EL2310 – Scientific Programming

Lecture 3: Scripts and Functions

Ramviyas Parasuraman (ramviyas@kth.se)

Royal Institute of Technology – KTH

Overview

Lecture 3: Scripts and Functions

- Wrap Up
- Output, Input and Commenting
- On Customized Help, Paths and Timing
- Scripts and Functions
- Basic Programming
- Making Movies
- Subfunctions
- Project and Group presentation

Last time

- Creating vectors and matrices:
  Ex: `linspace, eye, zeros, ones, diag, ...`
- Manipulating matrices:
  Ex: `', triu, tril, flipud, fliplr, rot90,...`
- Matrix operations:
  Ex: `min, max, sum, mean`
  `eig, svd, det, rank, trace, sqrtm, ...`
- Finding elements `find`

Last time

- Plotting:
  Ex: `plot, xlabel, ylabel, title, get/set, subplot`
- Loading data from file:
  `load('filename')`
- Pausing and drawing now:
  `pause, pause(n), drawnow`

Creating/Editing files

- MATLAB has a built-in text editor
- Create a new file or edit existing file with
  `edit <filename>`

Outputting text

- You often want to output text
- Useful to make user understand what is going on
- `disp('Some really nice text')`
- NOTE: Strings in MATLAB are in single quotes

Getting input from the user

- You can easily get input from user from keyboard
- `value = input('Some message that lets the user know what to input: ')`
- Input can be empty, scalar, vector, matrix, variable, etc. (parsed)
- Will repeat question until correct answer is given
- For string input do
  `s = input('Give us a string: ','s')`
- Then, input will not be parsed and just returned as string
- **To get graphical input?**

Adding comments

- Remember that people might want to read you code afterwards!
- You can (and should) add comments:
- Everything on the line after a % is interpreted as a comment
- `%{ multi line comments %}`

Good variable names

- Besides comments it is good to use meaningful variable names
- On the command line not so important as you are working with it actively
- You might have to understand a script/function after years or from someone else:
  - Not so good
    ```
    a=0:0.1:10;
    ```
  - Better
    ```
    speed=0:0.1:10;
    ```
  - Even better
    ```
    speed=0:0.1:10; % transl. speed of robot in m/s
    ```

Variable scope

- Each function has its own set of variables
  - (normally) functions can not access variables in base/main workspace
  - variable changes inside function do not affect base workspace
- This helps avoid name clashes (no need to track (all variable names in all functions called)
- These restrictions are called “scoping” and each variable has a “scope”
- Input arguments become local variables inside functions
  $\Rightarrow$ changes to input arguments are limited to function

Exercise 3.1

- Write scripts / functions that
  - Ask the reader to click in a window to enter some data
  - Display the points in the graph
  - Fit a line to them
  - Calculate the mean squared error between the points and the line

Learning by reading

- Remember that there are a lot of m-files in MATLAB
- You can look at all these to learn from
- Either find the file and look at it or do
  `type <function>`

Adding function description

- You can make sure that others can get useful “help” on your functions
- First comment line in file is used by `lookfor`
- Example:

```
function [k,m] =
calc_lineparameters(x,y)
% [k,m] = CALC_LINEPARAMETERS(x,y) fits data to
a line with least squares
% The resulting parameters describe the line
on the form
% y = k*x+m
```

Working directory

- Check current directory in file system with
  `pwd`
- Can change directory with
  `cd <direcory>`
- Can check where you are with
  `dir`

The path

- Similar to OS like windows and Unix/Linux there is a variable that tells where to look for files, the path variable
- Check what your current path is with
  `path`
- Add to the path
  `path(path, 'directory')`
  or
  `addpath <direcory>`
- You can also manipulate path with
  `pathtool`
- To check which m-file is used when executing a function:
  `which <function>`

What files are run?

