Welcome to

Carabao Workshop

What the Hell is a Carabao?

Drunken Carabao Tour

From: http://siargaophilippines.com/drunken-carabao-tour

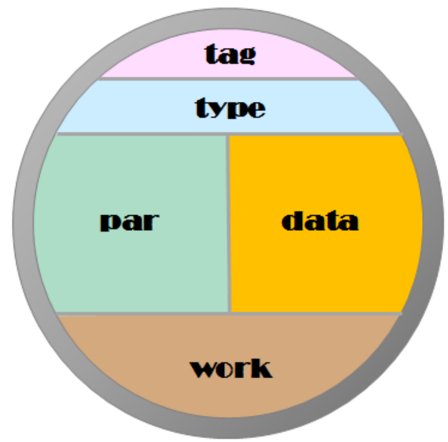
Carabao

carabao

- domestic swamp type water buffalo
- very powerful for Philippine farmers

Carabao

- Matlab type ‚domestic‘ class object
- very powerful for process, system & control engineers



Workshop Contents

Carbao is about toolbox engineering

- About 70% of each toolbox is based on similar building blocks

Carabao Building Blocks

- object manipulation (save, load, import, export, copy, cut, paste)
- rapid setup of a roll down menu providing a user interface (the graphical shell)
- manipulating and pre-processing of log data
- Those building blocks are now condensed in the MATLAB® Carabao toolbox.

Workshop Contents

Day 1: Mission Possible

Day 2: Meeting with Carabao

Day 3: Time for Espresso

Day 4: Shell Hard Core

Day 5: Riding the Carabao

Supported Files

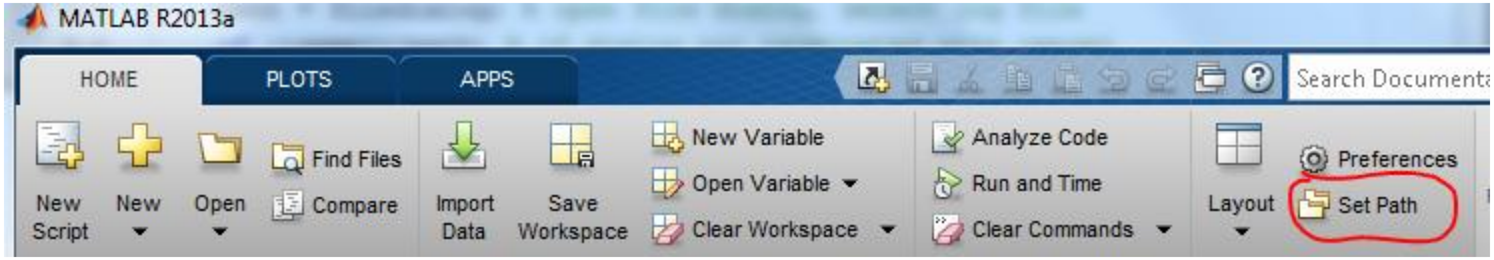
CaraMec.V1e.zip

- Carabao/v1e (Carabao toolbox)
- Doc (doc files)
- Imec (sample files)

Move Toolbox Files to an "m-file location"

- eg: m/Carabao/v1e

Set MATLAB path to toolbox folder



Day 1

Mission Possible

Mission ~~Im~~possible

Assume we got some log data

- two data streams: x, y
- each containing a stream of 1000 numbers

Mission #1

- Analyze this data:
- plot data
- calculating statistical numbers like
  - mean value
  - standard deviations
  - correlation coefficient.

