# Introduction to Matlab

•••

importing data, down-sampling, filtering, plotting signals

1. The idea of matlab
2. Good coding practices
3. A guide to writing unmaintainable code
4. Bugs and profiling
5. Filtering and its effects
6. Making beautiful plots
7. Graphical user interfaces

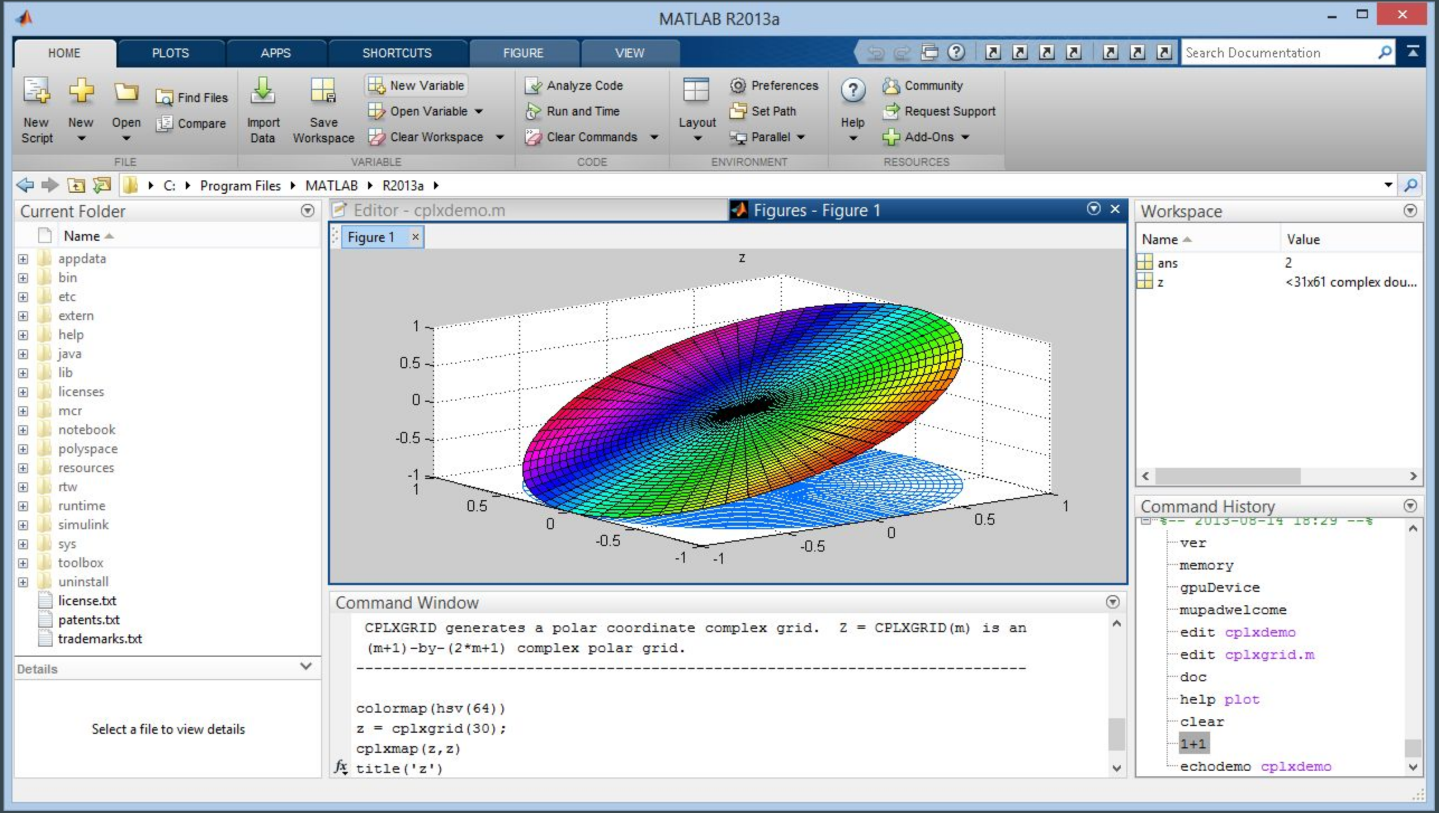
1. **The idea of matlab**
2. Good coding practices
3. A guide to writing unmaintainable code
4. Bugs and profiling
5. Filtering and its effects
6. Making beautiful plots
7. Graphical user interfaces

# What is matlab?

"MATLAB® is a high-level language and interactive environment for numerical computation, visualization, and programming" *www.mathworks.com*

- uses ‘English-like’ statements
  - as opposed to 0’s and 1’s
- specialized for scientific and technical computing
  - began as a tool for matrix manipulation
  - other tools such as C++ and Python are more general purpose
  - takes care of common processes for you (e.g. memory allocation)
- commercial software although Universities often purchase licenses

# What does it look like?

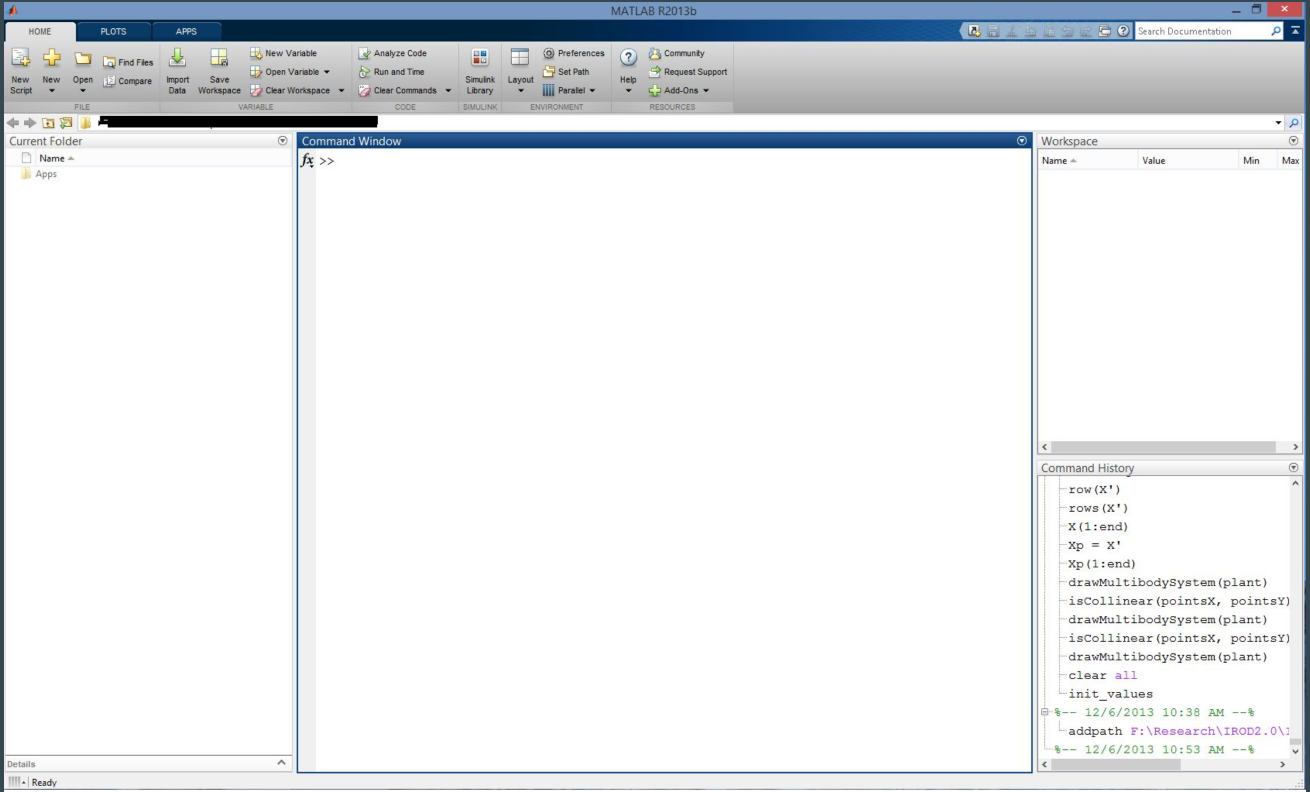


| Name | Value |
|---|---|
| ans | 2 |
| z | <31x61 complex dou... |

```
 CPLXGRID generates a polar coordinate complex grid.  Z = CPLXGRID(m) is an
 (m+1)-by-(2*m+1) complex polar grid.
--------------------------------------------------------------------------------

colormap(hsv(64))
z = cplxgrid(30);
cplxmap(z,z)
title('z')
```

```
%-- 2013-08-14 18:29 --%
ver
memory
gpuDevice
mupadwelcome
edit cplxdemo
edit cplxgrid.m
doc
help plot
clear
1+1
echodemo cplxdemo
```

# What does it look like?



[redacted]

```
fx >>
```

```
row(X')
rows(X')
X(1:end)
Xp = X'
Xp(1:end)
drawMultibodySystem(plant)
isCollinear(pointsX, pointsY)
drawMultibodySystem(plant)
isCollinear(pointsX, pointsY)
drawMultibodySystem(plant)
clear all
init_values
%-- 12/6/2013 10:38 AM --%
addpath F:\Research\IROD2.0\
%-- 12/6/2013 10:53 AM --%
```

# How does one best learn Matlab?

- Not watch someone talk about it...
- Treat Matlab like a foreign language
  - Be patient and learn basic structure by finding things that interest you
- Use common problem solving skills
  - Pick a problem that you are invested in
  - Break to problem into solvable parts in common language (commenting)
  - Use the functions help page [`edit mean`], or google to find functions for each step
  - Once the code works as intended, try to optimize the bottlenecks
  - Edit and re-edit the code to improve its readability and flexibility

