Matlab Lecture 6

Table of Contents

Version 0.1 by Markus Schellenberg, Winter Term 16/17

Clean up

```
clc, clear, close all
```

Strings, Characters, Cells, Arrays

Many of the examples in this chapter has been found on www.tutorialspoint.com. Some of them has been changed for this lecture. Type in:

```
s = 'hello'
whos s
```

```
s =

hello

  Name      Size            Bytes  Class    Attributes

  s         1x5                10  char
```

We can see that `s` is a string of the class char (character) it is arranged in a 1x5 matrix and it is 10 bytes long. Like any other format we can convert this string of character into something different. e.g. a string of ascii values

```
str_ascii = uint8(s)                % 8-bit ascii values
str_back_to_char= char(str_ascii)   % converts the ascii values back to char

str_16bit = uint16(s)               % 16-bit ascii values
str_back_to_char = char(str_16bit)
```

```
str_ascii =
```

```
  104   101   108   108   111


str_back_to_char =

hello


str_16bit =

    104    101    108    108    111


str_back_to_char =

hello
```

`uint` stands for unsigned integer and the number for the bit depth.

Rectangular Character Array

The strings that has been discussed so far are one-dimensional character arrays. To store more dimensional textual data in a program one need to create a rectangular character array.

Simplest way of creating a rectangular character array is by concatenating two or more one-dimensional character arrays, either vertically or horizontally as required.

You can combine strings vertically in either of the following ways:

- Using the MATLAB concatenation operator `[]` and separating each row with a semicolon (;). Please note that in this method each row must contain the same number of characters. For strings with different lengths, you should pad with space characters as needed.
- Using the char function. If the strings are of different lengths, char pads the shorter strings with trailing blanks so that each row has the same number of characters (source: www.tutorialspoint.com).

Example:

```
student_profile_bracket = ['Max Power          '; ...
                           '3rd Semester       '; ...
                           'Engineering Physics']

student_profile_char = char('Max Power', '3rd Semester',...
                            'Engineering Physics')


whos stud*      % information about every variable that begins with stud


        student_profile_bracket =

        Max Power
```

```
3rd Semester
Engineering Physics


student_profile_char =

Max Power
3rd Semester
Engineering Physics

  Name                          Size              Bytes  Class    Attributes

  student_profile_bracket       3x19                114  char
  student_profile_char          3x19                114  char
```

The two strings have the same class, size and length. However the char method is less error prone. If you use the bracket method and the length of each line must be the same, otherwise you will get the following error:

```
Error using vertcat

Dimensions of matrices being concatenated are not consistent.
```

You can combine strings horizontally in either of the following ways:

- Use the MATLAB concatenation operator, [] and separate the input strings with a comma or a space. This method preserves any trailing spaces in the input arrays.
- Use the string concatenation function, `strcat`. This method removes trailing spaces in the inputs.

Example

```
name =     'Max Power                          ';
semester = '3rd Semester                       ';
study =    'Engineering Physics                ';
```

Combination using brackets:

```
profile = [name ', ' semester ', ' study]


profile =

Max Power                          , 3rd Semester
```

Combination using `strcat`:

```
profile = strcat(name, ', ', semester, ', ', study)


profile =

Max Power,3rd Semester,Engineering Physics
```

The `strcat` command deletes the blank spaces inside strings and combines them afterwards.

Combining Strings into a Cell Array

It is clear that combining strings with different lengths could be a pain as all strings in the array has to be of the same length. We have used blank spaces at the end of strings to equalize their length.

However, a more efficient way to combine the strings is to convert the resulting array into a cell array.

MATLAB cell array can hold different sizes and types of data in an array. Cell arrays provide a more flexible way to store strings of varying length.

