Introduction to MATLAB® (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?
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?

3/12/2020
General background: what are we talking about?

What is MATLAB?

MATLAB = MATrix LABoratory
- 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.
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?
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?

3/12/2020
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?
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?

3/12/2020
MATLAB Graphical User Interface (GUI)

1. Command window
2. Command history
3. Workspace
4. File explorer
5. Toolbar
MATLAB syntax – Looking for help!

Don’t be scared to ask for help!

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

```matlab
>> 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 of function
```
MATLAB syntax – Looking for help!

Don’t be scared to ask for help!
In many case, the documentation texts are quite informative and educational.

```matlab
>> 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
MATLAB syntax – Matrices, vectors, arrays …

Scalar

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

Vector

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

Matrix

```matlab
>> 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

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

Matrix

>> A = [1, 2, 3 ; 4, 5, 6 ; 7, 8, 9]; % 3 x 3 matrix
MATLAB syntax – Matrices, vectors, arrays …

Scalar

\[
>> s = 5; \quad \% \text{no need to specify data type (default is double)}
\]

Vector

\[
>> a = [1, 2, 3]; \quad \% \text{row vector}
>> b = [4; 5; 6]; \quad \% \text{column vector}
\]

Matrix

\[
>> A = [1, 2, 3; 4, 5, 6; 7, 8, 9]; \quad \% 3 \times 3 \text{ matrix}
\]

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

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

Vector

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

Matrix

```matlab
>> 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)
• In MATLAB matrix and vector indexing start from 1 (not from 0).

• It uses a column-major convention (it affects reshaping and transpositions)

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

```
A =

68  59  96  82  48  76  73
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
```
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(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(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
```

```
A =

<table>
<thead>
<tr>
<th></th>
<th>68</th>
<th>59</th>
<th>96</th>
<th>82</th>
<th>48</th>
<th>76</th>
<th>78</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>66</td>
<td>23</td>
<td>55</td>
<td>25</td>
<td>36</td>
<td>76</td>
<td>94</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>76</td>
<td>14</td>
<td>93</td>
<td>84</td>
<td>39</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>26</td>
<td>15</td>
<td>35</td>
<td>59</td>
<td>57</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>51</td>
<td>26</td>
<td>20</td>
<td>55</td>
<td>8</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>96</td>
<td>70</td>
<td>85</td>
<td>26</td>
<td>92</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>35</td>
<td>90</td>
<td>26</td>
<td>62</td>
<td>29</td>
<td>54</td>
<td>34</td>
</tr>
</tbody>
</table>
```

```
>> 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(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
```

```
A =

|    68  96  82  48  76  78 |
|    66  55  36  76  94    |
|    17  14  84  39  13    |
|    12  15  59  57  57    |
|    50  26  20  55  8     |
|    96  85  92  6   2     |
|    35  90  26  62  54  34 |
```
• In MATLAB matrix and vector indexing start from 1 (not from 0).

• It uses a column-major convention (it affects reshaping and transpositions)

\[
\begin{bmatrix}
68 & 59 & 96 & 82 & 48 & 76 & 73 \\
66 & 23 & 55 & 25 & 36 & 76 & 94 \\
17 & 76 & 14 & 93 & 84 & 39 & 13 \\
12 & 26 & 15 & 35 & 59 & 57 & 57 \\
50 & 51 & 26 & 20 & 55 & 8 & 47 \\
96 & 70 & 85 & 26 & 92 & 6 & 2 \\
35 & 90 & 26 & 62 & 29 & 54 & 34
\end{bmatrix}
\]

\[
\begin{align*}
>> A(3,2) & \quad \% \text{Access a single element (3rd row, 2nd col)} \\
>> A(:,1) & \quad \% \text{Select the whole 1\textsuperscript{st} column} \\
>> A(2,2:5) & \quad \% \text{Select a subset of 2\textsuperscript{nd} row} \\
>> \text{sum}(A(2,:)) & \quad \% \text{Sum all elements of 2\textsuperscript{nd} row} \\
>> \text{max}(A(:,3)) & \quad \% \text{Max value of 3\textsuperscript{rd} column} \\
>> \text{find}(\text{isprime}(A)) & \quad \% \text{Index of prime numbers among all elements}
\end{align*}
\]

\[
\begin{align*}
\text{disp}((\text{isprime}(A)))
\end{align*}
\]

\[
\begin{align*}
\text{disp}((\text{find}(\text{isprime}(A))))'
\end{align*}
\]
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

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)

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

```
ans =
  4   10   18
```

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

**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!**
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 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

