Computer Algebra and Technical Computing (MTH1006)

B. Vorselaars

bvorselaars@lincoln.ac.uk

School of Mathematics and Physics, University of Lincoln

Today

- Deadline logbook is 8/12/2023. Final material to include for marking is next week's exercises. For the TCA it is better to also include the later material. See contents in session 1 for more information about the logbook. See also 'Logbook information' in 'Module Content'.
- Final in-class test for Matlab is on 12/12/2023
- Recap
- Scope
- Debugging
- File input/output

Recap

- pwd: present work directory. The current folder. demo

```
>> pwd
ans =
/Users/Bart/MyMatlabFolder
```

- ls: list the contents in a folder demo

```
>> ls
session1    session2    session3    session4
```

- cd: change directory

```
>> cd session1
>> pwd
ans =
/Users/Bart/MyMatlabFolder/session1
```

Recap

- `mkdir`: make directory

```
>> ls
session1    session2    session3    session4
>> mkdir session5
>> ls
session1    session2    session3    session4
     session5
```

- `path`: shows all folders that are searched:
- `addpath`: add a folder to the path

```
>> cd session4
>> addpath(pwd)
>> cd ..
>> myscript
This script displays one line
```

Download files online

See session 2 online for how to download `.m`, `.jpeg` and other files from online Matlab.

Recap

- Good programming practises: *incremental* programming, appropriately named variables and function names, and useful comments.
- Test if program is working using the `assert` function or `if` statements.

Recap

-
  ```
  function [x,y]=polar_to_Cartesian(r,theta)
  %Converts polar coordinates to Cartesian coordinates.
  assert(all(r(:)>=0),'The radial coordinate should be a non-negative number');
  x=r.*cos(theta);
  y=r.*sin(theta);

  >> [x,y]=polar_to_Cartesian(-1,pi/4)
  ??? Error using ==> polar_to_Cartesian at 5
  The radial coordinate should be a non-negative number
  ```

Scope I

The scope of a variable is the *workspace* where the variable is known. To see which variables are known within the current workspace use `who`

- Base workspace: all variables entered at the prompt and in a script. E.g., a and b

```
>> a=3; b=my_polynomial(a)
b =
    10

>> who
Your variables are:
a  b
```

Scope II

- Function workspace: all variables used in a function. E.g. x and y

```
function y=my_polynomial(x)
% Square the result and add 1
disp('start of function')
y=x.^2+1;
who
disp('end of function')
```

```
>> who
Your variables are:
a   b
>> x
??? Undefined function or variable 'x'.
```

Scope III

```
function y=my_polynomial(x)
% Square the result and add 1
disp('start of function')
y=x.^2+1;
who
disp('end of function')
```

```
>> c=my_polynomial(1);
start of function
Your variables are:
x  y
end of function
ans =
     2
>> who
Your variables are:
a     b     c
```

Meaning of debugging

Historically:

- A bug is an insect interfering with the electrical circuit.
- Debugging: getting rid of the insects so that the electrical circuit works properly.

Now:

- Debugging: getting rid of problems/errors in computer programs.

Debugging the code: primitive

How to debug a program?

- Primitive way: output variables in the code and add statement pause temporarily in the code.
- Benefit: using known programming constructs.
- Drawback: code needs modification before and after debugging.

Debugging the code: proper

Proper debugging

- Set *break-points* just before a line, by clicking on the line number. They are indicated by red dots or squares. demo
- Extra conditions are possible (right-mouse-click on dot/square)
- Matlab will stop and let you inspect variables. Debug prompt indicated by `K>>`:

```
K>> x
x =
     1
```

- Resuming execution line by line: press `F10` or type `dbstep` or click on *Step*
- Resuming program: `F5` or type `dbcont` or click on *Continue*.

Debugging example

How to determine the following sum using a `while` loop.

$$S = \sum_{m=1}^{2} \sum_{n=1}^{2} mn$$

```
n=0;
m=0;
s=0;
while n<2
  n=n+1;
  while m<2
     m=m+1;
     s=s+m*n
  end
end
```

Any bug? Let us put a breakpoint when the sum is updated demo

Debugging example

The `m=0;` should be within the first `while` loop.

Saving variables

Base workspace variables are deleted when exiting Matlab

- Saving variables: use `save`

```
>> x=3
x =
     3
>> save
Saving to: d:\matlab\matlab.mat
```

- Removing all variables: use `clear`

```
>> clear
>> x
??? Undefined function or variable 'x'.
```

- Reloading variables: use `load`

```
>> load
Loading from: matlab.mat
>> x
x =
     3
```

Input/output

Summary:

- `save`: save all variables from workspace to `.mat` file
- `load`: load all variables back to workspace

Only works for specially grafted binary Matlab `.mat` files. Often not compatible with other programs, such as Excel. What about text files?