Create Test Data

Some Test Data

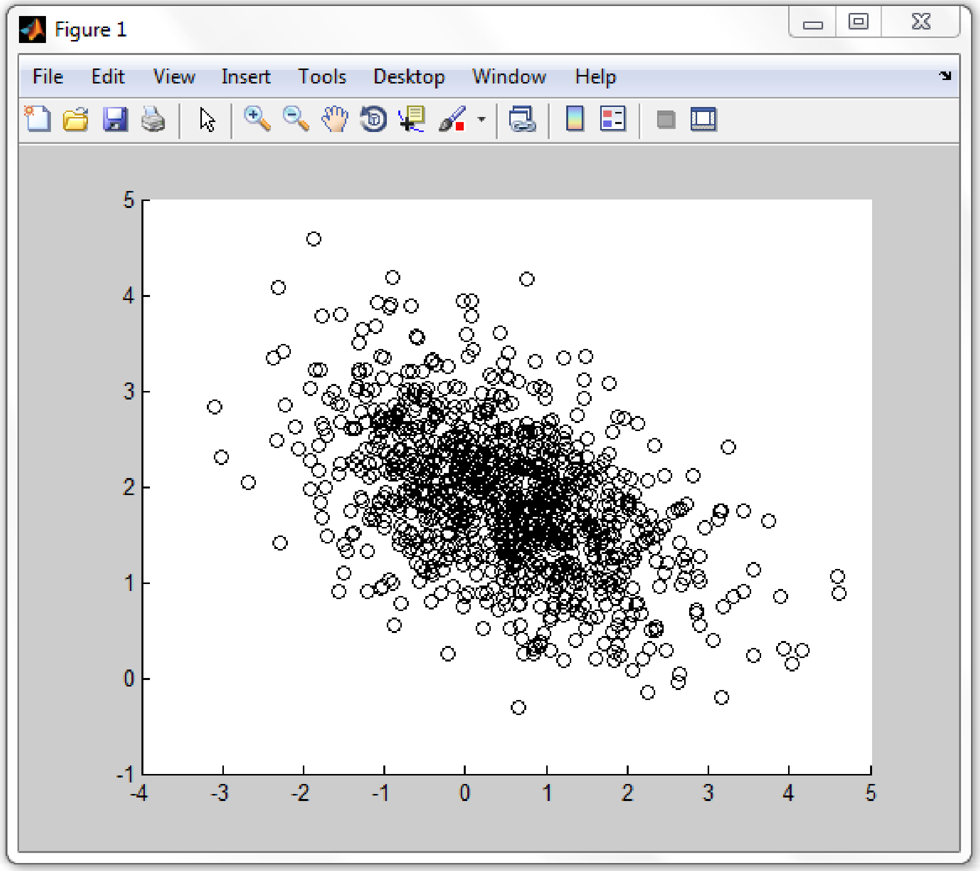
```
>> log = randn(1000,2);   % boring test data
```

Non Boring Test Data

```
>> rng('default');          % reset random generator
>> log = ones(1000,1)*randn(1,2) + randn(1000,2)*randn(2,2)
log =
   -1.7813 1.8336
    1.7186 1.2289
    0.6787 2.2148
   -0.6088 1.4329
    0.4465 1.1098
    0.7128 2.1951
      :      :
```

Draw Scatter Plot

```
>> x = log(:,1); y = log(:,2);
>> scatter(x,y,'k')                    % black scatter plot
```



Calculate Statistical Quantities

Mean Value

```
>> m = mean([x y]) % mean value
m =
0.4857 1.8666
```

Standard Deviation

```
>> s = std([x y]) % standard deviation (sigma)
s =
1.1480 0.7454
```

Correlation Coefficient

```
>> c = corrcoef(x,y)
c =
1.0000 -0.4803
-0.4803 1.0000
```

