EL2310 – Scientific Programming

Lecture 3: Scripts and Functions

Andrzej Pronobis
(pronobis@kth.se)

Royal Institute of Technology – KTH

Overview

Lecture 3: Scripts and Functions

- Wrap Up
- More on Plotting
- Scripts and Functions
- Output, Input and Commenting
- On Customized Help, Paths and Timing

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`
- Plotting:
  Ex: `plot, xlabel, ylabel, title, get/set` (handles)

Getting all columns or rows

- To get all rows in a matrix and for example the first column you can use
  `A(:,1)`
- Similarily to get all columns for the 3rd and 6th row you would do
  `A([3 6],:)`

A word on `diag`

- The command `diag` is used to create diagonal matrices
- Can also be used to extract the diagonal of a matrix

- What will diag(diag(A)) do?

Today

- More on plotting (3D)
- Scripts and functions
- More on loading files

Creating histograms

- Displaying histograms:
  `hist(v, b)`
  where `v` is vector with data and `b` is number of bins.
- If you want the histogram data use:
  `[n,x] = hist(v,b)`
  where `n` are frequency counts and `x` are bin locations.
- You can plot histogram data with `bar(x,n)`

Loading data

- We saw how you can load saved variables with
  `load <filename>`
- You can easily load data directly into MATLAB if the data is matrix-like, i.e. same number of columns for each row
- To load a file “filename.txt” do
  `load(’filename.txt’)`
- This will put the loaded matrix into a variable filename (the name of the file).
- Can also do
  `d = load(’filename.txt’);`

Task 1

- Load data from “gyrosignal.txt”
- Collected from a gyro while standing still
- Format: Each row contains `time` and `gyrosignal`
- The time is in seconds
- The gyrosignal is in rad/s (maybe biased)

- Task:
  1. Remove any bias
  2. Integrate the signal to verify the angle estimate

Modifying the axis

- MATLAB will automatically choose the axis range for you,
- but in some cases this is not what you want.
- Set using: `axis([x_min x_max y_min y_max])`
- Get current axis settings with: `a = axis;`
- Same x/y unit size `axis equal`
- Square figure with `axis square`
- Fit to figure `axis normal`
- You can “turn off” the axis with `axis off`

Saving/printing a figure

- You often want to save a figure
- This can be done from the figure menu or with `print` command.

- To create an eps file, select desired figure and do
  `print -deps <filename>` (black/white)
  `print -depsc <filename>` (color)
- For print options do `help print`

Getting input from a figure

- You can get information (coordinates) by clicking inside figure
- Use command

  ```
  xy = ginput
  ```

  (pressing ENTER terminates the command)

  or

  ```
  xy = ginput(n)
  ```

  (if you know beforehand how many data points)

`drawnow` and `pause`

- To force a figure to display its content now (flush event queue), use
  `drawnow`
- To pause execution and wait for ENTER in command window, use `pause`
- You can pause for *n* seconds with
  `pause(n)` (e.g. `pause(0.1)` to pause 0.1s)

Subplots

- Easy to put many plot in the same figure with `subplot(n,m,k)`
- Sets up for n by m plots in a figure and prepares to add plot k
- Example

```
subplot(2,1,1), plot(x1,y1)
subplot(2,1,2), plot(x2,y2)
```

3D plots

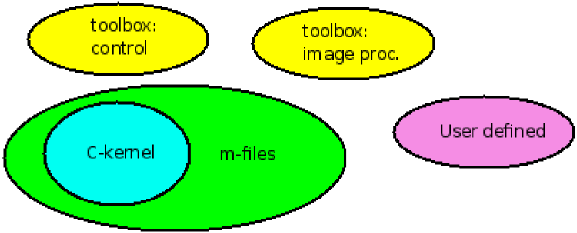
- Several functions to plot in 3D
- `plot3(x,y,z)`
- `mesh(X,Y,Z)`
- `surf(X,Y,Z)`
- `contour(X,Y,Z)`
- `mesh`, `surf` and `contour` plot the matrix Z against the values of X and Y.
- You can create values for X and Y with
  `[X,Y] = meshgrid(x,y);`
  where x and y are vectors and X and Y are matrices
- See also `colormap`

Task 2

- Display the function $z = 1 - x^2 + y^2$
- Use the interval $x, y \in [-1, 1]$

Scripts and functions

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



m-files

- You put your code in so called m-files
- Text file with file-ending .m
- Two types of m-files
  - scripts
  - functions

Scripts

- Commands listed are executed as if written on command line
- No need to type all commands over and over again
- Easy to 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: `function[out1, out2] = firstfunction(in1, in2)`
- The function normally matches filename
- 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)

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.
- Input will be 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

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

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
  ⇒ changes to input arguments are limited to function

Task 3

- 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;
```

Next time

- More on programming in MATLAB