The `cellstr` function converts a character array into a cell array of strings. (source: www.tutorialspoint.com)

Example:

```
name =     'Max Power                              ';
semester = '3rd Semester                           ';
study =    'Engineering Physics                    ';

profile = char(name, semester, study);
profile = cellstr(profile);
disp(profile)
```

```
    'Max Power'
    '3rd Semester'
    'Engineering Physics'
```

Now you have easy access to the content of the character array:

```
profile(1)
profile(2)
profile(1:2)
```

```
ans =

    'Max Power'


ans =

    '3rd Semester'


ans =

    'Max Power'
    '3rd Semester'
```

Again, you can convert the cell array to a char array and back. Try the following and understand the output:

```
whos profile
profile_char = char(profile);    % converts the profile into a char array

profile_char
profile_char(1)
profile_char(1:10)
profile_char(1,1:10)
profile_char(1:2,1:10)
```

```
  Name          Size              Bytes  Class    Attributes

  profile       3x1                 416  cell


profile_char =

Max Power
3rd Semester
Engineering Physics


ans =

M


ans =

M3Earnxdg


ans =

Max Power


ans =

Max Power
3rd Semest
```

Backconversion to a cell array:

```
profile_cell = cellstr(profile_char);
profile(1)
```

```
ans =

    'Max Power'
```

You can enter char and cell arrays also directly:

```
S = char('A','rolling','stone','gathers','momentum.')   % char array
C = {'A';'rolling';'stone';'gathers';'momentum.'}        % cell array
```

```
S =

A
rolling
stone
gathers
momentum.


C =

    'A'
    'rolling'
    'stone'
    'gathers'
    'momentum.'
```

Structures

Structures are multidimensional MATLAB arrays with elements accessed by textual *field designators* . For example,

```
exam.name = 'Tom Thomson';
exam.score = 83;
exam.grade = 1.7;
```

creates a scalar structure with three fields.

Like everything else in the MATLAB environment, structures are arrays, so you can insert additional elements. In this case, each element of the array is a structure with several fields. The fields can be added one at a time,

```
exam(2).name = 'Mary Miller';
exam(2).score = 91;
exam(2).grade = 1.3;
```

or an entire element can be added with a single statement:

```
exam(3) = struct('name','Jack Johnson','score',70,'grade',2.7);
```

Now the structure is large enough that only a summary is printed.

```
exam.score
```

```
ans =

    83


ans =

    91


ans =

    70
```

```
%it is the same as typing

exam(1).score, exam(2).score, exam(3).score
```

```
ans =

    83


ans =

    91


ans =

    70
```

which is a comma-separated list. If you enclose the expression that generates such a list within square brackets, MATLAB stores each item from the list in an array. In this example, MATLAB creates a numeric row vector containing the score field of each element of structure array S:

```
scores = [exam.score] % Do not forget the sqare brackets, you need them.
avg_score = sum(scores)/length(scores)
```

```
scores =

    83    91    70


avg_score =

   81.3333
```

To create a character array from one of the text fields (name, for example), call the char function on the comma-separated list produced by S.name:

```
names = char(exam.name)
```

```
names =

Tom Thomson
Mary Miller
Jack Johnson
```

Similarly, you can create a cell array from the name fields by enclosing the list-generating expression within curly braces:

```
names = {exam.name}
```

```
names =

    'Tom Thomson'    'Mary Miller'    'Jack Johnson'
```

To assign the fields of each element of a structure array to separate variables outside of the structure, specify each output to the left of the equals sign, enclosing them all within square brackets:

```
[N1 N2 N3] = exam.name
```

```
N1 =

Tom Thomson


N2 =

Mary Miller


N3 =

Jack Johnson
```

Dynamic Field Names / Structure

The most common way to access the data in a structure is by specifying the name of the field that you want to reference. Another means of accessing structure data is to use dynamic field names. These names express the field as a variable expression that MATLAB evaluates at run time. The dot-parentheses syntax shown here, makes "expression" a dynamic field name: structName.(expression). Index into this field by using the standard MATLAB indexing syntax. For example, to evaluate expression into a field name and obtain the values of that field at columns 1 through 25 of row 7, use structName.(expression)(7,1:25)

Example:

First, initialize the structure that contains student testscores for a 4-week period:

```
testscores.Tom_Thomson.week = [95 89 76 82];

testscores.Mary_Miller.week = [50 70 83 75];
```

The f_Average_score function shown below computes an average test score, retrieving information from the testscores structure using dynamic field names:

```
function avg = f_Average_score(testscores, student_name)
F_AVERAGE_SCORE Averages the testscores of students for given weeks

Read out the total number of weeks stored in variable testscores
number_of_weeks = length(testscores.(student_name).week);

% Get the scores out of the dynamic field
for k = 1:number_of_weeks
scores(k) = testscores.(student_name).week(k);
end

% Average the scores and store the result in the output variable
avg = sum(scores)/number_of_weeks;

end
```

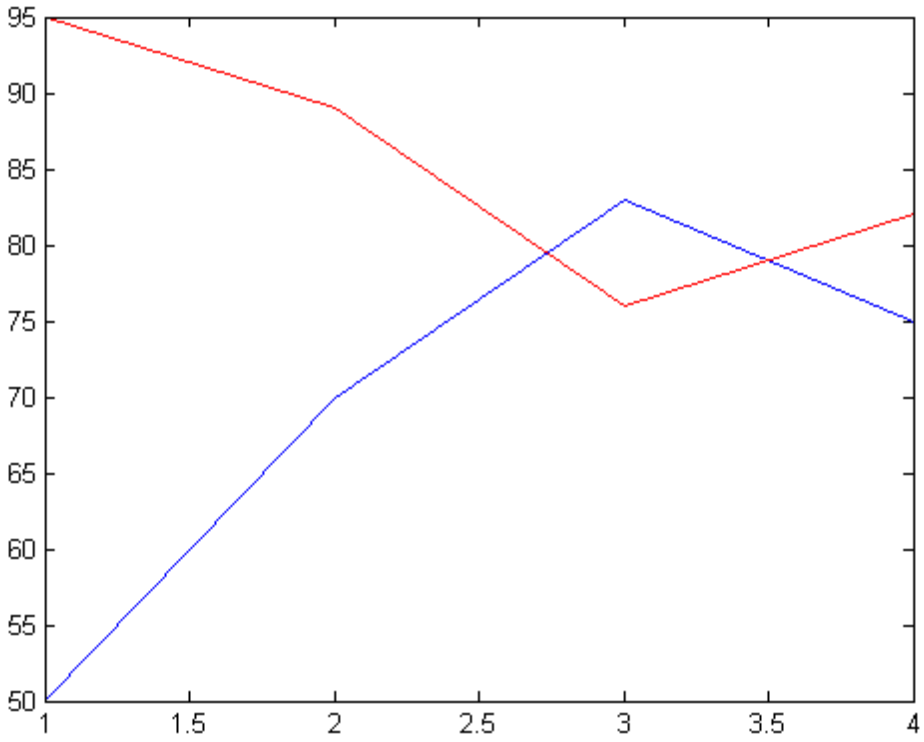
Now run f_Average_score, supplying the students name fields for the testscores structure at run time using dynamic field names:

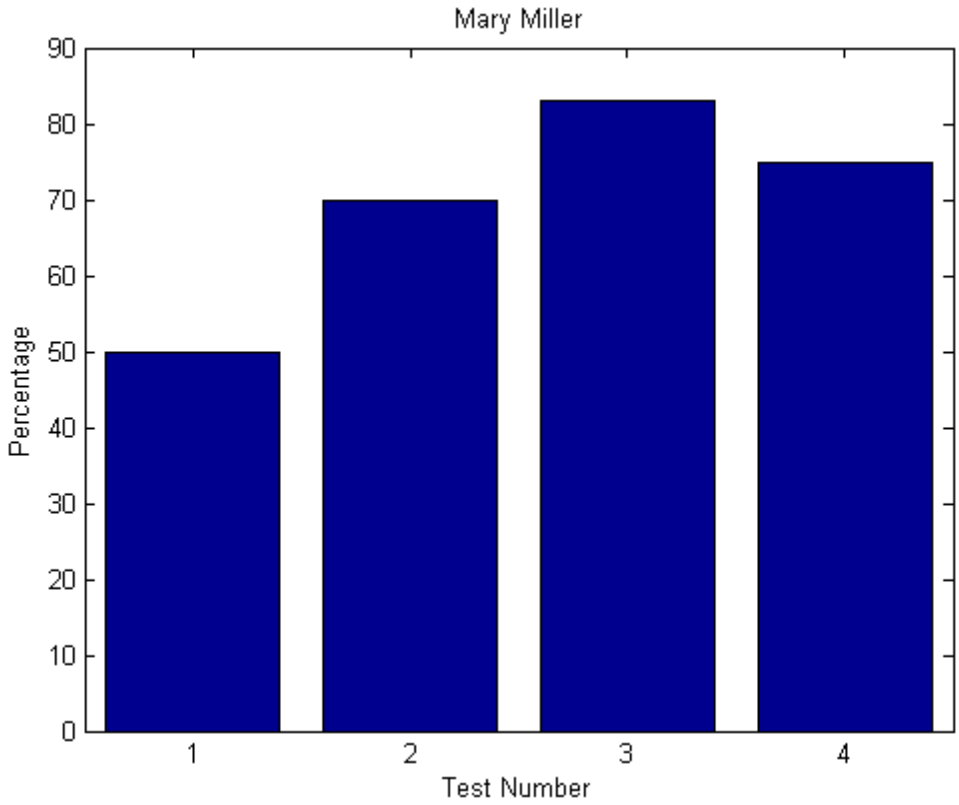
```
f_Average_score(testscores, 'Mary_Miller')
f_Average_score(testscores, 'Tom_Thomson')
```

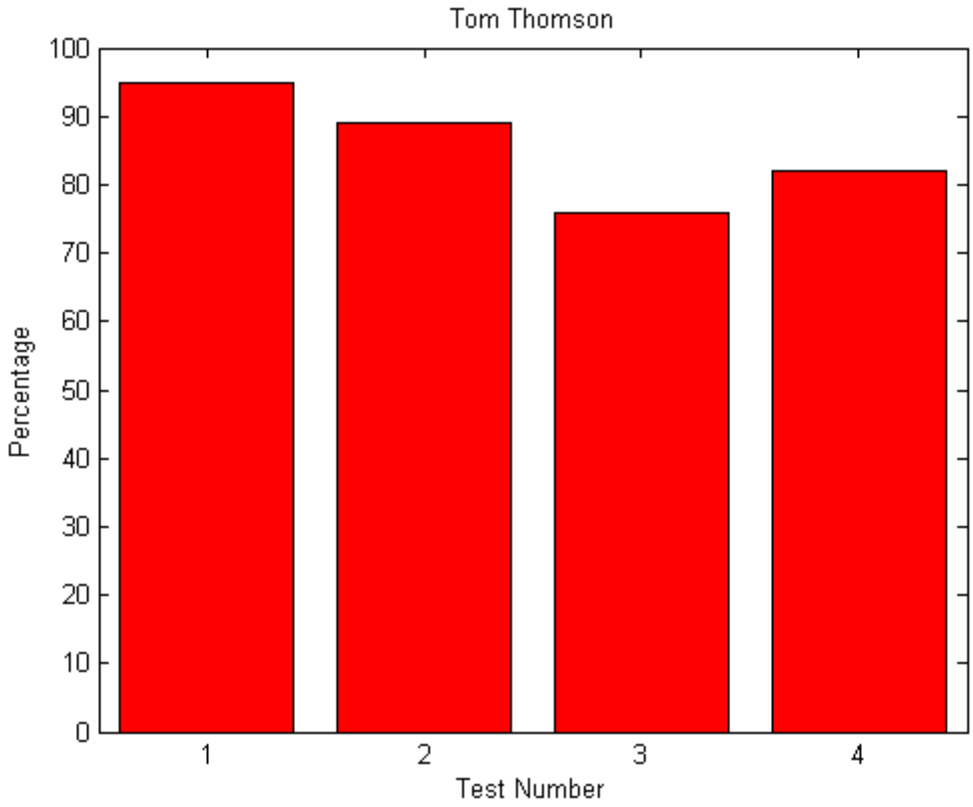
```
ans =

   69.5000


ans =

   85.5000
```