# When to use matlab?

Program Development Cycle

- **Analyze**: define the problem
- **Design**: plan the solution
- Choose the **interface** (*maybe Matlab isn't the best solution*)
- **Code**: translate the algorithm into a programming language
- **Debug and Test**: find errors and improve efficiency
- Complete the **documentation**: describe the program

# Fibonacci Numbers

A man put a pair of rabbits in a place surrounded on all sides by a wall. How many pairs of rabbits can be produced from that pair in a year if it is supposed that every month each pair begets a new pair which from the second month on becomes productive?

$$f_n = f_n - 1 + f_n - 2$$

# Fibonacci Numbers

```
function f = fibonacci(n)
% FIBONACCI Fibonacci sequence
% f = FIBONACCI(n) generates the first n Fibonacci numbers.

f = zeros(n,1);
f(1) = 1;
f(2) = 2;
for k = 3 : n
    f(k) = f(k-1) + f(k-2);
end
```

1. The idea of matlab
2. **Good coding practices**
3. A guide to writing unmaintainable code
4. Bugs and profiling
5. Filtering and its effects
6. Making beautiful plots
7. Graphical user interfaces

# Good coding practices...

Sommerville has identified four generalised attributes which are not concerned with what a program does, but how well the program does it:

- Maintainability.
- Dependability.
- Efficiency.
- Usability.

https://en.wikipedia.org/wiki/Best_coding_practices

# Good coding practices...

From Meek & Heath: "What happens before one gets to the coding stage is often of crucial importance to the success of the project."[8]

- how is development structured? (life cycle)
- what is the software meant to do? (requirements)
- the overall structure of the software system (architecture)
- more detailed design of individual components (design)
- choice of programming language(s)

https://en.wikipedia.org/wiki/Best_coding_practices

# Good coding practices...

Requirements...

"The first prerequisite you need to fulfill before beginning construction is a clear statement of the problem the system is supposed to solve."

https://en.wikipedia.org/wiki/Best_coding_practices

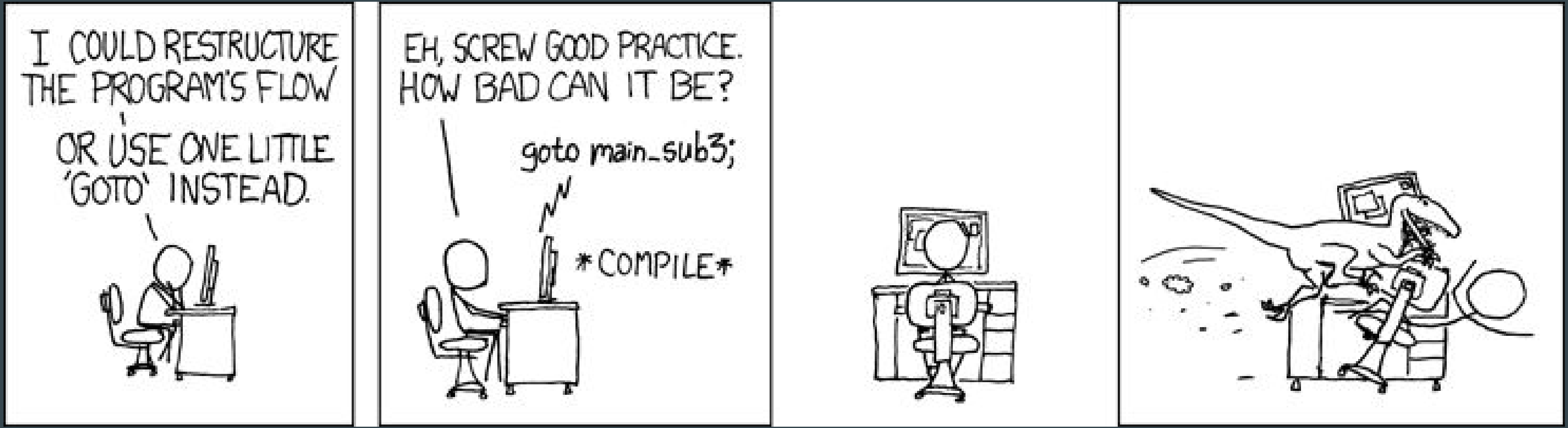
# Good coding practices...

Architecture...

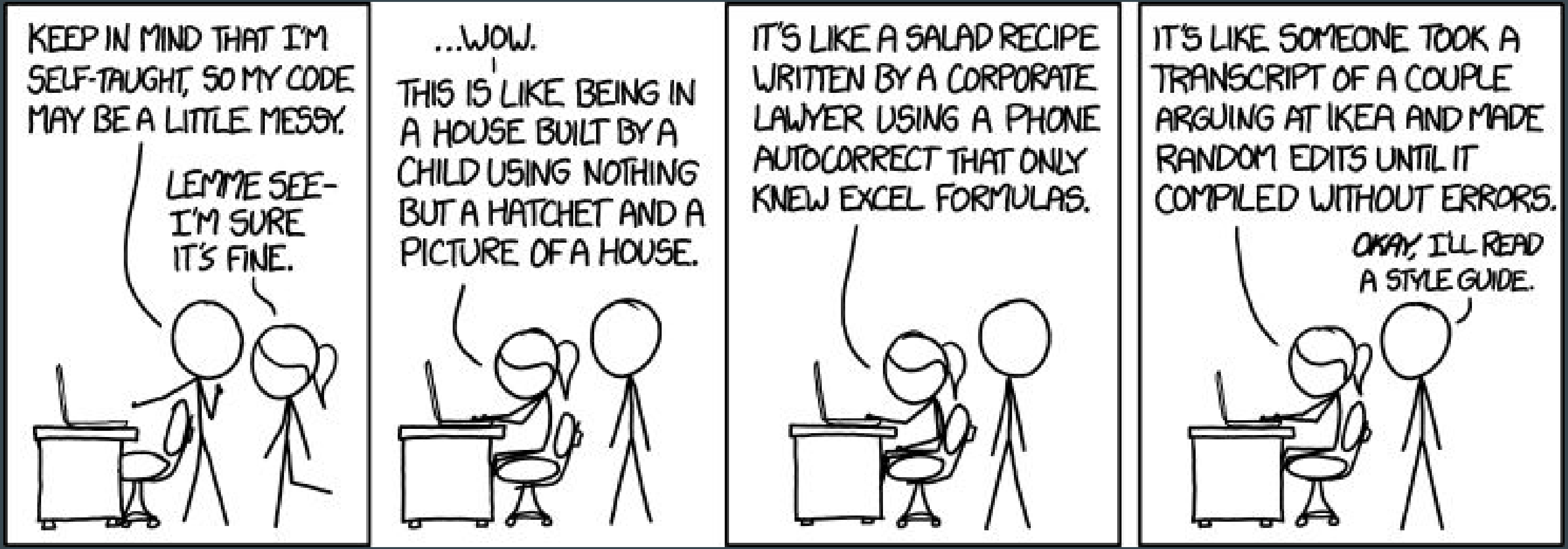
“there are two ways of constructing a software design: one way is to make it so simple that there are **obviously** no deficiencies; the other way is to make it so complicated that there are no **obvious** deficiencies. The first method is far more difficult.”

https://en.wikipedia.org/wiki/Best_coding_practices

# Good coding practices

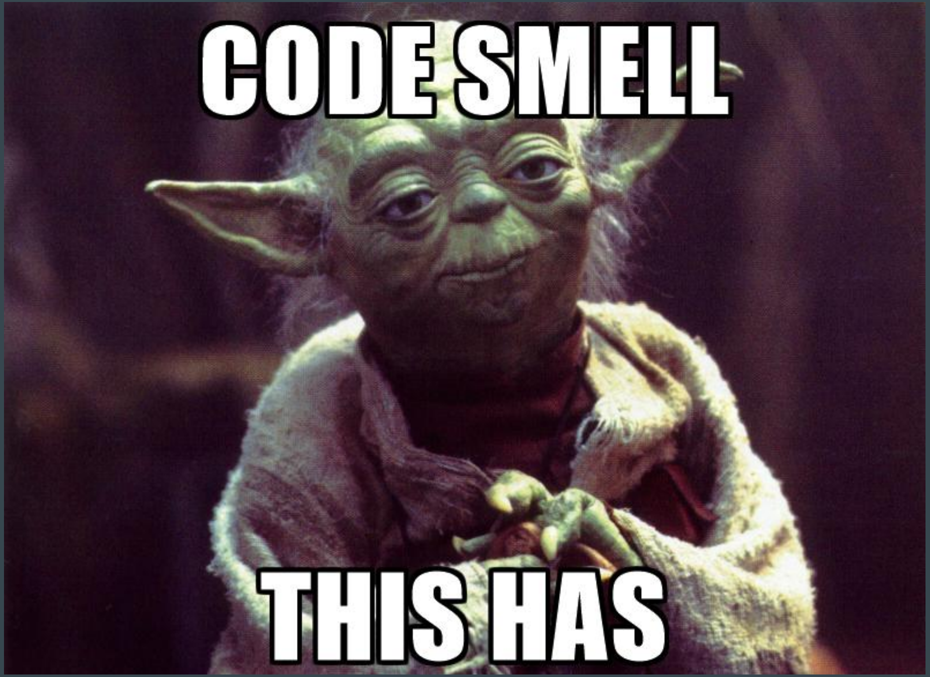

I COULD RESTRUCTURE THE PROGRAM'S FLOW
OR USE ONE LITTLE 'GOTO' INSTEAD.
EH, SCREW GOOD PRACTICE. HOW BAD CAN IT BE?
goto main_sub3;
*COMPILE*

# Good coding practices


KEEP IN MIND THAT I'M SELF-TAUGHT, SO MY CODE MAY BE A LITTLE MESSY.
LEMME SEE– I'M SURE IT'S FINE.
...WOW.
THIS IS LIKE BEING IN A HOUSE BUILT BY A CHILD USING NOTHING BUT A HATCHET AND A PICTURE OF A HOUSE.
IT'S LIKE A SALAD RECIPE WRITTEN BY A CORPORATE LAWYER USING A PHONE AUTOCORRECT THAT ONLY KNEW EXCEL FORMULAS.
IT'S LIKE SOMEONE TOOK A TRANSCRIPT OF A COUPLE ARGUING AT IKEA AND MADE RANDOM EDITS UNTIL IT COMPILED WITHOUT ERRORS.
OKAY, I'LL READ A STYLE GUIDE.

