 per million vehicle miles.

```

    histogram(rate,10)
    xlabel('Fatalities per Million Vehicle Miles')
    ylabel('Number of States')
  
```



**Find Correlations in the Data**

We can experiment with the data to see if any of the variables in the table are correlated with highway fatalities. It appears that highway fatality rates are lower in states with a higher percentage urban population.

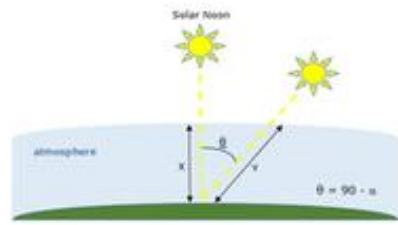
```

    varName = 'urbanPopulation';
    scatter(fatalities.(varName),rate)
    xlabel(varName)
    ylabel('Percent Fatalities per Million Vehicle Miles')

    hold on
    xmin = min(fatalities.(varName));
    xmax = max(fatalities.(varName));
    p = polyfit(fatalities.(varName),rate,1);
    plot([xmin xmax], polyval(p,[xmin xmax]))
  
```



**Air Mass and Solar Radiation**



The larger the air mass, the less radiation reaches the ground. The air mass can be calculated from the equation

$$AM = \frac{1}{\cos(90 - \alpha) + 0.5057(6.0799 + \alpha)^{-1.6364}}$$

Then the solar radiation (in Kw/m<sup>2</sup>) reaching the ground can be calculated from the empirical equation

$$sRad = 1.353 * 0.7^{AM^{0.678}}$$

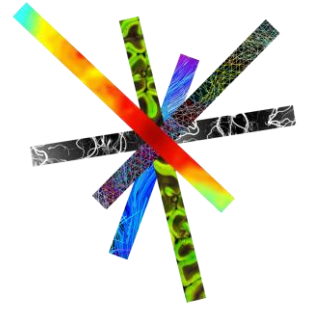
```

    AM = 1/(cosd(90-alpha) + 0.50572*(6.07955+alpha)^-1.6354);
    sRad = 1.353*0.7^(AM^0.678); % kW/m^2
    disp(['Air Mass = ' num2str(AM) ' Solar Radiation = ' num2str(sRad) ' kW/m^2'])
  
```

Live scripts can be exported to PDF, Microsoft® Word, HTML, or LaTeX.



# MATLAB's EDITOR – Live editor



MATLAB live scripts and live functions are **interactive documents** that combine MATLAB code with formatted text, equations, and images in a single environment called the Live Editor. In addition, live scripts **store and display output alongside the code that creates it.**

**Distribution of Fatalities**  
We can use a bar chart to see the distribution of fatality rates among the states. There are 11 states that have a fatality rate greater than 0.02 per million vehicle miles.

```
histogram(rate,10)  
xlabel('Fatalities per Million Vehicle Miles')  
ylabel('Number of States')
```

**Find Correlations in the Data**  
We can experiment with the data to see if any of the variables in the table are correlated with highway fatalities. It appears that highway fatality rates are lower in states with a higher percentage urban population.

```
varName = 'urbanPopulation';  
scatter(fatalities.(varName),rate)  
xlabel(varName)  
ylabel('Percent Fatalities per Million Vehicle Miles')  
  
hold on  
xmin = min(fatalities.(varName));  
xmax = max(fatalities.(varName));  
p = polyfit(fatalities.(varName),rate,1);  
plot([xmin xmax], polyval(p,[xmin xmax]))
```

**Solar Radiation**

mass, the less radiation reaches the ground. The air mass can be calculated from the equation

$$AM = \frac{1}{\cos(90 - \alpha) + 0.5057(6.0799 + \alpha)^{-1.6364}}$$

ation (in Kw/m<sup>2</sup>) reaching the ground can be calculated from the empirical equation

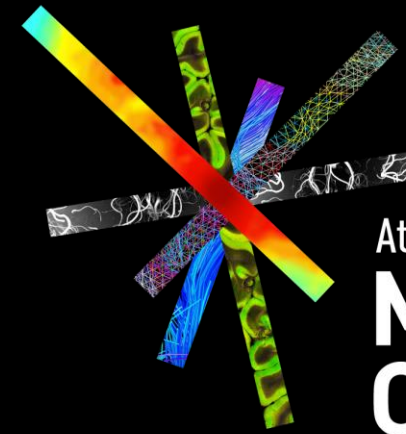
$$sRad = 1.353 + 0.7AM^{0.678}$$

```
90-alpha) + 0.50572*(6.07955+alpha)^-1.6354);  
0.7*(AM^0.678); % kW/m^2  
rs = ' num2str(AM) ' Solar Radiation = ' num2str(sRad) ' kW/m^2']
```

Live scripts can be exported to PDF, Microsoft® Word, HTML, or LaTeX.

# Overview

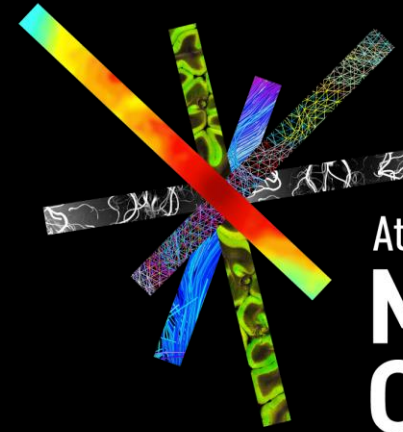
- **WHAT?**
- **WHY?**
- **HOW?**
  - GETTING STARTED
  - SCRIPTS, FUNCTIONS, AND THE EDITOR
  - VISUALIZATION TOOLS
- **BUT ... CAN I STILL USE IT, IF I DON'T KNOW HOW TO CODE?**



Athinoula A.  
**Martinos**  
**Center**  
For Biomedical Imaging

# Overview

- **WHAT?**
- **WHY?**
- **HOW?**
  - GETTING STARTED
  - SCRIPTS, FUNCTIONS, AND THE EDITOR
  - VISUALIZATION TOOLS
- **BUT ... CAN I STILL USE IT, IF I DON'T KNOW HOW TO CODE?**



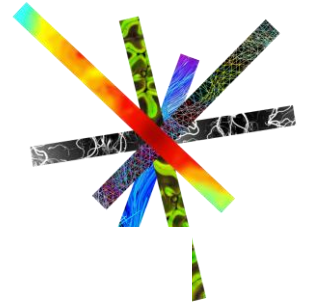
Athinoula A.

**Martinos  
Center**

For Biomedical Imaging

# MATLAB Graphics

## PLOTTING CURVES



This is an example of how to create an line plot with legend in MATLAB®.

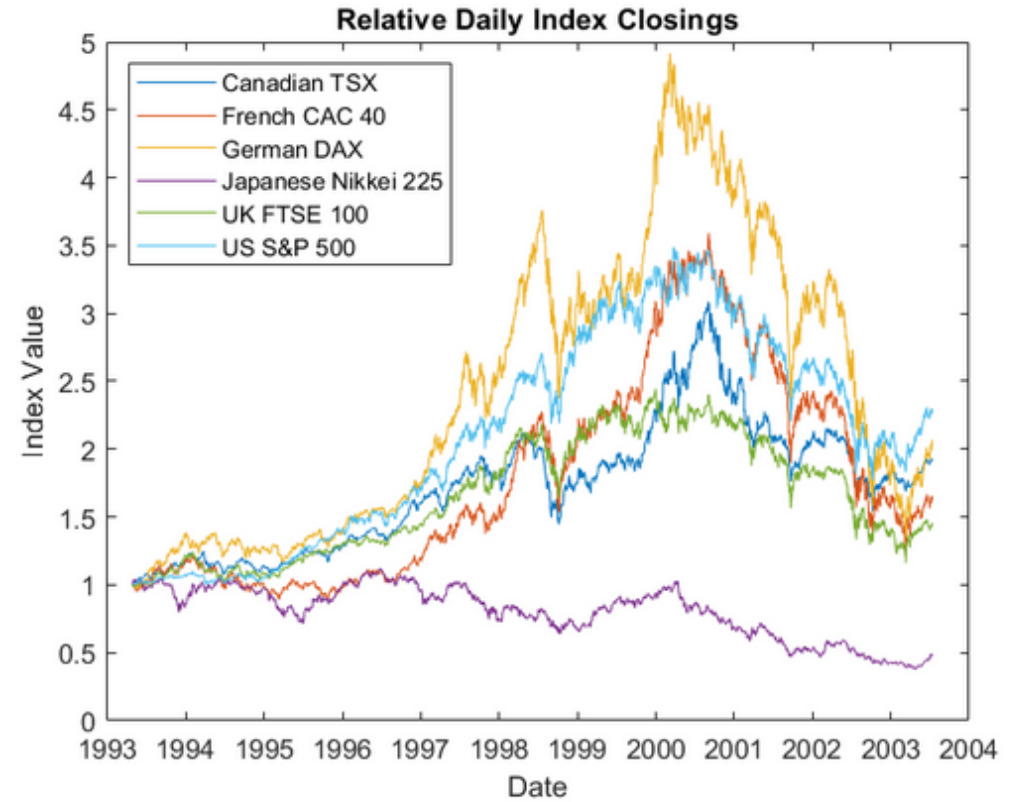
```
% Load data for the stock indices
load IndexData dates values series

% Plot the stock index values versus time
figure
plot(dates, values)

% Use dateticks for the x axis
datetick('x')

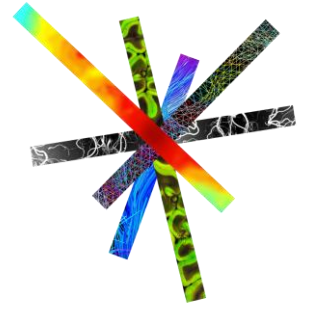
% Add title and axis labels
xlabel('Date')
ylabel('Index Value')
title('Relative Daily Index Closings')

% Add a legend in the top, left corner
legend(series, 'Location', 'NorthWest')
```



# MATLAB Graphics

## PLOTTING CURVES



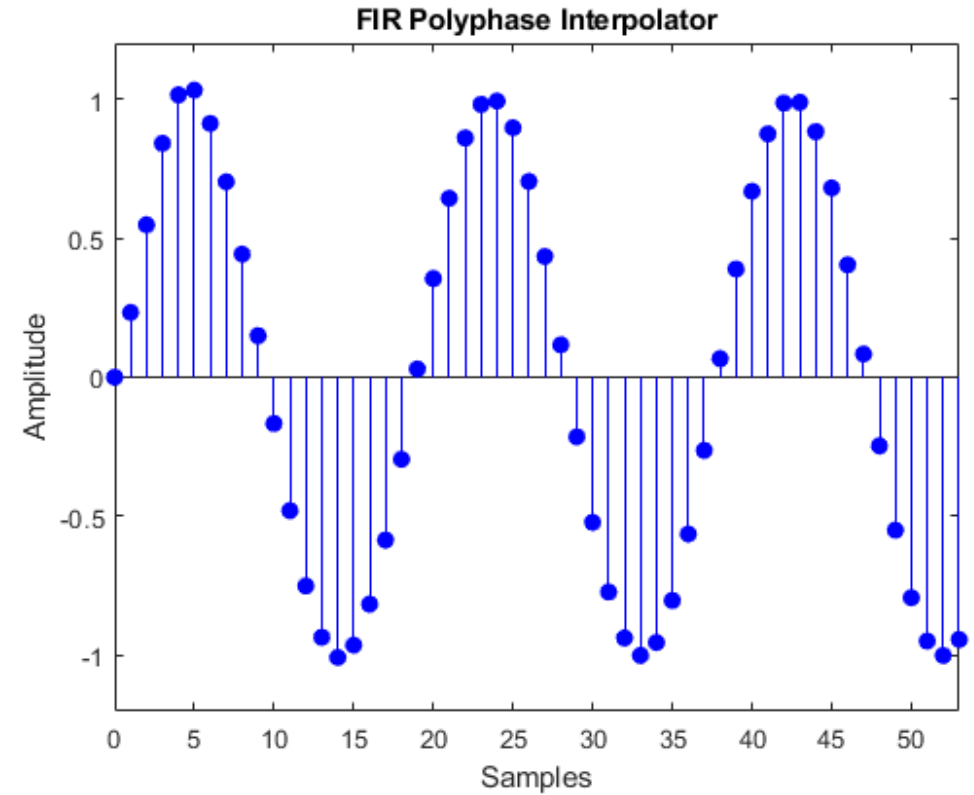
This is an example of how to create a simple stem plot in MATLAB®.

```
% Load amplitude data
load amplitudeData sample amplitude

% Create a stem plot using the stem function
figure
stem(sample, amplitude, 'filled', 'b')

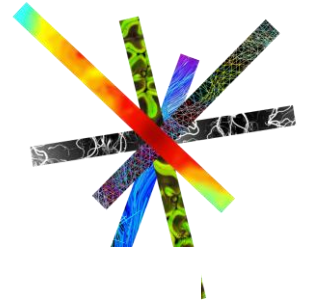
% Adjust the axis limits
axis([0 53 -1.2 1.2])

% Add title and axis labels
title('FIR Polyphase Interpolator')
xlabel('Samples')
ylabel('Amplitude')
```



# MATLAB Graphics

## PLOTTING CURVES



This is an example of how to create a curve with lower and upper bounds in MATLAB®.

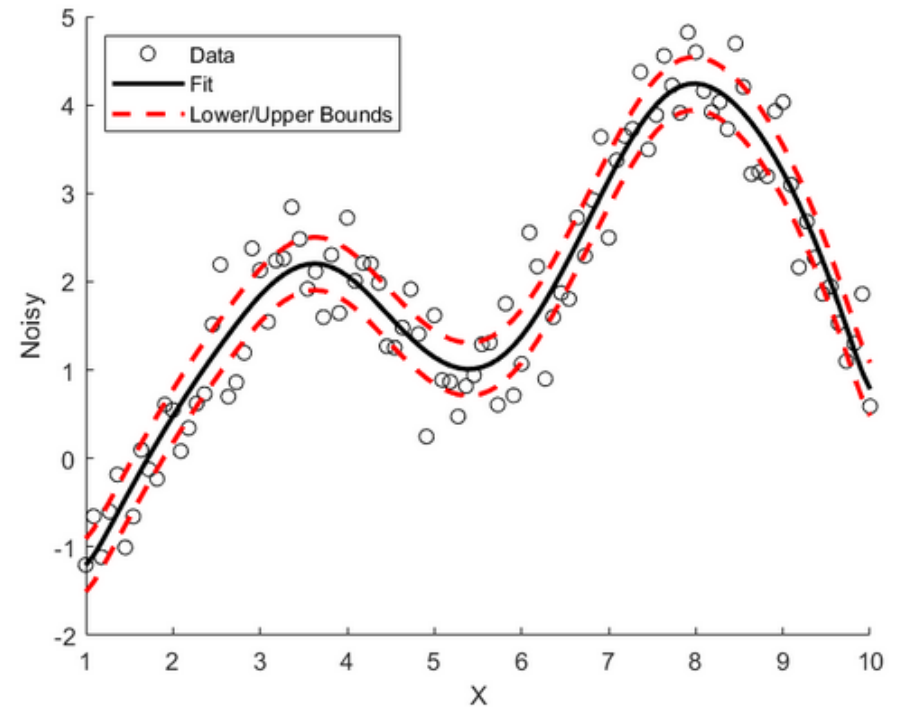
```
% Load the data for x, y, and yfit
load fitdata x y yfit

% Create a scatter plot of the original x and y data
figure
scatter(x, y, 'k')

% Plot yfit
line(x, yfit, 'Color', 'k', 'LineStyle', '-', 'LineWidth', 2)

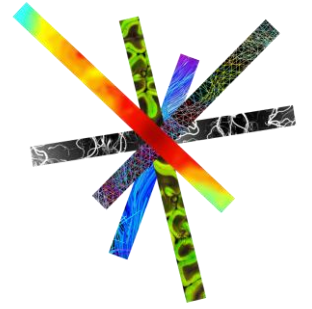
% Plot upper and lower bounds, calculated as 0.3 from yfit
line(x, yfit + 0.3, 'Color', 'r', 'LineStyle', '--', 'LineWidth', 2)
line(x, yfit - 0.3, 'Color', 'r', 'LineStyle', '--', 'LineWidth', 2)

% Add a legend and axis labels
legend('Data', 'Fit', 'Lower/Upper Bounds', 'Location', 'NorthWest')
xlabel('X')
ylabel('Noisy')
```



# MATLAB Graphics

## PLOTTING CURVES



This is an example of how to create a plot with two y axes in MATLAB®.

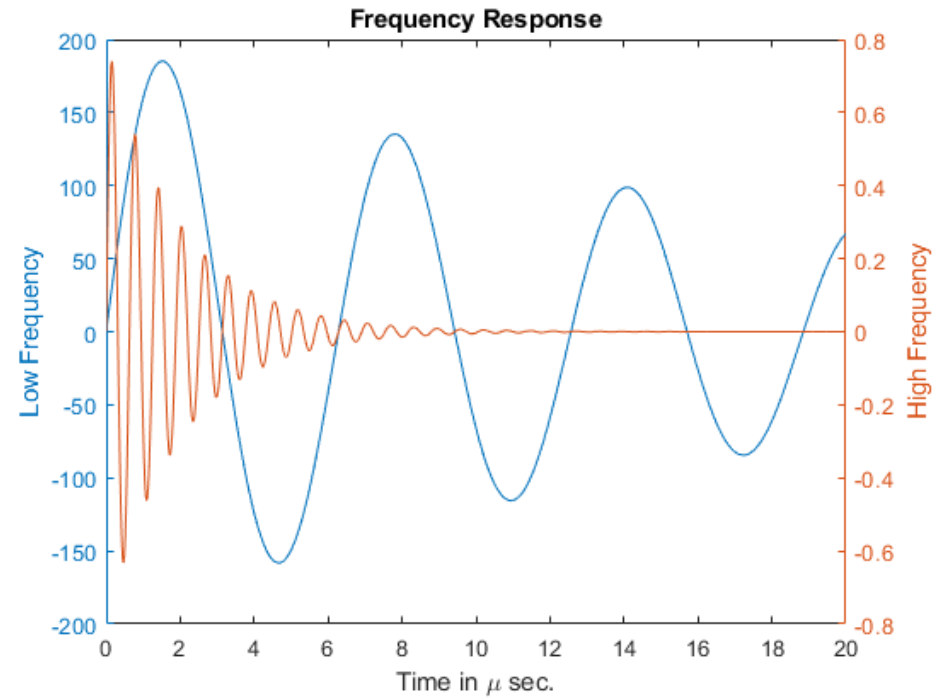
```
% Check version
if verLessThan('matlab','9.0')
    error(['yyaxis is available in R2016a or newer. ', ...
        'For older releases, use plotyy instead.'])
end

% Create some data for the two curves to be plotted
x = 0:0.01:20;
y1 = 200*exp(-0.05*x).*sin(x);
y2 = 0.8*exp(-0.5*x).*sin(10*x);

% Create a plot with 2 y axes using the yyaxis function
figure
yyaxis left
plot(x, y1)
ylabel('Low Frequency')

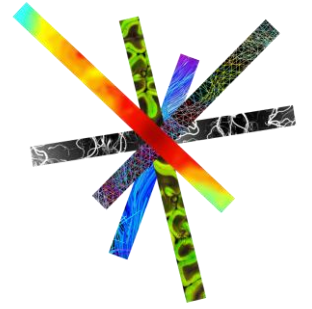
yyaxis right
plot(x, y2)
ylabel('High Frequency')

% Add title and x axis label
xlabel('Time in \mu sec.')
title('Frequency Response')
```



# MATLAB Graphics

## PLOTTING 3D DATASETS



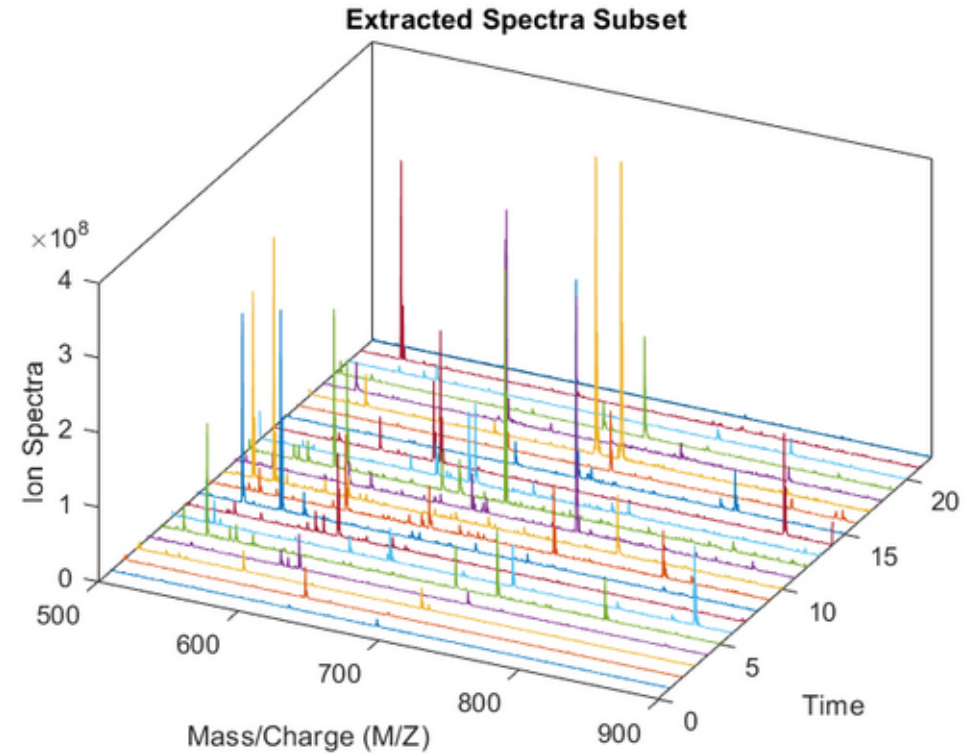
This is an example of how to create a 3D plot in MATLAB®.

```
% Load the spectra data
load spectraData masscharge time spectra

% Create the 3D plot
figure
plot3(masscharge, time, spectra)
box on

% Set the viewing angle and the axis limits
view(26, 42)
axis([500 900 0 22 0 4e8])

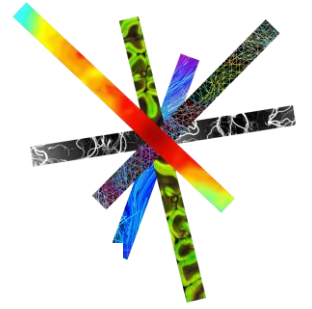
% Add title and axis labels
xlabel('Mass/Charge (M/Z)')
ylabel('Time')
zlabel('Ion Spectra')
title('Extracted Spectra Subset')
```





# MATLAB Graphics

## PLOTTING DATA / HISTOGRAMS / BARPLOTS



This is an example of how to create a vertical bar chart in MATLAB®.