Calculate Statistical Quantities

Mean Value

```
>> m = mean([x y]) % mean value
m =
0.4857 1.8666
```

Standard Deviation

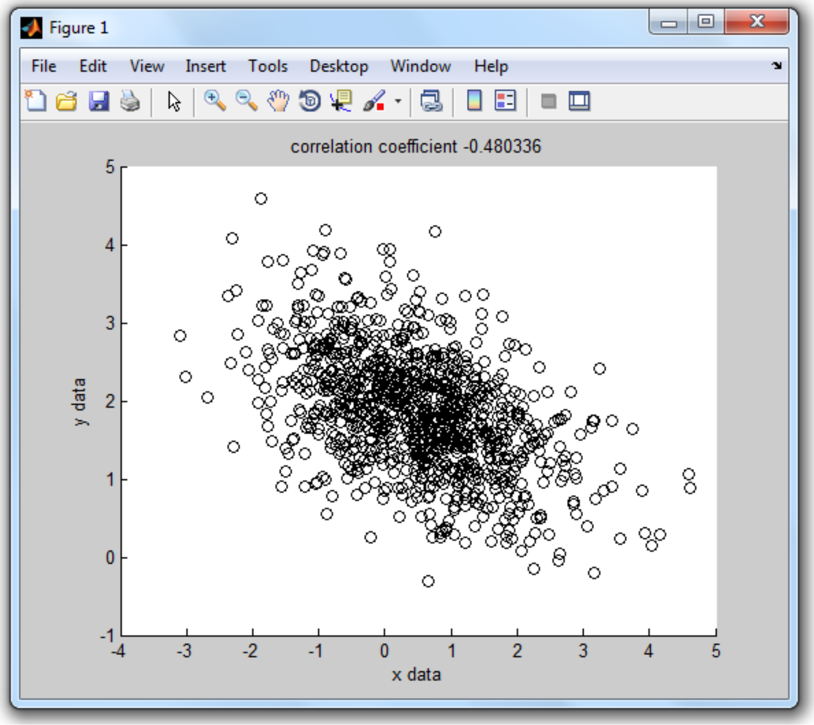
```
>> s = std([x y]) % standard deviation (sigma)
s =
1.1480 0.7454
```

Correlation Coefficient

```
>> c = corrcoef(x,y)
c =
1.0000 -0.4803
-0.4803 1.0000
```

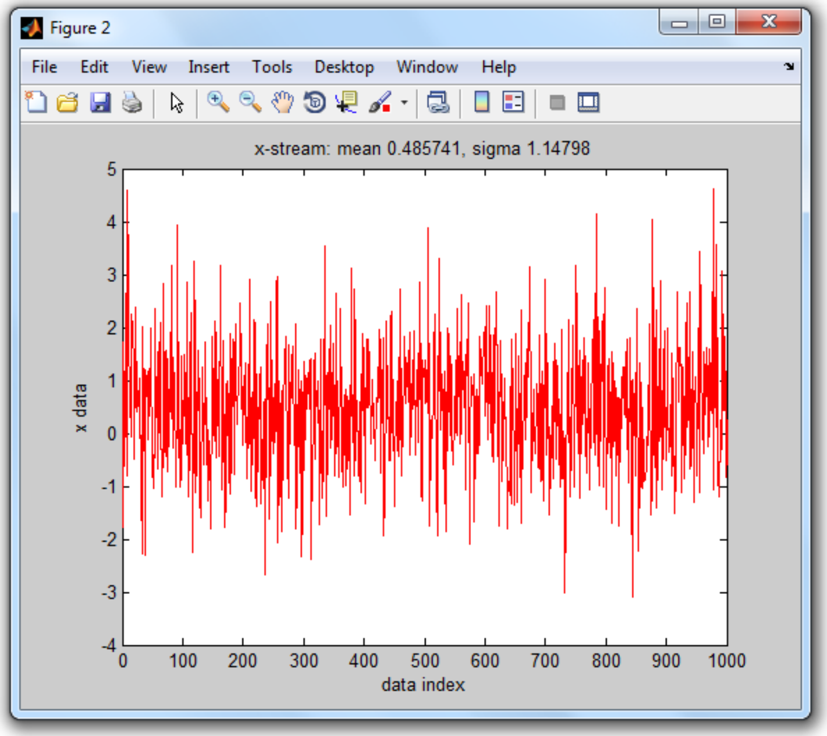
Scatter Plot with Labels

```
>> figure % open new figure
>> scatter(x,y,'k')
>> xlabel('x data');
>> ylabel('y data');
>> title(sprintf('correlation coefficient %g',c(1,2)));
```