- MATLAB cannot tell if an identifier is a variable or a function
- Resolved by picking first match from
  1. variable in current workspace
  2. built-in variable (like pi, i)
  3. built-in m-file
  4. m-file in current directory
  5. m-file in path

Timing your code

- When comparing algorithms execution time becomes important
- Start a stopwatch timer with: `tic`
- Stop stopwatch timer and get elapsed time with: `toc`
- An alternative is to look at spent CPU-time
  `cputime`
- Used as:

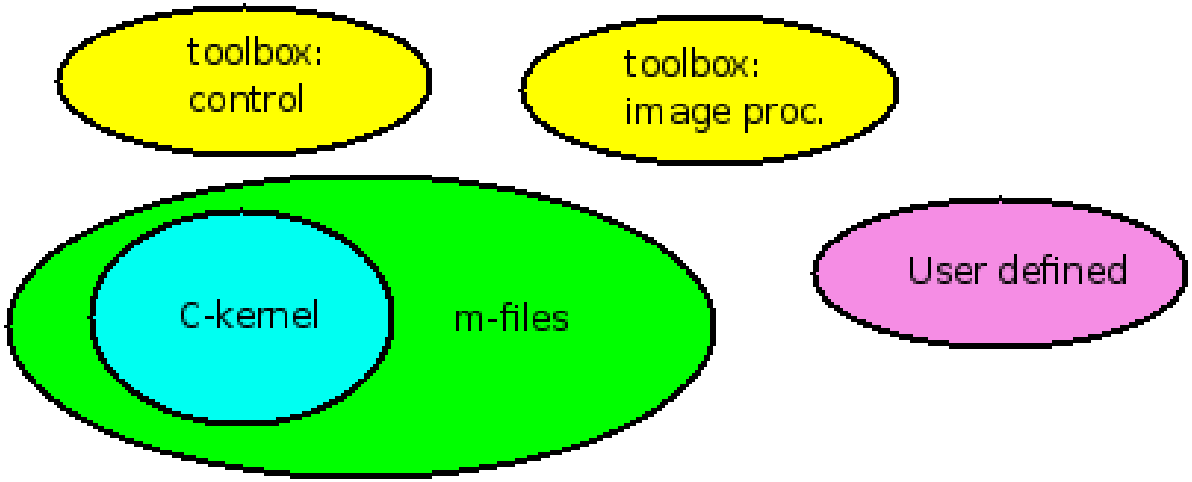
```
start_time = cputime;
...
disp('Spent CPU-time is') cputime-start_time;
```

Remember: What files are run

- Matlab cannot tell if an identifier is a variable or a function
- Resolved by picking first match from
  1. variable in current workspace
  2. built-in variable (like pi, i)
  3. built-in m-file
  4. m-file in current directory
  5. m-file in path

Scripts and functions

- Command windows ok for “calculator type” things
- Many commands $\Rightarrow$ execute a file with commands instead



m-files

- You put your code in so called m-files
- Two types of m-files
  - scripts
  - functions

Scripts

- Commands listed are executed as if written on command line
- Easy to reuse code and reproduce experiments
- A form of documentation of what you did

- Unexpected side effects?
- Ex: All variables cleared, changed, etc. in script also clear, changed in the workspace

Functions

- Used to "extend functionality" of MATLAB, with syntax:
  `[out1, out2] = function(in1, in2)`
- A function can have any number of input (`in1, in2`) and output (`out1, out2`) arguments:
- they can be scalar, vectors, matrices, strings, handles, functions, etc.

Scripts vs Functions

- Scripts:
  - Define experiment setups
  - Operate on base workspace variables
  - Solve very specific problem once
- Functions:
  - Easy to reuse functionality
  - Solve general problem
  - Arbitrary parameters
  - Use private variables (do not affect base workspace)

Script / Function Basics

- You run them by typing filename
- Can also run from the MATLAB editor
- A function should have the same filename as the function name
- Do not need to compile m-files (like for C/C++, JAVA, etc)
- Interpreted as they are run
- Downside: might not find error until run-time
  (editor will do syntax check)

Scripts

- Operates on base workspace variables
- Ex script:

```
% Calculate factorial
y = prod(1:n);
```

- What is needed for it to run?
- How will the workspace be changed?