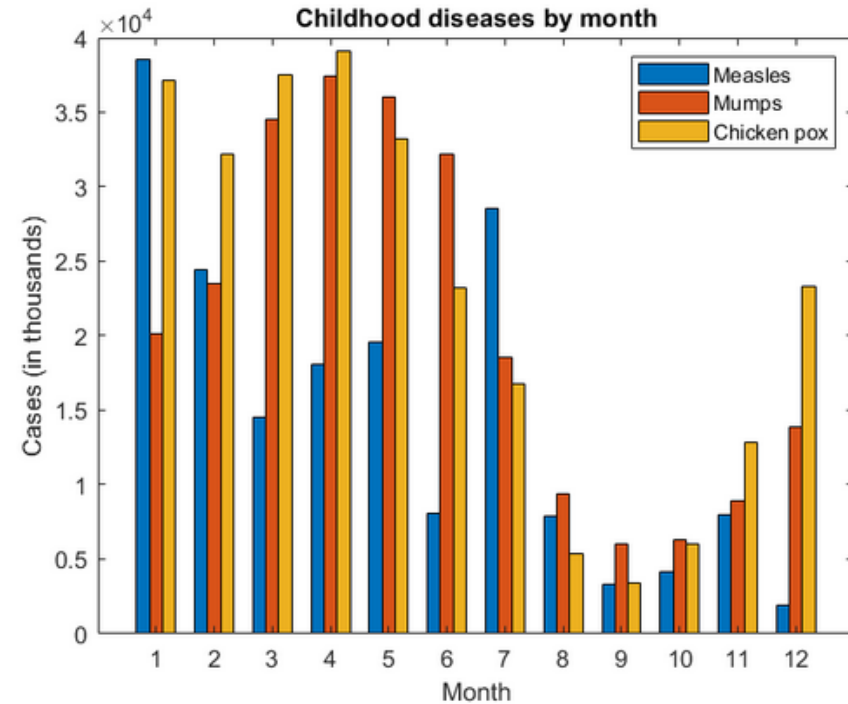
```
% Create data for childhood disease cases
measles = [38556 24472 14556 18060 19549 8122 28541 7880 3283 4135 7953 1884];
mumps = [20178 23536 34561 37395 36072 32237 18597 9408 6005 6268 8963 13882];
chickenPox = [37140 32169 37533 39103 33244 23269 16737 5411 3435 6052 12825 23332];

% Create a vertical bar chart using the bar function
figure
bar(1:12, [measles' mumps' chickenPox'], 1)

% Set the axis limits
axis([0 13 0 40000])
set(gca, 'XTick', 1:12)

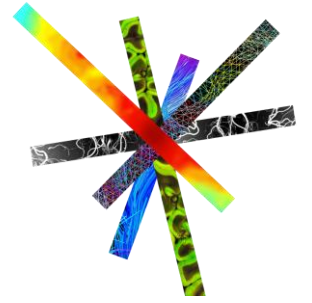
% Add title and axis labels
title('Childhood diseases by month')
xlabel('Month')
ylabel('Cases (in thousands)')

% Add a legend
legend('Measles', 'Mumps', 'Chicken pox')
```



# MATLAB Graphics

## PLOTTING DATA / HISTOGRAMS / BARPLOTS



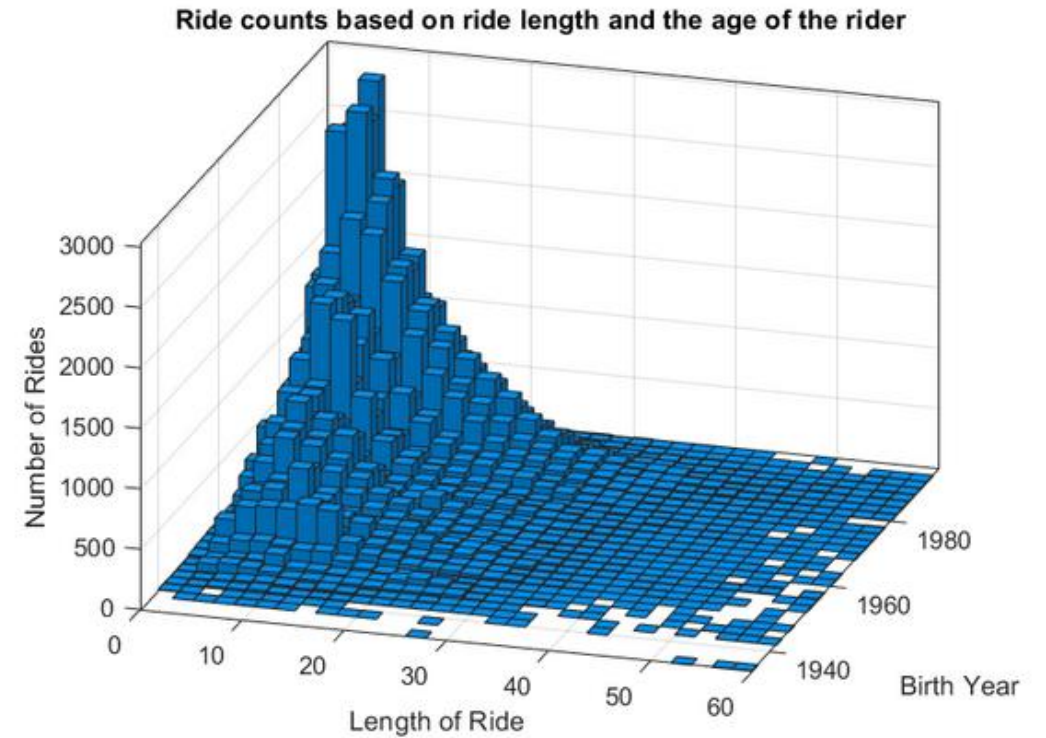
This is an example of how to create a bivariate histogram in MATLAB®.

```
% Check version
if verLessThan('matlab','8.6')
    error('histogram2 is available in R2015b or newer.')
end

% Load ride data from Boston's bike sharing program
load rideData rideData

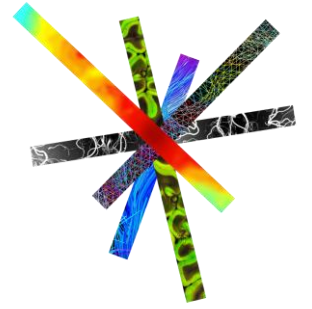
% Create bivariate histogram plot using the histogram2 function
histogram2(rideData.Duration, rideData.birth_date, 'Binwidth', [2 2])
xlabel('Length of Ride')
ylabel('Birth Year')
zlabel('Number of Rides')
title('Ride counts based on ride length and the age of the rider')

% Adjust view
view(17,30)
```



# MATLAB Graphics

## SHOWING TABULAR DATA AS HEATMAPS



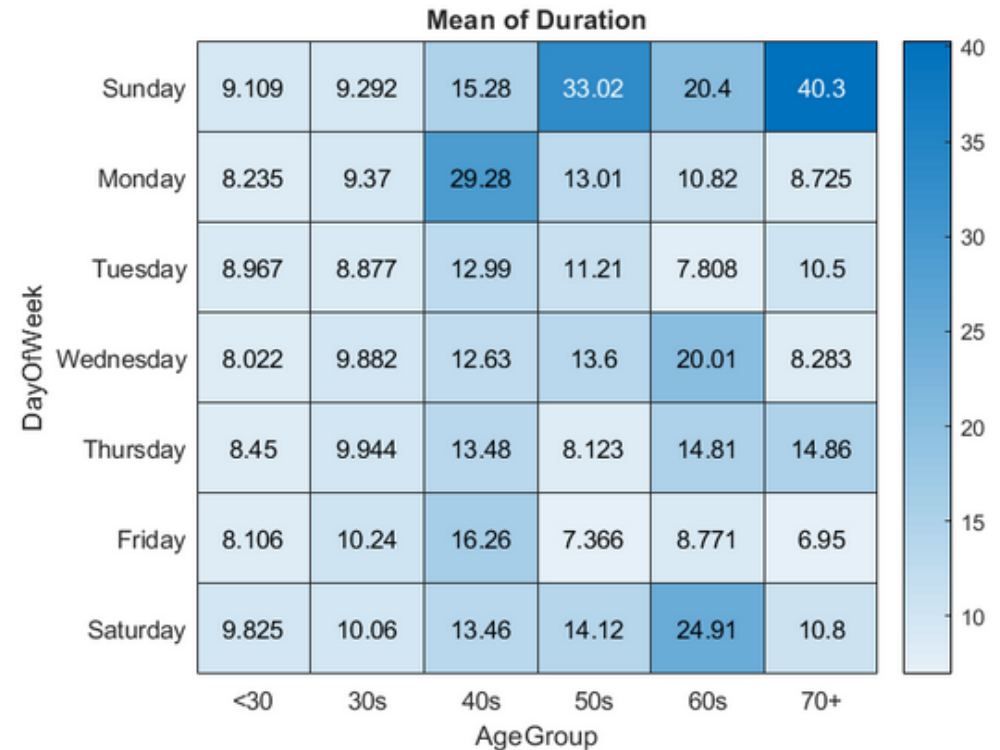
This is an example of how to create a heatmap chart in MATLAB®.

```
% Check version
if verLessThan('matlab','9.2')
    error('heatmap is available in R2017a or newer.')
end

% Load ride data from Boston's bike sharing program
load CambridgeData cambridge

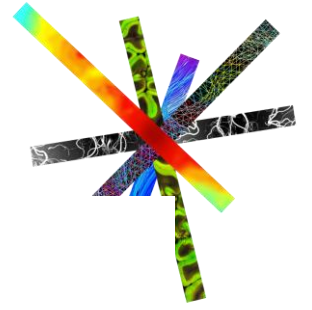
% Create a heatmap of DayOfWeek vs. AgeGroup, with color representing count
hm = heatmap(cambridge,'AgeGroup','DayOfWeek');

% Change the color to represent average Duration
hm.ColorVariable = 'Duration';
hm.ColorMethod = 'mean';
|
```



# MATLAB Graphics

## ASSEMBLING COMPLEX FIGURES USING SUBPLOTS

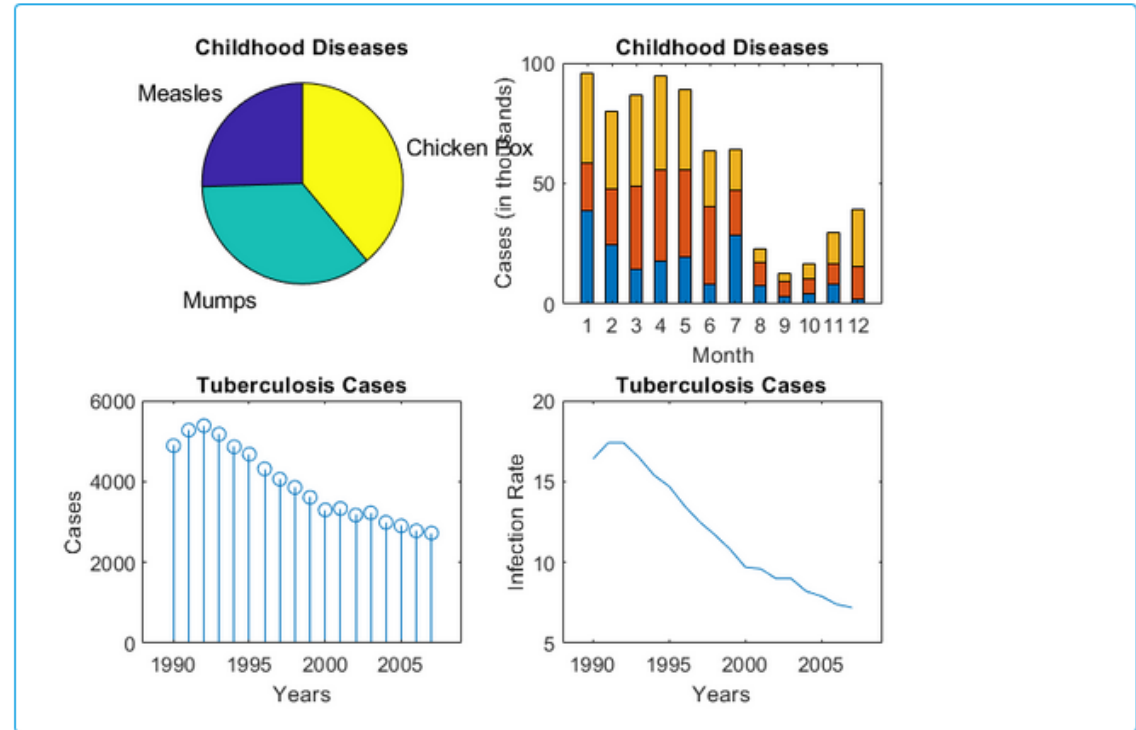


```
% Create the pie chart in position 1 of a 2x2 grid
figure
subplot(2, 2, 1)
pie([sum(measles) sum(mumps) sum(chickenPox)], {'Measles', 'Mumps', 'Chicken Po
title('Childhood Diseases')

% Create the bar chart in position 2 of a 2x2 grid
subplot(2, 2, 2)
bar(1:12, [measles/1000 mumps/1000 chickenPox/1000], 0.5, 'stack')
xlabel('Month')
ylabel('Cases (in thousands)')
title('Childhood Diseases')
axis([0 13 0 100])
set(gca, 'XTick', 1:12)

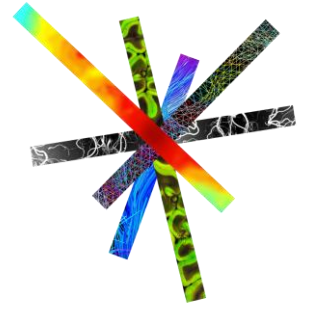
% Create the stem chart in position 3 of a 2x2 grid
subplot(2, 2, 3)
stem(years, cases)
xlabel('Years')
ylabel('Cases')
title('Tuberculosis Cases')
axis([1988 2009 0 6000])

% Create the line plot in position 4 of a 2x2 grid
subplot(2, 2, 4)
plot(years, rate)
xlabel('Years')
ylabel('Infection Rate')
title('Tuberculosis Cases')
axis([1988 2009 5 20])
```



# MATLAB Graphics

## VISUALIZING 2D/3D VECTOR FIELDS



This is an example of how to create a 2D quiver plot in MATLAB®.

```
% Create a grid of x and y points
[x, y] = meshgrid(-2:.2:2);

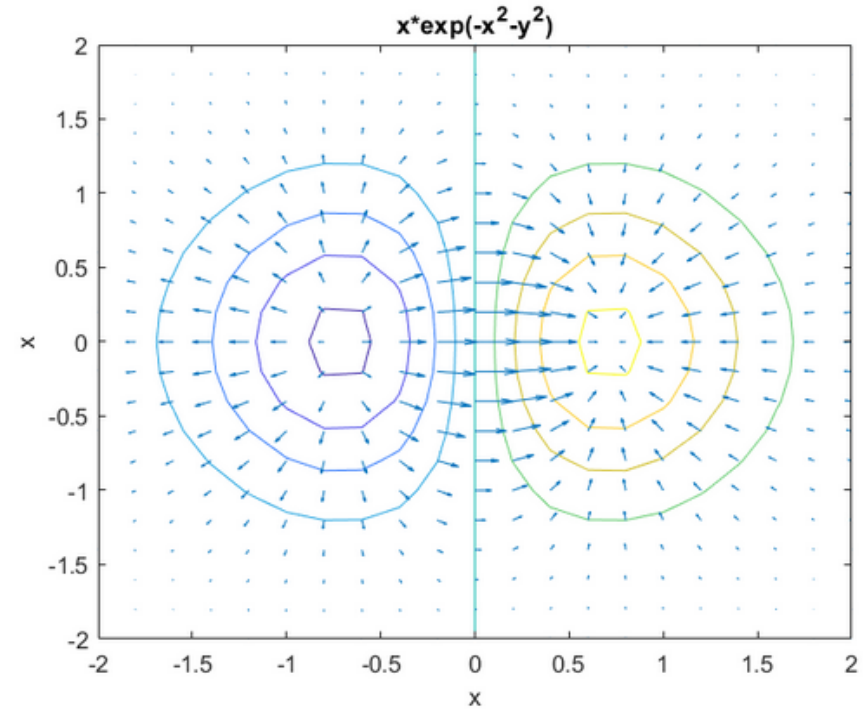
% Create the function z(x,y) and its gradient
z = x.*exp(-x.^2 - y.^2);
[dx, dy] = gradient(z, .2, .2);

% Create a contour plot of x, y, and z using the contour function
figure
contour(x,y,z)
hold on

% Create a quiver plot of x, y, and the gradients using the quiver function
q = quiver(x, y, dx, dy);

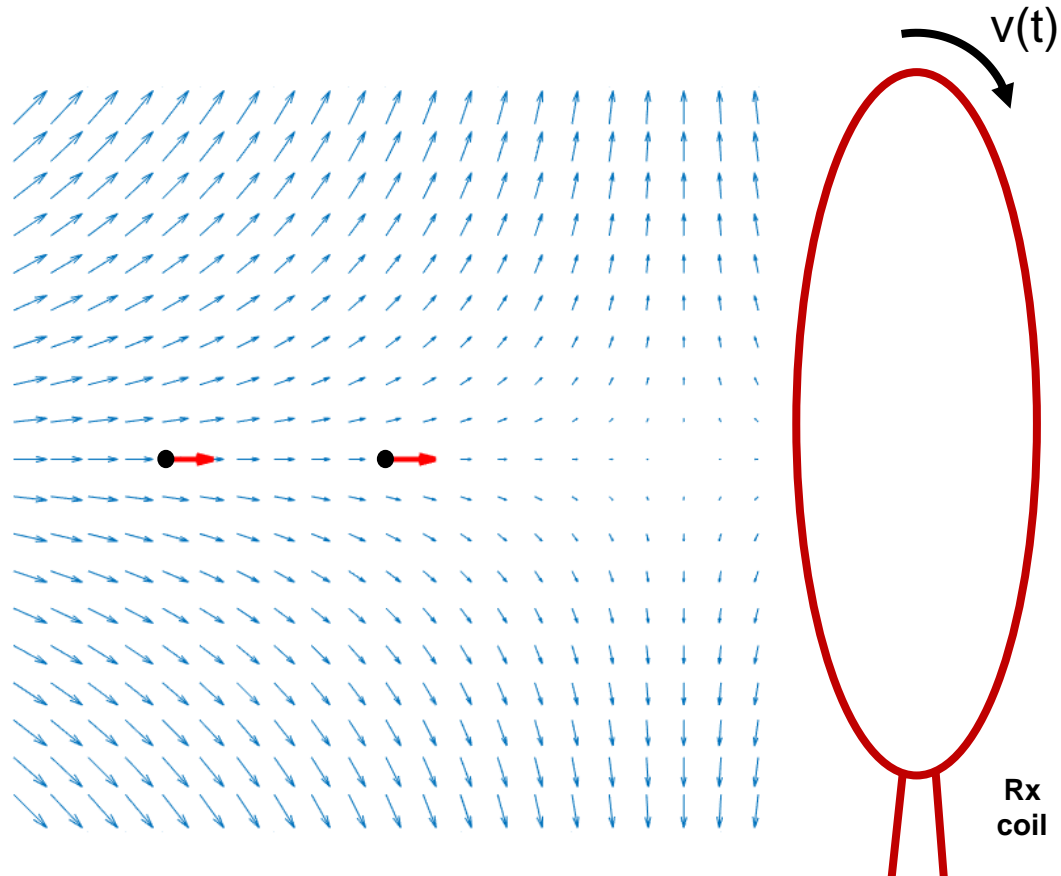
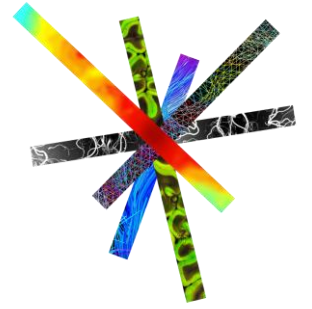
% Set the axis limits
xlim([-2 2])
ylim([-2 2])

% Add title and axis labels
title('x*exp(-x^2-y^2)')
xlabel('x')
ylabel('x')
```



# MATLAB Graphics

## VISUALIZING 2D/3D VECTOR FIELDS



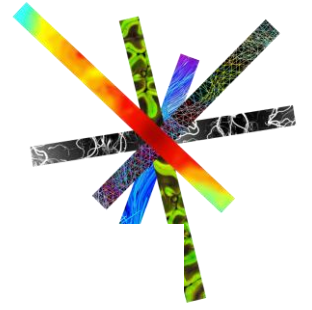
*Credits to Melissa Haskell*

*«Introduction to MATLAB», Why & How Series 2019*



# MATLAB Graphics

## SURFACE RENDERING WITH MATLAB



This is an example of how to create a surface contour plot in MATLAB®.