Plot the result

```
figure, plot(testscores.Mary_Miller.week), hold on
plot(testscores.Tom_Thomson.week, 'red'), hold off

figure, bar(testscores.Mary_Miller.week)
title('Mary Miller')
xlabel('Test Number')
ylabel('Percentage')
figure, bar(testscores.Tom_Thomson.week, 'red')
title('Tom Thomson')
xlabel('Test Number')
ylabel('Percentage')
```


95
90
85
80
75
70
65
60
55
50
1
1.5
2
2.5
3
3.5
4


Mary Miller
Percentage
90
80
70
60
50
40
30
20
10
0
1
2
3
4
Test Number



Structure Example

We want to analyse spectroscopic data. The data of each measurement is stored in it's own individual Textfile. The data is saved in the directory .../Spectra. We only want to load the *.txt files because only they contain the wanted spectras. Afterwards we want to plot them with the original filename (without the suffix) as a title.

With the command `uigetdir` (= User Interface to GET a DIRectory) you can run a little window to choose a directory with mouse and keyboard. Unfortunately the publish command would be stopped when you use it. A good solution would be to add the path by hand for the publish command and comment it later for more user convinience. You can also check for it by using the `exist` command. The `fullfile` command builds a full filename from parts.

```
%path = fullfile(pwd, '\Spectra'); % comment: uigetdir, uncomment: publish
path = fullfile('.\Spectra'); % comment: uigetdir, uncomment: publish

if ~exist('path', 'var') % if the variable path does not exist run uigetdir
    path = uigetdir
end
```

With the following method, you can scan through a choosen directory and store every file with the suffix *.txt inside the struct variable `f`

```
f = dir(fullfile(path, '*.txt'));    % Filename
number_of_spectra = length(f);       %
```

You have acces to the content of `f` by using a simple dot. `f.` + TAB will give you a list of structs content. Try the following and understand the output

```
whos f
disp(f)
f.name
f(1).name
```

```
  Name      Size             Bytes  Class     Attributes

  f         3x1               2243  struct

3x1 struct array with fields:

    name
    date
    bytes
    isdir
    datenum


ans =

yr1999ew.txt


ans =

yr2000ew.txt


ans =

yr2001ew.txt


ans =

yr1999ew.txt
```

Now we want to import every file that are listet in the struct variable `f` by using the funktion `importdata`. There are many different posibilites to do the same thing in MATLAB without this specific command.

```
for k = 1:number_of_spectra
    fullFileName = fullfile(path, f(k).name);
    Spectra(:,:,k)= importdata(fullFileName);
end
```

The spectral data is now stored in the variable Spectra. To display the first ten wavelengths and the corresponding spectrometer counts of the first spectrum just type in:

```
Spectra(1:10,:,1)
```

