EL2310 – Scientific Programming

Lecture 4: Programming in Matlab

Andrzej Pronobis
(pronobis@kth.se)

Royal Institute of Technology – KTH

Overview

Lecture 4: Programming in Matlab

- Wrap Up
- More on Scripts and Functions
- Basic Programming

MatLab Tasks

- Lecture 2
- Lecture 3
- Lecture 4

Last time

- Loading data from file:
  `load('filename')`
- Graphical input and keyboard input:
  `ginput and input`
- Pausing and drawing now:
  `pause, pause(n), drawnow`
- Scripts and functions
  - scope
  - inputs and outputs
  - comments with %
- The path:
  `path, addpath, pathtool`

Today

- More on programming in matlab
  - flow control: selection and repetition
  - checking input and output arguments
- Tips for faster code
- Timing, debugging and profiling

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

A word on rounding

- `round`: round to the nearest integer
- `fix`: round to nearest integer towards zero
- `floor`: round to the nearest integer towards -inf
- `ceil`: round to the nearest integer towards +inf

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

- Lines in script executed as if on command line
- 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
- Current values of variables corresponding to returned values are used
- 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

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 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

Skip to next iteration

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

Next time

- Closing our MatLab introduction
- with some more on programming
- and making movies.

Task 2.1

- Generate a vector of normally distributed values
- Check mean and standad deviation
- Generate two sequences and check covariance

Task 2.2 + 2.3

- Load data from “linedata.mat”
- Try to fit line to the data
- Plot the data from the line
- Plot your line approximation

Task 3.1

- 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 find the angle estimate

Task 3.2

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

Task 3.3

- 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

Task 4.1

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

Task 4.2

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

Task 4.3

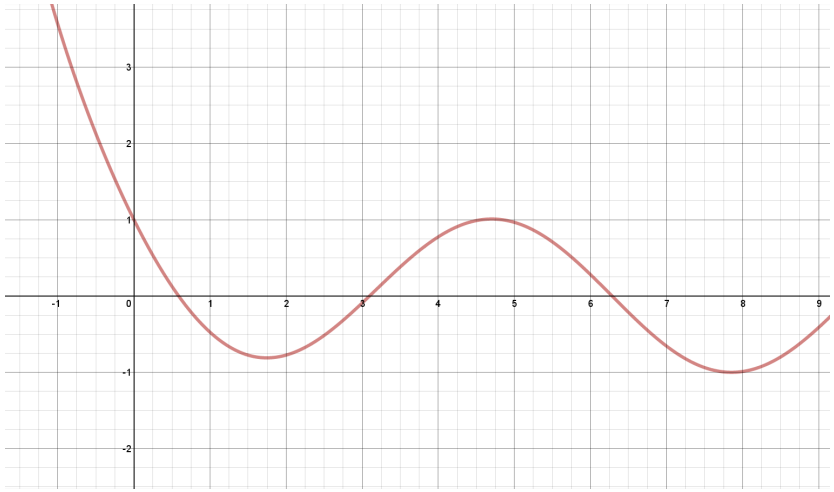
- Example of pre-allocation:

```
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. What if you built it directly in one command?

Task 4.4

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



- Newtons 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 $|x_n - x_{n-1}| \leq tol$