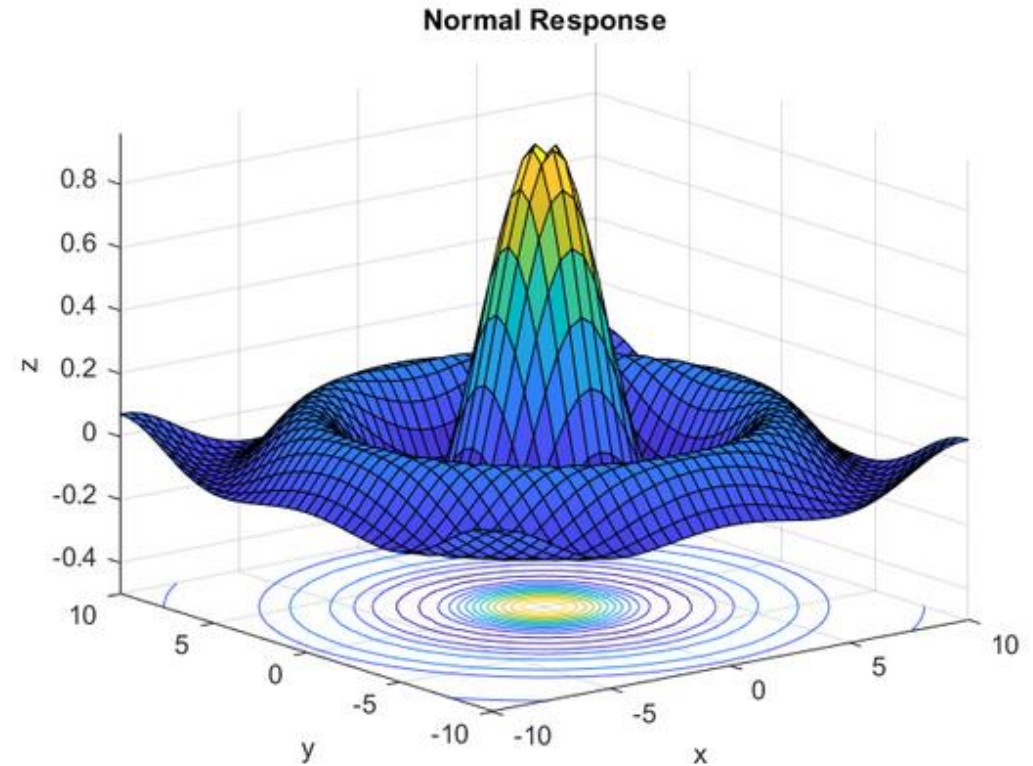
```
% Create a grid of x and y data
y = -10:0.5:10;
x = -10:0.5:10;
[X, Y] = meshgrid(x, y);

% Create the function values for Z = f(X,Y)
Z = sin(sqrt(X.^2+Y.^2)) ./ sqrt(X.^2+Y.^2);

% Create a surface contour plot using the surf function
figure
surf(X, Y, Z)

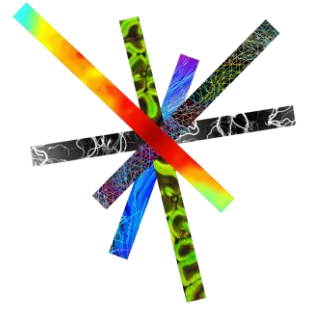
% Adjust the view angle
view(-38, 18)

% Add title and axis labels
title('Normal Response')
xlabel('x')
ylabel('y')
zlabel('z')
```



# MATLAB Graphics

## SURFACE RENDERING WITH MATLAB



This is an example of how to create a 3D mesh plot in MATLAB®.

```
% Create a grid of x and y data
y = -10:0.5:10;
x = -10:0.5:10;
[X, Y] = meshgrid(x, y);

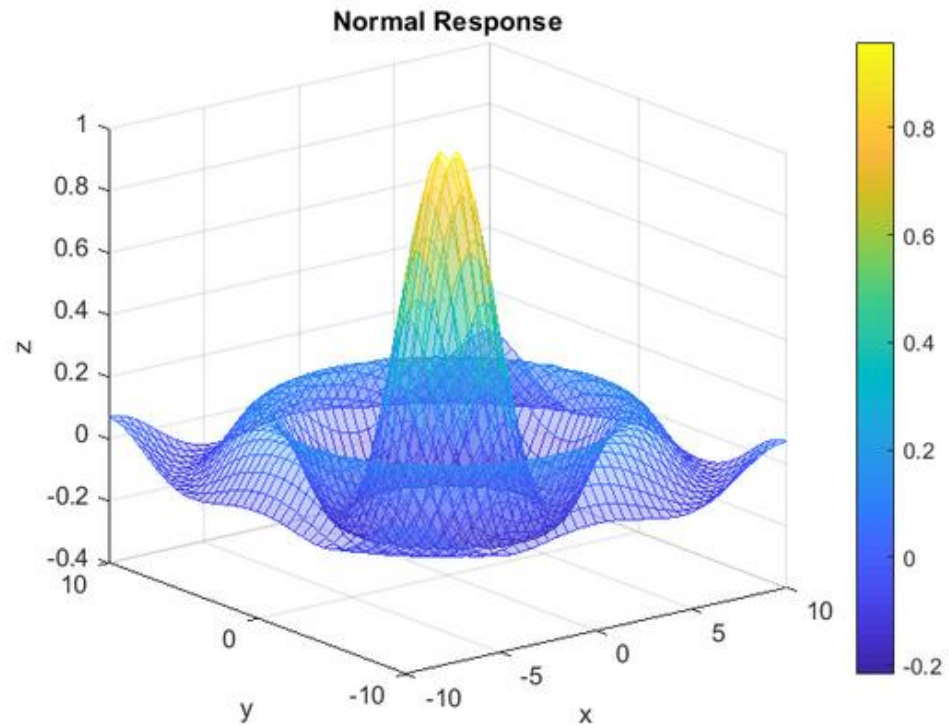
% Create the function values for Z = f(X,Y)
Z = sin(sqrt(X.^2+Y.^2)) ./ sqrt(X.^2+Y.^2);

% Create a surface contour plot using the mesh function
figure
s = mesh(X, Y, Z, 'FaceAlpha', '0.3');

% Adjust the view angle
view(-38, 18)

% Add title and axis labels
title('Normal Response')
xlabel('x')
ylabel('y')
zlabel('z')

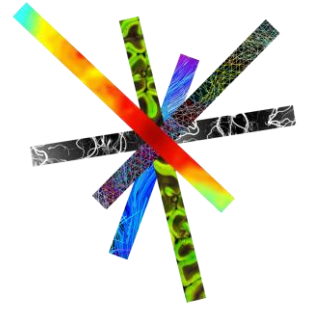
% Customize the plot
colorbar
s.FaceColor = 'flat';
```



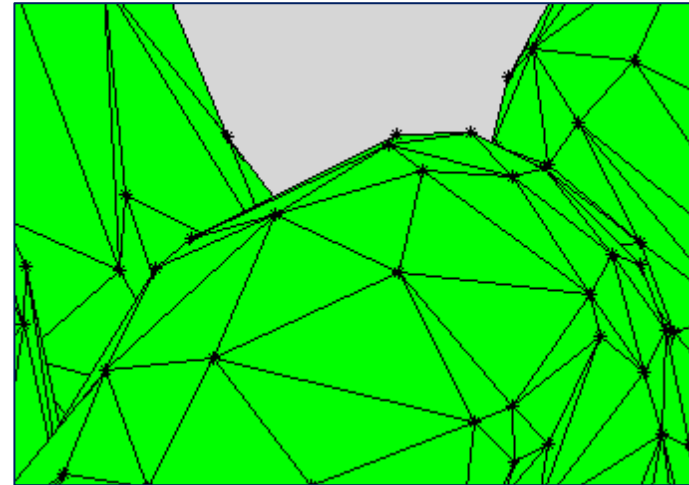
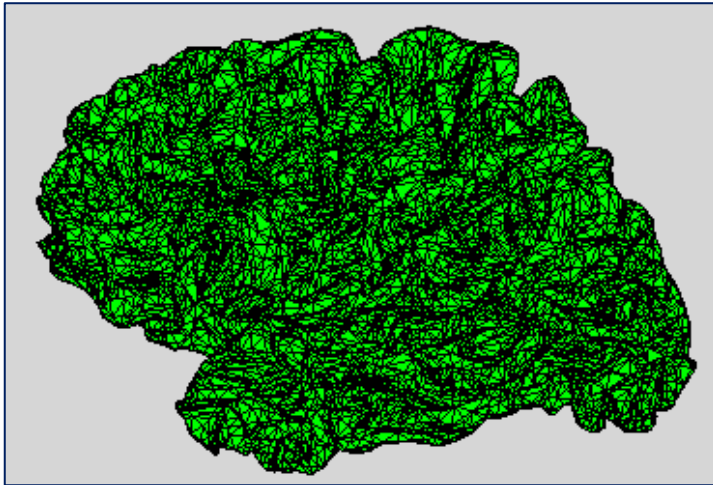


# MATLAB Graphics

## SURFACE RENDERING WITH MATLAB



Discrete surface consists of “**vertex points**” and “**edges**”:



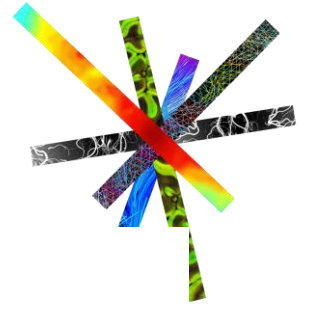
If you want to render your own mesh/surface in MATLAB, you need two lists of numbers:

- “**Vertices**” are the coordinates of surface points.
- “**Faces**” tell which three vertices form a given triangle.

*Credits to Melissa Haskell*

# MATLAB Graphics

## DISPLAY IMAGES



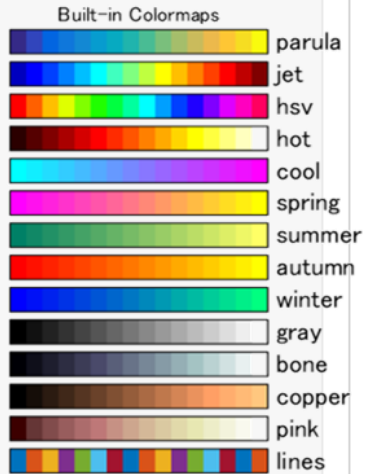
This is an example of how to display multiple images in a subplot in MATLAB®.

```
% Read the data for the original image  
load spine X  
original = X;
```

```
% Create the first image display using the image command  
figure  
ax(1) = subplot(1, 2, 1);  
image(original)  
axis square off  
title('Original image')  
colorbar('SouthOutside')
```

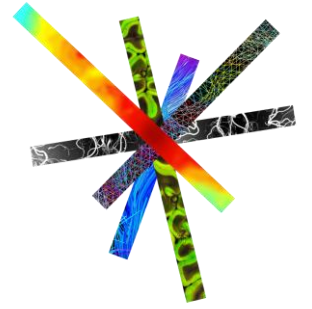
```
% Create the second image display using the imagesc  
ax(2) = subplot(1, 2, 2);  
imagesc(original, [0,40])  
axis square off  
title('Scaled image')  
colorbar('SouthOutside')
```

```
colormap(ax(1), 'bone')  
colormap(ax(2), 'bone')
```



# MATLAB Graphics

## HOW CRAZY CAN YOU GO?



figure

```
% Create isosurface patch
p = patch(isosurface(x, y, z, spd, 40));
isonormals(x, y, z, spd, p)
set(p, 'FaceColor', 'red', 'EdgeColor', 'none')

% Create isosurface end-caps
p2 = patch(isocaps(x, y, z, spd, 40));
set(p2, 'FaceColor', 'interp', 'EdgeColor', 'none')

% Adjust aspect ratio
daspect([1 1 1])

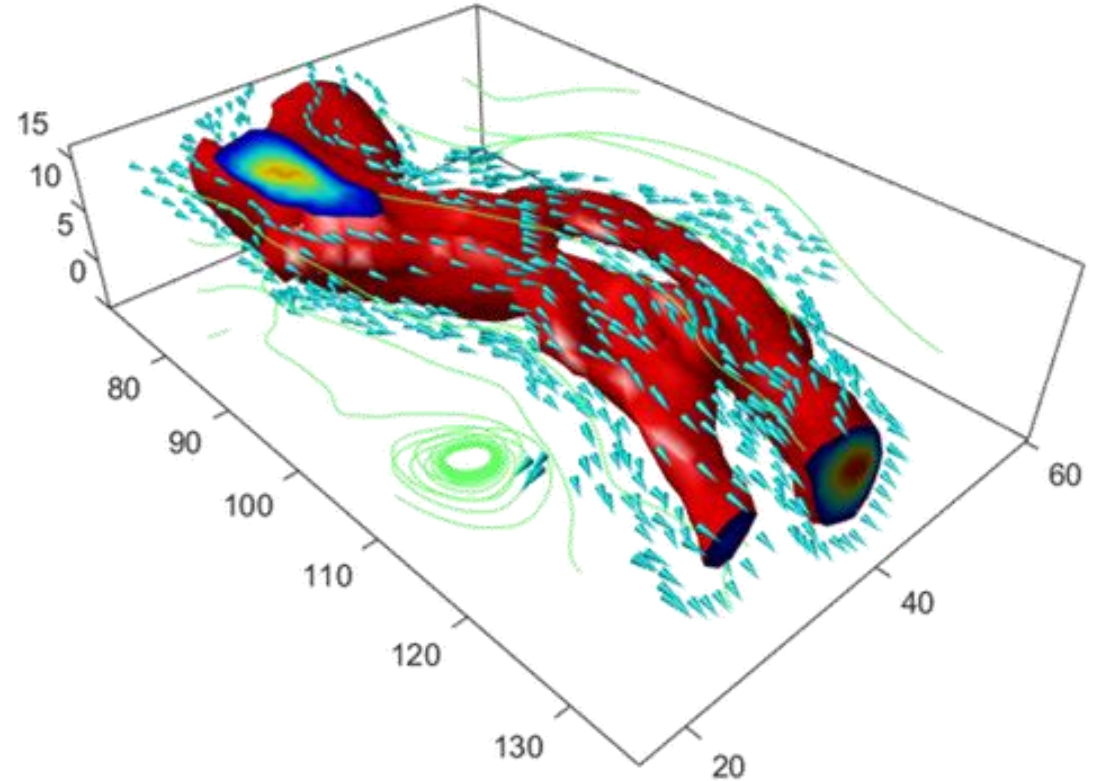
% Downsample patch|
[f, verts] = reducepatch(isosurface(x, y, z, spd, 30), .2);

% Create coneplot (velocity cone)
h = coneplot(x, y, z, u, v, w, verts(:, 1), verts(:, 2), verts(:, 3), 2);
set(h, 'FaceColor', 'cyan', 'EdgeColor', 'none')

% Create streamline
[sx, sy, sz] = meshgrid(80, 20:10:50, 0:5:15);
h2 = streamline(x, y, z, u, v, w, sx, sy, sz);
set(h2, 'Color', [.4 1 .4])

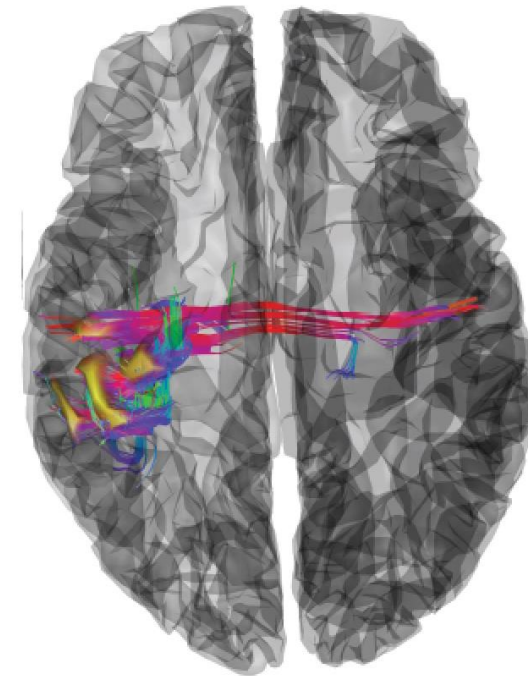
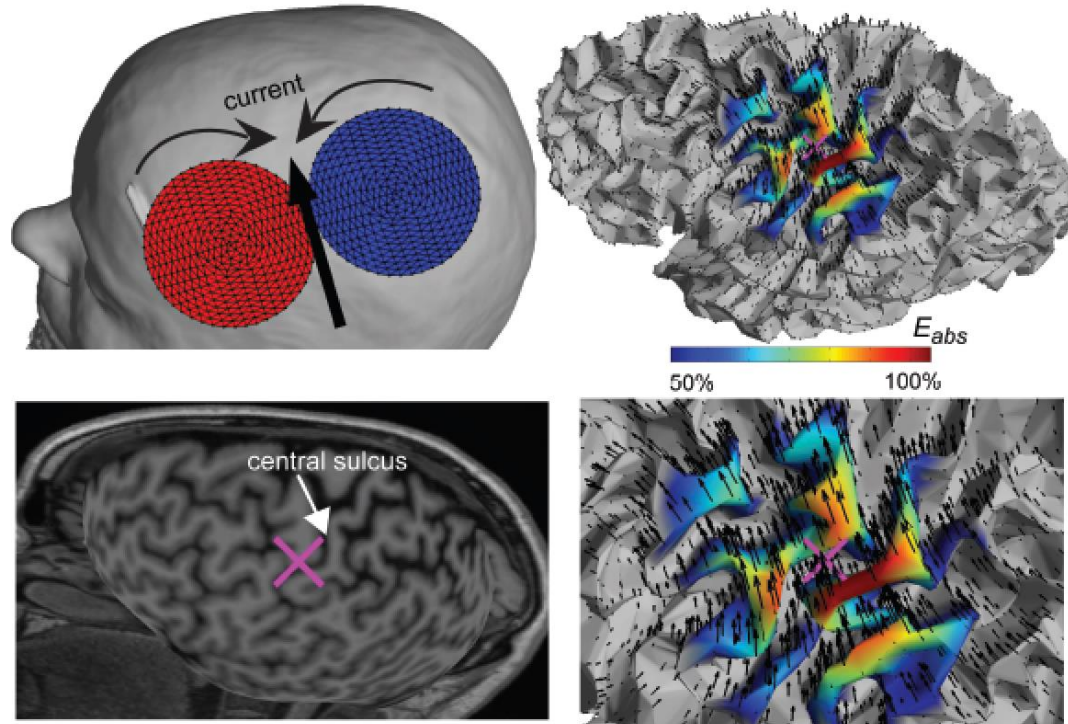
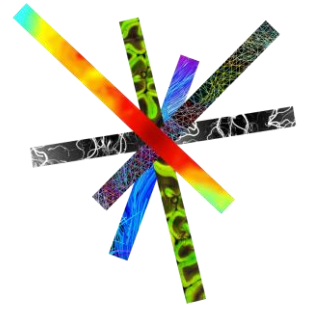
% Adjust colormap and axes settings
colormap(jet)
box on
axis tight
camproj perspective
camva(34)
campos([165 -20 65])
camtarget([100 40 -5])
camlight left
lighting gouraud
```

[isosurface](#)  
[isonormals](#)  
[isocaps](#)  
[coneplot](#)  
[streamline](#)  
[patch](#)  
[reducepatch](#)



# MATLAB Graphics

HOW CRAZY CAN YOU GO?

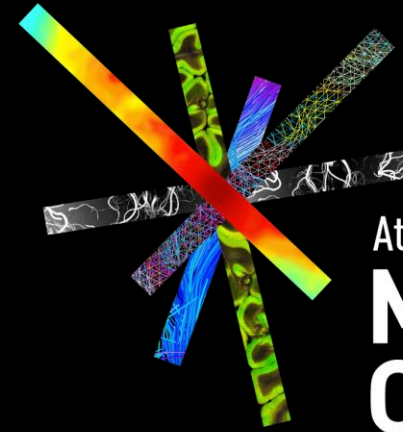


*Credits to Melissa Haskell*

*«Introduction to MATLAB», Why & How Series 2019*

# Overview

- **WHAT?**
- **WHY?**
- **HOW?**
  - GETTING STARTED
  - SCRIPTS, FUNCTIONS, AND THE EDITOR
  - VISUALIZATION TOOLS
- **BUT ... CAN I STILL USE IT, IF I DON'T KNOW HOW TO CODE?**



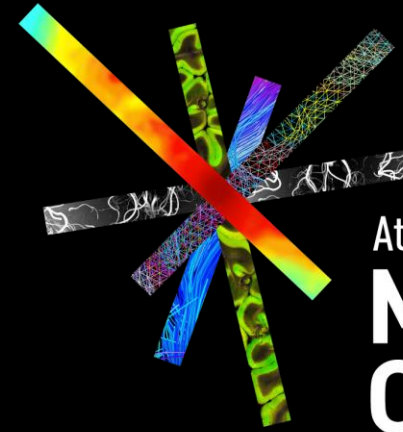
Athinoula A.

**Martinos  
Center**

For Biomedical Imaging

# Overview

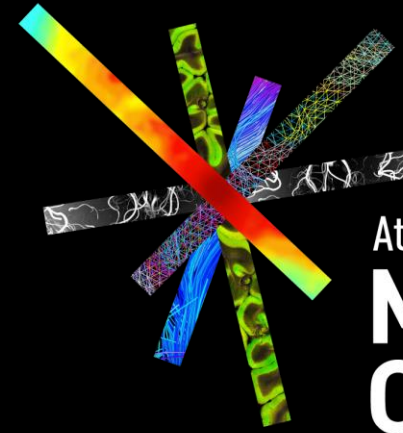
- **WHAT?**
- **WHY?**
- **HOW?**
  - GETTING STARTED
  - SCRIPTS, FUNCTIONS, AND THE EDITOR
  - VISUALIZATION TOOLS
- **BUT ... CAN I STILL USE IT, IF I DON'T KNOW HOW TO CODE?**



Athinoula A.  
**Martinos**  
**Center**  
For Biomedical Imaging



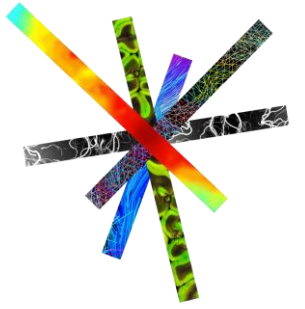
# Overview



Athinoula A.  
**Martinos**  
**Center**  
For Biomedical Imaging

- **WHAT?**
- **WHY?**
- **HOW?**
  - GETTING STARTED
  - SCRIPTS, FUNCTIONS, AND THE EDITOR
  - VISUALIZATION TOOLS
- **BUT ... CAN I STILL USE IT, IF I DON'T ~~KNOW HOW~~ TO CODE?**

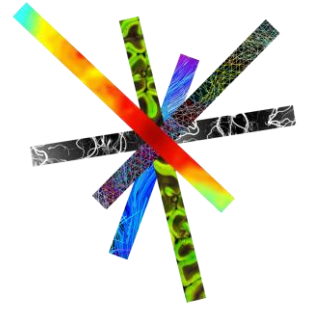
*want*



# MATLAB'S OWN OPTIONS



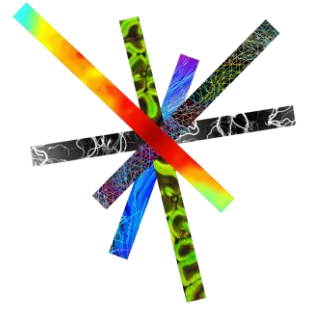
# Matlab APPS



- A MATLAB app is a **self-contained MATLAB program** with a user interface that automates a task or calculation.
- **All the operations** required to complete the task (getting data into the app, performing calculations on the data, and getting results) **are performed within the app.**



# Matlab APPS



## MACHINE LEARNING AND DEEP LEARNING

- Classification Learner
- Deep Network Designer
- Neural Net Clustering
- Neural Net Fitting
- Neural Net Pattern Recog...
- Neural Net Time Series
- Regression Learner

## MATH, STATISTICS AND OPTIMIZATION

- Curve Fitting
- Distribution Fitter
- Optimization
- PDE Modeler

## CONTROL SYSTEM DESIGN AND ANALYSIS

- Control System Designer
- Control System Tuner
- Fuzzy Logic Designer
- Linear System Analyzer
- Model Reducer
- MPC Designer
- Neuro-Fuzzy Designer
- PID Tuner
- SLAM Map Builder
- System Identification

## SIGNAL PROCESSING AND COMMUNICATIONS

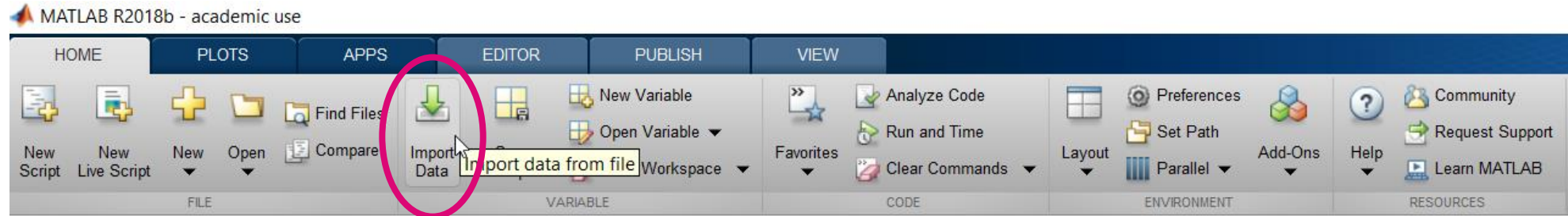
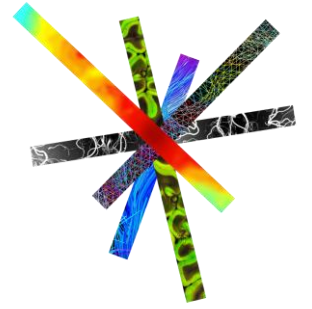
- Bit Error Rate Analysis
- Eye Diagram Scope
- Filter Builder
- Filter Designer
- LTE Downlink RMC Generator
- LTE Test Model Generator
- LTE Throughput Analyzer
- LTE Uplink RMC Generator
- Radar Equation Calculator
- Radar Waveform Analyzer
- RF Budget Analyzer
- Sensor Array Analyzer
- Signal Analyzer
- Signal Multiresolutio...
- Sonar Equation Calculator
- Wavelet Analyzer
- Wavelet Signal Denoiser
- Window Designer
- Wireless Waveform Ge...

