

130207/Thomas Munther  
Halmstad University  
School of Information Science, Computer and Electrical Engineering

## Laboratory 3

**Length about 2 hours with supervision.**

**Regardless if you completed the whole Lab-PM during these 2 hours you should try to find the time to work through this Lab-PM thoroughly afterward.**

**The last page contains hand-in assignments. You should try to solve them within a week to be able to follow the pace in the course.**

**Everything that follows after >> is written in the Matlab Command Window and indicate what you as a user should write. The % is intended as a comment to that specific command.**

The concept of variables is fundamental in all programming.

In matlab you introduce variables and assign values to them all by yourself.

When you assign a value to a variable, its previously assigned value will then be overwritten.

You can at any time find out the actual value of a variable simply by writing it in the command window ( without ; ).

The class is decided by the assigned value.

```
>> a= 1;      % the class is double and the storage size is 8 bytes. Open workspace !
```

In this laboratory we will investigate strings and conversions of strings among other things. A string is stored as a row vector, so that every element represents a character. Each character is accessible by the name of the vector and its index.

**Exercise 1:** Assign a variable to a string of characters !

```
>> words='This is a string'
```

Press enter to see the content of the variable words. Words has the size 1x16 elements.

The class is char ( character) and storage size 32 bytes.

```
>> words(2)    % gives the second element in the variable words.
```

```
ans = h
```

```
>> words(2)='t' % replaces the second element in the vector with t.
```

Eventually in many programs we need some form of interactivity. For instance to get input from the user. Matlab takes data from the matlab prompt.

```
>> x= input('Give me a name !: ','s'); % Matlab expects a number, string or a matrix  
% from the keyboard.
```

```
% The second argument says it
```

```
% should be a string.
```

```
>> disp('The name is: '), disp(x) % Gives text as output and the value of x.
```

We can also introduce the escape character sequences \n for the command input.

This can be written as:

```
>> x= input('Give me a name !: \n','s'); % Skips a row.
```

```
>> disp([x , ' That was a nice name !']) % Displays the name + string.
```

Read a matrix from the command window and then display it afterwards.

Use input and disp ! It is no difference from reading a number.

```
>> A=input('Give me a matrix\n'); disp(A)
```

We shall now look on another example where we have a interaction with the command window. Now we will use a trigonometric function as a string input. The m-file looks like below:

```
-----  
% The m-file was created 060201 by Thomas Munther  
% The user can choose between three trigonometric functions and have these plotted.  
% The command fplot have two input arguments. The first one is a string and the second  
% one is a vector that gives the plot interval.
```

```
fcn= input('Choose one of the functions: sin, cos or tan : ', 's')  
fplot(fcn,[0 40])  
-----
```

We stated earlier that a string of characters is a row vector and this is nothing more than a special case of a matrix, so we can create matrices consisting of strings, but note that these strings must have the same lengths. Otherwise each element in the matrix will differ and we can not perform the basic operations such as addition, subtraction and so on.

Now we present some typical swedish names as strings:

```
>> S1='Anders ' % 8 characters, two are blank.  
>> S2='Nils ' % 8 characters, four are blank.  
>> S3='Johanna ' % 8 characters, one is blank  
>> S4='Klara ' % 8 characters, three are blank.
```

```
>> A=[ S1 S2 S3 S4] % a matrix A, its elements are strings.
```

It is possible to assemble strings of characters in many different ways, but sometimes you would also like to have numbers inserted in the strings. This can be arranged by using conversions. We shall now try the command *num2str*. It means we make a coversion from a numerical value to a string.

```
>> x=num2str(123.4567,3) % converts the number to a string with three characters.
```

```
>> u=['The number is ', x, ', who would have guessed !'];  
disp(u)
```

There are lots of conversions in matlab. In the table below we show a few of them:

int2str(n)	Converts an integer n to a string
hex2num(hstr)	Converts hexadecimal number hstr to a float.
hex2dec(hstr)	Converts hexadecimal string to decimal integer.
dec2hex(n)	Converts decimal integer to hexadecimal string.
dec2base(n,base)	Converts decimal integer to base B string.
bin2dec(str)	Converts binary string to decimal integer.
dec2bin(n)	Converts decimal integer to a binary string.

mat2str(A,n)	Convert a 2-D matrix to a string in MATLAB syntax.
--------------	--

There are of course many other conversions, but we will stop here and try a few of them:

```
>> dec2hex(10)
```

```
>> dec2hex(12)
```

For those of you who have not heard about hexadecimal numbers: it simply means that the base is 16, but the representation uses the numbers 0,1,2...9, A,B.....F instead of the numbers 0, 1, 2.....9, 10,11,...15.