# Code smell

"smells are certain structures in the code that indicate violation of fundamental design principles and negatively impact design quality".



# Code smell

"smells are certain structures in the code that indicate violation of fundamental design principles and negatively impact design quality".

```
function load_file(name_of_file)

try
    load(name_of_file);

catch
    display(‘something bad...
        happened’)

end
```

```
function load_file(name_of_file)

if ~isa(name_of_file, ‘string’)
    display(‘input not a string’)
    return
end

if isa(name_of_file, ‘file’)
    display(‘cannot find file’)
    return
end

load(name_of_file);
```

# Spaghetti code

“a pejorative phrase for source code that has a complex and tangled control structure”

# Spaghetti code

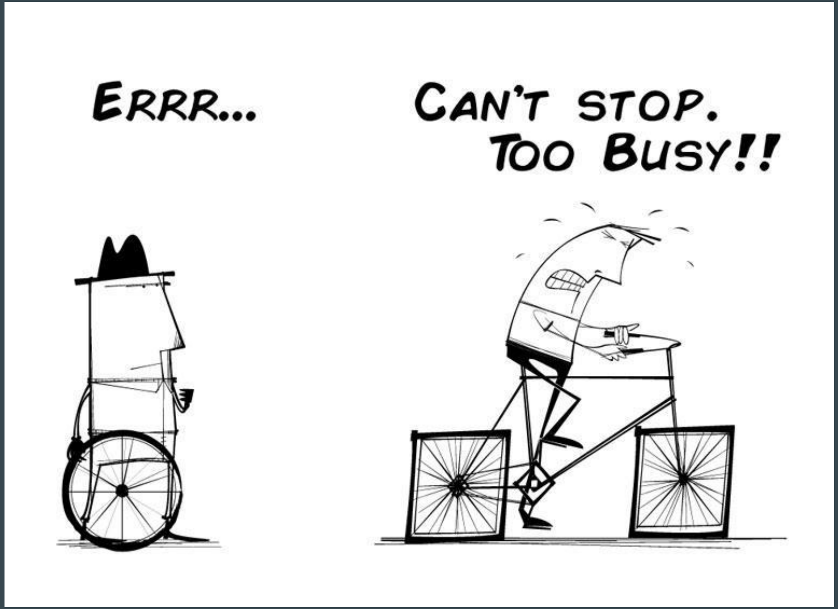
“a pejorative phrase for source code that has a complex and tangled control structure”

```
counter = 0;

while true:

    counter = counter + 1;

    fprintf(‘%i’, counter);

    if counter > 10
        break
    end
end
```

```
for counter = 1 : 10

    fprintf(‘%i’, counter);

end

% or even better

numbers_to_display = 1 : 10;
fprintf('%i ',
numbers_to_display);
```

# Technical Debt

"a concept in programming that reflects the extra development work that arises when code that is easy to implement in the short run is used instead of applying the best overall solution"



# Technical Debt

"a concept in programming that reflects the extra development work that arises when code that is easy to implement in the short run is used instead of applying the best overall solution"

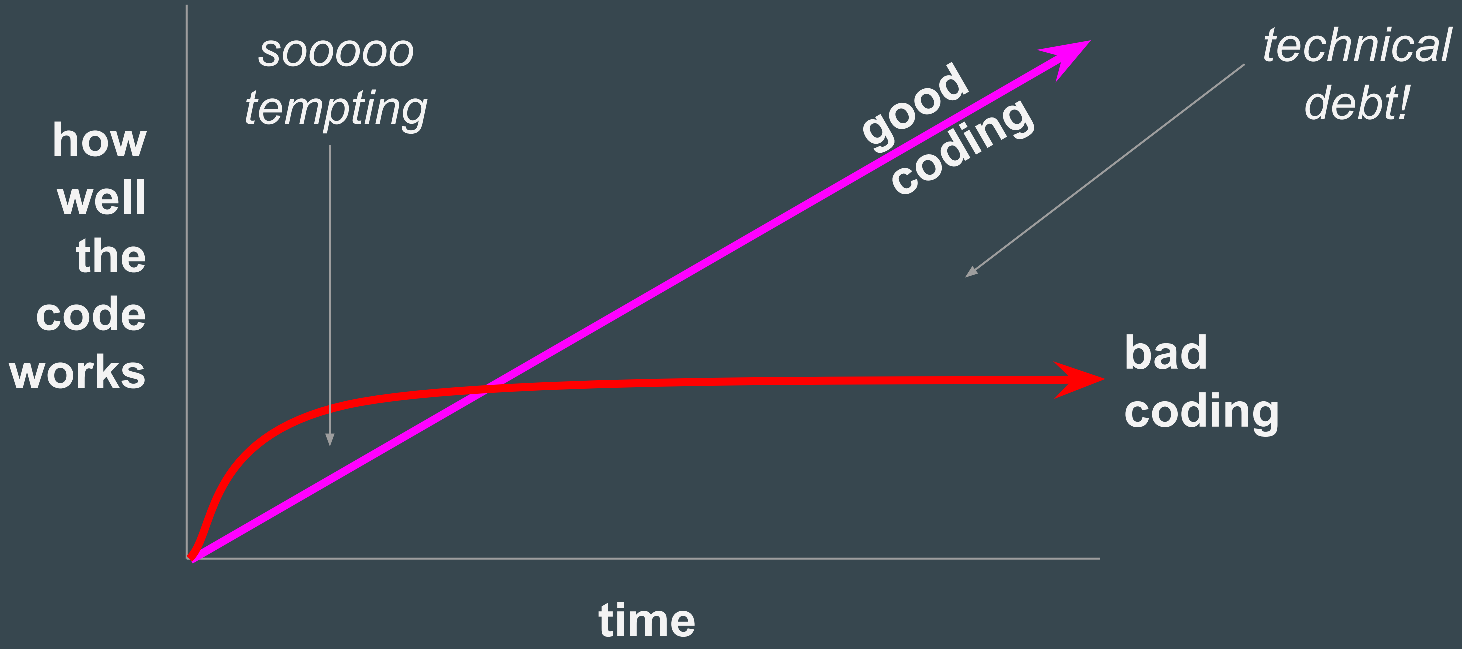
```
% do stuff
[var1, var2] = pwelch(x,50,25,50,200);
```

# Technical Debt

"a concept in programming that reflects the extra development work that arises when code that is easy to implement in the short run is used instead of applying the best overall solution"

```
window_length = 0.25 % time in seconds
sampling_rate = 200; % sampling rate of the data

delta_window = floor(sampling_rate * window_length);
delta_overlap = floor(delta_window / 2);

% calculate the fast fourier transform using the p-welch method
[delta_band, delta_range] = pwelch(...
    EEG.data’ ,...     % data (transposed so channels are columns)
    delta_window ,...  % window length
    delta_overlap ,... % overlap
    delta_window ,...  % points in calculation (window length)
    sampling_rate );   % sampling rate
```

# Technical Debt



1. The idea of matlab
2. Good coding practices
3. A guide to writing unmaintainable code
4. Bugs and profiling
5. Filtering and its effects
6. Making beautiful plots
7. Graphical user interfaces

# A guide to writing unmaintainable code

Secure a job for life by making your code only intelligible to you (and barely that)...

General Principle

*Quidquid latine dictum sit, altum sonatur.*

- Whatever is said in Latin sounds profound.

# A guide to writing unmaintainable code

## Variable Naming

*"When I use a word," Humpty Dumpty said, in a rather scornful tone, "it means just what I choose it to mean - neither more nor less."*

- Lewis Carroll -- Through the Looking Glass, Chapter 6

# A guide to writing unmaintainable code

Variable Naming

Single letter variables: `i, j, p`

Random names from a baby book: `fred, susie, billy`

Creative misspelling: `data_meen, filtre_parametre, coluor_seting`

Be abstract: `dataFunction32, important_function_blue`

Use underscores whenever possible: `x_1, x1_, x1_b, x_1b, xb_1, x_1_b`

# A guide to writing unmaintainable code

Camouflage

*"The longer it takes for a bug to surface, the harder it is to find.*

- Roedy Green

# A guide to writing unmaintainable code

## Camouflage

```
channel_mean = max(data);


mary_poppins = (superman + …
    batman) / rice_cooker;


timer_is_zero = 10;


important_variable = sqrt(nanmean(data));
clear important_variable
```

# A guide to writing unmaintainable code

Documentation

*Any fool can tell the truth, but it requires a man of some sense to know how to lie well.*
- Samuel Butler (1835 - 1902)

*Incorrect documentation is often worse than no documentation.*
*- Bertrand Meyer*

*"If you can't say anything nice, don't say anything at all".*
- Thumper from Disney’s Bambi

# A guide to writing unmaintainable code

Coding Obfuscation

*Sedulously eschew obfuscatory hyperverbosity and prolixity.*

- Roedy Green

# A guide to writing unmaintainable code

## Coding Obfuscation

```
some_data = rand(100, 1);
some_variable = 0;

for n = 1 : 100

    if some_data(n) > some_variable

        some_variable = some_data(n);
    end
end
```