## IMAGE PROCESSING AND COMPUTER VISION

- Camera Calibrator
- Color Thresholder
- DICOM Browser
- Image Browser
- Image Acquisition
- Image Batch Processor
- Image Labeler
- Image Region Analyzer
- Image Segmenter
- Image Viewer
- Map Viewer
- OCR Trainer
- Registration Estimator
- Stereo Camera Calibrator
- Video Labeler
- Video Viewer
- Volume Viewer

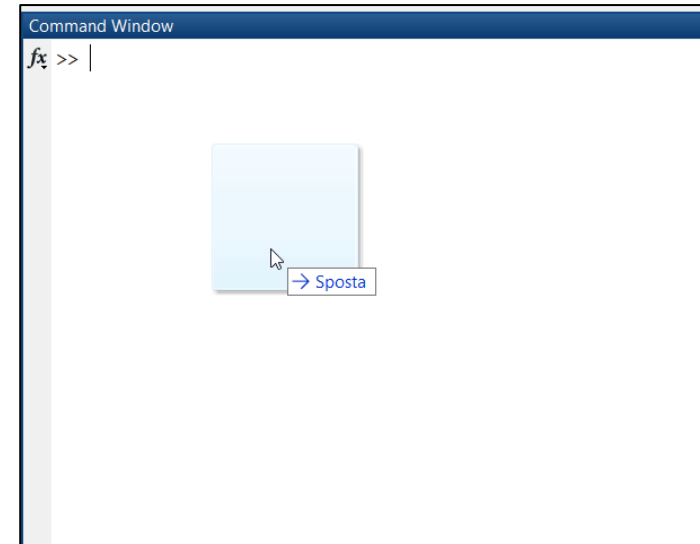
# Matlab APPS

## INTERACTIVE DATA IMPORT



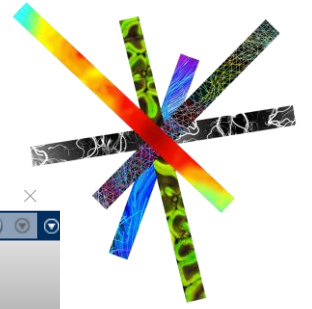
Click 'Import Data' from the toolbar and select file, or ...

... Drag & Drop data file in Command Window



# Matlab APPS

## INTERACTIVE DATA IMPORT



Import - D:\--Cloud\GoogleDrive\_personal\PRESENTATIONS,TALKS,2020\_03\_12\_IntroToMATLAB\_WhyNHow\examples\data.csv

**IMPORT** | **VIEW**

Delimited | Column delimiters: Comma | Range: A2:U22 | Output Type: Table |  Replace | unimportable cells with NaN |  Import Selection

Fixed Width | Variable Names Row: 1 |  Text Options

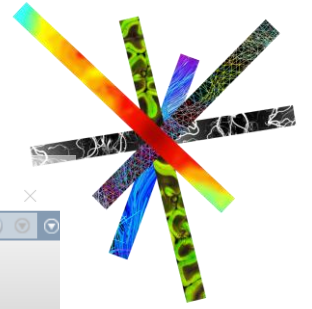
DELIMITERS | SELECTION | IMPORTED DATA | UNIMPORTABLE CELLS | IMPORT

data.csv

	data																				
	time	ts1	ts2	ts3	ts4	ts5	ts6	ts7	ts8	ts9	ts10	ts11	ts12	ts13	ts14	ts15	ts16	ts17	ts18	ts19	t
Number	Number	Number	Number	Number	Number	Number	Number	Number	Number	Number	Number	Number	Number	Number	Number	Number	Number	Number	Number	Number	Number
1	time	ts1	ts2	ts3	ts4	ts5	ts6	ts7	ts8	ts9	ts10	ts11	ts12	ts13	ts14	ts15	ts16	ts17	ts18	ts19	ts20
2	1790	4.44	4.0194	3.4529	4.3665	4.0912	4.0518	4.07	3.5966	3.7349	3.3375	4.3549	3.6917	3.5253	3.9849	4.1481	4.1406	3.6543	3.5621	3.965	4.136
3	1800	5.2668	4.6787	5.5902	5.3637	5.7056	4.6873	5.7753	4.7356	5.4916	5.9903	5.9428	6.3217	4.4341	4.2881	5.7204	5.6247	5.2752	5.0764	4.1571	5.586
4	1810	7.5232	6.5081	8.3269	6.4535	7.7604	7.2286	7.5634	5.9267	6.9466	8.2151	7.5496	7.0731	7.7481	6.8135	6.5532	7.4084	9.1318	6.5213	8.4164	6.159
5	1820	9.2513	8.9724	7.9744	8.7772	8.1789	9.1674	9.2151	9.194	9.8588	8.004	10.585	9.366	10.746	9.31	9.75	9.9795	8.4992	10.888	10.831	8.671
6	1830	11.583	11.314	12.32	12.681	13.597	13.041	12.237	11.541	9.5919	15.407	14.023	11.742	15.005	15.24	14.96	11.777	13.613	11.625	12.148	13.16
7	1840	11.845	16.637	16.956	16.772	16.943	16.672	18.462	18.208	17.897	15.245	16.447	15.745	14.48	18.664	17.292	16.249	15.259	20.209	17.481	16.50
8	1850	24.547	21.021	18.498	21.1	21.344	22.661	21.55	22.366	27.382	23.624	24.091	20.899	20.012	22.968	22.387	22.854	25.481	22.235	24.901	26.08
9	1860	30.5	30.503	34.042	31.967	29.222	28.157	35.126	36.955	34.663	34.851	30.461	32.511	26.772	35.511	32.834	29.24	32.428	26.842	32.608	35.61
10	1870	37.838	36.815	36.999	43.489	43.546	37.352	41.652	44.431	42.116	39.168	35.127	44.765	38.439	34.567	37.538	39.881	41.117	36.212	41.288	36.35
11	1880	52.236	48.143	59.799	48.939	46.135	54.048	51.644	51.023	48.997	61.724	53.386	52.848	47.11	48.452	52.425	62.073	48.8	54.891	49.634	54.59
12	1890	53.972	59.733	60.441	61.613	55.12	73.874	62.92	61.121	64.038	80.214	63.324	68.273	71.166	71.785	62.052	59.867	64.442	69.542	62.656	71.67
13	1900	70.456	85.373	79.11	59.268	77.632	67.18	78.779	84.756	77.856	77.051	74.578	86.198	64.941	87.418	75.861	80.921	87.191	77.218	76.669	78.43
14	1910	102.56	97.615	81.49	84.874	110.5	113.87	124.45	81.452	92.887	74.455	94.684	69.004	75.97	98.719	96.239	82.483	71.069	94.644	84.735	106.9
15	1920	112.02	106.32	99.095	90.973	105.97	121.83	104.51	112.82	96.922	101.84	116.14	103.93	107.87	110.89	120.1	119.86	88.436	112.39	120.74	116.9
16	1930	107.07	104.79	108.45	118.06	126.59	124.87	103.69	114.58	118.47	112.39	127.63	127.14	137.45	115.6	128.35	110.9	127.9	104.88	122.88	125.4
17	1940	102.68	110.29	136.87	138.62	119.34	127.73	156.92	126.43	129.4	121.63	134.26	141.15	121.13	141.51	153.41	134.45	123.08	124.04	140.74	143.2
18	1950	142.09	121.09	170.32	173.66	175.93	140.17	159.89	140.58	143.46	133.71	154.92	131.04	131.63	138.22	120.13	141.38	146.23	123.12	137.82	153.6
19	1960	182.83	225.63	168.37	211.19	181.24	193.91	167.4	189.3	193.98	180.4	179.92	160.99	176.33	189.28	170.96	188.17	152.21	170.96	159.75	171.5
20	1970	224.32	224.93	213.95	202.6	215.87	190.76	258.66	189.05	257.04	248.19	189.12	221.21	171.45	210.78	209.84	205.23	186.45	224.46	203.14	212.3
21	1980	228.62	232.32	215.08	219.25	204.94	216.04	238.98	202.41	196.53	210.29	244.32	223.86	226.89	252.31	207.58	225.5	217.35	242.75	220.77	227.3
22	1990	220.79	224.47	251.24	269.03	269.94	270.68	256.02	262.45	251.89	241.72	283.74	262.46	269.3	238.11	216.97	322.04	230.65	305.6	278.42	239.6

# Matlab APPS

## INTERACTIVE DATA IMPORT



Import - D:\--Cloud\GoogleDrive\_personal\PRESENTATIONS,TALKS,2020\_03\_12\_IntroToMATLAB\_WhyNHow\examples\data.csv

IMPORT VIEW

Delimited Column delimiters: Range: A2:U22 Output Type:  Replace unimportable cells with NaN

Import Selection

Import Selection

Import Data

Generate Script

Generate Function

Output Type:

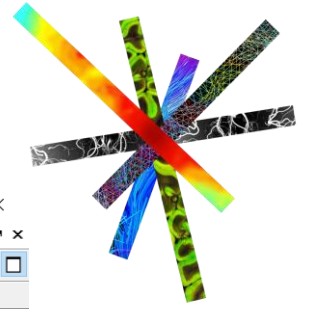
- Table
- Table
- Column vectors
- Numeric Matrix
- String Array
- Cell Array

Replace unimportable cells with NaN

	ts3	ts4	ts5	ts6	ts7	ts8	ts9	ts10	ts11	ts12	ts13	ts14	ts15	ts16	ts17	ts18	ts19	ts20																																																																												
11	1880	52.236	48.143	59.799	44.146	3.5966	3.7349	3.3375	4.3549	3.6917	3.2834	29.24	32.428	26.842	32.608	35.61	37.538	39.881	41.117	36.212	41.288	36.35	52.425	62.073	48.8	54.891	49.634	54.59	62.052	59.867	64.442	69.542	62.656	71.67	75.861	80.921	87.191	77.218	76.669	78.43	96.239	82.483	71.069	94.644	84.735	106.9	120.1	119.86	88.436	112.39	120.74	116.9	128.35	110.9	127.9	104.88	122.88	125.4	153.41	134.45	123.08	124.04	140.74	143.2	120.13	141.38	146.23	123.12	137.82	153.6	170.96	188.17	152.21	170.96	159.75	171.5	209.84	205.23	186.45	224.46	203.14	212.3	207.58	225.5	217.35	242.75	220.77	227.3	216.97	322.04	230.65	305.6	278.42	239.6

# Matlab APPS

## CURVE FITTING



Curve Fitting Tool

File Fit View Tools Desktop Window Help

untitled fit 1

Fit name:

X data:

Y data:

Z data:

Weights:

Interpolant:

Method:

Center and scale

Auto fit

Results

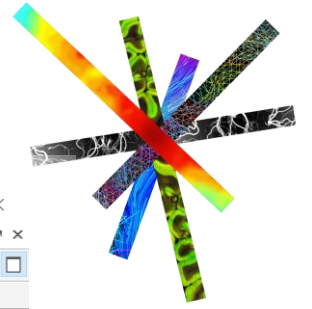
Select data to fit curves or surfaces.

Table of Fits

Fit name ^	Data	Fit type	SSE	R-square	DFE	Adj R-sq	RMSE	# Coeff	Validation Data	Validation SSE	Validation RMSE
<input checked="" type="checkbox"/> untitled fit 1		linearinterp									

# Matlab APPS

## CURVE FITTING



Curve Fitting Tool

File Fit View Tools Desktop Window Help

untitled fit 1

Fit name: untitled fit 1

X data: (none)

Y data: (none)

Polynomial

Gaussian

Interpolant

Polynomial

Power

Rational

Smoothing Spline

Sum of Sine

Weibull

Auto fit

Fit

Stop

Fit name: untitled fit 1

X data: cdate

Y data: pop

Z data: (none)

Weights: (none)

Polynomial

Degree: 3

Robust: Off

Center and scale

Fit Options...

Custom Equation

z = f( x , y )

= 1 a + b\*sin(m\*pi\*x\*y)

2 + c\*exp(-(w\*y)^2)

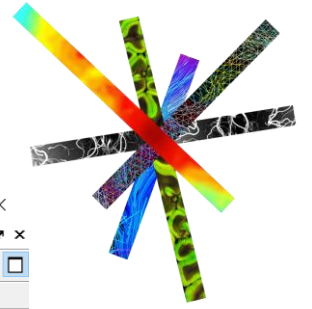
Fit Options...

Fit name ^	Data	Fit type
untitled fit 1		linearinterp

Validation RMSE

# Matlab APPS

## CURVE FITTING



Curve Fitting Tool

File Fit View Tools Desktop Window Help

polynomial

Fit name: polynomial

X data: cdate

Y data: pop

Z data: (none)

Weights: (none)

Polynomial

Degree: 2

Robust: Off

Center and scale

Auto fit

Fit

Stop

Fit Options...

Results

Linear model Poly2:  
 $f(x) = p1 \cdot x^2 + p2 \cdot x + p3$   
 where x is normalized by mean 1890 and std 62.05

Coefficients (with 95% confidence bounds):

p1 = 25.18 (23.58, 26.79)  
 p2 = 75.43 (74.04, 76.83)  
 p3 = 61.74 (59.7, 63.79)

Goodness of fit:  
 SSE: 159  
 R-square: 0.9987  
 Adjusted R-square: 0.9986  
 RMSE: 2.972

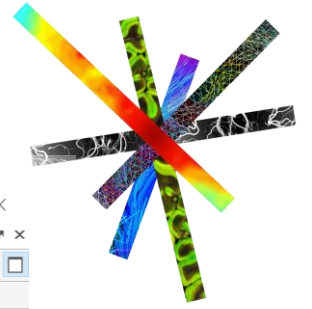
Table of Fits

Fit name ^	Data	Fit type	SSE	R-square	DFE	Adj R-sq	RMSE	# Coeff	Validation Data	Validation SSE	Validation RMSE
polynomial	pop vs. cdate	poly2	159.0293	0.9987	18	0.9986	2.9724	3			



# Matlab APPS

## CURVE FITTING



Curve Fitting Tool

File Fit View Tools Desktop Window Help

polynomial x +

Fit name: polynomial

X data: cdate

Polynomial

Degree: 2

Robust: Off

Auto fit

Fit

Stop

Fit Options...

**Results**

Linear model Poly2:

$$f(x) = p1*x^2 + p2*x + p3$$

Coefficients (with 95% confidence bounds):

p1 = 0.006541 (0.006124, 0.006958)

p2 = -23.51 (-25.09, -21.93)

p3 = 2.113e+004 (1.964e+004, 2.262e+004)

Goodness of fit:

SSE: 159

R-square: 0.9987

Adjusted R-square: 0.9986

RMSE: 2.972

**Save Fit to MATLAB Workspace**

Save fit to MATLAB object named: fittedmodel

Save goodness of fit to MATLAB struct named: goodness

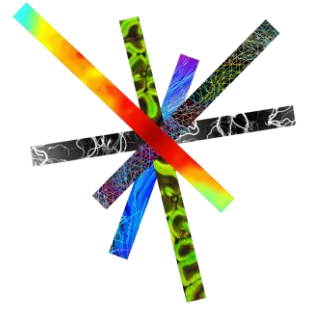
Save fit output to MATLAB struct named: output

OK Cancel

DFE	Adj R-sq	RMSE	# Coeff	Validation Data	Validation SSE	Validation RMSE
18	0.9986	2.9724	3			

# Matlab APPS

## DICOM BROWSER



DICOM Browser - Browser

DICOM

Load Collection Export Volume

Browser

Patient/Study/Series details

Studies

	StudyDateTime	PatientName	PatientSex	Modality	StudyDescription	StudyInstanceUID
1	30-Apr-1993 11:27:24	Anonymized		CT	RT ANKLE	1.2.840.113619.2.1.1.322987881.621.736170080.681
2	14-Dec-2013 15:47:31	GORBERG MITZI	F	MR	CSP	1.2.840.113619.2.244.3596.11880862.13689.1386517653.214
3	03-Oct-2011 19:18:11		M	MR	RIGHT KNEE	1.3.6.1.4.1.9590.100.1.2.320418845013189618318250681693358291211
4	03-Oct-2011 19:18:11		M	MR	RIGHT KNEE	1.3.6.1.4.1.9590.100.1.2.320498134711034521212730362051554545799
5	30-Jan-1994 11:25:01	Anonymized		US	Echocardiogram	999.999.3859744

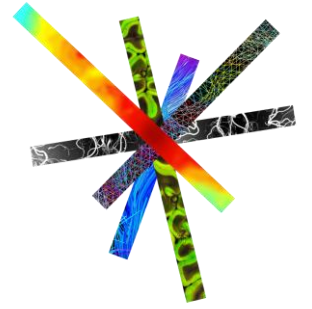
Series viewer

Series

	SeriesDateTime	Rows	Columns	Frames	SeriesDescription	SeriesInstanceUID
1	14-Dec-2013 15:	512	512	22	AX T2	1.2.840.113619.2.244.3596.11880862.13689.1386517653.217

# Matlab APPS

## DICOM BROWSER



DICOM Browser - Browser

DICOM

Load Collection Export Volume

Browser

Patient/Study/Series details

Study	StudyDateTime	PatientName	PatientSex	Modality	StudyDescription	StudyInstanceUID
1	30-Apr-1993 11:27:24	Anonymized		CT	RT ANKLE	1.2.840.113619.2.1.1.322987881.621.736170080.681
2	14-Dec-2013 15:47:31	GORBERG MITZI	F	MR	CSP	1.2.840.113619.2.244.3596.11880862.13689.1386517653.214
3	03-Oct-2011 19:18:11		M	MR	RIGHT KNEE	1.3.6.1.4.1.9590.100.1.2.320418845013189618318250681693358291211
4	03-Oct-2011 19:18:11		M	MR	RIGHT KNEE	1.3.6.1.4.1.9590.100.1.2.320498134711034521212730362051554545799
5	30-Jan-1994 11:25:01	Anonymized		US	Echocardiogram	999.999.3859744

Series viewer

DICOM

Load Collection Export Volume

FILE

Browser

Patient/Study/

Studies

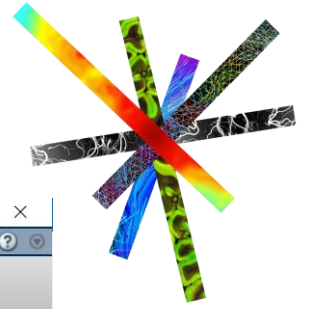
- Export Series to Workspace
- Export Series to Volume Viewer App
- Export Series to Video Viewer App

Series	SeriesDateTime	Rows	Columns	Frames	S
1	14-Dec-2013 15:47:31	512	512	22 AX	

Study	StudyDateTime	PatientName	PatientSex
1	30-Apr-1993 11:27:24	Anonymized	
2	14-Dec-2013 15:47:31	GORBERG MITZI	F
3	03-Oct-2011 19:18:11		M
4	03-Oct-2011 19:18:11		M
5	30-Jan-1994 11:25:01	Anonymized	

# Matlab APPS

## VOLUME VIEWER



Volume Viewer - Rendering Editor

VOLUME VIEWER

Load Volume  Specify Dimensions X-axis 1 units/vx  Upsample To Cube Y-axis 1 units/vx  Use File Metadata Z-axis 12.0135 units/vx

Volume Slice Planes Linear Grayscale CT Bone MRI Background Color Default Layout

FILE SPATIAL REFERENCING VIEW VOLUME RENDERING BACKGROUND COLOR LAYOUT

XY Slice 3D Volume

XZ Slice

YZ Slice

Orientation Axes

Rendering Editor

Volume Rendering

Opacity

Image Intensity

Lighting

# Matlab APPS

## IMAGE SEGMENTER

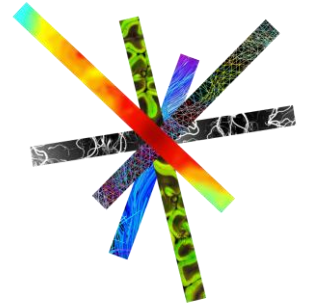


Image Segmenter - Segmentation

SEGMENTATION

Load Image Load Mask New Segmentation Include Texture Features Threshold Graph Cut Local Graph Cut Flood Fill Morphology Active Contours Zoom in Zoom out Pan Opacity Show Binary Export

Data Browser

Segmentations

1 Segmentation 1

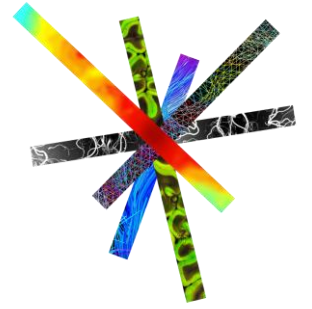
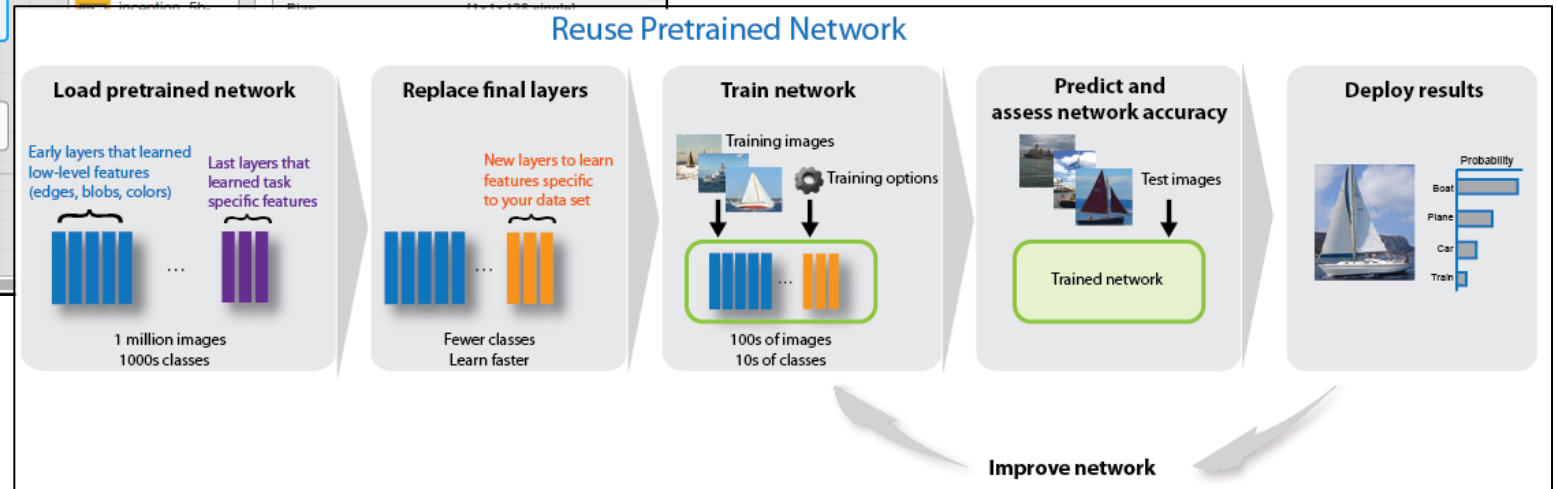
History

1 Load

2 Threshold image - global threshold

# Matlab APPS

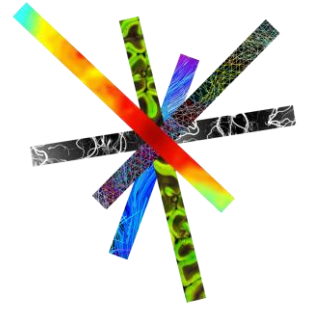
## DEEP LEARNING NETWORK DESIGNER



# Mathworks File Exchange Platform

SHARING CODE, CUSTOM TOOLBOXES AND APPS WITH OTHER USERS



MathWorks® Products Solutions Academia Support Community Events Get MATLAB MS

File Exchange Clear Filters x Search File Exchange File Exchange Q

MATLAB Central Files Authors My File Exchange Contribute About Trial software

Filter by Source  
 Community 2,403  
 MathWorks 30

Filter by Category  
 < Clear Categories

Using MATLAB  
 Language Fundamentals 877  
 Data Import and Analysis 997  
 Mathematics 1,388  
 Graphics 1,856  
 Programming 366  
 App Building 409  
 Software Development Tools 144  
 External Language Interfaces 431  
 Environment and Settings 119  
 Installation, Licensing, and Activation 10  
 Parallel Computing 135  
 Application Deployment 59  
 Database Access and Reporting 122

Applications  
 Science and Industry 3,368  
 Image Processing and 2,433

2,433 RESULTS

### Image Processing and Computer Vision (2,433)

**export\_fig**

Exports figures nicely to a number of vector & bitmap formats

1518 Downloads ★★★★★

**Simulink Support Package for Arduino Hardware**

Run models on Arduino boards.