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

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 / b \% equiv to a*pinv(b)
>> A + 3  % Add a scalar
>> A * 3  % Multiply a scalar
>> sin(A)  % Elementwise sine
>> exp(A)  % Elementwise exponential
>> inv(A)  % Inverse of a matrix

Elementwise ops [ MUST BE same shape! ]
>> A + B  % Matrices addition
>> A .* B  % Matrices multiplication
>> A ./ B  % Matrices division

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

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

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

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

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

### MUST BE of compatible shape!

- `A + B` % Matrices addition
- `A .* B` % Matrices multiplication
- `A .* a` % Matrix-vector product
- `A / B` % A*inv(B)
- `A \ B` % inv(A)*B

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

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

```matlab
data = struct with fields:
    x: [1x100 double]
    y: [1x100 double]
    title: 'y = sin(x)'
```
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`
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`
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`
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 = 1×2 cell
   {'2017-08-16'}    {1x3 double}

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

C = 3×2 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 array.

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

ans = 1×2 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 = 1×3
    56   67   78
Use an if-else condition to check the value of some variable within the code:

```matlab
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
```
Use an if-else \textbf{condition to check the value} of some variable within the code:

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

```matlab
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

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

% row index
% col index
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.

```matlab
% find the root of the polynomial x^3 - 2x - 5
a = 0; fa = -Inf;
b = 3; fb = Inf;

while b-a > eps*b
    x = (a+b) / 2;   fx = x^3-2*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?

3/12/2020
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?
MATLAB’s Graphical User Interface (GUI)
MATLAB’s EDITOR – M-file

SCRIPT M-FIILE

- 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

✓ 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

- 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, allows reproducibility

- Convenient for debugging

- Run directly (even portion of it, up to a single line): no need to be compiled

Be careful!
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 ( ).

Save this as m-file: compute_square.m
Once we save the function m-file, it may be called from a script or another function:

```plaintext
>> 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.
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
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.
MATLAB’s EDITOR – Standard editor

TIPS AND TRICKS

Interactive debugging

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

```matlab
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)

[Image of MATLAB interface showing command window and workspace]
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.

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.

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?**
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?
MATLAB Graphics

PLOTTING CURVES

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

```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')
```

![Relative Daily Index Closings](image)
MATLAB Graphics

PLOTTING CURVES

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

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

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

```matlab
% Check version
if verLessThan('matlab','9.0')
    error(['yaxis 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')
```
This is an example of how to create a 3D plot in MATLAB®.

```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')
ylabel('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 10549 8122 28541 7880 3283 4135 7953 1884];
mumps = [28178 23536 34561 37395 36072 32237 18597 9408 6005 6268 8963 13882];
chickenPox = [37140 32169 37533 30103 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')
This is an example of how to create a bivariate histogram in MATLAB®.

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

```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';
```

### Mean of Duration

<table>
<thead>
<tr>
<th>DayOfWeek</th>
<th>&lt;30</th>
<th>30s</th>
<th>40s</th>
<th>50s</th>
<th>60s</th>
<th>70+</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sunday</td>
<td>9.109</td>
<td>9.292</td>
<td>15.28</td>
<td>33.02</td>
<td>20.4</td>
<td>40.3</td>
</tr>
<tr>
<td>Monday</td>
<td>8.235</td>
<td>9.37</td>
<td>29.28</td>
<td>13.01</td>
<td>10.82</td>
<td>8.725</td>
</tr>
<tr>
<td>Tuesday</td>
<td>8.967</td>
<td>8.877</td>
<td>12.99</td>
<td>11.21</td>
<td>7.808</td>
<td>10.5</td>
</tr>
<tr>
<td>Wednesday</td>
<td>8.022</td>
<td>9.882</td>
<td>12.63</td>
<td>13.6</td>
<td>20.01</td>
<td>8.283</td>
</tr>
<tr>
<td>Friday</td>
<td>8.106</td>
<td>10.24</td>
<td>16.26</td>
<td>7.366</td>
<td>8.771</td>
<td>6.95</td>
</tr>
<tr>
<td>Saturday</td>
<td>9.825</td>
<td>10.06</td>
<td>13.46</td>
<td>14.12</td>
<td>24.91</td>
<td>10.8</td>
</tr>
</tbody>
</table>

- **AgeGroup**: <30, 30s, 40s, 50s, 60s, 70+
% 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 Pox'}
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')
ylabel('Cases (in thousands)')
title('Childhood Diseases')
axis([0 13 0 1000])
set(gca, 'XTick', 1:12)

% Create the stem chart in position 3 of a 2x2 grid
subplot(2, 2, 3)
stem(years, cases)
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)
ylabel('Infection Rate')
title('Tuberculosis Cases')
axis([1988 2009 5 20])
This is an example of how to create a 2D quiver plot in MATLAB®:

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