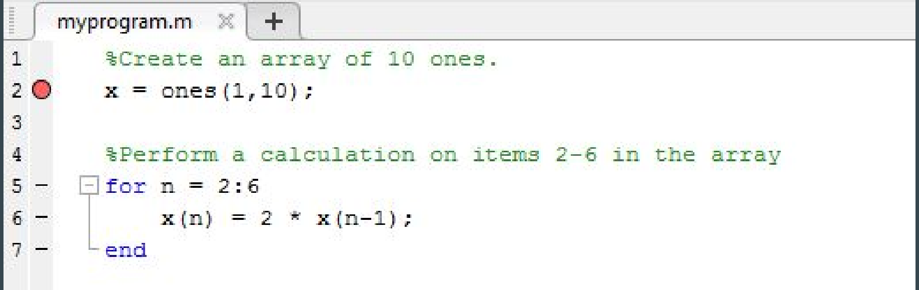
1. The idea of matlab
2. Good coding practices
3. A guide to writing unmaintainable code
4. **Bugs and profiling**
5. Filtering and its effects
6. Making beautiful plots
7. Graphical user interfaces

# common matlab error messages...

```
>> A(1))
??? A(1))
Error: Unbalanced or misused parentheses or brackets.

>> A = [1,3];
>> A(3)
??? Index exceeds matrix dimensions

>> A(0)
??? Subscript indices must either be real positive integers or logicals.

>> A(1:2, 1:2) = [1,2,3,4];
??? Subscripted assignment dimension mismatch.
```

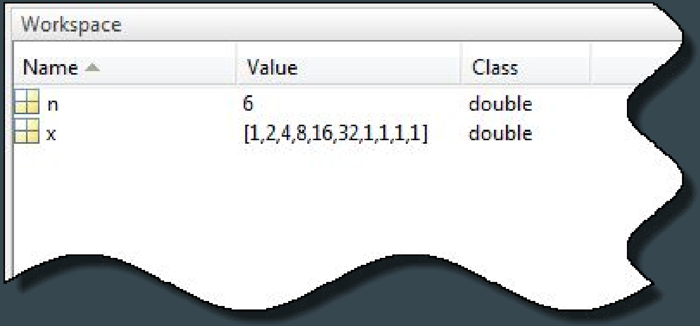
# bug in your code? don't panic...

Setting a “break point” in functions...



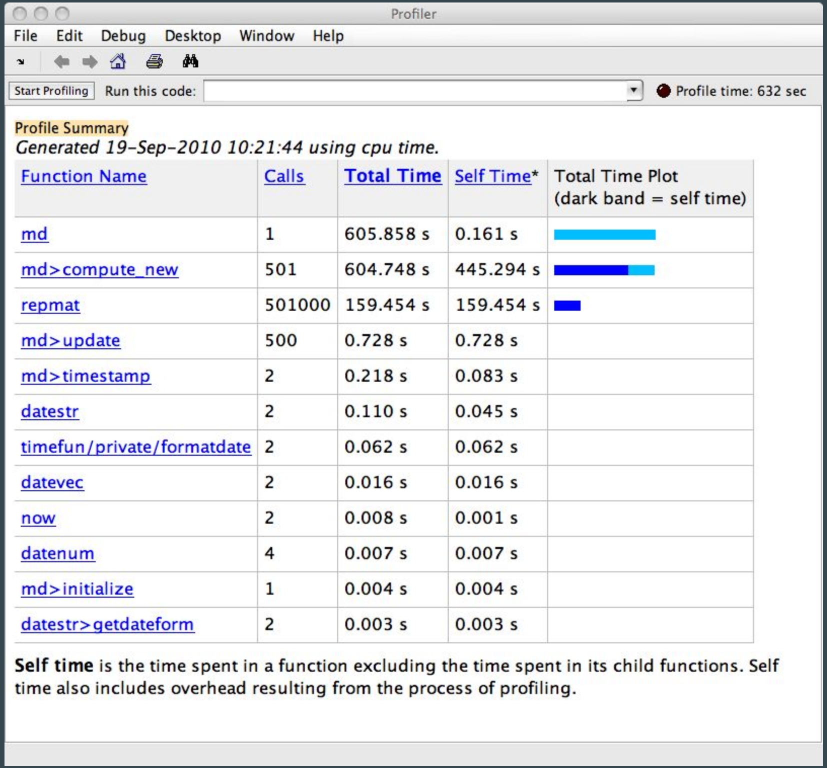
```
%Create an array of 10 ones.
x = ones(1,10);

%Perform a calculation on items 2-6 in the array
for n = 2:6
    x(n) = 2 * x(n-1);
end
```

Then examine your workspace as if you were in the middle of that function...



| Name | Value | Class |
|---|---|---|
| n | 6 | double |
| x | [1,2,4,8,16,32,1,1,1,1] | double |

# code a little slow? no worries, use the “profiler”

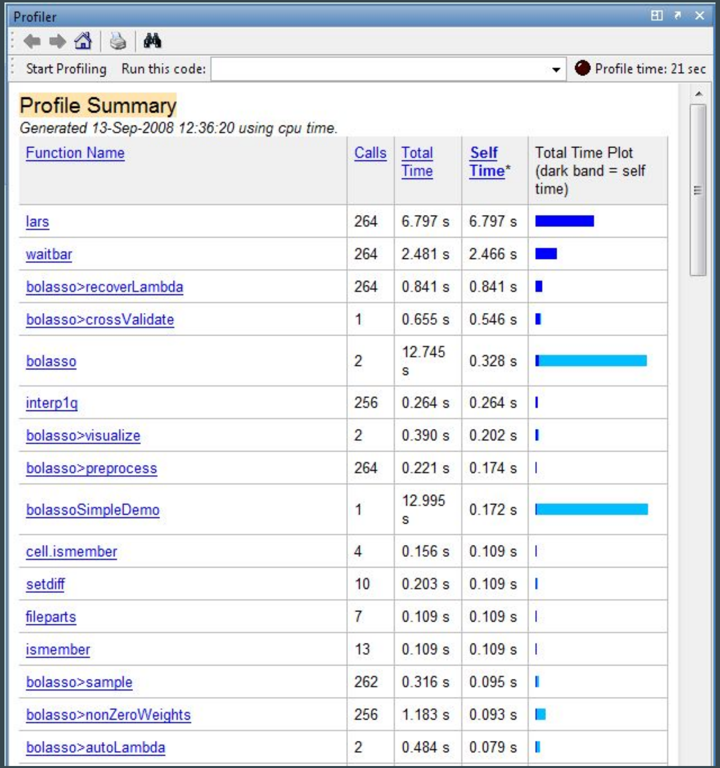


Profile Summary

*Generated 19-Sep-2010 10:21:44 using cpu time.*

| Function Name | Calls | Total Time | Self Time* | Total Time Plot (dark band = self time) |
|---|---|---|---|---|
| md | 1 | 605.858 s | 0.161 s | |
| md>compute_new | 501 | 604.748 s | 445.294 s | |
| repmat | 501000 | 159.454 s | 159.454 s | |
| md>update | 500 | 0.728 s | 0.728 s | |
| md>timestamp | 2 | 0.218 s | 0.083 s | |
| datestr | 2 | 0.110 s | 0.045 s | |
| timefun/private/formatdate | 2 | 0.062 s | 0.062 s | |
| datevec | 2 | 0.016 s | 0.016 s | |
| now | 2 | 0.008 s | 0.001 s | |
| datenum | 4 | 0.007 s | 0.007 s | |
| md>initialize | 1 | 0.004 s | 0.004 s | |
| datestr>getdateform | 2 | 0.003 s | 0.003 s | |

**Self time** is the time spent in a function excluding the time spent in its child functions. Self time also includes overhead resulting from the process of profiling.



Profile Summary

*Generated 13-Sep-2008 12:36:20 using cpu time.*

| Function Name | Calls | Total Time | Self Time* | Total Time Plot (dark band = self time) |
|---|---|---|---|---|
| lars | 264 | 6.797 s | 6.797 s | |
| waitbar | 264 | 2.481 s | 2.466 s | |
| bolasso>recoverLambda | 264 | 0.841 s | 0.841 s | |
| bolasso>crossValidate | 1 | 0.655 s | 0.546 s | |
| bolasso | 2 | 12.745 s | 0.328 s | |
| interp1q | 256 | 0.264 s | 0.264 s | |
| bolasso>visualize | 2 | 0.390 s | 0.202 s | |
| bolasso>preprocess | 264 | 0.221 s | 0.174 s | |
| bolassoSimpleDemo | 1 | 12.995 s | 0.172 s | |
| cell.ismember | 4 | 0.156 s | 0.109 s | |
| setdiff | 10 | 0.203 s | 0.109 s | |
| fileparts | 7 | 0.109 s | 0.109 s | |
| ismember | 13 | 0.109 s | 0.109 s | |
| bolasso>sample | 262 | 0.316 s | 0.095 s | |
| bolasso>nonZeroWeights | 256 | 1.183 s | 0.093 s | |
| bolasso>autoLambda | 2 | 0.484 s | 0.079 s | |

1. The idea of matlab
2. Good coding practices
3. A guide to writing unmaintainable code
4. Bugs and profiling
5. Filtering and its effects
6. Making beautiful plots
7. Graphical user interfaces