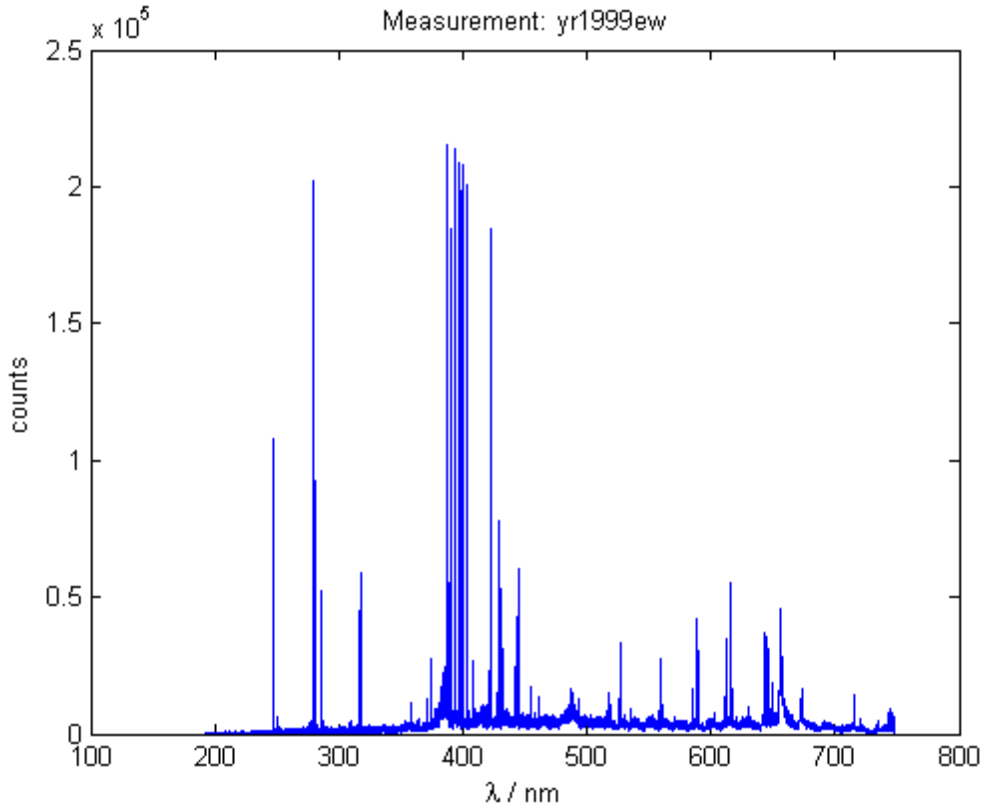
```
ans =

  192.3024   -9.0000
  192.3094   -1.0000
  192.3164   24.0000
  192.3233   35.0000
  192.3303    6.0000
  192.3373   -4.0000
  192.3442    6.0000
  192.3512   20.0000
  192.3582    4.0000
  192.3651    6.0000
```

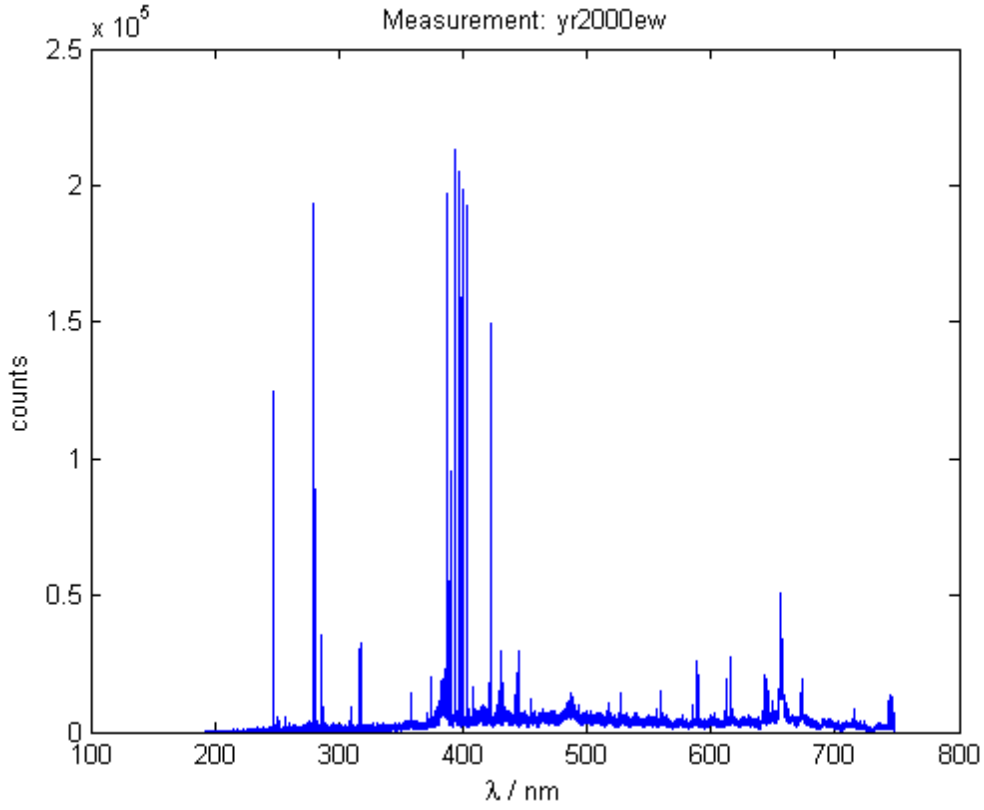
The original data may contain negative values, which should be impossible, because there is no thing like a negative count. Why this is normal for a measurement is explained during the lecture. Check for negative values

