# EL2310 – Scientific Programming

## Lecture 4: Moving from Matlab to C


Ramviyas Parasuraman (ramviyas@kth.se)

Royal Institute of Technology – KTH

# Overview

## Lecture 4: Programming in Matlab

- Wrap Up
- More on Functions
- Profiling and Debugging

## Introduction to C programming

- Roots of C
- Getting started with C

# Wrap Up

- getting input from users: `input` and `ginput`
- giving info to users: `disp`
- `nargin` and `nargout`
- `if` and `switch` branching

# Wrap Up

- `for` and `while` loops
- `break` and `continue` in loops
- `path` and `pathtool`

# Wrap Up

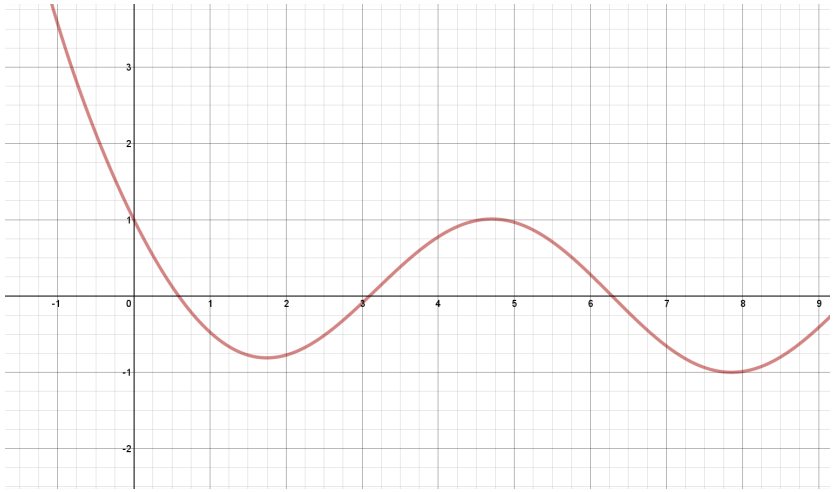
- general scripting
- functions and subfunctions

```
function [out,..]  = myfunction(in,..)
```

- making movie using `getframe` and `movie2avi`

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

# Passing functions as arguments

- In the Newton method task from last time we would have to write a new primary function for every new function we would like to solve
- Can be avoided by instead passing a function name as an argument

# syntax

- `B(@A)` - passing function handle

```
function A(x)
  <commands>
end

function B(fcn)
  fcn(<args>)
end
```

# Exercise 4.1

- Re-implement the Newton function (Task 4.4) with function as argument
- Now we can solve any $f(x) = 0$ assuming we define a function that returns evaluation of $f(x)$ and $f'(x)$

# Symbolic manipulation

- Matlab (with the right toolbox) can also do symbolic calculations
- Declare symbol with e.g. `syms t`
- Example

  `syms t` - Declare symbolic variable

  `f = t*sin(t)`

  `diff(f,'t')` - Differentiation

  `subs(f,'t',3)` - Substitution

# Profiling

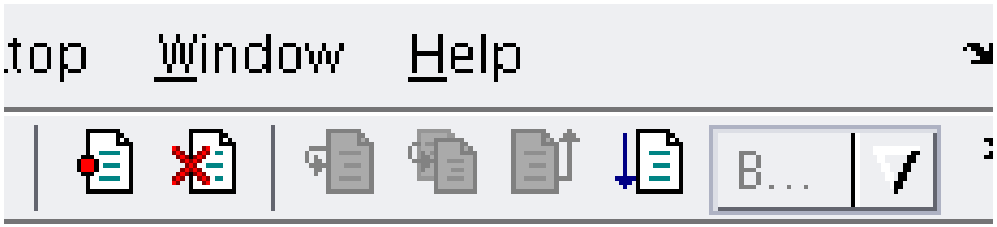
- Often useful to be able to tell what takes time in your program
- Can use `profile`
- `profile on` - Starts profiler
- `profile off` - Stops profiler
- `profile viewer` - Displays results
- For more info do `help profile`
- Use `fcn_busy` as a function in our Newton task and profile the code!

# Debugging

- Very rare that you get everything right immediately
- Debugging often accomplished by printing intermediate results
- Compare outputs with expected values

# Debugging continued

- The MATLAB editor has debugging support so that you can step through the code to see what happens
- You can set 'breakpoint(s)'
  - Program will stop at the breakpoint
  - which will allow you to check variables, etc.
- Step line by line
- Step in/out of functions
- ...



