theory

adam okulicz-kozaryn

adam.okulicz.kozaryn@gmail.com

this version: Thursday 4th December, 2025 17:25

outline

key scientific computing rules to get your started

and yet one more variation on general rules

Know Your Data!

- cant use it well if you dont know it well
  - (not just data; the field: theory, lit, method, etc)
  - thats how you beat IT folks (MS/PhD just in IT)
- again, invest a lot of time into your data
  - use data that you're passionate about
  - or that can make $ (now or in future career)
- think about it! don't be mindless!
  - ask questions, be investigative, be critical
- double check, cross check, give to others to check

outline

key scientific computing rules to get your started

and yet one more variation on general rules

Wilson put it well

- print out Box 1 from these 2 art
  - hang it at your office, home, and elsewhere
- `https://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.1001745`
- `https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1005510`
- and Know Your Data (2nd slide)
- and variations on these, and other general rules follow

simplicity, cleanliness, and organization!

- keep it as simple as possible
  - especially if overwhelmed or struggling
- say retain only 5vars and 25obs
  - much easier to understand such data
- simplicity transpancy clarity:
  - use fancy code: eg loops iff they simplify
- have chunks of code only once
- code it all from raw to final (replication principle)
- organize: sections, comments, and logical order (eg rewrite, move code around)

be fast/efficient

- the fancier the code, the more time/effort to write it
- don't do fancy things unless they save time in the long run
- it's all about managing complexity
- automate as much as you can
- simplify and be clear
- have general modules (sections or separate files)
  - that can be reused for different projects
- be lazy: don't reinvent the wheel–google often

outline

key scientific computing rules to get your started

and yet one more variation on general rules

more principles

- from 2 books about general programming (classics and free!)
  - ○ `http://catb.org/esr/writings/taoup/`
  - ○ `http://www.htdp.org/2003-09-26/Book/curriculum-Z-H-1.html`

clarity

- “design for transparency and discoverability”
- ○ write clean code [eg split 1 fn over many l for readability]
- ○ avoid fancy code
- ○ fancy code is buggier
- ○ clarity is better than cleverness
- eg:
- ○ group logical chunks together

modularity

- “write simple parts that are cleanly connected”
- “controlling complexity is the essence of computer programming”
  - debugging dominates development
- eg:
  - clear sections of one file
  - or many files instead of one file without sections

modularity

- code should be organized logically not chronologically
- ○ do free writing, but then reorganize
- ○ like with papers, code should be rewritten, eg:
- ○ no data management in data vis part
- ○ move rename, replace, etc earlier

composition

- “design programs to be connected to other programs”
- notebook or its sec will produce output for another notebook or sec
- eg: you clean up data in one file to make data ready for another one to vis
  - or just have one big file
- but the workflow needs to be logically organized

optimization (fancier, fewer lines)

- yes, but “get it working before optimizing” !
- eg:
  - first make mpl hist for one var, make it working
  - and then deploy it for 10 vars with a loop

extensibility

- “design for the future because it will be sooner than you think”

- ◦ you will reuse your code in the near future
- ◦ so write it clean
- ◦ have sections, etc
- ◦ use lots of comments
- ◦ reorganize, rewrite
- ◦ optimize

silence

- “when a program has nothing surprising to say, it should say nothing”
- drop unnecessary code
- ◦ if you think it may be useful in the future comment it out
- do not generate unnecessary output, do not lose your reader in unnecessary clutter
- ◦ if the output has nothing useful to say it should be dropped
- ◦ (or commented out)

automation (again)

- “rule of generation: avoid hand-hacking”
- because humans make mistakes and computers don’t, computers should replace humans wherever possible
- automate anything that you can
- but stay human, focus on fun creative part, eg vis
- dont automate everything; eg dont crank out bunch of vis mindlessly

reuse (copy-paste), don't reinvent the wheel

- if someone has already solved a problem once, reuse it!
- it is very unlikely you are doing something completely new
  - ○ again google what you are mapping and see images
- often all you need to do is to adjust somebody else's map

save time: reuse, don’t reinvent the wheel

- ask people for code:
    - your supervisor
    - journal article authors
    - your colleagues, friends, etc
- share your code
    - you may want to protect some parts of it
    - (critical, innovative research ideas, etc)
    - but share as much as possible
- acknowledge others’ work

defensive programing

- “people are dumb-make program bullet-proof”
- so important to interpret critically your maps
  - if sth looks weird/unlikely, there’s probably mistake
- check double check
- google your maps see images
- go to conferences, publish