# How to filter in matlab...

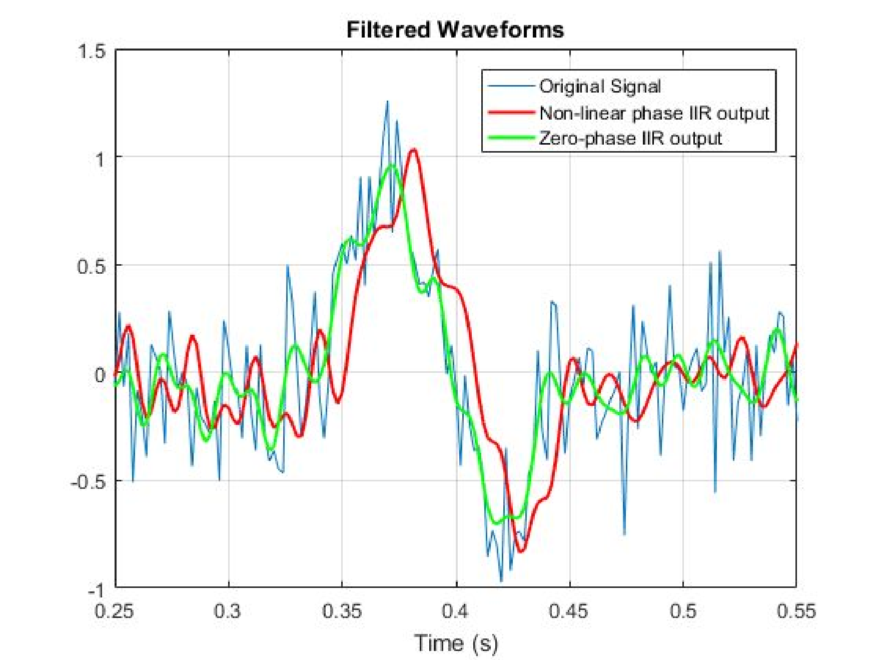
Practical guide to matlab filtering

*(http://ch.mathworks.com/help/signal/examples/practical-introduction-to-digital-filtering.html)*

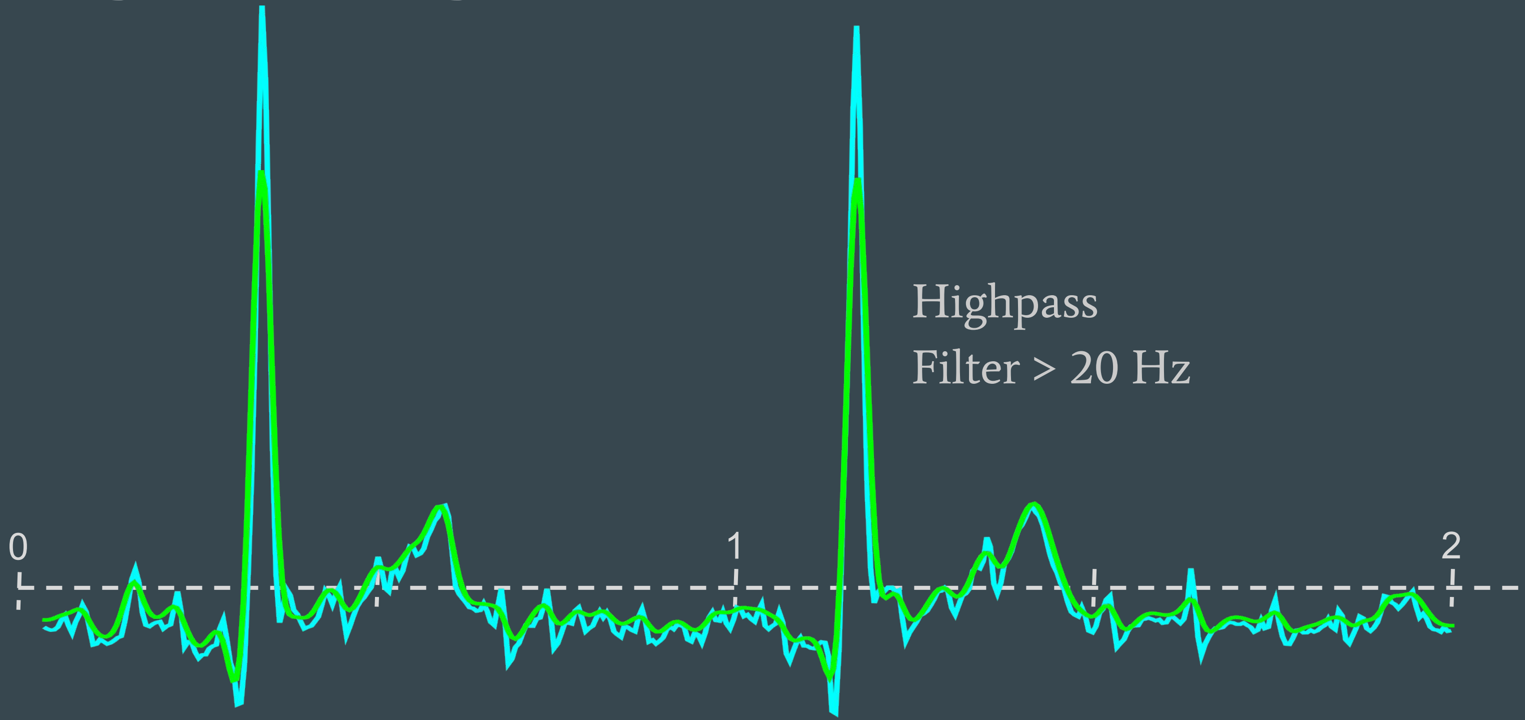
```
sampling_rate = 500; % sample rate in Hz
num_samples = 500;  % number of signal samples
sample_data = ecg(num_samples)' + 0.25 * randn(num_samples, 1);  % noisy waveform
time_range = (0 : num_samples - 1) / sampling_rate;    % time vector

freq_normalized = 75 / (sampling_rate/2);    % Normalized frequency
filter_design = designfilt('lowpassfir', 'FilterOrder', 70, ...
     'CutoffFrequency',freq_normalized);

filtered_data_nl = filter(filter_design, sample_data); % non-linear phase filter
filtered_data_zp = filtfilt(filter_design, sample_data);  % zero-phase implementation
```

# How to filter in matlab...

```
figure
plot(time_range, sample_data);
hold on
plot(time_range, filtered_data_nl,...
    'r', 'linewidth' ,1.5);
plot(time_range, filtered_data_zp,...
    'g', 'linewidth' ,1.5);
title('Filtered Waveforms');
xlabel('Time (s)')
legend('Original Signal','Non-linear
phase IIR output',...
  'Zero-phase IIR output');
ax = axis;
axis([0.25 0.55 ax(3:4)])
grid on
```

