First Steps in Scientific Programming

Patricio F. Ortiz

University of Sheffield, June 19, 2018
Overview

- Where to start
- The learning curve
- Elements of a computer
- The “terminal”, CL, UNIX tools
- Concepts of programming irrespective of language, code optimisation
- Thinking big and long term
- Miscellaneous issues
Where to start: the audience

- Young (inexperienced) scientists or science/engineering students
- Limited exposure to programming
- Eager to learn (or not)
- Pressured to produce ASAP
- “Land” in projects, 0-input on choices
- In their future, they will lead projects involving computers, programming decisions and acquisition of resources
The learning curve

• The environment, how to do things in a computer

• What “things” you can do

• How to make these things to happen

• How to organise information

• Forget spreadsheets, this is “the real word”

• The available tools, OS, languages, different architectures

• What to learn (???)
Main computer components

- Not just a magical box, everything below is finite

- Hardware:
  1. The processor(s): CPU, GPU
  2. The storage components
  3. Memory

- Software:
  A. Programming languages, compilers, interpreters
  B. Special libraries, text editors, apps, paths, ETC.
The terminal

- A psychological barrier for millennials. It needs demystifying
- Command line (!GUI), but real power if embraced
- UNIX: Microsoft ❤️ Linux. Future of the cloud, IOT
- Plenty of tools that save you from programming:
  - Man, ls, head, tail, find, awk, grep, ps, shells, sed, mv, cp, od
  - Remote work: ssh, scp, sftp, rsync
  - Versioning: SVN, git
  - Editing: vim, emacs, nano, textwrangler, ETC. Personal preference
  - Typesetting: tex and family, open-office, ETC.
Elements of programming

• Variables

• Objects

• Sub-programs / methods (variable scope)

• Flow control: conditionals, loops, exception

• Demo code, prototypes, production code

• Low data-volume v high data-volume

• Systematic Code testing
Code optimisation

- Code optimisers do exist, but they do not replace good programming practices and they may introduce undesired “features”

- Consider using look-up tables

- Some operations are really “expensive”, avoid them. (pow, exponentials, trigonometric, polynomials).

- Learn about existing libraries, avoid reinventing the wheel

- Manage memory well, whatever variable/object created occupies space. Beware of memory leaks.

- Be aware of overheads
Prepare for the long run

• Longevity of code

• Longevity of data

• Use adequate data characterisation/description for sharing

• Good practices to share data

• If data volume is significant, store in binary. Binary is the natural way for a computer to store information, human readable format is not.

• Whenever possible, add uncertainty information to your data. Somebody might want to run a model using your data.
Miscellaneous issues

- Things do fail, ergo, learn how to fix (debug) ASAP
- Even your computer can fail. Backup, better, user versioning
- Learn about “accelerators” (TAB key, use of make, ETC)
- Learn to profile your code (memory, exec-time, IO, etc.)
- When all else fails, ask for help, but write a clear description of the problem “It doesn’t work” is not enough.
- Learn how to alter someone else’s code effectively
- Plan your programs as if they were projects, learn about flowcharts, use pseudo coding, try thinking of different scenarios.
- Learn to handle time and time dependent situations.