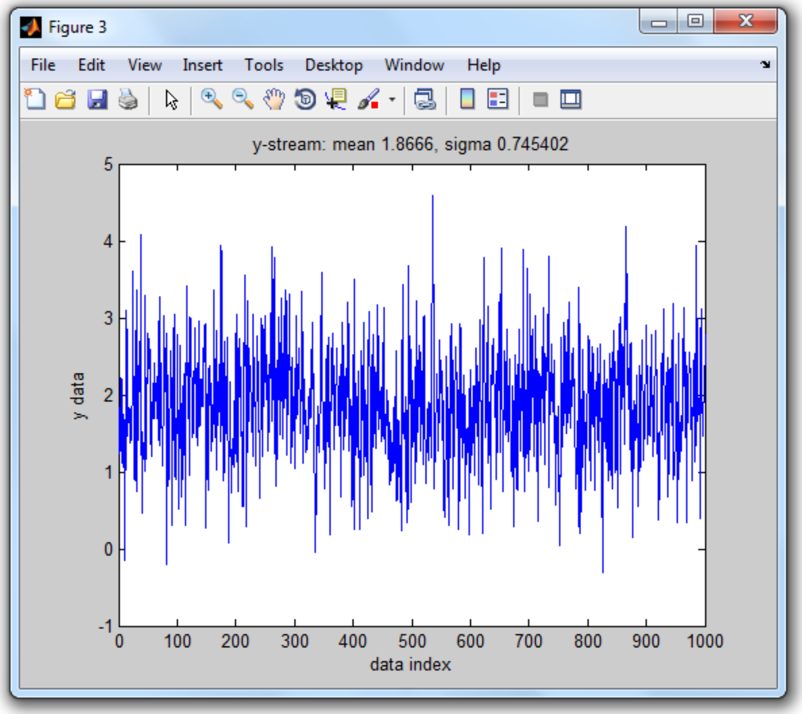
Plot X-Stream with Labels

```
>> figure % open new figure
>> plot(x,'r');
>> xlabel('data index');
>> ylabel('x data');
>> title(sprintf('x-stream: mean %g, sigma %g',m(1),s(1)));
```



Plot X-Stream with Labels

```
>> figure % open new figure
>> plot(y,'b');
>> xlabel('data index');
>> ylabel('y data');
>> title(sprintf('y-stream: mean %g, sigma %g',m(2),s(2)));
```



Plot X-Stream with Labels

Mission #1 completed

- data creation
- comand line plotting / labeling

Mission ~~Im~~possible

Mission #2

- Provide an easy-to-use MATLAB tool
- Allows any user to analyse a given data log file *)
- provide 3 plots with proper labeling (same as we did in mission #1)

*) any user who received short instructions

Write a MATLAB Function

Write an M-File Function (which could look as follows)

```
function analysis                          % log data analysis
   path = filedialog;                      % open file dialog, select log file
   if ~isempty(path)                       % if dialog not canceled
      [x,y,par] = read(path);              % read data (x,y) and parameters
      figure                               % open new figure
      scatterplot(x,y,par);                % draw black scatter plot
      figure                               % open new figure
      streamplot(x,'x','r',par);           % plot x-stream in red color
      figure                               % open new figure
      streamplot(y,'y','b',par);           % plot y-stream in blue color
   end
end
```

Organizing Files

We have

- log data files
- function files (M-file functions)
- class methods

Put in a project directory with version folders

```
imec
imec/log
imec/v1a
```

Creating Log Data Files

- log data file shall begin with a parameter definition (title of our log data)
- syntax: ‘$‘ <parameter> ‘=‘ <value>

```
function create(path)  % create random data log file (v1a/create.m)
%
%  CREATE   Create random data & log to a log file: create(path)
%
   [~,name] = fileparts(path);
   log = ones(1000,1)*randn(1,2) + randn(1000,2)*randn(2,2);
   x = log(:,1);  y = log(:,2);

   fid = fopen(path,'w');                      % open log file for write
   if (fid < 0)
      error('cannot open log file');
   end

   fprintf(fid,'$title=%s\n',upper(name));
   fprintf(fid,'%10f %10f\n',log');            % write x/y data
   fclose(fid);                                % close log file
end
```

Actual Log Data Creation

Type in command line

```
>> rng('default');    % reset random generator
>> create('data1.log');
>> create('data2.log');
>> create('data3.log');
>> create('data4.log');
>> create('data5.log');
```

We get a log data file like:

```
>> type data1.log
$title=DATA1
 -1.781269   1.833640
  1.718563   1.228938
  0.678696   2.214829
 -0.608834   1.432873
     :           :
```

Reading Log Data

Function to read log data into variables x, y & par

```
function [x,y,par] = read(path)          % read log data (v1a/read.m)
   fid = fopen(path,'r');
   if (fid < 0)
      error('cannot open log file!');
   end
   par.title = fscanf(fid,'$title=%[^\n]');
   log = fscanf(fid,'%f',[2 inf])';      % transpose after fscanf!
   x = log(:,1); y = log(:,2);
end
```