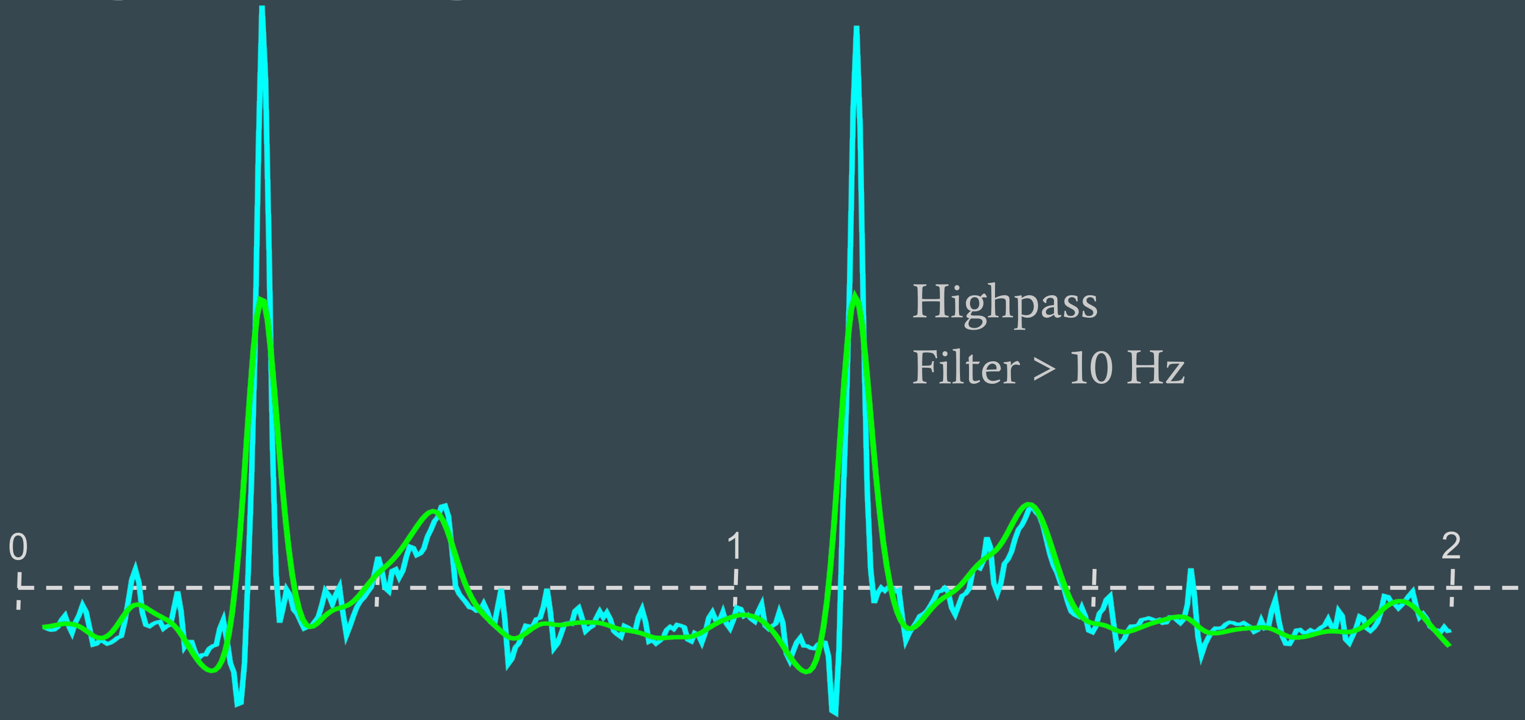


# filtering an ECG signal

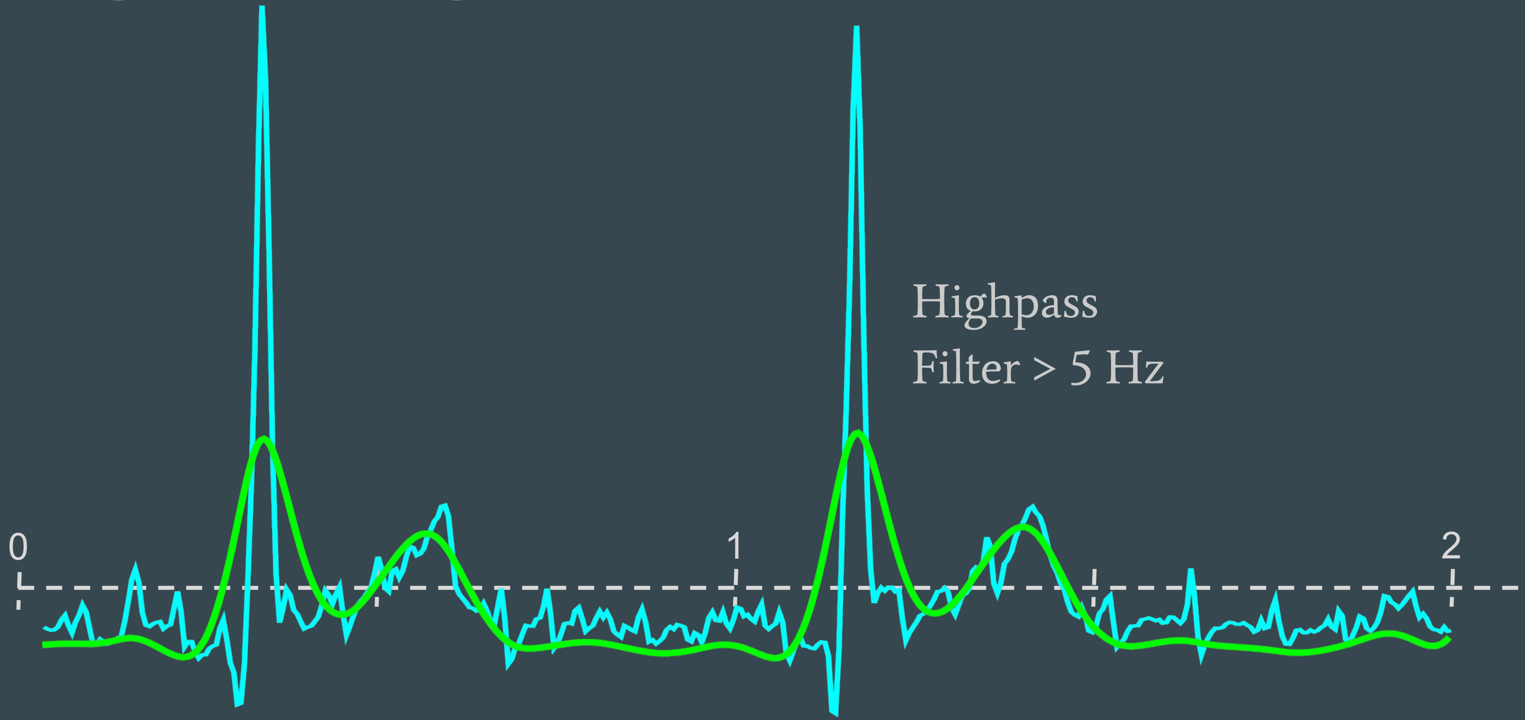


Highpass
Filter > 20 Hz

# filtering an ECG signal

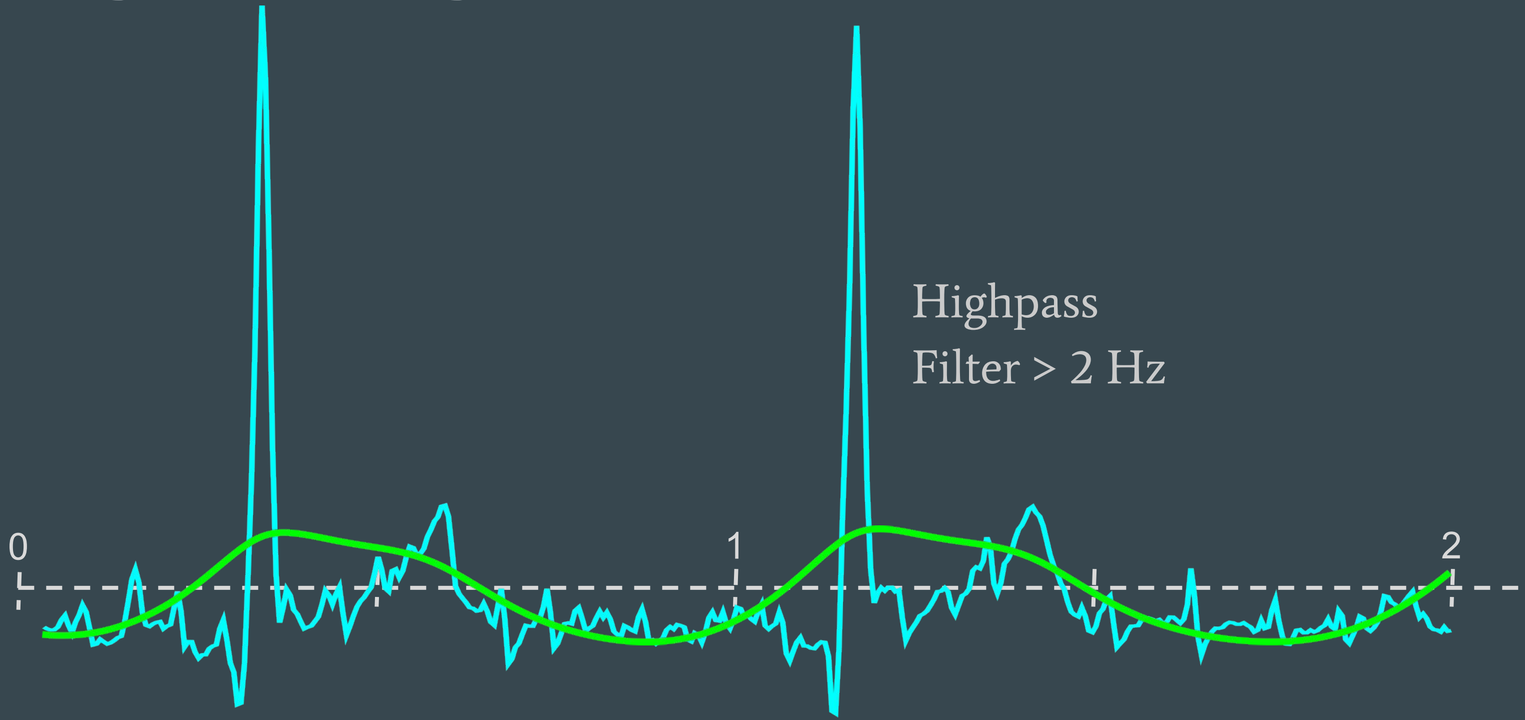


# filtering an ECG signal



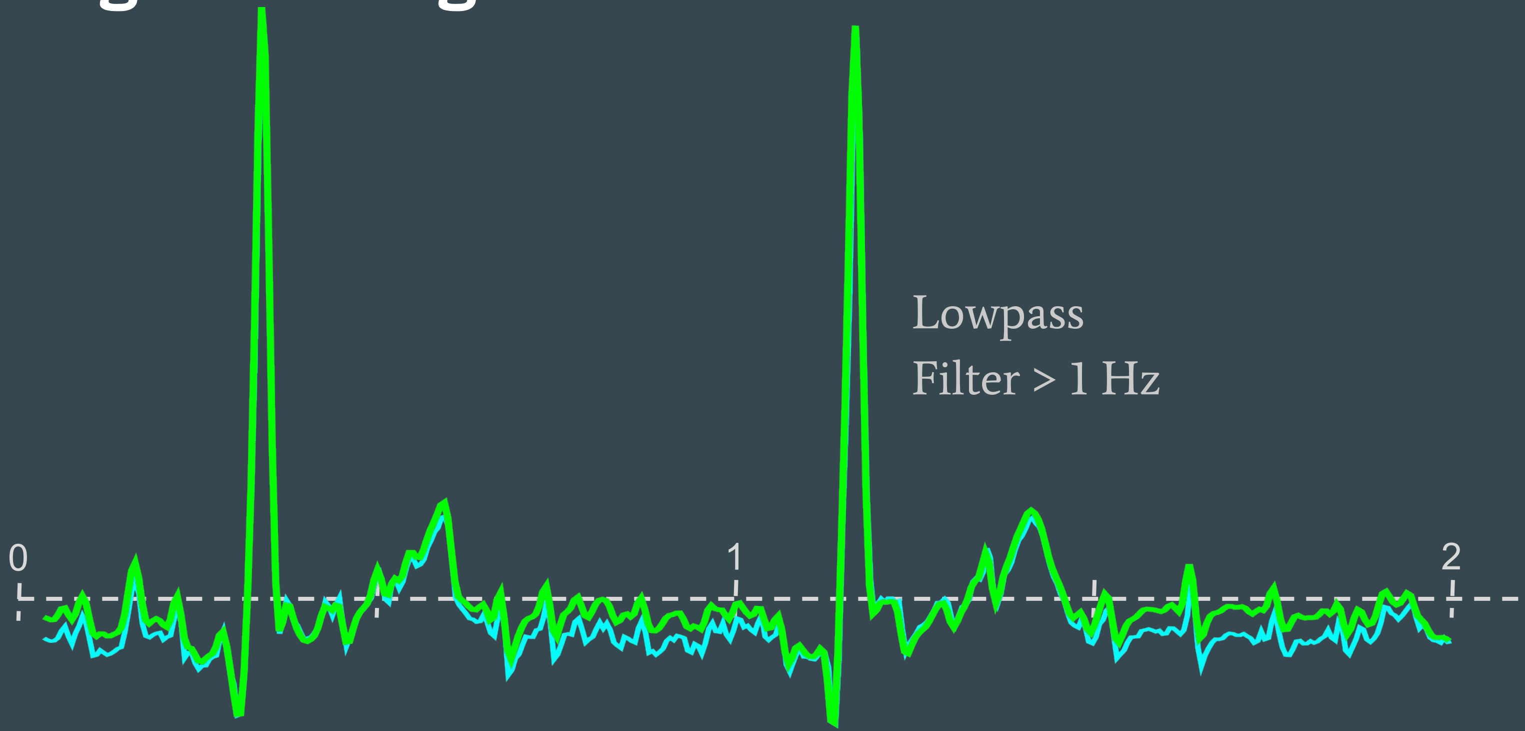
Highpass
Filter > 5 Hz

# filtering an ECG signal



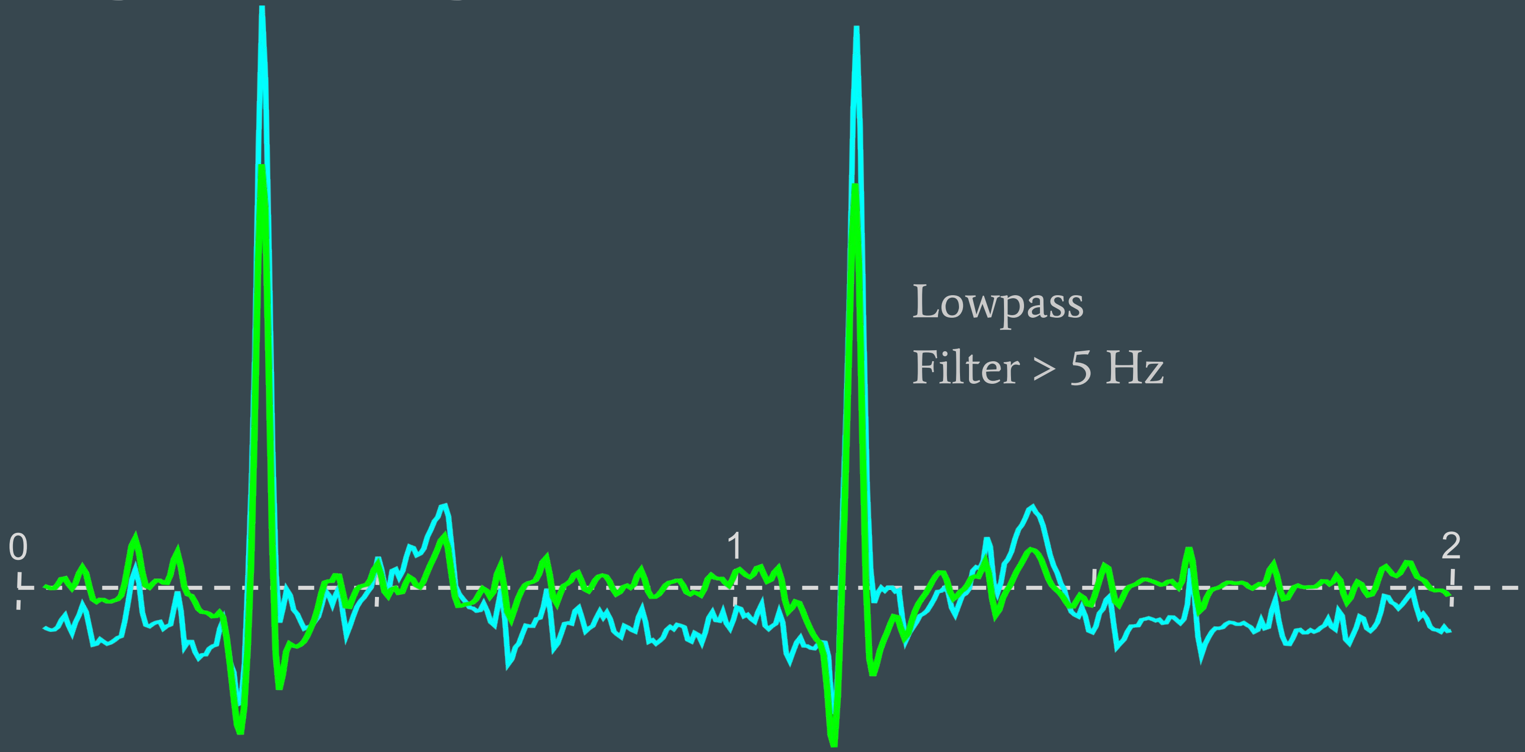
Highpass
Filter > 2 Hz

# filtering an ECG signal



Lowpass