Also try to go back from a hexadecimal representation to decimal representation.

Do not forget that the argument is a string.

What becomes dec2base(4,3) ?

```
>> dec2base(4,3) % converts the number 4 (base 10) to a number with base 3.
```

There are logical functions for strings and functions to pick substrings.

We have functions to add blanks and subtract characters. We can also alter from small to capital letters and vice versa. We can also decide if we have letters in a string.

Comparison of strings are also possible and to decide if they are equal and many other things.

I choose to exemplify the following:

Blanks(n)	Gives a string with n blanks
deblank(str)	Subtracts all blanks at the end of the string.
lower(str)	All letters are changed to small.
upper(str)	All letters are changed to capital.
ischar(str)	If string contains character => gives 1 in return, 0 otherwise.
isletter(str(i))	If element number i in the string is a letter=> gives 1 in return.
isspace(str)	True for white space characters.
strcmp(str1,str2)	returns 1 if strings S1 and S2 are the same and 0 otherwise.
strcmpi(str1,str2)	returns 1 if strings S1 and S2 are the same except for case and 0 otherwise.
strfind(str1,str2)	returns the starting indices of any occurrences of the string str2 in the string str1.
findstr(str1,str2)	returns the starting indices of any occurrences of the shorter of the two strings in the longer.

Introduce a string str1='What's your name ? ' % Put some blanks after the letters.

and yet another string str2= 'What's your name ?' % without any blanks.

Try some of the commands in the previous table.

Especially the commands deblank,lower, upper, ischar, isletter and isspace with the argument str1.

```
>> isletter(str1(3))
```

```
>> isletter(str1(4))
```

Also try to compare strings. What will be result of the following:

```
>> strcmpi(deblank(str1),str2) % deblank(str1), creates a new string.
```

and finally to examine the last to commands in the table.

```
>> str3='name'
```

Try !

```
>> strfind(str1,str3)
```

```
>> findstr(str1,str3)
```

**Exercise 2:**

eval(str)	Performs the command in the string str.
eval(str,alt)	As above, but eval has an alternative alt if the first string str does not work.

In the m-file below there is an example of how to use the command eval. The m-file also contains an repetition consisting of a for-loop. It is executed for n=1,2,3 and 4 and then the program is finished. Execute the program to find out how it works !

```
% Magic(N) is a N-by-N matrix constructed from the integers 1 through N^2 with  
% equal row, diagonal and column sums.  
% Produces valid magic squares for all N> 0 except N=2.
```

```
for n=1:4  
    eval(['M' num2str(n) '=magic(n)'])  
end  
% Note that magic is a built-in function in Matlab.
```

-----  
**Please notice that the matlab command *num2str* is inside a string vector and evaluated with the Matlab command *eval*.**

We have in the two previous Matlab laboratories studied the concept function and used it to solve some simple problems. The function is called with or without several input arguments, but there are of course other possibilities to create functions. We will create an inline function that can be used in a similar way as the traditional function m-file.

g=inline(expr, arg1,arg2)	Creates an inline function g from the expression expr that can be called with two variables. Here arg1 and arg 2.
---------------------------	---

Let us illustrate this with an example:

```
>> g=inline('3*sin(x)+cos(y)', 'x', 'y')    % Definition of the function.  
                                           % The two variables are x and y.
```

```
g =
```

Inline function:

```
g(x,y) = 3*sin(x)+cos(y)
```

Now we can use this function simply by writing :

```
>> g(pi, 2*pi)    % x=pi and y=2*pi
```

```
ans=
```

```
1.0000
```

We can also evaluate the function g for different x-y pairs and get the function plotted.

```
>> x=0:10, y=0:10;    % Contains all integer values from 0 to 10.  
>> g(x,y)            % Results in a vector with 11 elements.
```

If we would like to see what we have calculated with a plot.

```
>> out=g(x,y, plot(out))
```

**Exercise 3:** Write an m-file that evaluates a function  $3*\sin(x)+2*\cos(x)+\tan(x)$  every tenth degree between 0 and 360. Plot the function versus x with suitable title and labels. Display the calculated values in a matrix according to: function values and x in two columns.

**Exercise 4:** We have earlier used matrices containing numerical elements or string of characters, but the elements must be of the same class, but matlab also permits other forms of matrices to exist with elements of different classes like cell arrays, but you can not perform the operations like multiplication, subtraction, addition and so on these. But remember that all rows and columns must have the same length as for an ordinary matrix. How can we make a cell array ?