1456 Downloads ★★★★☆

**MATLAB Support Package for USB Webcams**

Acquire images and video from UVC compliant webcams.

744 Downloads ★★★★★

MathWorks® File Exchange Search File Exchange File Exchange Q

**File Exchange badges are here!**  
 View badges you can earn by participating in the File Exchange community.

**export\_fig** ★★★★★ 749 Ratings

version 2.0.0.0 (85.3 KB) by Yair Altman  
 1518 Downloads Updated 30 May 2018

Exports figures nicely to a number of vector & bitmap formats  
[https://github.com/altmany/export\\_fig](https://github.com/altmany/export_fig) view license on GitHub

+ Follow Download from GitHub

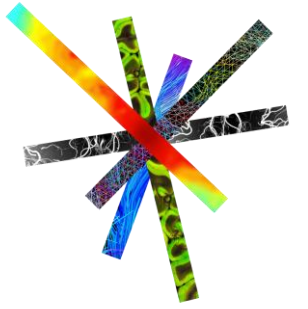
Overview Functions

**Editor's Note:** Popular File 2011 2012 2013 2014 2015 2016 2017 2018

This file was selected as MATLAB Central **Pick of the Week**

This function saves a figure or single axes to one or more vector and/or bitmap file formats, and/or outputs a rasterized version to the workspace, with the following properties:

- Figure/axes reproduced as it appears on screen
- Cropped/padded borders (optional)
- Embedded fonts (pdf only)
- Improved line and grid line styles

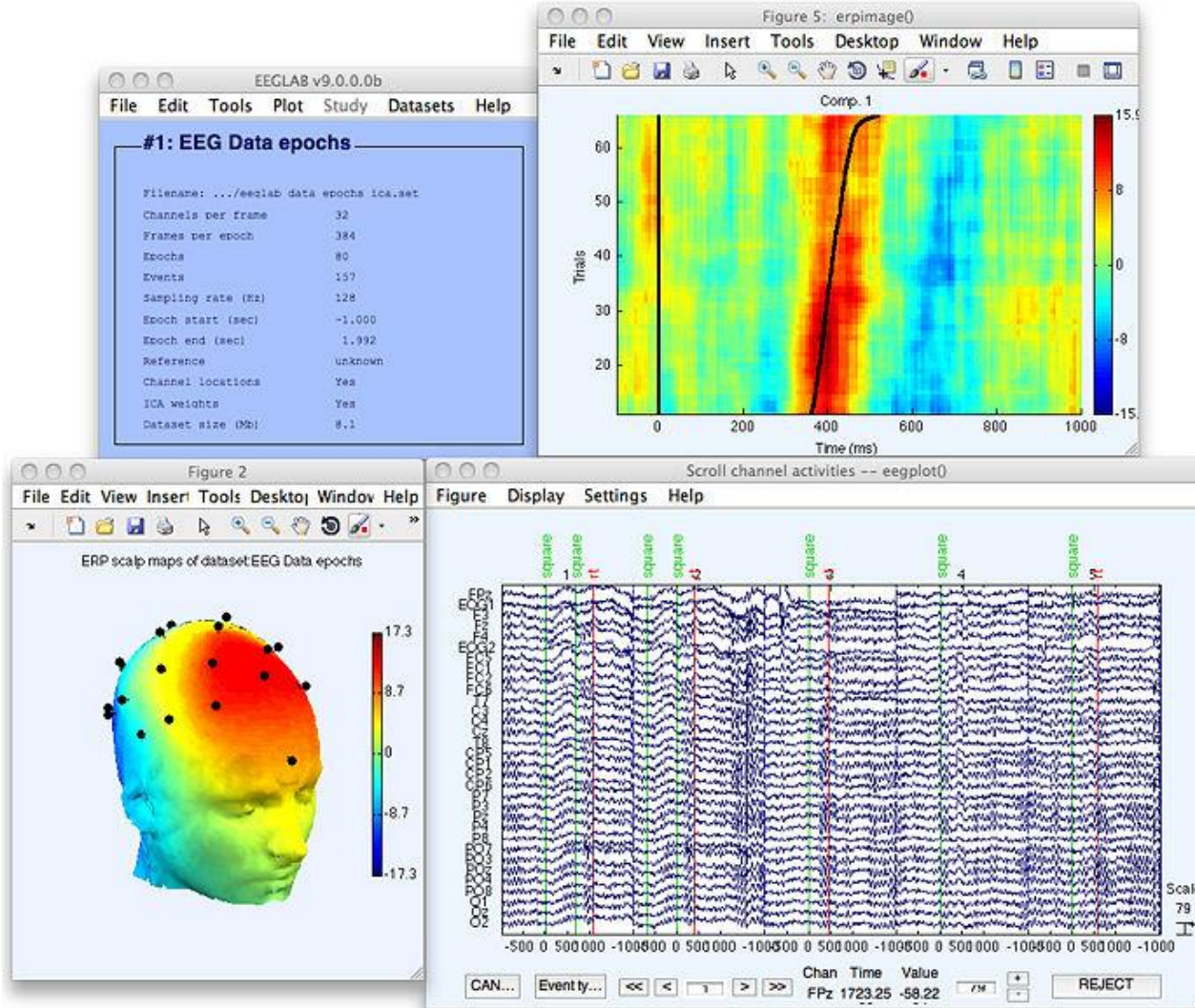
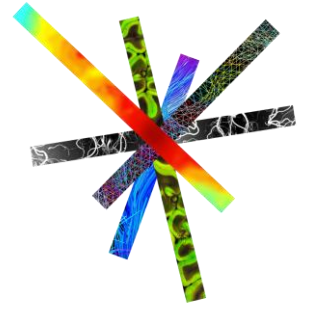


# (Neuro)Science community options



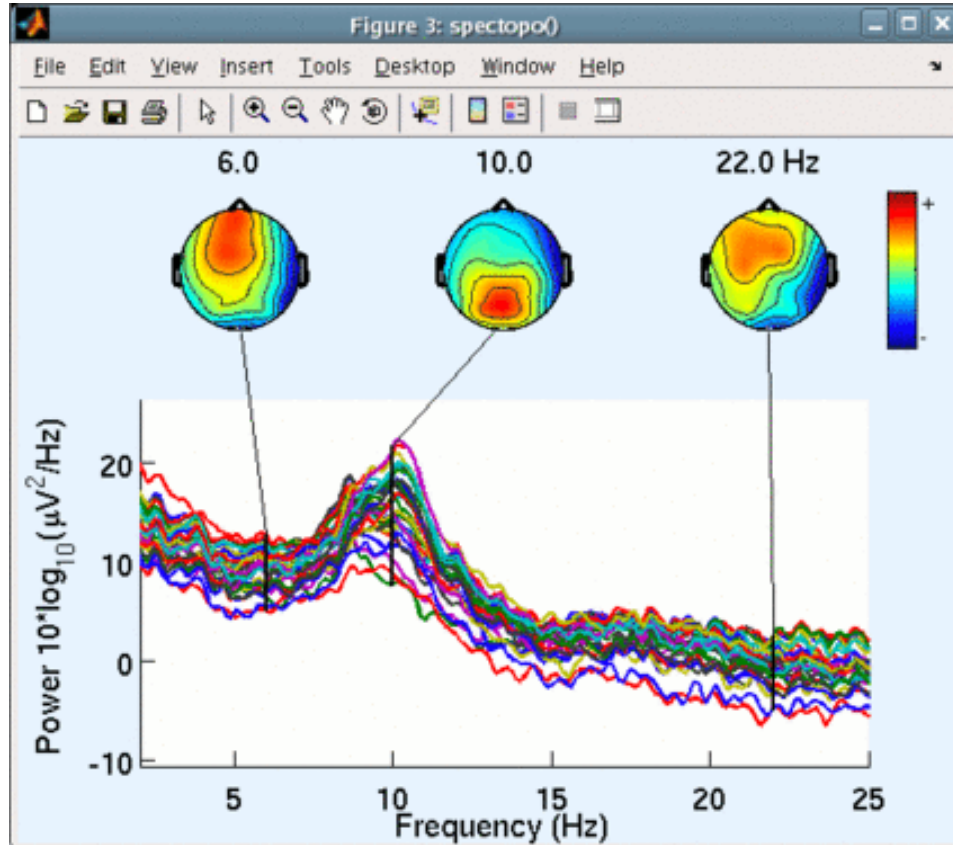
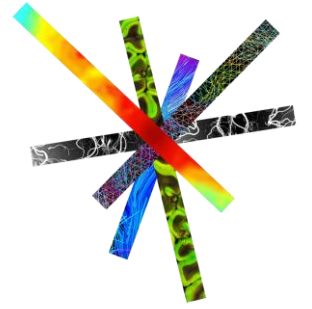
# EEGLAB

<https://sccn.ucsd.edu/eeglab/index.php>



**EEGLAB** is an interactive Matlab toolbox for processing continuous and event-related **EEG**, **MEG** and other electrophysiological data:

- It provides a **GUI** to interactively process high-density EEG
- It allows building and running **batch or custom data analysis scripts**
- It offers a structured environment for **storing, accessing, measuring, manipulating and visualizing** event-related EEG data
- It's an **open-source platform** through which researchers can share new methods as **EEGLAB plug-in functions**



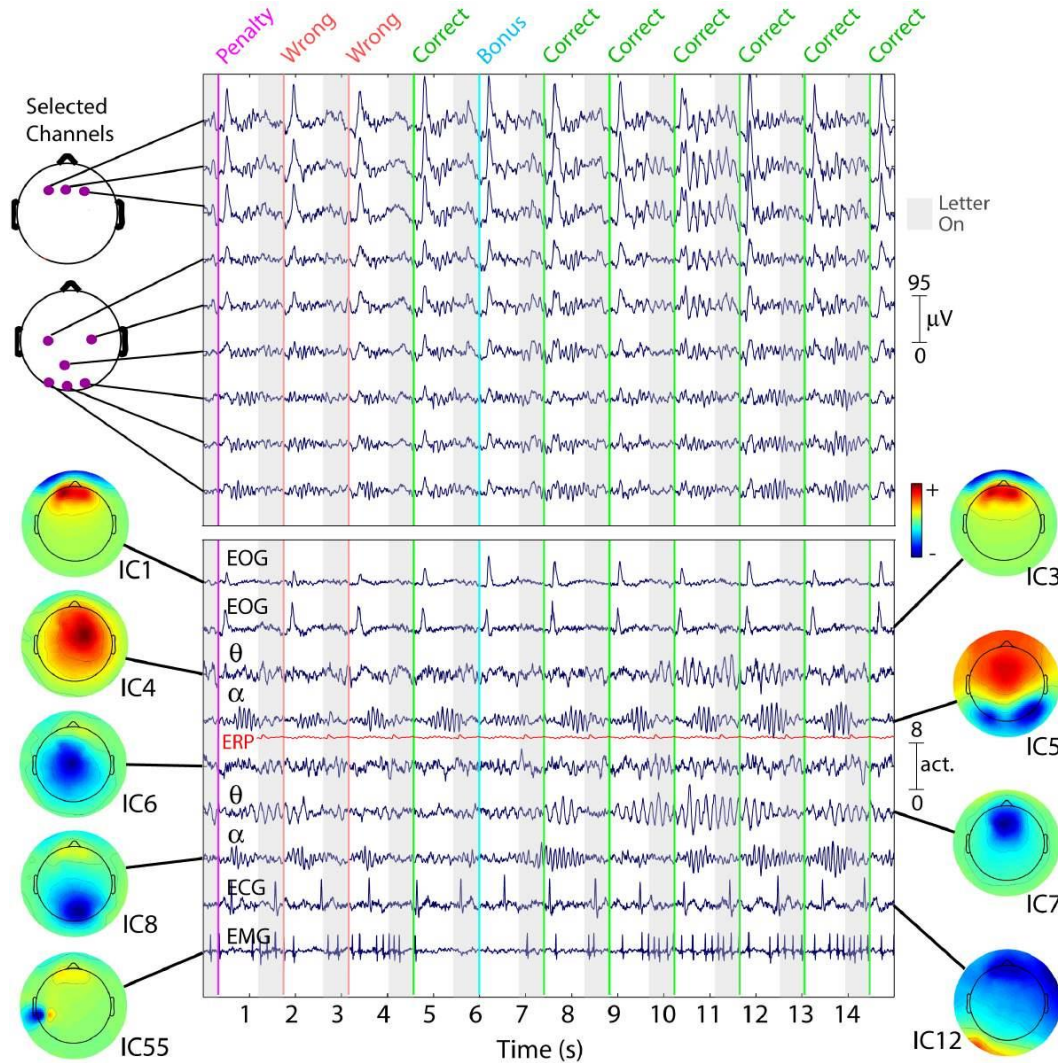
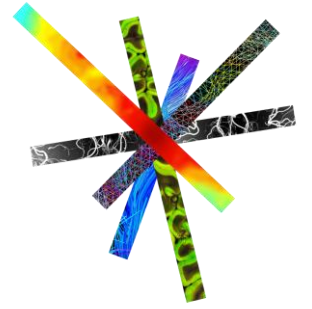
**EEGLAB** is an interactive Matlab toolbox for processing continuous and event-related **EEG**, **MEG** and **other electrophysiological data**:

- It provides a **GUI** to interactively process high-density EEG
- It allows building and running **batch or custom data analysis scripts**
- It offers a structured environment for **storing, accessing, measuring, manipulating and visualizing** event-related EEG data
- It's an **open-source platform** through which researchers can share new methods as **EEGLAB plug-in functions**



# EEGLAB

<https://sccn.ucsd.edu/eeglab/index.php>

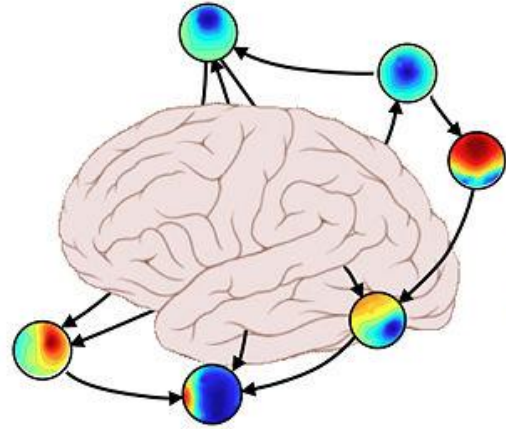
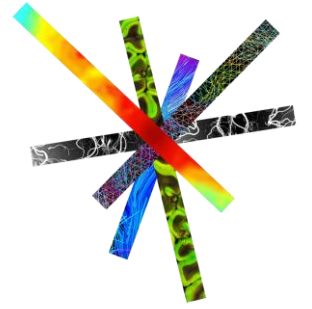


**EEGLAB** is an interactive Matlab toolbox for processing continuous and event-related **EEG**, **MEG** and other electrophysiological data:

- It provides a **GUI** to interactively process high-density EEG
- It allows building and running **batch or custom data analysis scripts**
- It offers a structured environment for **storing, accessing, measuring, manipulating and visualizing** event-related EEG data
- It's an **open-source platform** through which researchers can share new methods as **EEGLAB plug-in functions**

# EEGLAB - SIFT

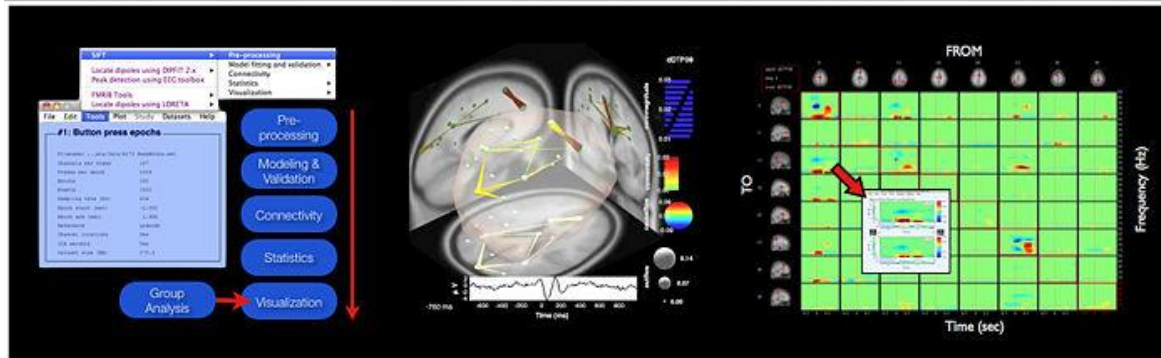
<https://sccn.ucsd.edu/wiki/SIFT>



**SIFT**  
Source Information Flow Toolbox

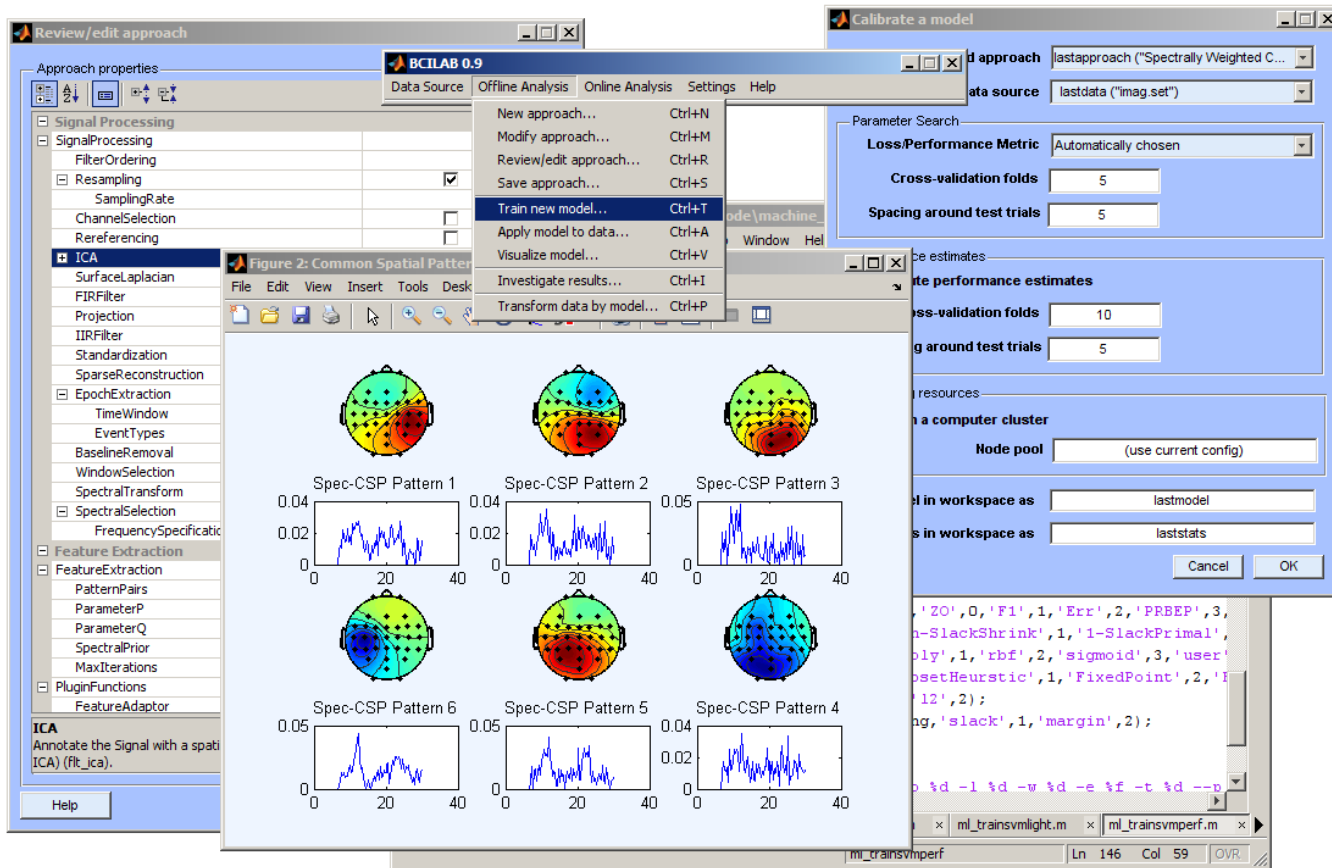
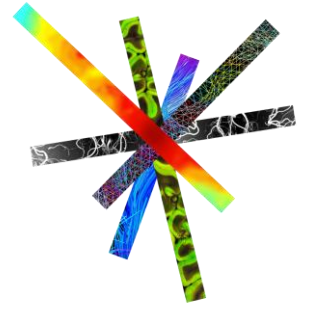
Version 0.1-Alpha

EEGLAB-compatible toolbox for **analysis and visualization of multivariate causality and information flow** between sources of electrophysiological (EEG/EMG/MEG) activity.