We get a log data file like:

```
>> [x,y,par]=read('data1.log');

>> par
par =
    title: 'DATA1'
```

```
>> [x(1:5),y(1:5)]
ans =
   -1.7813    1.8336
    1.7186    1.2289
    0.6787    2.2148
   -0.6088    1.4329
    0.4465    1.1098
       :         :
```

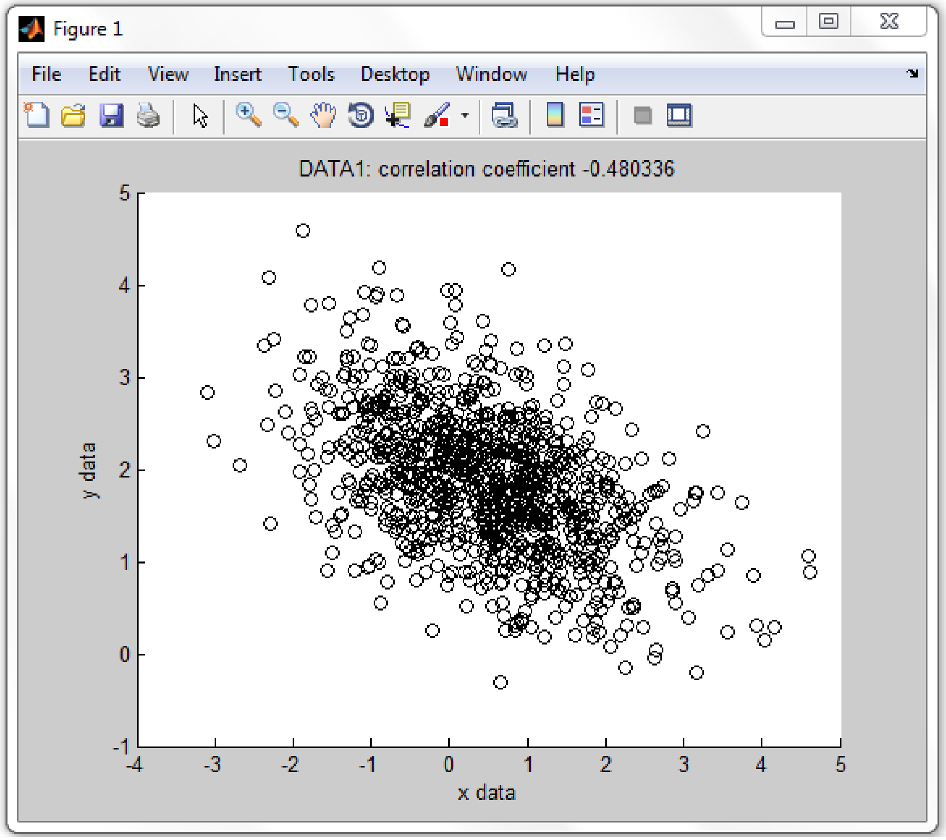
Scatter Plot

Function to draw scatter plot with labeling

```
function scatterplot(x,y,par) % black scatter plot (v1a/scatterplot.m)
%
% SCATTERPLOT   Draw a black scatter plot: scatterplot(x,y,par)
%
   scatter(x,y,'k');              % black scatter plot
   c = corrcoef(x,y);             % correlation coefficients

   xlabel('x data');
   ylabel('y data');
   title(sprintf('%s: correlation coefficient %g',par.title,c(1,2)));
end
```