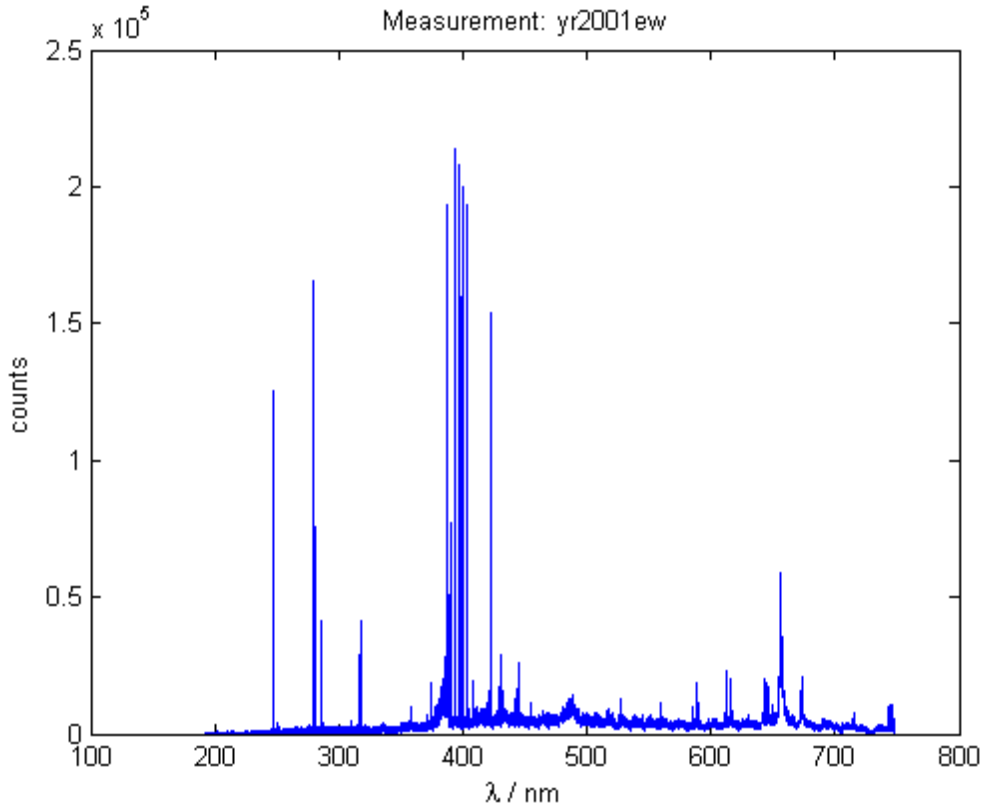
```
negative_values = find(Spectra < 0);
```