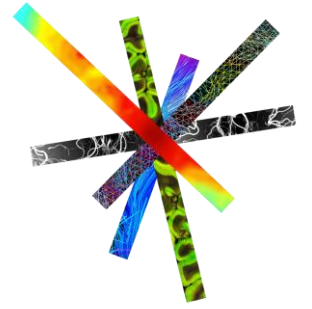
# EEGLAB - BCILAB

<https://sccn.ucsd.edu/wiki/BCILAB>



MATLAB toolbox and EEGLAB plugin for the design, prototyping, testing, experimentation with, and evaluation of Brain-Computer Interfaces (BCIs), and other systems in the same computational framework.

# MNE



MEG + EEG ANALYSIS & VISUALIZATION

Open-source Python software for exploring, visualizing, and analyzing human neurophysiological data: MEG, EEG, sEEG, ECoG, and more.

⚡ Speed

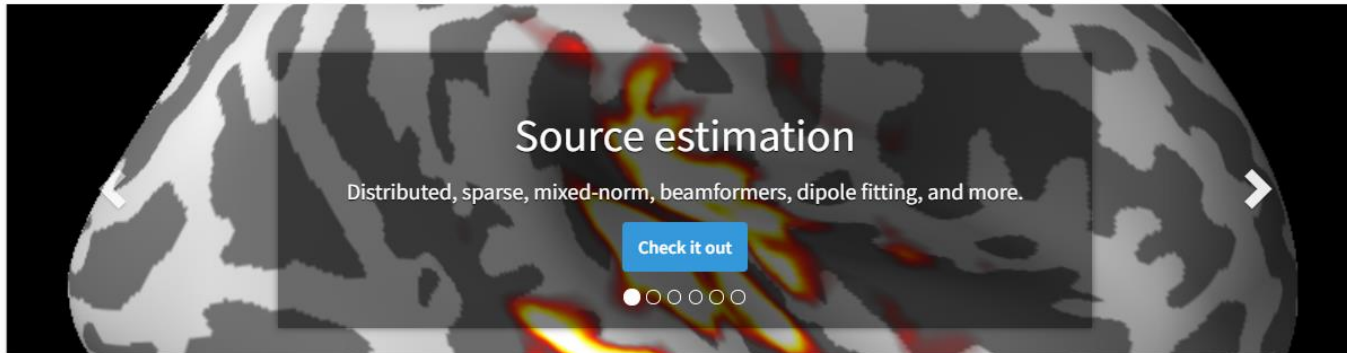
Multi-core CPU & GPU.

👁 Usability

Clean scripting & visualization.

⚙ Flexibility

Broad data format & analysis support.



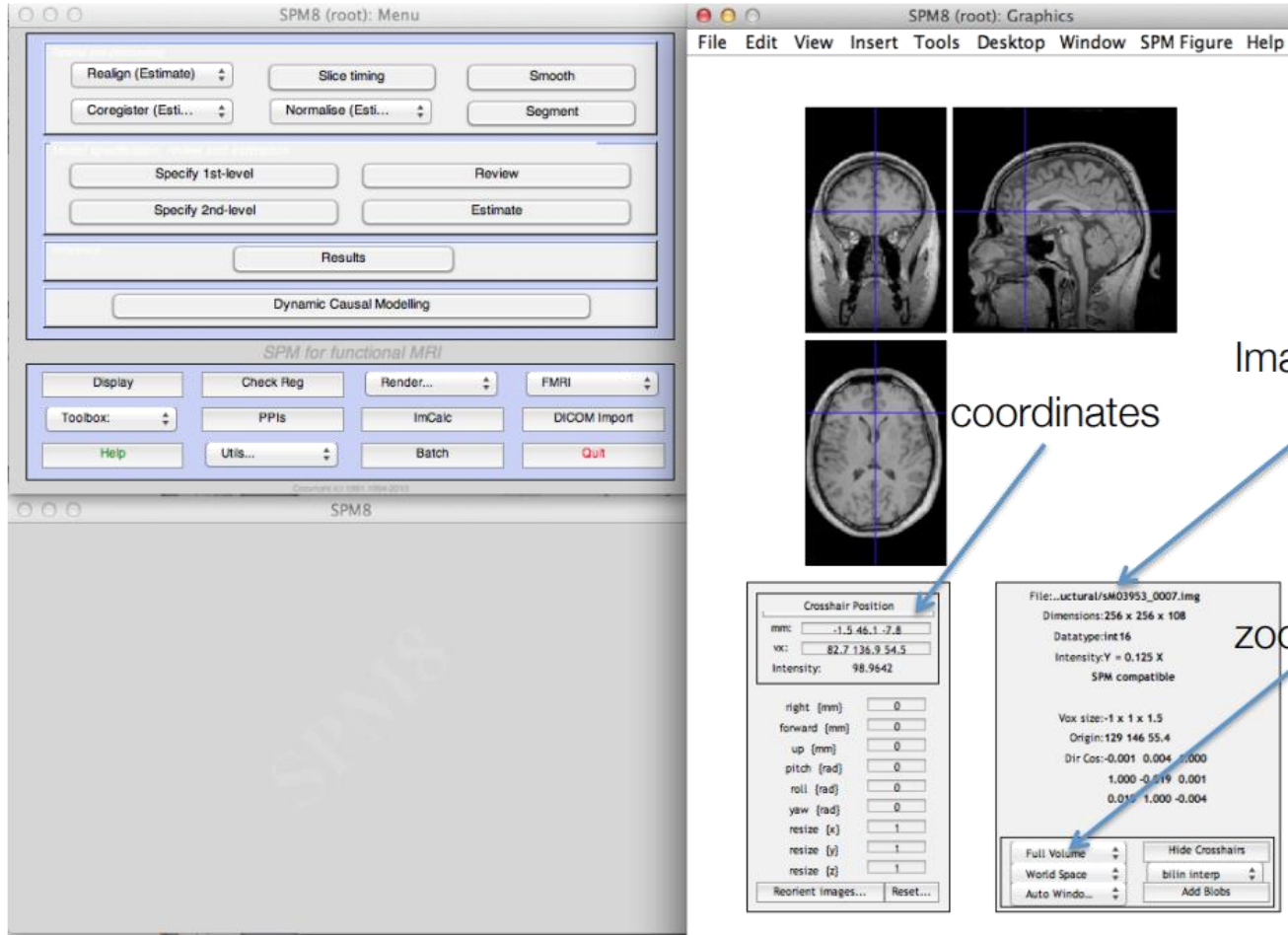
## MEG/EEG source analysis

</usr/pubsw/packages/mne/stable/share/matlab/>

<https://mne.tools/stable/index.html>



# Statistical Parametric Mapping (SPM)



</usr/pubsw/common/spm>

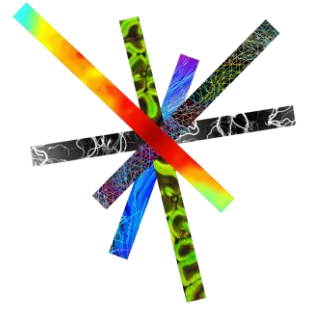
<https://www.fil.ion.ucl.ac.uk/spm/>

The SPM software is a suite of MATLAB functions and subroutines, designed for the **analysis of brain imaging data sequences.**

The sequences can be a **series of images from different cohorts, or time-series from the same subject.**

The current release is designed for the analysis of **fMRI, PET, SPECT, and MEG.**

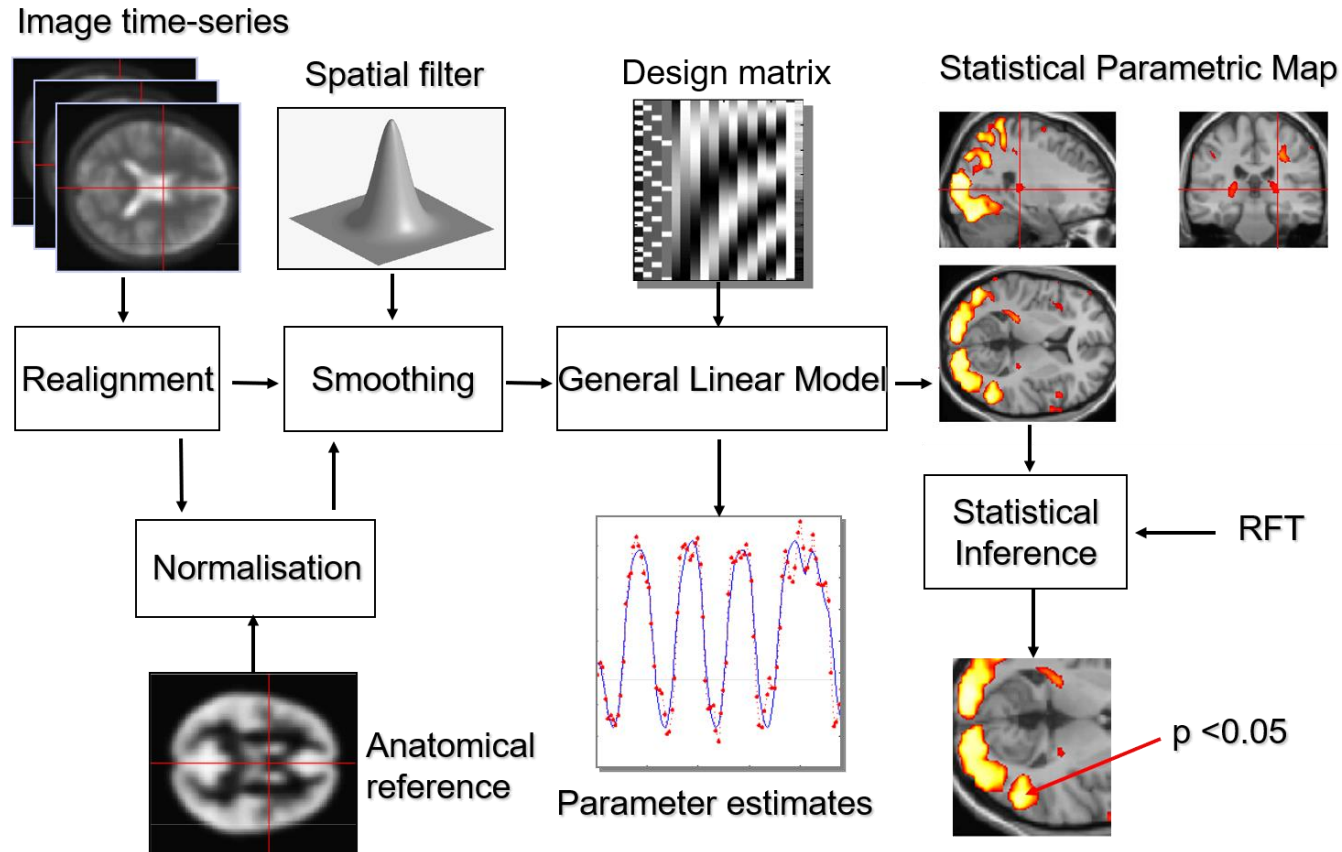
# Statistical Parametric Mapping (SPM)



Preprocessing

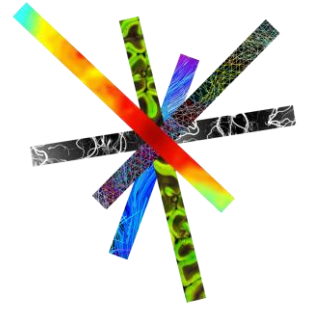
GLM

Statistics





# GIFT (Group ICA of fMRI Toolbox)



TReNDS Home > Software > GIFT

Tuesday, March 3, 2020

Group ICA Of  
fMRI Toolbox  
(GIFT)

Download  
Updates  
Documentation

Version  
Compatibility  
Version

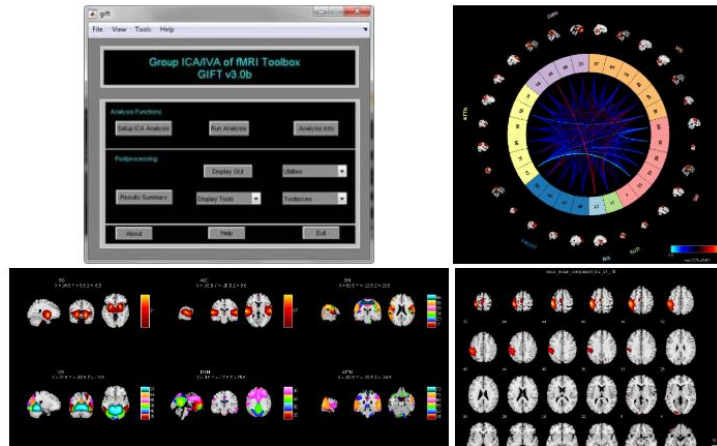
History  
Publications  
Email List  
FAQ

## Group ICA Of fMRI Toolbox(GIFT)

GIFT is an application supported by the NIH under grant 1R01EB000840 to Dr. Vince Calhoun and Dr. Tulay Adali. It is a MATLAB toolbox which implements multiple algorithms for independent component analysis and blind source separation of group (and single subject) functional magnetic resonance imaging data. GIFT works on MATLAB R2008a and higher. Many ICA algorithms were generously contributed by Dr. Andrzej Cichocki. These are also available in Dr. Cichocki's [ICALAB toolbox](#). For any question or comments please contact Vince Calhoun ([vcalhoun@gsu.edu](mailto:vcalhoun@gsu.edu)) or Srinivas Rachakonda ([srachakonda@gsu.edu](mailto:srachakonda@gsu.edu)).

GroupICATv4.0b (GIFTv3.0b) is now released. Till the date of Mar 13, 2019, the Group ICA Toolbox has been downloaded 15,503 times independently by worldwide researchers. Please see [version history](#) page for more information.

GIFT is also registered on github. Github link will be available when you click on the download button.

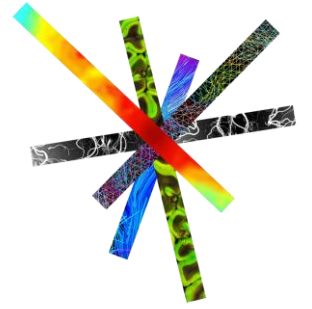


It is a MATLAB toolbox which implements multiple algorithms for **independent component analysis and blind source separation of group** (and single subject) **fMRI** data.

<https://trendscenter.org/software/gift/>

# Masamune

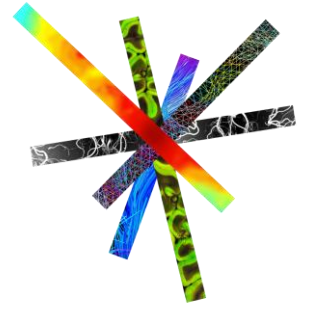
[http://people.fas.harvard.edu/~kastman/nwlab\\_pipeline/pet-recon-mgh.html](http://people.fas.harvard.edu/~kastman/nwlab_pipeline/pet-recon-mgh.html)



Matlab tool for reconstruction of  
'BrainPET' PET-MR data (Bay 6)

# Masamune

[http://people.fas.harvard.edu/~kastman/nwlab\\_pipeline/pet-recon-mgh.html](http://people.fas.harvard.edu/~kastman/nwlab_pipeline/pet-recon-mgh.html)



**Masamune**  
Reconstruction and Analysis Tools for the BrainPET  
developed by members and collaborators of the A.A. Martinos Center MR/PET Core

**Subject Initialization**  
Organize MR Files

**BrainPET QC**  
QC  
Blood GUI

**Atten Map** (highlighted)  
Pseudo-CT  
Atlas CT  
UTE PET  
PET-coreg based  
UTE based  
CT based  
Add Coils  
Anti-Aliasing

**Generate ROIs/Segmentation**  
AIF ROI  
Freesurfer  
Import Aseg  
FS 2 PET  
MR Segment/MNI Atlas Gen  
Brain Extract  
Seg/ALL/BRD

**File Conversion/Xform**  
Generic  
DCM 2 Flat  
CT Reslicer  
Coreg Vols  
DICOM 2 i File in PET Space  
DCM 2 ISO  
ISO 2 PET  
RSL 2 PET  
Compress i File  
Resample iFile  
iFiles2nii(4D)  
Avg PET Vol  
Surf/Vol Smooth

**TAC Analysis**  
Model TACs  
FS TACs  
Gen Atlas  
FS PVC  
IDIF

**Recon**  
AIF  
One Frame  
TAC Recon  
Sino/LOR Recon

**Convert i File-LAS (Internal)**  
iFile2Nifti

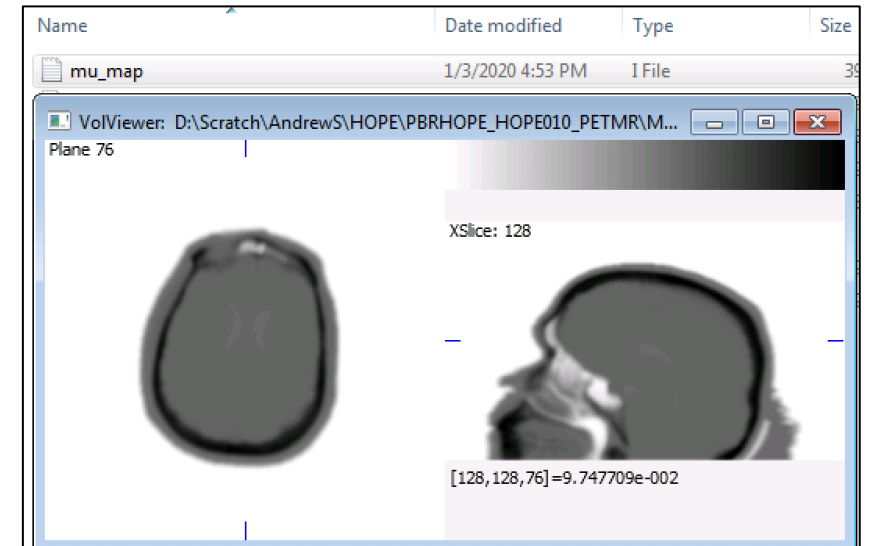
**Export i File-RAS (FS/SPM/etc)**  
iFile2Nifti

**Motion Correction & Derive MC Estimation**  
Time Series MC  
PAC-MMAN  
PET-Based  
MR-Based

(c) 2012 Daniel Chonde <chonde@nmr.mgh.harvard.edu>

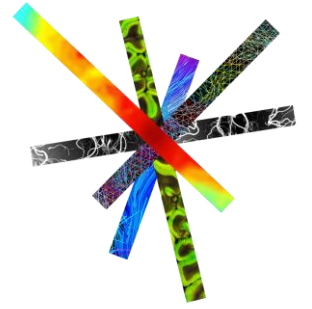
## Matlab tool for reconstruction of 'BrainPET' PET-MR data (Bay 6)

Attenuation correction



# Masamune

[http://people.fas.harvard.edu/~kastman/nwlab\\_pipeline/pet-recon-mgh.html](http://people.fas.harvard.edu/~kastman/nwlab_pipeline/pet-recon-mgh.html)



## Matlab tool for reconstruction of 'BrainPET' PET-MR data (Bay 6)

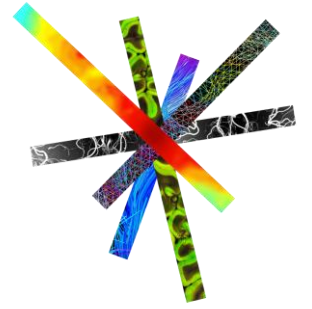
Attenuation correction

Motion correction

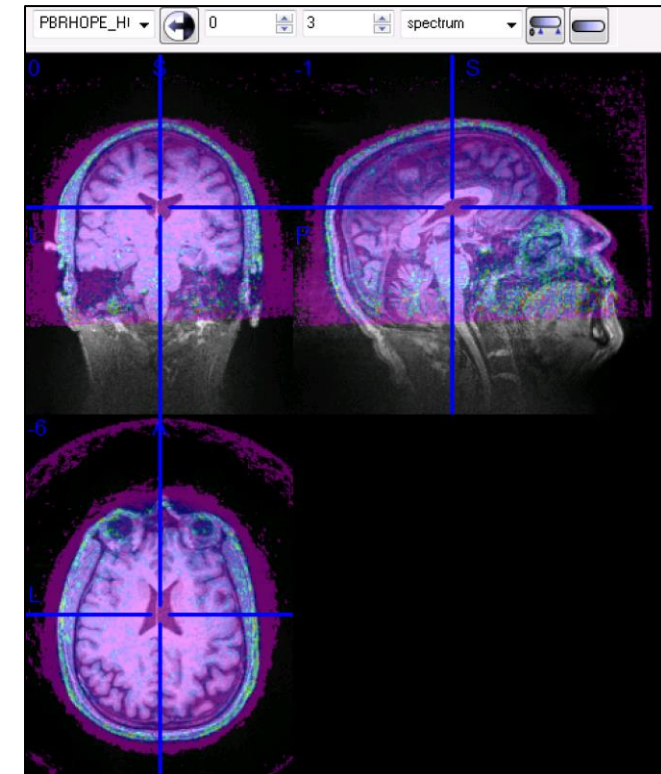


# Masamune

[http://people.fas.harvard.edu/~kastman/nwlab\\_pipeline/pet-recon-mgh.html](http://people.fas.harvard.edu/~kastman/nwlab_pipeline/pet-recon-mgh.html)

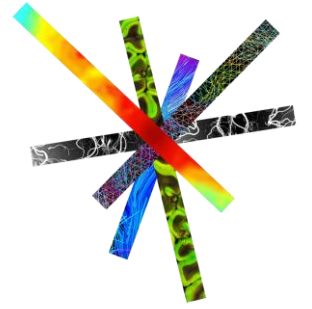


Matlab tool for reconstruction of  
'BrainPET' PET-MR data (Bay 6)



# Masamune

[http://people.fas.harvard.edu/~kastman/nwlab\\_pipeline/pet-recon-mgh.html](http://people.fas.harvard.edu/~kastman/nwlab_pipeline/pet-recon-mgh.html)



## Matlab tool for reconstruction of 'BrainPET' PET-MR data (Bay 6)

Attenuation correction

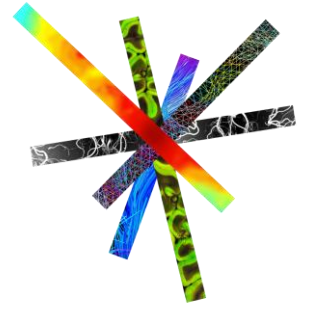
Motion correction

Data reconstruction

File conversion

# Masamune

[http://people.fas.harvard.edu/~kastman/nwlab\\_pipeline/pet-recon-mgh.html](http://people.fas.harvard.edu/~kastman/nwlab_pipeline/pet-recon-mgh.html)



**Masamune**  
Reconstruction and Analysis Tools for the BrainPET  
developed by members and collaborators of the A.A. Martinos Center MR/PET Core