Testing Scatter Plot



How to test

```
>> scatterplot(x,y,par);
```

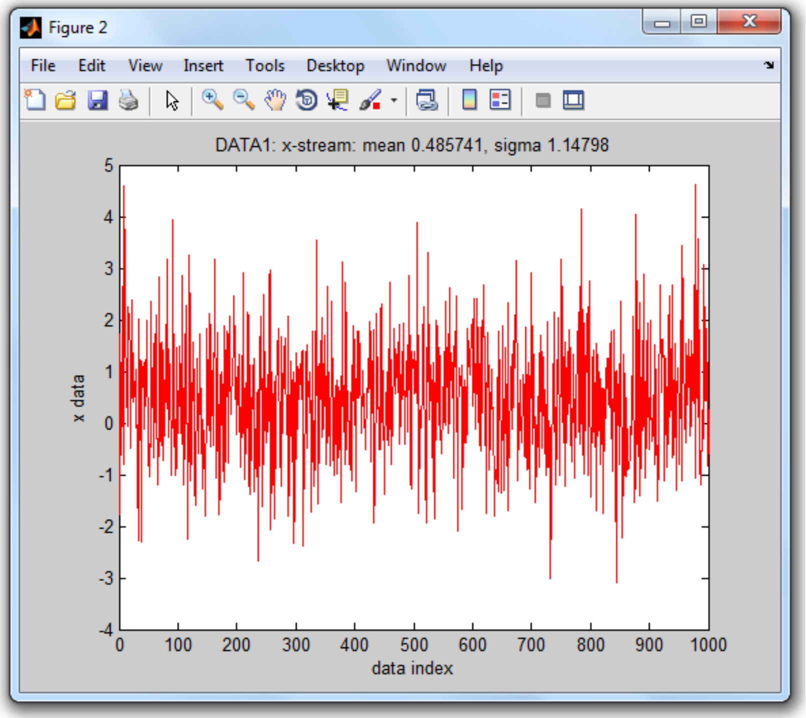
Stream Plot

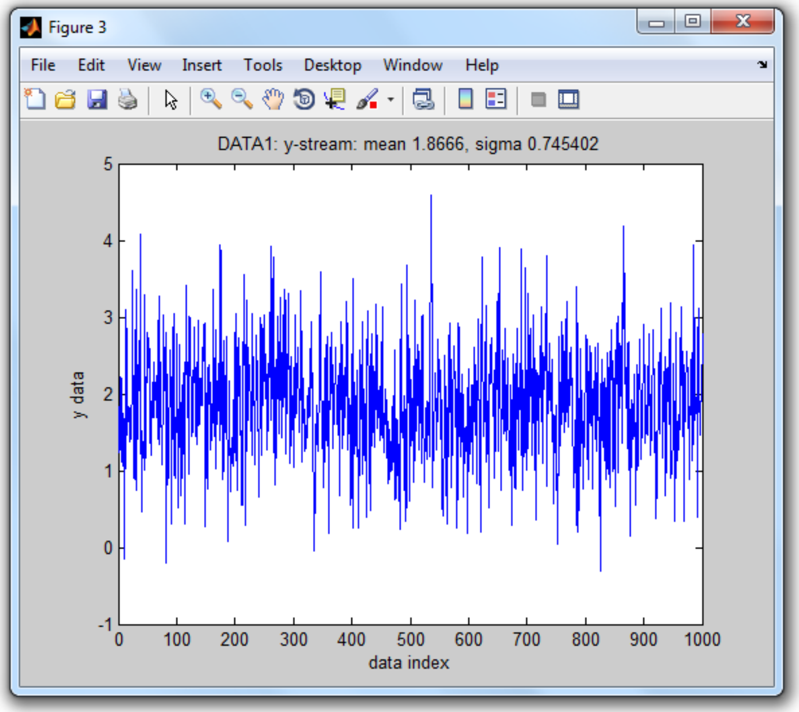
Function to draw x/y-stream with labeling

```
function streamplot(x,sym,col,par) % stream plot (v1a/streamplot.m)
%
% STREAMPLOT   Plot data stream: streamplot(x,'x','r')
%
   plot(x,col);                        % stream plot
   m = mean(x);                        % mean value
   s = std(x);                         % standard deviation (sigma)

   xlabel('data index');
   ylabel([sym,' data']);
   format = '%s: %s-stream: mean %g, sigma %g';
   text = sprintf(format,par.title,sym,m,s);
   title(text);
end
```

Testing Stream Plot





How to test

```
>> figure; streamplot(x,'x','r',par);
>> figure; streamplot(y,'y','b',par);
```

File Dialog

Function to open a file dialog

```
function path = filedialog   % select a log file (v1a/filedialog.m)
%
% FILEDIALOG   Dialog to select data log file: path = filedialog
%
   [file, dir] = uigetfile('*.log', 'Open .log file');
   if isequal(file,0)
      path = '';
   else
      path = [dir,file];
   end
end
```