If there are negative values then shift the whole spectrum upwards in order to have the biggest negative value set to zero. You will get the biggest negative value by calculating `min(min(min(Spectra(:,2,k))))`

```
if isempty(negative_values) == 0
    for k = 1:number_of_spectra
        Spectra(:,2,k) = Spectra(:,2,k) - min(min(min(Spectra(:,2,k))));
        % it is minus, because the value is negative
    end
end

% plot the spectras
for k = 1:number_of_spectra
    figure
    plot(Spectra(:,1,k), Spectra(:,2,k))
    xlabel('\lambda / nm')
    ylabel('counts')
    title(['Measurement: ', f(k).name(1:(end-4))])
end
```


Measurement: yr1999ew
x 10^5
2.5
2
1.5
1
0.5
0
counts
100
200
300
400
500
600
700
800
λ / nm

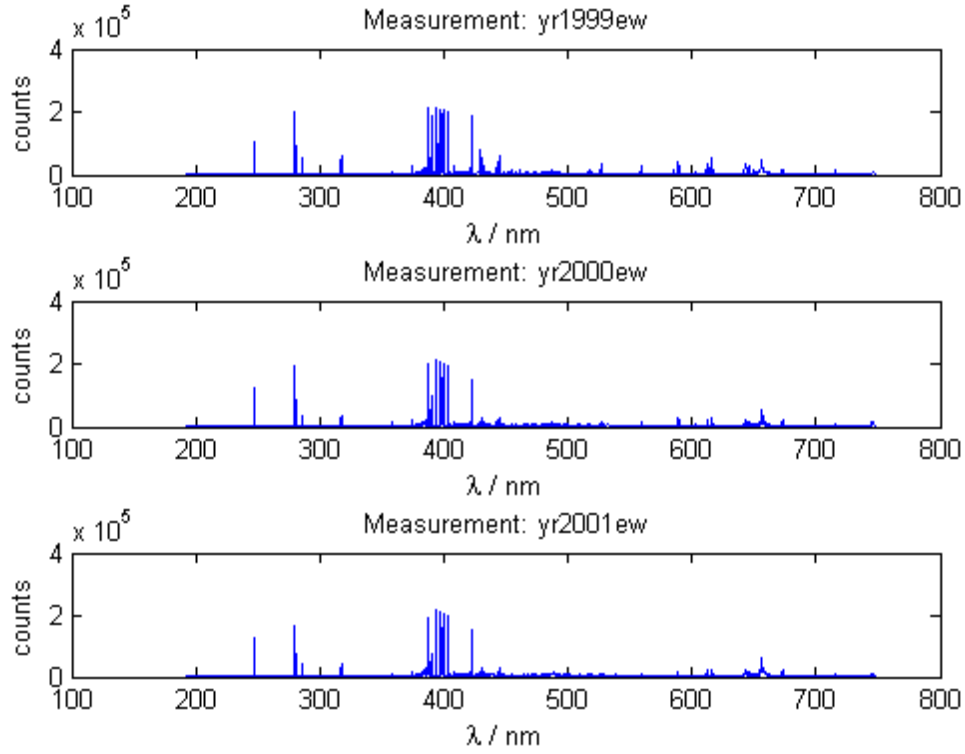

Measurement: yr2000ew
x 10^5
2.5
2
1.5
1
0.5
0
counts
100
200
300
400
500
600
700
800
λ / nm



You will notice that it is actually quite easy to have access to the filenames that correspond to each individual dataset. Since every information that was handed over from the `dir` command to the variable `f`. `f(k).name` gives you the name for each position `k` and `(end-4)` cuts off the last four elements of `f(k).name`, in this case '.txt'.

You can also use the the `subplot` command to display the spectras. Make sure, that you do not have an huge number of datasets in your choosen directory, otherwise you might crash MATLAB by reaching the limit of the internal memory.

```
figure
for k = 1:number_of_spectra
    subplot(3,1,k)
    plot(Spectra(:,1,k), Spectra(:,2,k))
    xlabel('\lambda / nm')
    ylabel('counts')
    title(['Measurement: ', f(k).name(1:(end-4))])
end
```



Published with MATLAB® R2013a