>> D=cell(3,4) % gives an empty cell array, with 3 rows and 4 columns.

Assignment can be achieved by assigning each and every cell like:

>> D{1,1}=123.23

>> D{1,2}='kalle'

We can of course also make the whole cell array from the beginning.

>> A={'Charles', 'Anderson', 23, 'football'; 'Gwen', 'Nolan', 34, 'golf' }

A =

```
'Charles' 'Anderson' [23] 'football'
'Gwen'    'Nolan'    [34] 'golf'
```

Different applicable commands to cell arrays can be found in the table below:

cell(m,n)	Creates an empty m by n cell array.
celldisp(A)	Displays all elements in the cell array A.
cellplot(A)	Gives a graphical picture of the cell array A.
cell2struct(A,post,dim)	Creates a structure, if dim=1, the information is read columnwise to field 1 in the structure.

>> cellplot(A) % the result can be seen in figure 1.

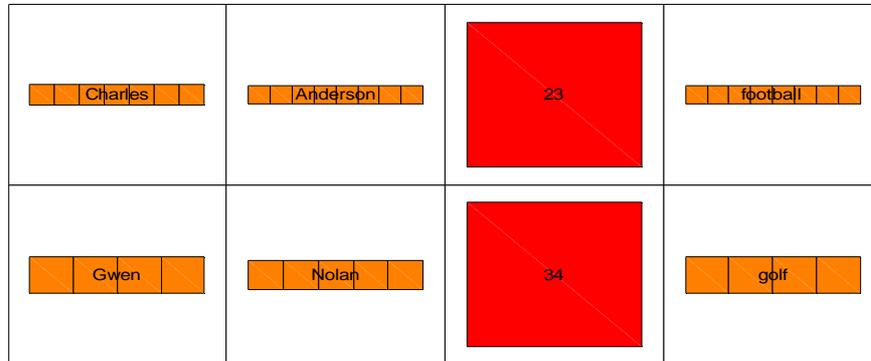


Figure 1

We will also exemplify how `cell2struct` can be used. If the concept structure is not known from programming you will also have a possibility later on in the course to get to know the concept. Structure is a variable containing different fields. Think of it as a method of organizing the variable. Each field can be accessed by point notation.

```
>> E=cell2struct(A,{'first_name','last_name','age','sport'},2)
% Notice that dimension is now 2.
```

```
E =
2x1 struct array with fields:
    first_name
    last_name
    age
    sport
```

Display the different fields within the structure:

```
E(1)                                E(2)
ans =                                ans=
    first_name: 'Charles'             first_name: 'Gwen'
    last_name: 'Anderson'            last_name: 'Nolan'
    age: 23                           age: 34
    sport: 'football'                 sport: 'golf'
```

**Home assignments for laboratory 3. Should be handed in within a week.**

The following must be done in order to pass the laboratory.

You must hand in a m-file for each problem. The m-file should begin with 2 comment lines where it is stated when this file is created and by whom. Mail the the m-files to me. I prefer if you send them in a compressed format like zip.files.

1. Create an inline function  $2*\sin(3*x) + 4*\cos(y) + 23*\tan(z)$ , that can be calculated simply by writing `h(x,y,z)` in the matlab command window !

2. Read 12 arbitrary numbers from the keyboard to a vector one at a time. The numbers should all be in the range 0-20. The m-file should ask for a new number until the vector has 12 elements. The input from the keyboard must come after a request. When the vector has 12 elements it should present them as a matrix displayed in the command window. There should also be a plot of the numbers. Present them with '\*' in the plot. They should be connected with a dotted line.

The second part of the m-file should have a graphical reading.

Mark the twelve points by a click with the mouse button in the plot ( use `ginput` !).

Then round these twelve points towards nearest integer. Put these twelve rounded values together with the twelve original values in a matrix ( different columns). Display the matrix in the command window and put a comment there as well.

**Hint:** Use a for-loop and `ginput`. Use help to find out how these work !

3) Create an m-file that asks for an arbitrary function that depends on variables x,y and z.. The user should enter the function .Your m-file should localize y and z in the function and then switch these against x. Plot the the new function versus x. Where  $x=0:0.1:20$ .

The user gives an arbitrary three-variable function and your m-file converts it to a single-variable function.

Add of course a suitable scaling of the axis and labels and titles !

Example:

Like if I as a user enters the function:  $\sin(x) + 3*y + 4*z$

Your m-file should replace "y" and "z" to x:  $\sin(x) + 3*x + 4*x$