Filter > 1 Hz

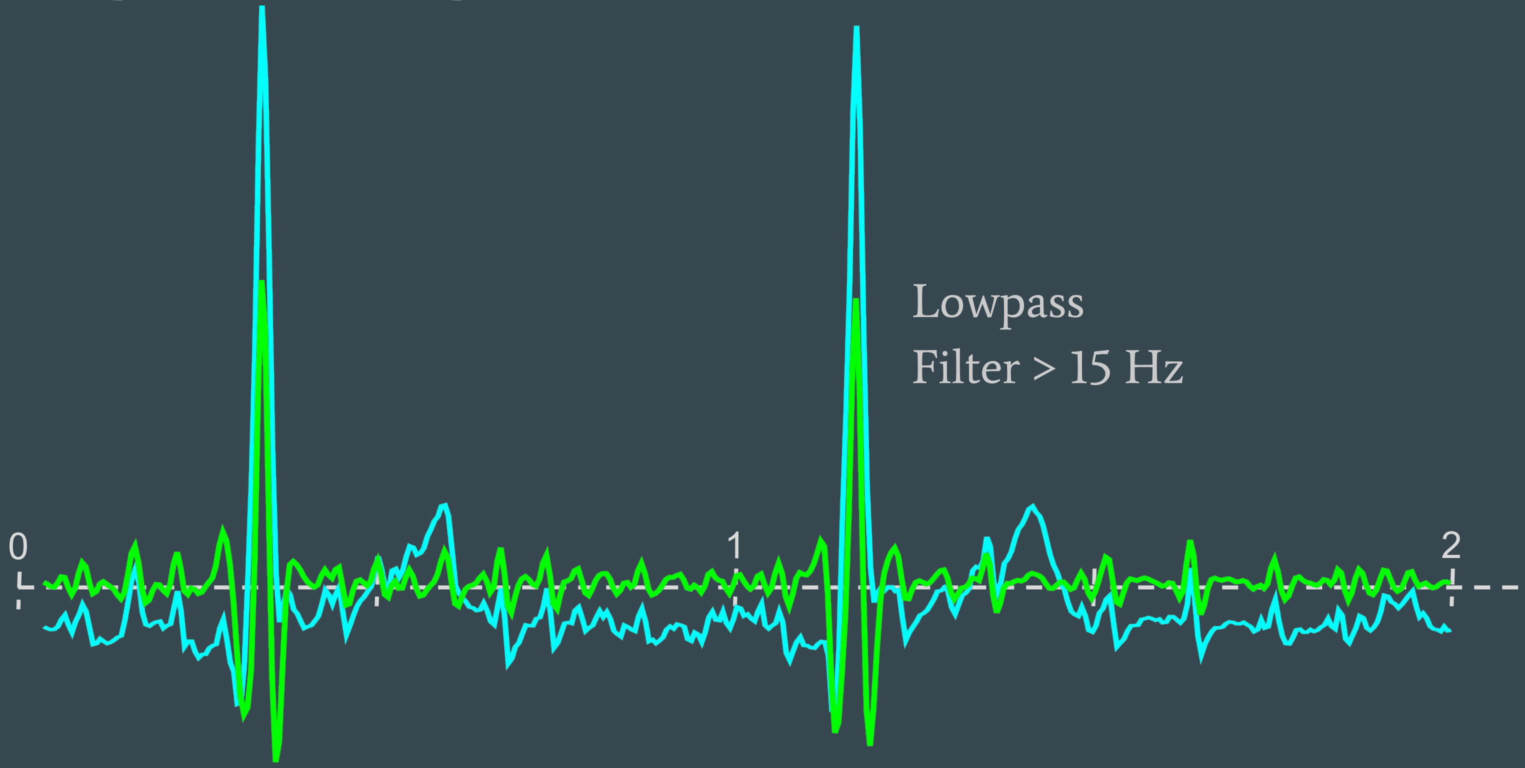
# filtering an ECG signal



Lowpass

Filter > 5 Hz

# filtering an ECG signal



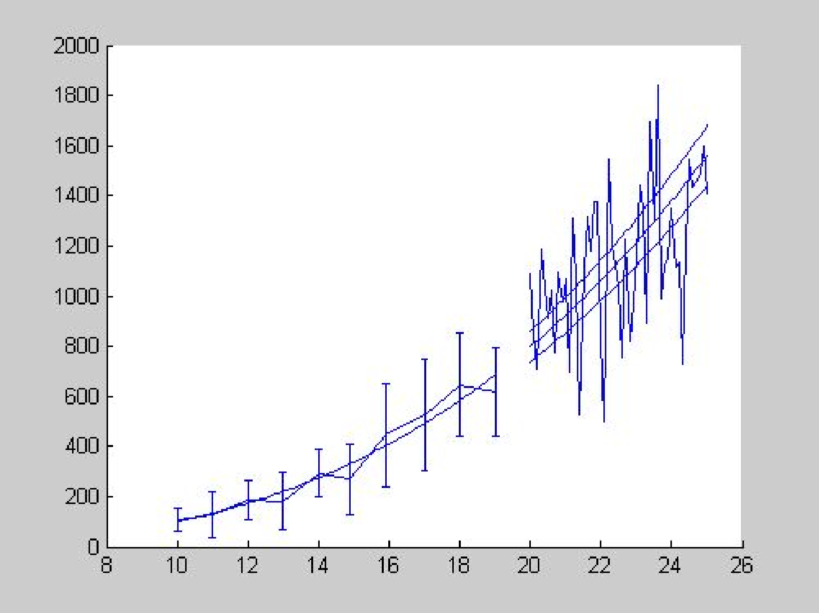
Lowpass

Filter > 15 Hz

1. The idea of matlab
2. Good coding practices
3. A guide to writing unmaintainable code
4. Bugs and profiling
5. Filtering and its effects
6. Making beautiful plots
7. Graphical user interfaces