```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
xlabel('x')
ylabel('y')
zlabel('z')
```

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

```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(-30, 18)

% Add title and axis labels
title('Normal Response')
xlabel('X')
ylabel('Y')
zlabel('Z')

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

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

DISPLAY IMAGES

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

```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?

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

% Adjust colormap and axes settings
colormap(jet)
box on
axis tight
camproj perspective
camzoom(30)
cam Targets([165 -20 65])
cam Target([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?

3/12/2020
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?**

3/12/2020
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?
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 Recognition
- 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
- FF. Budget Analyzer
- Sensor Array Analyzer

## 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
- Nap Viewer
- OCR Trainer
- Registration Estimator
- Stereo Camera Calibrator
- Video Labeler
- Video Viewer
- Volume Viewer
Click ‘Import Data’ from the toolbar and select file, or ...

... Drag & Drop data file in Command Window
Matlab APPS

INTERACTIVE DATA IMPORT

| time | t1  | t2  | t3  | t4  | t5  | t6  | t7  | t8  | t9  | t10 | t11 | t12 | t13 | t14 | t15 | t16 | t17 | t18 | t19 |
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 10000| 9.32| 48.143| 59.709| 48.899| 46.155| 43.046| 51.644| 51.023| 48.907| 61.724| 53.386| 52.846| 47.211| 48.452| 52.425| 62.073| 48.809| 49.634| 54.599|
Matlab APPS

INTERACTIVE DATA IMPORT
Matlab APPS

CURVE FITTING

- Polynomial
- Gaussian
- Interpolant
- Polynomial
- Power
- Rational
- Smoothing Spline
- Sum of Sine
- Weibull

**Fit name:** untitled fit 1
**X data:** cdate
**Y data:** pop
**Z data:** (none)
**Weights:** (none)

**Custom Equation**

\[ z = f(x, y) \]

\[ = a + b \times \sin(m \times \pi \times x \times y) + c \times \exp(-w \times y) \times z \]
Matlab APPS
CURVE FITTING

Results

Linear model Poly2:
\[ f(x) = p_2 x^2 + p_1 x + p_0 \]
where \( x \) is normalized by mean 1896 and std 62.05

Coefficients (with 95% confidence bounds):
\[ p_2 = 25.18 \ (23.38, 26.97) \]
\[ p_1 = 75.43 \ (74.94, 76.82) \]
\[ p_0 = 67.74 \ (59.7, 63.79) \]

Goodness of fit:
- SSE: 159
- R-square: 0.9987
- Adjusted R-square: 0.9986
- RMSE: 2.912

Table of Fits

<table>
<thead>
<tr>
<th>Fit name</th>
<th>Data</th>
<th>Fit type</th>
<th>SSE</th>
<th>R-square</th>
<th>DFE</th>
<th>Adj R-sq</th>
<th>RMSE</th>
<th># Coeff</th>
<th>Validation Data</th>
<th>Validation SSE</th>
<th>Validation RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>polynomial</td>
<td>pop vs. date</td>
<td>poly2</td>
<td>159.929</td>
<td>0.9987</td>
<td>18</td>
<td>0.9956</td>
<td>2.9724</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Matlab APPS
CURVE FITTING

Results

Linear model Poly2:
\[ f(x) = p_1 x^2 + p_2 x + p_3 \]
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
Matlab APPS

IMAGE SEGMENTER
Matlab APPS
DEEP LEARNING NETWORK DESIGNER

Reuse Pretrained Network

Load pretrained network

Replace final layers

Train network

Predict and assess network accuracy

Deploy results

Early layers that learned low-level features (edges, blobs, color)

Last layers that learned task specific features

New layers to learn features specific to your data set

Training images

Training options

Test images

Trained network

1 million images
100k classes

Fewer classes
Learn faster

10k of images
10k of classes

Improve network
Mathworks File Exchange Platform

SHARING CODE, CUSTOM TOOLBOXES AND APPS WITH OTHER USERS

File Exchange

Image Processing and Computer Vision (2,433)

export_fig

Simulink Support Package for Arduino Hardware

Run models on Arduino boards.

MATLAB Support Package for USB Webcams

Acquire images and video from UVC compliant webcams.

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

This file was selected as MATLAB Central Pick of the Week

- 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 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

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

https://sccn.ucsd.edu/wiki/SIFT
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 source analysis
/usr/pubsw/packages/mne/stable/share/matlab/

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

- Image time-series
- Spatial filter
- Design matrix
- Realignement
- Smoothing
- Normalisation
- Anatomical reference
- Parameter estimates

GLM

General Linear Model

Statistics