**Subject Initialization**  
Organize MR Files

**Atten Map**  
Pseudo-CT  
Atlas CT  
UTE PET  
PET-coreg based  
UTE based  
CT based  
Add Coils  
Anti-Aliasing

**Recon**  
AIF  
One Frame  
TAC Recon  
Sino/LOR Recon

**Generate ROIs/Segmentation**  
AIF ROI  
Freesurfer  
Import Aseg  
FS 2 PET  
MR Segment/Mini Atlas Gen  
Brain Extract  
Seg/ALL/BRD  
TAC Analysis  
Model TACs  
FS TACs  
Gen Atlas  
FS PVC  
IDIF

**File Conversion/Xform**  
Generic  
DCM 2 Flat  
CT Reslicer  
Coreg Vols  
DICOM 2 i File in PET Space  
DCM 2 ISO  
ISO 2 PET  
RSL 2 PET  
RSL Nifti 2 I  
Compress i File  
Resample iFile  
iFiles2nii(4D)  
Avg PET Vol  
Surf/Vol Smooth  
Convert i File-LAS (Internal)  
iFile2Nifti  
Export i File-RAS (FS/SPM/etc)  
iFile2Nifti  
Motion Correction & Derive MC Estimation  
Time Series MC  
PAC-MMAN  
PET-Based  
MR-Based

**BrainPET QC**  
QC  
Blood GUI

(c) 2012 Daniel Chonde <chonde@nmr.mgh.harvard.edu>

## Matlab tool for reconstruction of 'BrainPET' PET-MR data (Bay 6)

Attenuation correction

Motion correction

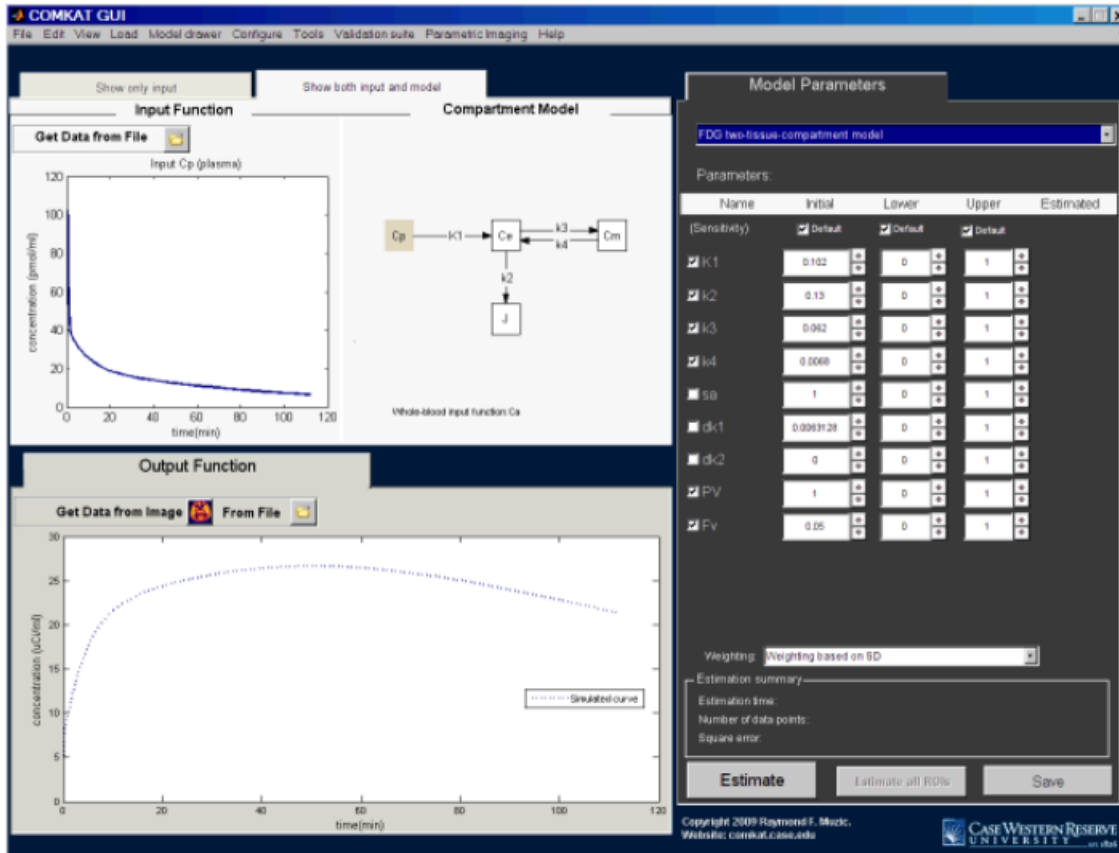
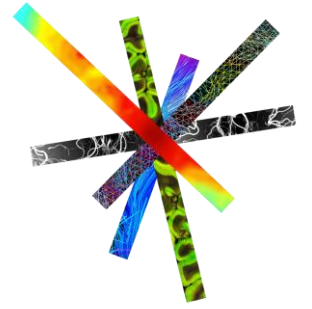
Data reconstruction

File conversion

ROI segmentation

TAC analysis

# Comkat (COmpartmental Model Kinetic Analysis Tool)

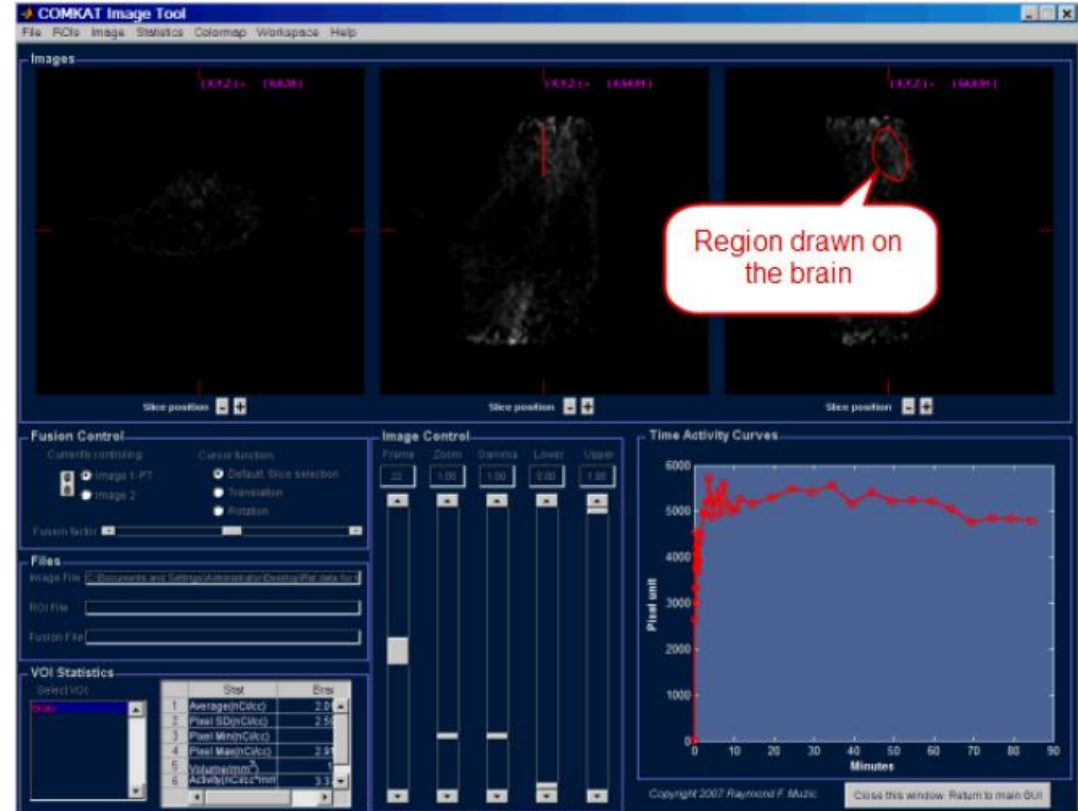
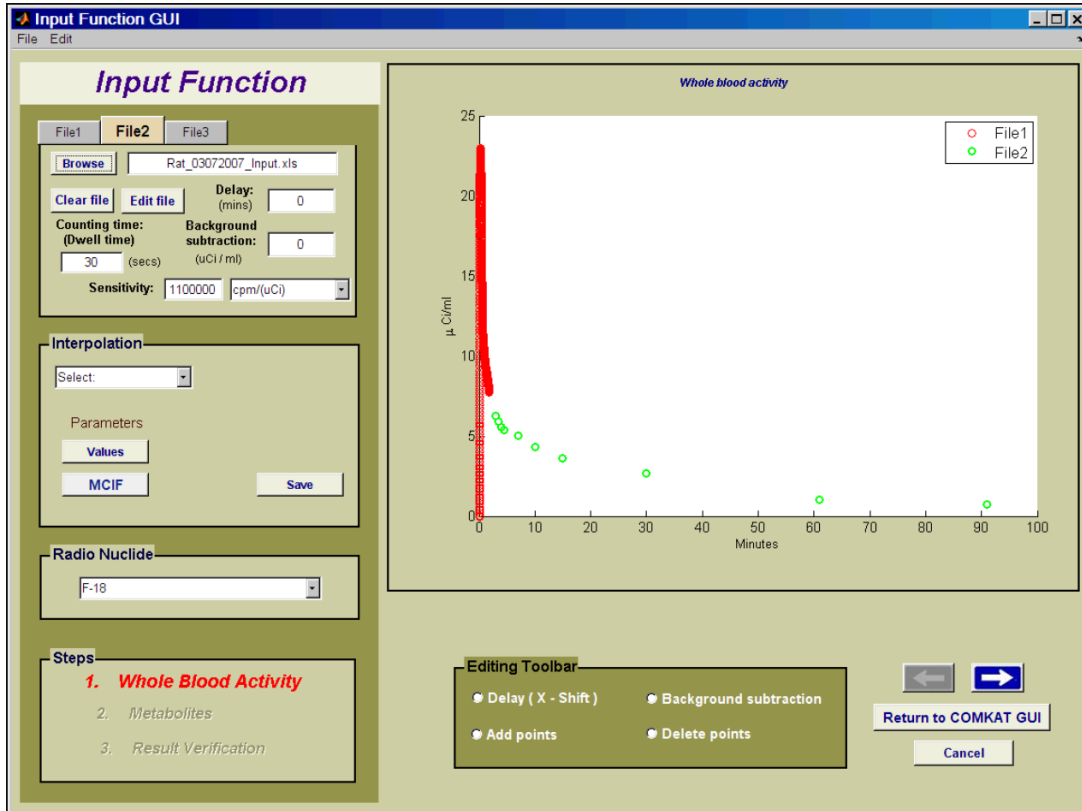
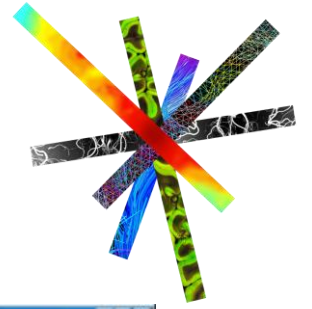


MATLAB software for **compartmental modeling oriented to nuclear medicine applications** (PET & SPECT). It supports models of a wide range complexity including **multiple injection, receptor model with saturation**:

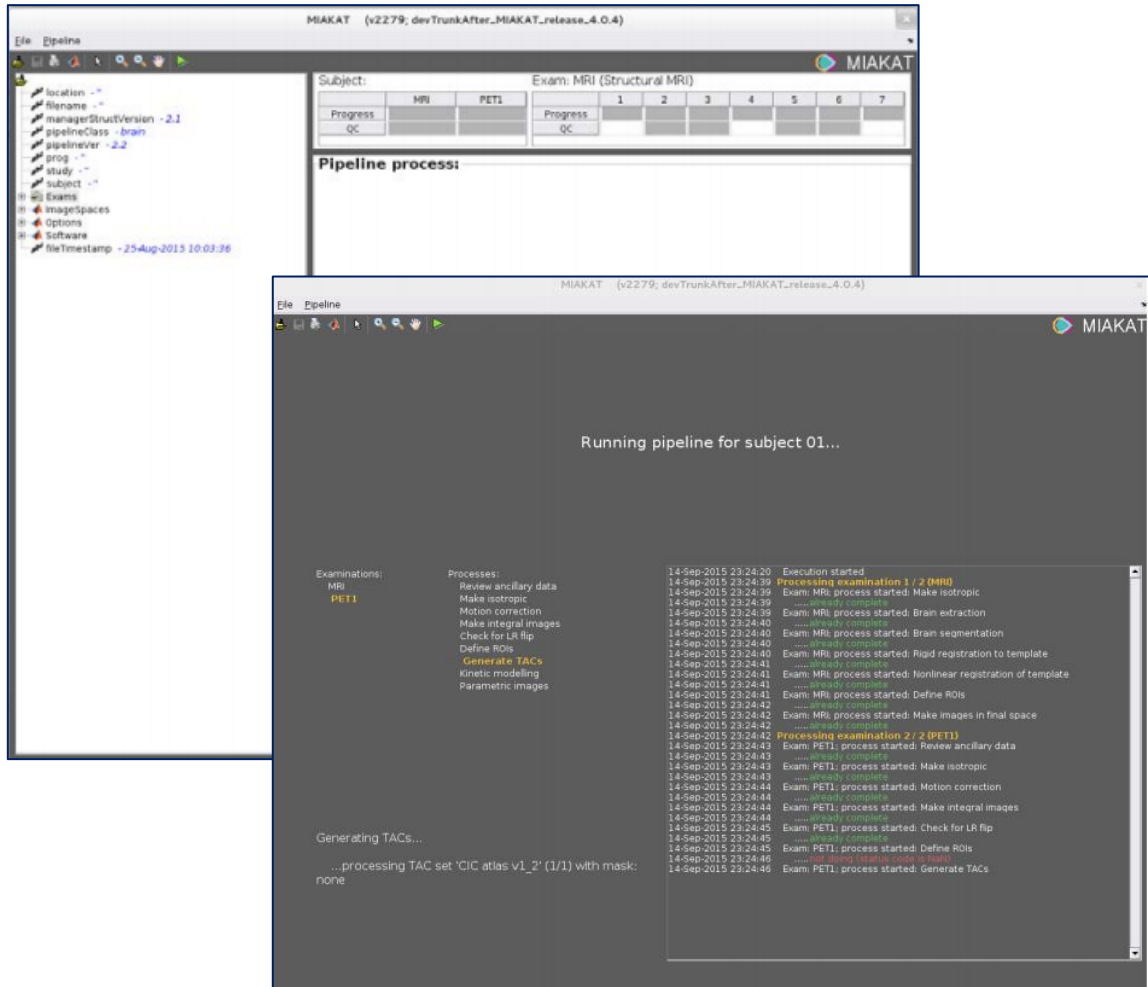
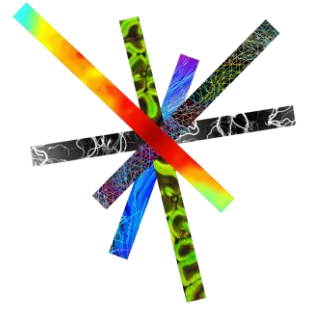
- It supports **many image formats**, including DICOM
- Using either **the command line interface or GUI**, models are easily specified, solved or used to fit experimental data.
- **No mathematical derivations are required** on the part of the user.



# Comkat (COmpartmental Model Kinetic Analysis Tool)



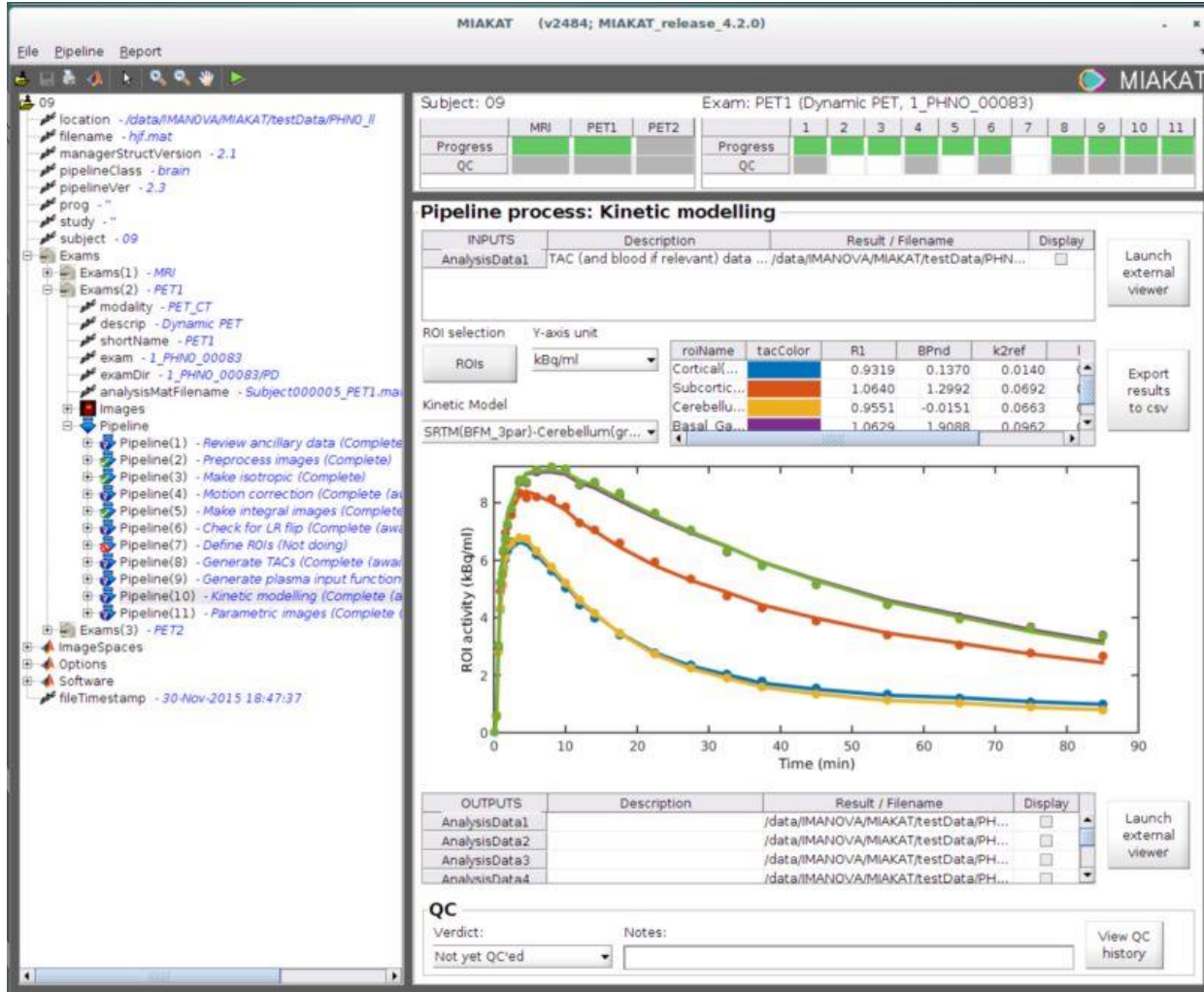
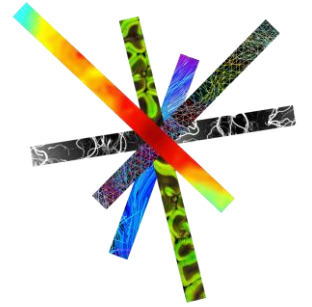
# MIAKAT



MIAKAT is a **fully quantitative suite of analysis tools for PET neuroimaging data** bringing together state of the art tools in a single user-friendly software environment.

It is implemented in MATLAB and it has a central **GUI that facilitates “point and click” operation.**

The user can **configure an analysis pipeline for a given research study**, and then simply replicate it for each dataset.

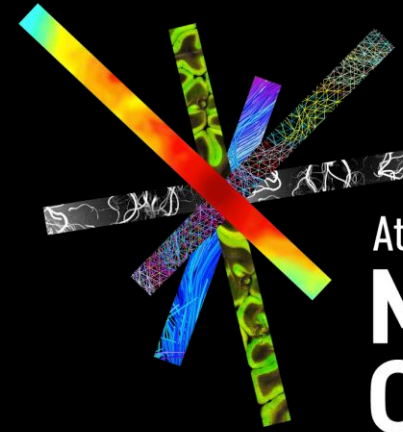


## STANDARD BRAIN PIPELINE

- take the **primary experimental data** (dynamic PET, structural MR images, arterial blood measurements)
- perform a **sequence of processes** which ultimately produce results in **regional (or voxel-wise) parameters**

- Brain Extraction*
- Brain Tissue Segmentation*
- Motion Correction*
- Regional ROI Definition via Atlas*
- Blood/Plasma Function Modelling*
- ROI Tracer Kinetic Modelling*
- Parametric Imaging*

# Overview

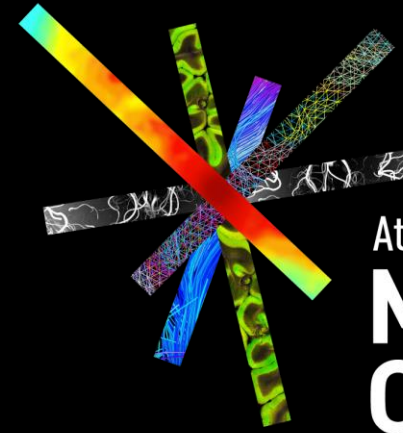


Athinoula A.  
**Martinos  
Center**  
For Biomedical Imaging

- **WHAT?**
- **WHY?**
- **HOW?**
  - GETTING STARTED
  - SCRIPTS, FUNCTIONS, AND THE EDITOR
  - VISUALIZATION TOOLS
- **BUT ... CAN I STILL USE IT, IF I DON'T ~~KNOW HOW~~ TO CODE?**

*want*

# Overview

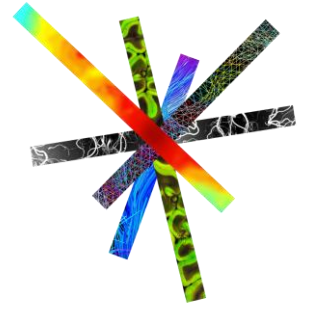


Athinoula A.  
**Martinos  
Center**  
For Biomedical Imaging

- **WHAT?**
- **WHY?**
- **HOW?**
  - GETTING STARTED
  - SCRIPTS, FUNCTIONS, AND THE EDITOR
  - VISUALIZATION TOOLS
- **BUT ... CAN I STILL USE IT, IF I DON'T ~~KNOW HOW~~ TO CODE?**

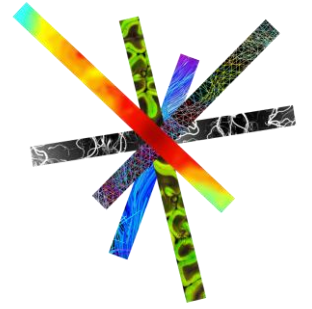
*want*

**Bonus: Miscellaneous "Advanced" Topics**



# Using MATLAB at Martinos Center

# Opening MATLAB



If you are logged into *any* Linux workstation in **Martinos Center**

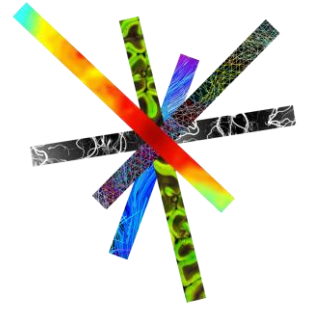
`$ matlab &` ← opens **DEFAULT** MATLAB version (**NOT** necessarily the **LATEST** version)

Other versions can be found as well (executable are in </usr/pubsw/bin/>):

<b>matlab</b>	matlab7.0	matlab7.11	matlab7.2	matlab7.4	matlab7.7
matlab7.9	matlab8.2	matlab8.4	matlab8.6	matlab9.2	matlab9.4
matlab9.6	<b>matlab.new</b>	matlab6.5.1	matlab7.1	matlab7.14	matlab7.3
matlab7.5	matlab7.8	matlab8.0	matlab8.3	matlab8.5	matlab9.0
matlab9.3	matlab9.5	matlab9.7			



# Using MATLAB from 'your' laptop



## Use a Network License

This version only works when you are connected to the network inside the Partners firewall.