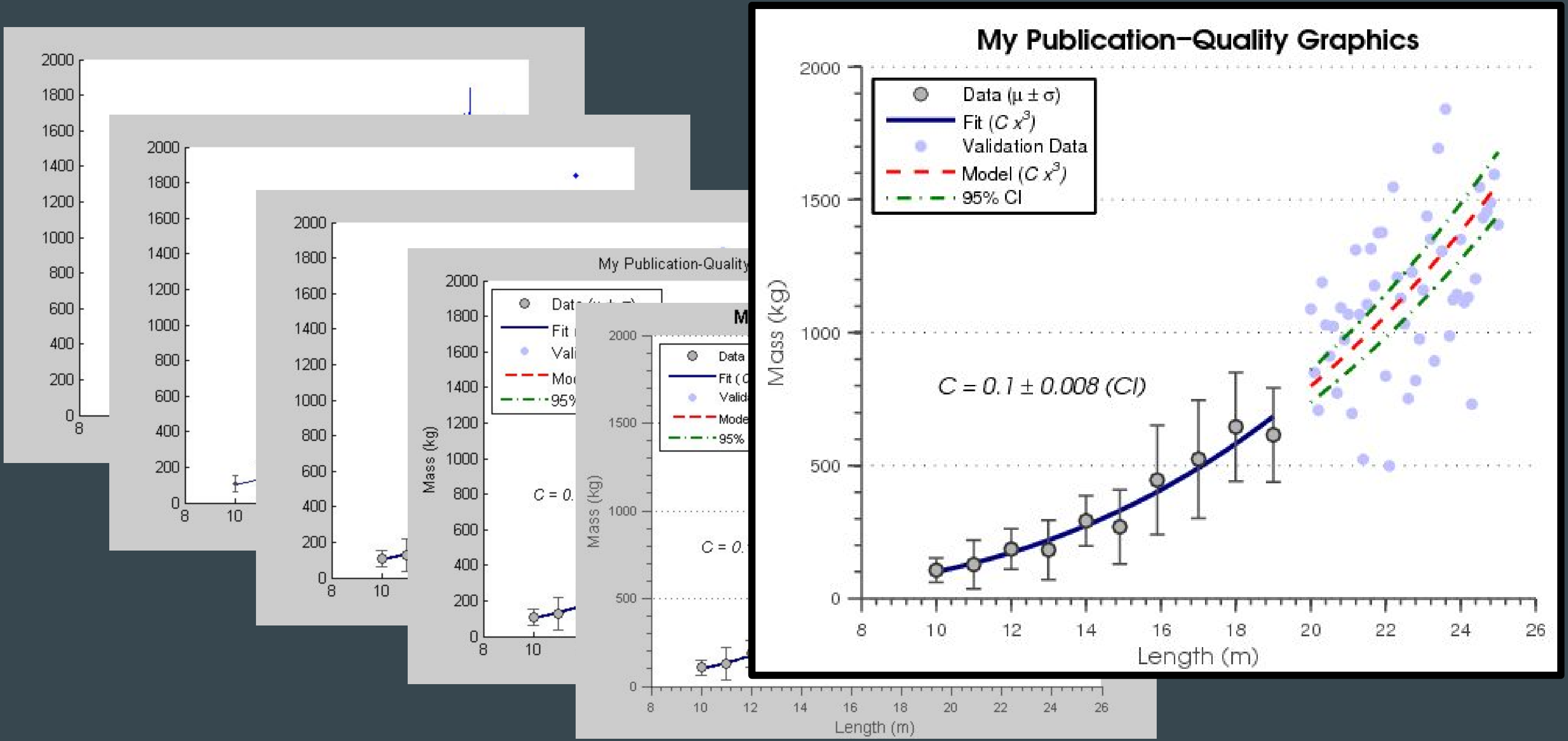
# Matlab capabilities in plotting

```
load data

figure;
hold on;

hFit = line(xfit  , yfit   );
hE = errorbar(xdata_m, ydata_m,
ydata_s);

hData  = line(xVdata, yVdata );
hModel = line(xmodel, ymodel );
hCI(1) = line(xmodel, ymodelL);
hCI(2) = line(xmodel, ymodelU);
```

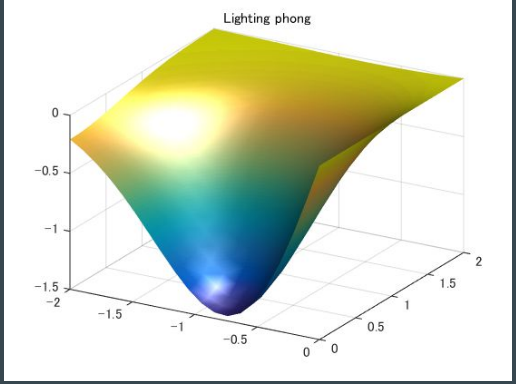


http://blogs.mathworks.com/loren/2007/12/11/making-pretty-graphs/

# Matlab capabilities in plotting

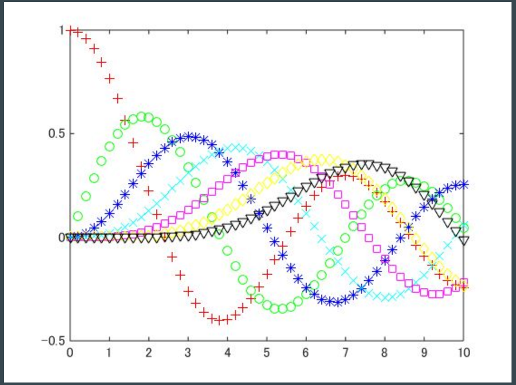


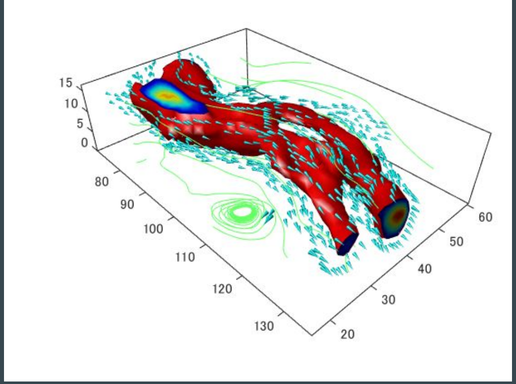
# Matlab capabilities in plotting



3D surface plots with lighting effects...

colourful line plots using distinct markers





mixed plots with streamlines, contours and objects

# Getting started

Always try to pre-specify plotting components... don’t be surprised by results

```
% open a figure window
handles.figure = figure( ...
     'color', 'w', ...
     'units', 'normalized', ...
     'position', [0.25, 0.25, 0.5, 0.5]);

% open an axes window
handles.axes = axes( ...
     'parent', handles.figure, ...
     'nextplot', 'add', ...
     'xlim', [0, 100], ...
     'ytick', []);

% plot some data
handles.line = plot(rand(100, 1));
```

```
% alternatively (but worse)...
% plot some data
plot(rand(100, 1));

% get the handles from the figure
handles.figure = gcf;
handles.axes = gca;
handles.line = gco;

% set some properties
set(handles.figure, ...
     'color', 'w', ...
     'units', 'normalized', ...
     'position', [0.25, 0.25, 0.5, 0.5]);
```

# Getting started

See the properties with `get` on the handles or google the plot functions

```
% open a figure window
get(handles.figure)

     Alphamap: [1x64 double]
     BeingDeleted: 'off'
     BusyAction: 'queue'
     ButtonDownFcn: ''
     Children: [0x0 GraphicsPlaceholder]
     Clipping: 'on'
     CloseRequestFcn: 'closereq'
     Color: [1 1 1]
     Colormap: [64x3 double]
     CreateFcn: ''
     CurrentCharacter: ''
     CurrentPoint: [-0.0017857 -0.002381]
     DeleteFcn: ''
```

```
…
     DockControls: 'on'
     FileName: ''
     GraphicsSmoothing: 'on'
     HandleVisibility: 'on'
     IntegerHandle: 'on'
     Interruptible: 'on'
     InvertHardcopy: 'on'
     KeyPressFcn: ''
     KeyReleaseFcn: ''
     MenuBar: 'figure'
…

set(handles.figure, ‘color’, [0, 0, 0]);
```

1. The idea of matlab
2. Good coding practices
3. A guide to writing unmaintainable code
4. Bugs and profiling
5. Filtering and its effects
6. Making beautiful plots
7. Graphical user interfaces

# Getting started

GUIs are “little more” than figures with buttons and “callbacks”

```
handles.figure = figure(...
     'color', 'w');

handles.button = uicontrol(...
     'Parent', handles.figure,...
     'Style', 'pushbutton',...
     'String', 'plot',...
     'Units', 'normalized',...
     'Position', [0.4 0.4 0.2 0.2], ...
     'Callback', 'plot(rand(100, 1))');
```