Functions

- Header:
  ```
  function [out1, out2] = myfunction(in1, in2)
  ```
- Defines max number of input and output arguments but not minimum
- A variable used in the function must be passed in or be given value in some other way (see later)
- Remember that local variables exist only in local scope
- Ex function:
  ```
  function [y] = factorial(n)
  % y=factorial(n) - Calculates the factorial
  y = prod(1:n);
  ```
- `y` is a local variable here

Returning values from a function

- You can return values from a function at any time with: `return`
- Interrupts the execution and returns to where the function was invoked
- There is an implicit `return` at the end of the function

Branching with `if`

- Often want to control the flow depending on the value of some variable
- if-elseif-else construction

```
if <logical expression>
    <commands>
elseif <logical expression>
    <commands>
else
    <commands>
end
```

- Can have any number of commands in between
- Can have many `elseif`
- Do not forget the last `end`

Relations

==	equal to
<	less than
<=	less than or equal to
~=	not equal to
>	greater than
>=	greater than or equal to

Logical operators and functions

Logical Operators

`&` and

`|` or

`~` not

Short-circuit Operators

`A && B` - B not evaluated if A==0

`A || B` - B not evaluated if A==1

Functions

`xor(A,B)` exclusive or (exactly one of 2 is true)

`isempty(A)` check if argument is an empty array []

`any(A,dim)` check if any element is non-zero

`all(A,dim)` check if all elements are non-zero

Verify correct input

- What happens with our function factorial for argument -2
- Look at “real” factorial function
- Need to check that inputs are correct

Exercise 3.2

- Write function that returns a string with the season given the average temperature

Branching with `switch`

- Useful with many == expressions

```
switch <variable>
  case <value>
    <commands>
  case <value>
    <commands>
  otherwise
    <commands>
end
```

- Commands associated with first true `case` executed, or `otherwise` if no true `case`
- All commands until next `case`, `otherwise` or `end` are executed

`nargin` and `nargout`

- Can check how many input and output arguments were given
- `nargin`: number of inputs arguments
- `nargout`: number of output arguments
- Typically:
  - Let `nargin` and `nargout` define what is done
  - Check `nargin` and give default values if not given

Repetition with `for`

- `for`-loops allow you to loop over some value

```
for <loop variable> = <vector>
  <commands>
end
```

- You can define the vector in any of the many ways we saw before
- Ex: (cumulative sum)

```
sum = 0;
for v = values
  sum = sum + v;
end
```

Repetition with `while`

- ```
  while <condition>
    <commands>
  end
  ```
- `for`-loops good when you know which values to loop over in advance
- `while`-loops when repeating something until some criteria is fulfilled
- Ex: Repeat approximation until the error is small enough
  ```

Tips

- Avoid loops
- Use matrix operations
- Pre-allocate memory

Breaking/Skipping a repetition

- Sometimes you want to break out of a repetition
- Use `break` command
- Will continue after the end statement of the `for/while` loop

- Sometimes you want to start the next iteration
- Use `continue` command
- Will go up to the `for/while` statement again

A word on rounding

- `round(x)`: round to the nearest integer
- `fix(x)`: round to nearest integer towards zero
- `floor(x)`: round to the nearest integer $\leq x$
- `ceil(x)`: round to the nearest integer $\geq x$

Task 3.1

- Investigate `nargin` and `nargout`
- What happens if not all inputs and outputs are used?

Task 3.2

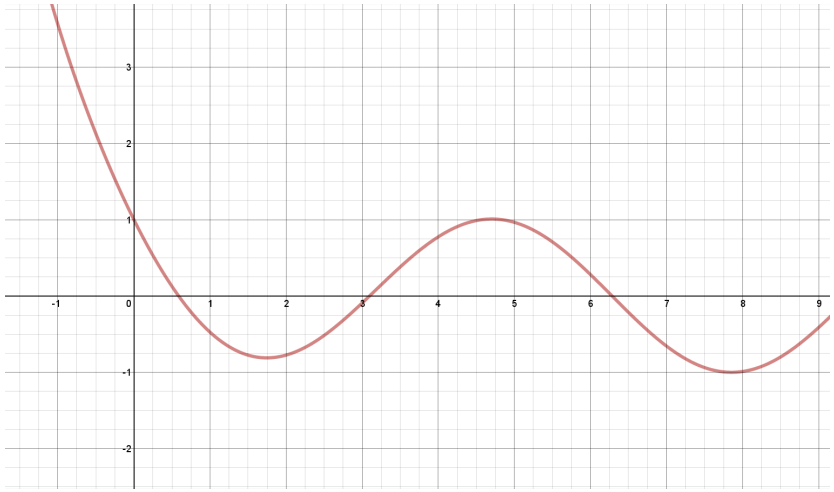
- Example of pre-allocation:

```
tic
<initialization>
n = 1000;
for k=1:n
  X(k,k) = 2*k;
end
toc
```

- Investigate with initializations
  1. `X=[]`
  2. `X=zeros(n,n)`
  3. can you make it faster?

Task 3.3

- Write a function that finds a solution to: $f(x) = e^{-x} - sin(x) = 0$



- Newton's method: $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$
- Assume initial guess x_0 is given
- Iterate at most `maxit` time
- Stop if $|x_n - x_{n-1}| \leq tol$

Task 3.4

- Finding solutions to equations is another key problem (besides regression) you should know how to handle.
- Have a look in the Matlab documentation what other methods are already implemented and available with Matlab. Compare the performance with your implementation of Newton's method.
- What optimization methods exist in Matlab? Can you determine minima and maxima of $f(x)$ in some interval?

Making movies with MATLAB

- Check out
  - `frame=getframe(figure_handle)` Get a movie frame
  - `movie(frames)` Play movie frames
  - `movie2avi(frames)` Make a movie file
  - `avi=avifile(filename)` Create AVI file
  - `addframe(avi, frame)` Add a frame to the AVI file

Exercise 3.3 (let's see how)

- Make a movie for `surf(X,Y,Z)` with `Z=sin(X-x0)` for varying `x0`

Movie making example

```
[X,Y] = meshgrid(0:0.1:10,0:0.1:10);
n = 0;
for x0 = 0:0.1:10
  Z = sin(X-x0);
  surf(X,Y,Z)
  n = n + 1;
  F(n) = getframe(gcf);
  drawnow
end

movie2avi(F, 'sinmovie.avi','FPS',10)
```

Subfunctions

- Can have many functions in an m-file
- Only one function is the *primary function*
- Subfuctions begin with a new function header
- Subfunctions cannot be called from outside, only from other subfunctions and the primary function
- Only the primary function can be called from outside

Why subfunctions?

- Can make the code easier to read/write
- Can have everything in one file
- Encapsulation
- Remember that only primary function can be called from outside

Task 3.5

- Rewrite the Newton code so that the code to calculate f(x) and f'(x) are in a subfunction.

Next time

- Profiling and debugging
- Closing introduction to MATLAB

Matlab project

- The project instructions are now on the course webpage
- The deadline is Monday 12.09.2016, 11:59 pm
- Expected zipped file containing three .m files (*wifi_simulator.m*, *wififitting.m*, *ProgramMain.m*), ten .mat datasets, README.txt
- Add a brief report on organization and how the program works in the README file
- Comments are important!

Presentation 1: Principal Component Analysis

- PCA overview, applications and implementation in Matlab (compare with Matlab's own)
- Who? Inigo Alvarez, Meng Ba, Moaaj Mohamed Ali Bhana, Sima Brunner
- When? 8th Sep (Thursday)

Presentation 2: Overview on various toolboxes

- What? Present various toolboxes in Matlab. Pick a four and provide details and functionality. Demonstrate a few interesting examples and explain how it works.
- Who? Jiacheng Cai, Sourjya Chowdhury, Anthony Clerc, Hanna Dotevall
- When? 12th Sep (Monday)