# Schedule

- Introduction to C - main part of this course
- Deadline to submit your MATLAB project solutions: Mon, September 12th, 11:59.
- Submit a README file to run your code.
- Include a brief report in your README.
- Don't work in groups!

# Announcements

- Refer course webpage for materials: www.csc.kth.se/ ramviyas/el2310.html
  - Online courses
  - Reference Books and manuals
  - Coding convention guides
  - Linux and Emacs
- Virtual machine for C/C++ projects is online
- Homework:
  - Install and run the virtual machine (or use Linux...)
  - Start Emacs
  - Type, compile and run a Hello-world program
  - Check out coding conventions!

# The roots of C

- First compiler developed by Dennis Ritchie at Bell Labs (1969-1973)
- Was based on two languages:
  - BCPL, written by Martin Richards at University of Cambridge
  - B, written by Ken Thompson at Bell Labs in 1970 for the first UNIX system
- Original C language was known as “K&R” C (Kernigan & Ritchie C) since the K&R book was the only language specification

# ANSI C

- American National Standards Institute (ANSI) formed a committee in 1983 and work completed in 1988
- Aim: to define “an unambiguous and machine-independent definition of the language C”
- Resulted in ANSI C standard
- Extensions to the standard: C99, C11

# The C language

- Initially developed for UNIX systems
- Most applications today were written in C
- "Systems programming language"
  - Constructs map efficiently to machine instructions
  - A replacement for the assembly language
- Many later languages borrow from C:
  - C++, C#, D, Go, Java, JavaScript, Perl, PHP, Python, Unix C Shell
- Considered low level language (in contrast to e.g. MATLAB)

# Types

**Types:**

- Classify type of data e.g. *integer*, *char*, *string*, etc.
- Machine data types: bits, words (32-bit/64-bit)
- Compiler maps language data types to machine data types

**Operators:**

- Interaction between objects of certain types (e.g. **+**,**-**)

# Types

- Typing systems differ between programming languages
- Strongly / Weakly typed
  - Unclear definition
  - Restrictions on interaction between data types
  - MATLAB "weakly" typed
  - C/C++ "strongly" typed
- Statically / Dynamically typed
  - Type checking during compile time or run time.

Strongly, statically typed languages are more likely to catch errors at compile time while weakly typed languages allow further flexibility.

# Tips to learn C

- Practice!
- Practice!
- Practice!
- Practice!
- Practice!
- A good idea: Define your own little project.

# Steps to a running program

- **Write**
- **Compile**
- **Link**
- **Execute**

From: http://www.physics.drexel.edu/courses/Comp_Phys/General/C_basics/compile.html

# Compiling the code

- Parsing of the statements for syntax
- Translation of the statements into machine language
- Setting up the addresses of all variables
- Optimization of the code (in modern compilers)

# Linking

- Assembles the routines produced during the compilation
- Resolves missing calls to either language-specific libraries or system-wide functions

# Optimization

- You can tell the compiler to optimize the code
- Better NOT to optimize until the program runs as expected
- Optimization changes the code internally for better efficiency
- However makes debugging much harder!
- Can typically specify different levels of optimization
- Optimization can in same cases change behavior of code

# Hello world

- The Hello world program
- Typically the first program written in all languages
- First one written in B

- Input: nothing
- Output: prints “Hello world” on the screen

# Hello world in C

```
#include <stdio.h>

main()
{
  printf("Hello world\n");
}
```

# The gcc compiler

- GNU (GNU's Not Unix) - Unix-like OS developed by the GNU Project
- GNU offers a freely available compiler called `gcc`
- usage example: `gcc hello.c`
- If the program is correct, will produce a binary file: `a.out`

# Running the program in Linux

- `./a.out`
- The prefix `./` instructs the system to run the program `a.out` in the current directory
- Just like in MATLAB there is a `PATH` variable that tells the system where to look for programs to run
- In Unix/Linux systems this `PATH` does NOT normally contain the current directory.

# Compiler arguments

- Compiler takes many arguments
  - `-o <output (object) filename>`
  - `-Wall` - enable all warnings
  - `-O, -O1, -O2, -O3` - optimization level
  - `-c <filename.c>` - only compile filename.c (not link)
  - `-lname` - link to library called libname
  - `-L<directory>` - tell the linker where to find libraries
- For now let us focus on `-o`

# Compiling a program cont'd

- To create executable named `hello` from `hello.c`
- `gcc -o hello hello.c`

# Next lecture

- Programming environment in C
- Basic datatypes
- Input and Output
- Branching and loops