- Statistical Parametric Map
- RFT
- Statistical Inference
- p < 0.05
GIFT (Group ICA of fMRI Toolbox)

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/
Matlab tool for reconstruction of ‘BrainPET’ PET-MR data (Bay 6)

http://people.fas.harvard.edu/~kastman/nw labs_pipeline/pet-recon-mgh.html
Matlab tool for reconstruction of ‘BrainPET’ PET-MR data (Bay 6)

Attenuation correction

http://people.fas.harvard.edu/~kastman/nwlabs_pipeline/pet-recon-mgh.html
Matlab tool for reconstruction of ‘BrainPET’ PET-MR data (Bay 6)

Attenuation correction
Motion correction
Matlab tool for reconstruction of ‘BrainPET’ PET-MR data (Bay 6)

http://people.fas.harvard.edu/~kastman/nwlabspipeline/pet-recon-mgh.html
Matlab tool for reconstruction of ‘BrainPET’ PET-MR data (Bay 6)

Attenuation correction
Motion correction
Data reconstruction
File conversion

http://people.fas.harvard.edu/~kastman/nwlabs_pipeline/pet-recon-mgh.html
Matlab tool for reconstruction of ‘BrainPET’ PET-MR data (Bay 6)

Attenuation correction
Motion correction
Data reconstruction
File conversion
ROI segmentation
TAC analysis
Comkat  (CCompartmental 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.

http://comkat.case.edu/
Comkat (COmpartmental Model Kinetic Analysis Tool)
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.

http://www.miakat.org/
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

• 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?

3/12/2020
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?

Bonus: Miscellaneous “Advanced” Topics
Using MATLAB @ 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/):

<table>
<thead>
<tr>
<th>Version</th>
<th>Executable</th>
</tr>
</thead>
<tbody>
<tr>
<td>matlab</td>
<td>matlab7.0</td>
</tr>
<tr>
<td>matlab7.9</td>
<td>matlab8.2</td>
</tr>
<tr>
<td>matlab9.6</td>
<td>matlab.new</td>
</tr>
<tr>
<td>matlab7.5</td>
<td>matlab7.8</td>
</tr>
<tr>
<td>matlab9.3</td>
<td>matlab9.5</td>
</tr>
<tr>
<td></td>
<td>matlab7.11</td>
</tr>
<tr>
<td></td>
<td>matlab8.4</td>
</tr>
<tr>
<td></td>
<td>matlab6.5.1</td>
</tr>
<tr>
<td></td>
<td>matlab8.0</td>
</tr>
<tr>
<td></td>
<td>matlab9.5</td>
</tr>
<tr>
<td></td>
<td>matlab7.2</td>
</tr>
<tr>
<td></td>
<td>matlab8.6</td>
</tr>
<tr>
<td></td>
<td>matlab7.1</td>
</tr>
<tr>
<td></td>
<td>matlab8.3</td>
</tr>
<tr>
<td></td>
<td>matlab8.7</td>
</tr>
<tr>
<td></td>
<td>matlab7.4</td>
</tr>
<tr>
<td></td>
<td>matlab9.2</td>
</tr>
<tr>
<td></td>
<td>matlab7.14</td>
</tr>
<tr>
<td></td>
<td>matlab9.0</td>
</tr>
</tbody>
</table>
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) 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:

```bash
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
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)
```

```python
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)
```

matlab_main.m
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

**mymod.py**

```python
# 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**

```python
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)
```

**matlab_main.m**

```matlab
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

**mymod.py**

```python
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})
```

**python_main.py**

```python
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})
```

**matlab_main.m**

```matlab
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

```python
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)
```

```matlab
array = 1:10;
save('matlab_output.mat','array');
[status,result] = system('python python_main.py');
```

```matlab
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

```python
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)
```

```matlab
mymod.py

python_main.py

matlab_main.m
```
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

```python
import numpy as np

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

def root(num):
    return np.sqrt(num)

my_mod = np.array([1, 2, 3])
square_result = square(my_mod)
root_result = root(square_result)
```

```matlab
clear
clc

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

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

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?

```matlab
% < 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 ... >
```
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.
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.B. You cannot call scripts directly in a parfor-loop. However, you can call functions.
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 EXecutable.

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

```c
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
Thanks for joining!

MSCIPIONI@mgh.harvard.edu
@mscipioTW

3/12/2020