How to test

```
>> path = filedialog
path =
.../play/log/data2.log
```

Our Analysis Function

Everything is ready now

- all building blocks are available now
- we can call our building blocks in our ANALYSIS function

```
function analysis                      % log data analysis (v1a/analysis.m)
   path = filedialog;                  % open file dialog, select log file
   if ~isempty(path)                   % if dialog not terminated with cancel
      [x,y,par] = read(path);          % read data (x,y) and parameters
      figure                           % open new figure
      scatterplot(x,y,par);            % draw black scatter plot
      figure                           % open new figure
      streamplot(x,'x','r',par);       % plot x-stream in red color
      figure                           % open new figure
      streamplot(y,'y','b',par);       % plot y-stream in blue color
   end
end
```

How to test

```
>> analysis
```

Awesome

Mission #2 completed

- MATLAB function (tool)
- Can select any log data file
- Draws scatter and stream plot
- Provides labels with statistical results

What More?

Question #1

- What about data encapsulation?

Question #2

- What about user interaction?

Data Encapsulation?

Log file

- data was initially encapsulated (parameters, data)

After reading from log file

- data & parameters are stored in different variables
- OK for a few variables/parameters
- loosing overview for many log data & variables/parameters

User Interaction?

Limited User Interaction

- the only user interaction is a file dialog
- there is no other interactiv control by user

Imagine

- imagine that instead of 3 graphs we have 20 different graphs and our tool pops-up all of these graphs ...

Wish

- menu driven analysis tool
- user interaction (GUI)

Procedural Programming

Our programming Style (so far)

- ... was procedural programming

Data Representation

- data is usually represented by variables or fields of a structure

Operations

- are typically represented as functions
- they take variables/structures as input args
- and return modified variables/structures as output args
- programs are sequences of function calls (controlled by some control structures)

Object Oriented Programming

Object Oriented Programming Style

- typically study a family of applications (say data analysis tools)
- Identify patterns (e.g. of data analysis tools)
- Determine what components & functionality is used repeatedly and in common

Base Class

- define a base class (super class) that the defines the common properties (data elements) and methods (function elements)

Derived Class

- used to implement a specific application
- uses specific (overloaded) properties & methods

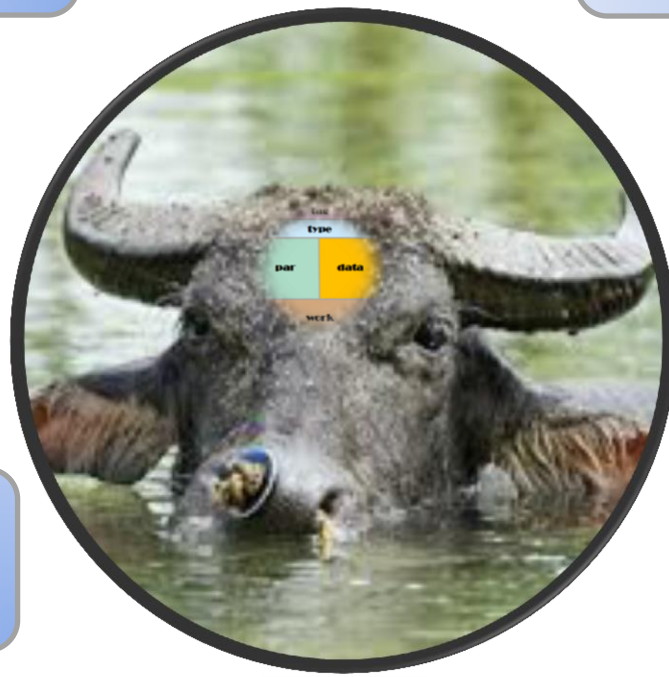
Outlook: Carabao Base Class

menu construction

object initialization

check & choice menus

utilize menu building block



provide menu building block

shell settings & object options

mass storage

object arguments

callback handling

shell's screen refreshing

local function dispatching

shell's menu rebuilding

Conclusions

- We used a procedural approach to implement a simple data analysis tool
- the tool reads data and parameters from a log file, performs calculations on the data (statistical quantities) and plots graphics
- the tool does not support user interaction

- We see the need for a menu driven tool which supports user interaction
- we got an idea how object oriented programming can support basic functionality

- we are looking forward to learn more about *Carabao* class