Input/output text files

- `save(filename,'-ascii','variablename')`. Note: *variable name* given as a *string*!

```
>> my_variable=[1, 2, 3; 4, 5, 6];
>> save('data.txt','-ascii','my_variable')
>> type data.txt  % shows the content of
   a textfile
  1.00000e+00  2.00000e+00  3.00000e+00
  4.00000e+00  5.00000e+00  6.00000e+00
```

- `variablename=load(filename)`. Note: Matlab automatically finds out whether it is an ASCII or a `.mat` file

```
>> my_loaded_variable=load('data.txt')
my_loaded_variable =
     1     2     3
     4     5     6
```

Summary file types relevant to Matlab

- Text file. Example: using `diary session4.txt`; plain text file
- Script file. Example: `myscript.m`; `.m` file, runnable by Matlab
- Function file. Example: `myfunction.m`; `.m` file, called from command prompt, script or other function using `myfunction`
- Data file, Matlab format. Example: `mydata.mat`. `.mat` file, contains variables with content. This can be generated using the `save` command, and loaded again in memory using `load`.
- Data file, text format. Example: `mydata.txt`. `.txt` file, contains the contents of a single variable. This can be generated using the `save` command with the option `-ascii`, and loaded again in memory using `load`.
- Figure image. Example: `myfigure.jpg`. `.jpg` file containing an image of the figure.
- Figure file. Example: `myfigure.fig`. `.fig` file, containing all the data needed to plot a figure. The benefit over a standard image file is that it can still be edited.

Advanced file IO

To simply write numbers from a vector or matrix to a file:

- Store variable my_var to a file called outputdata.txt:

  ```
  save('outputdata.txt', '-ascii',
      'my_var'); % Note: variable name as a
      string!
  ```

- Load data from file inputdata.txt and store it in a variable called my_var:

  ```
  my_var=load('inputdata.txt')
  ```

In some situations this is too limited:

- Can only read/write scalars or vectors or matrices.
- Anything else is not possible, especially for the -ascii type.
- Can only read from the whole file at once, or write to the whole file at once. Not always wanted.

Advanced file IO

Advanced file input/output is typically carried out using three phases:

- Open the file
- Read/write or append to the file
- Close the file

Opening a file

First stage is to open a file.

- The operating system tries to retrieve the file, to see if it exists.
- Matlab command: `fopen`

```
fid = fopen('myfilename', 'permission
    string')
```

- `'myfilename'` the actual name of the file. Alternatively, one can use a variable with string containing the filename.
- `'permission string'`. This is a string denoting what one can do with the file.
  - `'r'`: file can be read (default)
  - `'w'`: file can be written to
  - `'a'`: appending to the file
- `fid`: a variable that will contain an identifier for the file (a reference). This will be used later to do more actions.

Opening a file

Example

- If a positive number is returned, then this implies that the file is successfully openened (for *r*eading in the following case)

```
>> fid=fopen('mydatafile.txt','r')
fid =
     5
```

- If the file identifier is -1, this indicates that something went wrong. One reason could be that the file does not exist.

```
>> fid=fopen('non-existing-file.txt','r')
fid =
     -1
```

Closing a file

Closing a file is particularly important if you write to a file. Other programs can then safely read the fully written file.

- Syntax:

  ```
  closeresult = fclose(fid);
  ```

- `fid`: the variable containing the file indentifier.
- `closeresult`: the variable containing the result of attempting to close the file. If it is 0, then closing is successful. Otherwise something went wrong

Closing a file

Example

-
  ```
  >> fid=fopen('mydatafile.txt','r')
  fid =
       5
  >> fclose(fid)
  ans =
       0
  ```

  The file successfully closed

  ```
  >> fclose(fid)
  Error using fclose
  Invalid file identifier.  Use fopen to
     generate a valid file identifier.
  ```

  The file was already closed. It can only be closed once after opening.

End of the file?

If we want to read from a file, it is convenient to know whether we are at the end of the file

- `feof`: test whether or not at the end of the file
- Syntax: `is_at_end = feof(fid);`
- Example

  ```
  >> fid=fopen('mydatafile.txt','r')
  >> feof(fid)
  ans =
       0    % implies end of file hasn't
          been reached
  ```

- If the end of the file has been reached:

  ```
  >> feof(fid)
  ans =
       1
  ```

This can be used to process files with an unknown length.

Reading line from a file

One can read a file line-by-line:

- `fgetl`: read one line
- Syntax:

  ```
  aline = fgetl(fid);
  ```

  Read a string from the file with identifier `fid` and store the result in the string variable `aline`

Reading line from a file

Example. The file `test.txt` contains:

```
a first line
a second line
```

-
  ```
  >> fid=fopen('test.txt','r')
  fid =
       6
  >> fgetl(fid)
  ans =
  a first line
  >> fgetl(fid)
  ans =
  a second line
  >> feof(fid)
  ans =
       1
  ```

String manipulation

Convenient string manipulation:

- `strtok`: split the string into the first and remaining words
- Syntax: `[token, remaining] = strtok(mystr)` Process a string variable `mystr` and put the first word into the variable `token`, and the rest in `remaining`
- Example

```
>> [first, rest]=strtok('This is a
   sentence')
first =
    'This'
rest =
    ' is a sentence'
>> [first2, rest2]=strtok(rest)
first2 =
    'is'
rest2 =
    ' a sentence'
```