<https://www.nmr.mgh.harvard.edu/intranet/computer/software/matlab> (Intranet login required).

## Use remote access to your work desktop:

**No Machine** (software from Partners)

<https://www.nmr.mgh.harvard.edu/intranet/computer/remote-access/nomachine>

**VNC** (GUI access to Martinos workstations)

<http://www.nmr.mgh.harvard.edu/martinos/userInfo/computer/vnc/windows.php>

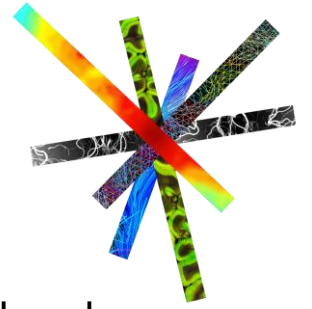
## Standalone License

If you need a copy of Matlab that will work wherever you go **you need to buy a standalone license.**

Contact Alyssa Silverman ([Alyssa.Silverman@mathworks.com](mailto:Alyssa.Silverman@mathworks.com)) for a quote and then submit the quote to whomever handles purchasing for your department.



# MATLAB & launchpad



The center has limited numbers of MATLAB licenses.

All users are limited to no more than 20 MATLAB licenses in use at once over all locations (launchpad, tensor or your group workstations).

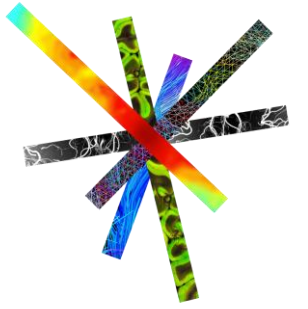
You can run MATLAB jobs in the cluster (launchpad)

<http://www.nmr.mgh.harvard.edu/martinos/userInfo/computer/launchpad.php>

- Submit any jobs that use MATLAB to the queue **matlab**.
- If your job requires any toolbox licenses, you are limited to just **ONE** such job running on the cluster.
- To automate MATLAB jobs on the cluster, first **create a \*.m script file with your actual Matlab commands** to run. The last line of the script should be 'exit'. Give a command like this to **pbsubmit's -c option**:

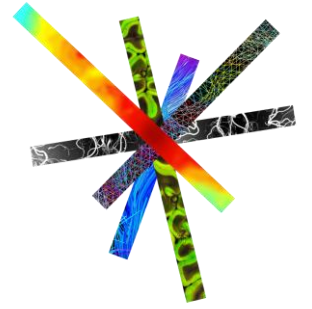
```
matlab.new -nodisplay -nodesktop -nojvm -r matlabfile
```

- Another option is to "**compile**" your Matlab program into a **stand-alone executable**. *This will not use up a license normally.* <https://www.nmr.mgh.harvard.edu/martinos/itgroup/deploytool.html>



# Calling MATLAB from SHELL

# Running MATLAB scripts from SHELL



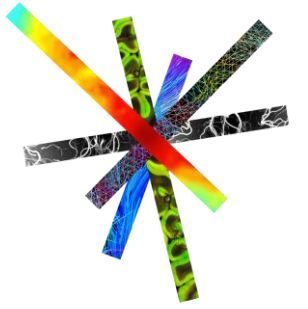
The MATLAB Editor is nice but:

- Let us assume that you have a **complicated SHELL processing stream** using **FSL & FreeSurfer** tools.
- You want to do a little bit of something in the middle with MATLAB that neither FSL or FS can do.
- Then it is more convenient to **run your MATLAB script from UNIX command line** as part of your main script, without starting an interactive MATLAB session.

```
matlab.new -nodesktop -nodisplay -r "run /full/path/to/script/my_script"
```

## NOTE

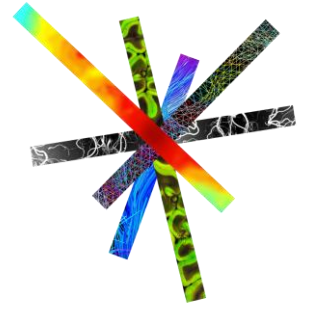
1. NO **.m** extension in the script file name
2. Make sure last line of the file **my\_script.m** is **exit;**



# Calling Python (or anything else) from MATLAB

# Calling Python from MATLAB

## USING THE DEFAULT PYTHON SUB-SYSTEM



- Specific *only for Python*
- Similar functionalities also available for a handful of other languages

matlab\_main.m

### mymod.py

```
"""Python module demonstrates passing
   MATLAB types to Python functions"""

import numpy as np

def square(num):
    num = np.asarray(num)
    return np.power(num, 2)

def root(num):
    num = np.asarray(num)
    return np.sqrt(num)
```

```
clear
clc

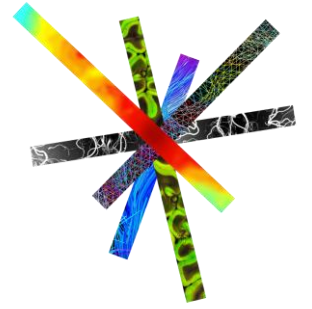
mod = py.importlib.import_module('mymod');
py.reload(mod);

array = 1:10;
array_squared = double(py.mymod.square(array));
array_root = double(py.mymod.root(array));

disp('Array')
disp(array)
disp('Array square')
disp(array_squared)
disp('Array root')
disp(array_root)
```

# Calling Python from MATLAB

## USING THE GENERIC SYSTEM CALL



- We can use the `system` call
- Can also do this with scripts from other programming languages

`matlab_main.m`

### `mymod.py`

```
"""Python module demonstrates passing
MATLAB types to Python functions"""

import numpy as np

def square(num):
    num = np.asarray(num)
    return np.power(num, 2)

def root(num):
    num = np.asarray(num)
    return np.sqrt(num)
```

### `python_main.py`

```
from mymod import square, root ←
import scipy.io as spio

filename = 'matlab_output.mat'
input = spio.loadmat(filename,
                      struct_as_record=False,
                      squeeze_me=True)

input = input['array']
input_squared = square(input)
input_root = root(input)

spio.savemat('python_output.mat',
             {'square': input_squared,
             'root': input_root})

# print(input_squared)
# print(input_root)
```

```
clear
clc

array = 1:10;
save('matlab_output.mat', 'array');

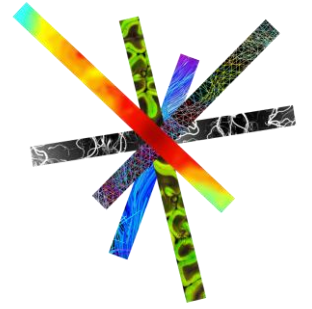
[status, result] = system('python python_main.py');

python_output = load('python_output.mat');
array_squared = python_output.square;
array_root = python_output.root;

disp('Array')
disp(array)
disp('Array square')
disp(array_squared)
disp('Array root')
disp(array_root)
```

# Calling Python from MATLAB

## USING THE GENERIC SYSTEM CALL



- We can use the `system` call
- Can also do this with scripts from other programming languages

`matlab_main.m`

### `mymod.py`

```
"""Python module demonstrates passing
MATLAB types to Python functions"""

import numpy as np

def square(num):
    num = np.asarray(num)
    return np.power(num, 2)

def root(num):
    num = np.asarray(num)
    return np.sqrt(num)
```

### `python_main.py`

```
from mymod import square, root ←
import scipy.io as spio

filename = 'matlab_output.mat'
input = spio.loadmat(filename,
                      struct_as_record=False,
                      squeeze_me=True)

input = input['array']
input_squared = square(input)
input_root = root(input)

spio.savemat('python_output.mat',
             {'square': input_squared,
              'root': input_root})

# print(input_squared)
# print(input_root)
```

```
clear
clc

array = 1:10;
save('matlab_output.mat', 'array');

[status, result] = system('python python_main.py');

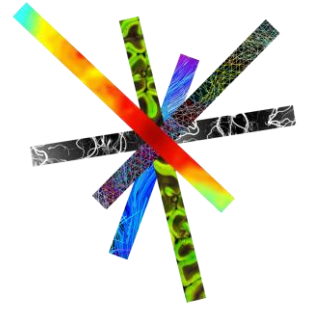
python_output = load('python_output.mat');
array_squared = python_output.square;
array_root = python_output.root;

disp('Array')
disp(array)
disp('Array square')
disp(array_squared)
disp('Array root')
disp(array_root)
```



# Calling Python from MATLAB

## USING THE GENERIC SYSTEM CALL



- We can use the `system` call
- Can also do this with scripts from other programming languages

`matlab_main.m`

### `mymod.py`

```
"""Python module demonstrates passing
MATLAB types to Python functions"""

import numpy as np

def square(num):
    num = np.asarray(num)
    return np.power(num, 2)

def root(num):
    num = np.asarray(num)
    return np.sqrt(num)
```

### `python_main.py`

```
from mymod import square, root ←
import scipy.io as spio

filename = 'matlab_output.mat'
input = spio.loadmat(filename,
                      struct_as_record=False,
                      squeeze_me=True)

input = input['array']
input_squared = square(input)
input_root = root(input)

spio.savemat('python_output.mat',
             {'square': input_squared,
             'root': input_root})

# print(input_squared)
# print(input_root)
```

```
clear
clc

array = 1:10;
save('matlab_output.mat', 'array');

[status, result] = system('python python_main.py');

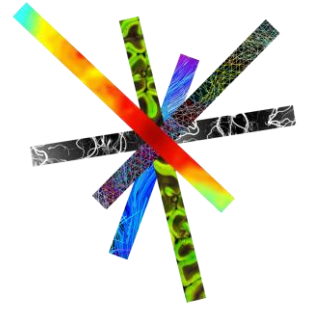
python_output = load('python_output.mat');
array_squared = python_output.square;
array_root = python_output.root;

disp('Array')
disp(array)
disp('Array square')
disp(array_squared)
disp('Array root')
disp(array_root)
```



# Calling Python from MATLAB

## USING THE GENERIC SYSTEM CALL



- We can use the `system` call
- Can also do this with scripts from other programming languages

`matlab_main.m`

### `mymod.py`

```
"""Python module demonstrates passing
MATLAB types to Python functions"""

import numpy as np

def square(num):
    num = np.asarray(num)
    return np.power(num, 2)

def root(num):
    num = np.asarray(num)
    return np.sqrt(num)
```

### `python_main.py`

```
from mymod import square, root
import scipy.io as spio

filename = 'matlab_output.mat'
input = spio.loadmat(filename,
                     struct_as_record=False,
                     squeeze_me=True)

input = input['array']
input_squared = square(input)
input_root = root(input)

spio.savemat('python_output.mat',
            {'square': input_squared,
            'root': input_root})

# print(input_squared)
# print(input_root)
```

```
clear
clc

array = 1:10;
save('matlab_output.mat', 'array');

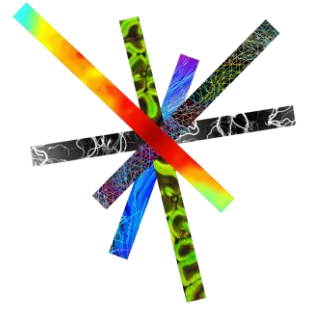
[status, result] = system('python python_main.py');

python_output = load('python_output.mat');
array_squared = python_output.square;
array_root = python_output.root;

disp('Array')
disp(array)
disp('Array square')
disp(array_squared)
disp('Array root')
disp(array_root)
```

# Calling Python from MATLAB

## USING THE GENERIC SYSTEM CALL



- We can use the `system` call
- Can also do this with scripts from other programming languages

matlab\_main.m

mymod.py

```
"""Python module demonstrates passing
MATLAB types to Python functions"""

import numpy as np

def square(num):
    num = np.asarray(num)
    return np.power(num, 2)

def root(num):
    num = np.asarray(num)
    return np.sqrt(num)
```

python\_main.py

```
from mymod import square, root ←
import scipy.io as spio

filename = 'matlab_output.mat'
input = spio.loadmat(filename,
                      struct_as_record=False,
                      squeeze_me=True)

input = input['array']
input_squared = square(input)
input_root = root(input)

spio.savemat('python_output.mat',
             {'square': input_squared,
             'root': input_root})

# print(input_squared)
# print(input_root)
```

Don't do this

if you just need Python, but ...

```
clear
clc

array = 1:10;
save('matlab_output.mat', 'array');

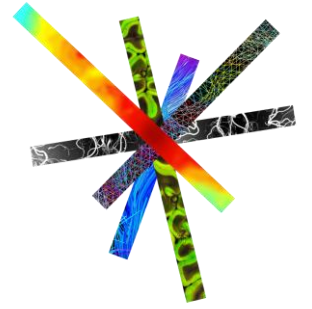
[status, result] = system('python python_main.py');

python_output = load('python_output.mat');
array_squared = python_output.square;
array_root = python_output.root;

disp('Array')
disp(array)
disp('Array square')
disp(array_squared)
disp('Array root')
disp(array_root)
```

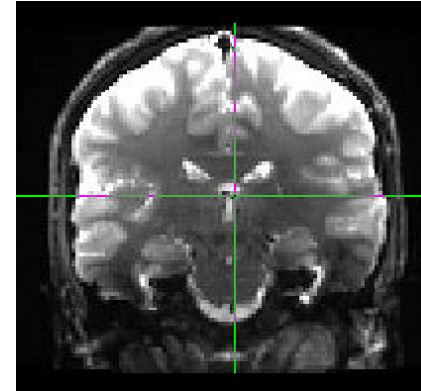
# Executing UNIX commands from MATLAB

## USING THE GENERIC SYSTEM CALL

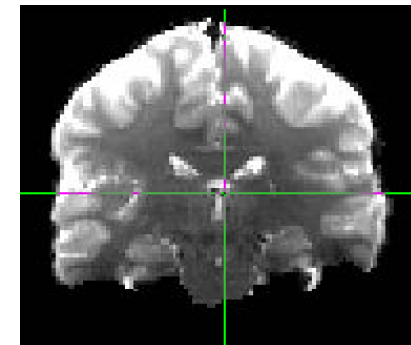


What if I need an **FSL command** in the middle of an elaborate MATLAB processing pipeline?

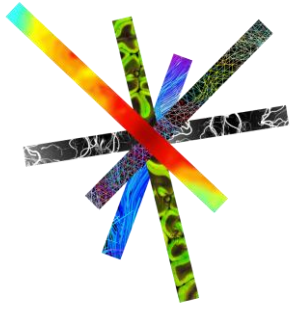
```
% < Do some preprocessing, and save results  
%   somewhere on disk. >  
  
% MAKE A STRING FOR THE FSL BET COMMAND  
command_string_bet = 'bet b0.nii.gz b0_brain.nii.gz -m';  
  
% EXECUTE THE FSL COMMAND USING SYSTEM  
[status,result] = system(command_string_bet);  
  
% < Load back the results and continue ... >
```



b0



b0\_brain

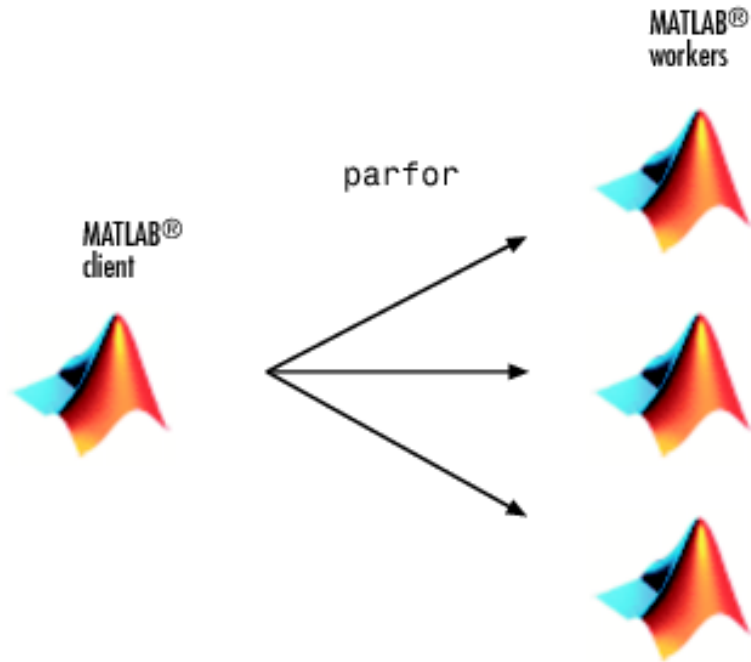


# Speeding-up your code

- **Use functions** instead of scripts.
- **Pre-allocate** the final size of arrays.
- **Vectorize**: Instead of writing loop-based code, consider using MATLAB matrix and vector operations.
- Place independent operations outside loops.
- **Avoid** programmatic use of *cd*, *addpath*, and *rmpath*, when possible: *changing the MATLAB path during run time results in code recompilation.*

# Parallel computing toolbox

## USING PARALLEL FOR-LOOP (PARFOR)



```
n = 2000;  
A = 500;  
  
a = zeros(1,n);  
tStart = tic;  
for i = 1:n  
    a(i) = max(abs(eig(rand(A))));  
end  
tEnd = toc(tStart);  
fprintf('Standard FOR loop:\n%d minutes and %.2f seconds\n',...  
        floor(tEnd/60), rem(tEnd,60));
```

Standard FOR loop:  
5 minutes and 44.89 seconds

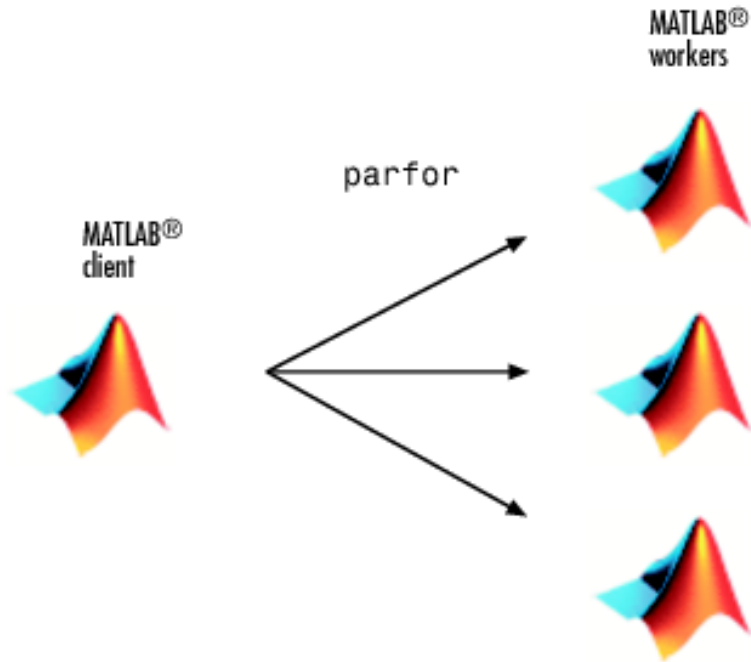
```
a = zeros(1,n);  
tStart = tic;  
parfor i = 1:n  
    a(i) = max(abs(eig(rand(A))));  
end  
tEnd = toc(tStart);  
fprintf('Parallel PARFOR loop:\n%d minutes and %.2f seconds\n',...  
        floor(tEnd/60), rem(tEnd,60));
```

Parallel PARFOR loop:  
1 minutes and 44.28 seconds

N.B. You *cannot call scripts* directly in a parfor-loop. However, you *can call functions*.

# Parallel computing toolbox

## USING PARALLEL FOR-LOOP (PARFOR)



```
n = 2000;  
A = 500;  
  
a = zeros(1,n);  
tStart = tic;  
for i = 1:n  
    a(i) = max(abs(eig(rand(A))));  
end  
tEnd = toc(tStart);  
fprintf('Standard FOR loop:\n%d minutes and % 2f seconds\n',  
        tEnd/60, tEnd-60*tEnd/60);
```

Standard FOR loop:  
5 minutes and 44.89 seconds

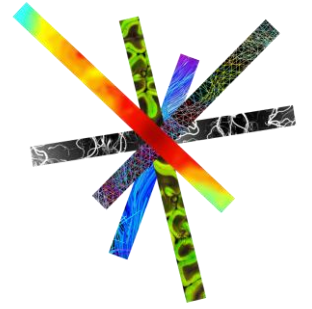
```
a = zeros(1,n);  
tStart = tic;  
parfor i = 1:n  
    a(i) = max(abs(eig(rand(A))));  
end  
tEnd = toc(tStart);
```

Parallel PARFOR loop:  
1 minutes and 44.28 seconds

N.B. You *cannot call scripts* directly in a parfor-loop. However, you *can call functions*.

# Parallel computing toolbox

## USING PARALLEL FOR-LOOP (PARFOR)



### DO use 'parfor' loops

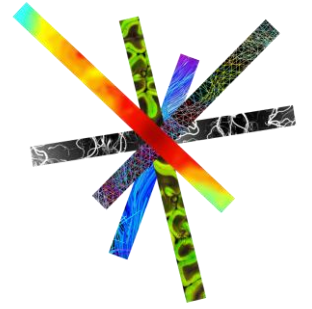
- many loop iterations of a **simple calculation**
- Loop iterations are “**independent**”

### DON'T use 'parfor' loops

- **An iteration** in your loop **depends on the results of other** iterations
- You plan of using the *matlab* queue on ***launchpad***
- There's no Parallel Computation TOOLBOX **license** available ...

# MATLAB Executable (MEX) File Functions

CALL C/C++ OR FORTRAN MEX FILE FUNCTIONS FROM MATLAB



**MEX** stands for **MATLAB EX**ecutable.

A MEX file is **a function**, created in MATLAB, **that calls a C/C++ program or a Fortran subroutine**. A MEX function behaves just like a MATLAB script or function.

## Two main components:

- A gateway routine, **mexFunction**, that interfaces C/C++ and MATLAB data
- Some *non-MATLAB* source code, that performs the desired computations

```
void mexFunction(  
    int nlhs, mxArray *plhs[],  
    int nrhs, const mxArray *prhs[])  
{  
    /* more C code ... */  
}
```

## PROS

Fast calculations  
Easy to learn and use

## CONS

Slow implementation compared to M-files  
Platform dependent (re)compilation



Athinoula A.

# Martinos Center

For Biomedical Imaging

That's all folks!

## Thanks for joining!



[MSCIPIONI@mgh.harvard.edu](mailto:MSCIPIONI@mgh.harvard.edu)



[@mscipioTW](https://twitter.com/mscipioTW)

3/12/2020