LIÈGE université
GIGA

Good practices in scientific computing

GIGA Doctorate School

Christophe Phillips, Ir Ph.D.

The scientific process

Ideas → Hypotheses → Experiments → Data → Analysis → Comprehension → Dissemination

What could go wrong?

- bad idea

- ill-formulated
- no hypothesis
- (X-ploratory)

- flawed instrumentation
- poor experimental design
- inconsistent with hypothesis

- software bugs
- inappropriate use of methods
- incl. P-hacking

- misinterpretation
- extrapolation & HARKing
- post-diction & “story telling”

- publish anyway
- many details missing
- behind pay walls

Introduction

Science relies on (digital) data and their analysis.

$\rightarrow$ use & write scientific software!

Do we know

- what we want? $\rightarrow$ Mostly yes.
- how to calculate it? $\rightarrow$ We are working on it.
- how to build the “tool”? $\rightarrow$ Usually done “as it flows”...

  $\rightarrow$ Software development best practices!

Goals

Improve

- productivity of scientific programming and
- reliability of the resulting code

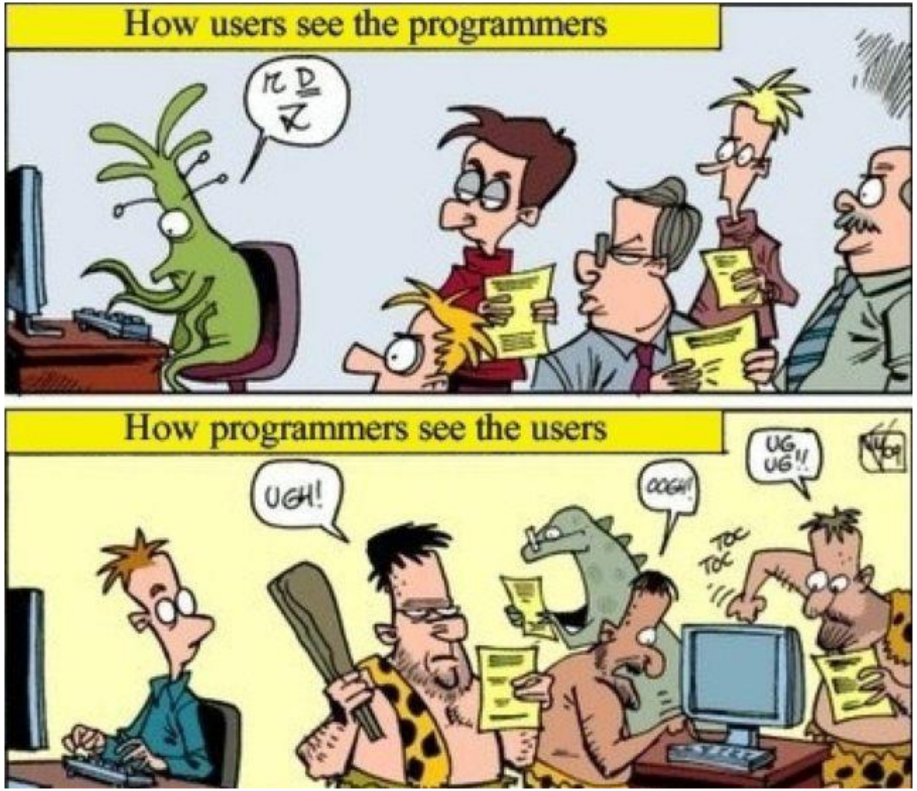
→ *speed up* result production

→ *boost* confidence in results

→ *ensure* results reproducibility

→ *increase* your scientific impact

Programmers vs. Users



Scientific computing best practices

1. Write programs for people, not computers
2. Let the computer do the work
3. Make incremental changes
4. Don’t repeat yourself or others
5. Plan for mistakes
6. Optimize software only after it works correctly
7. Document design and purpose, not mechanics
8. Collaborate

G. Wilson et al., “Best Practices for Scientific Computing”, PLOS Biology, 12:e1001745, 2014

Scientific computing best practices

1.	Write programs for people, not computers	A
2.	Let the computer do the work	B
3.	Make incremental changes	C
4.	Don’t repeat yourself or others	B
5.	Plan for mistakes	C
6.	Optimize software only after it works correctly	D
7.	Document design and purpose, not mechanics	A
8.	Collaborate	C

G. Wilson et al., “Best Practices for Scientific Computing”, PLOS Biology, 12:e1001745, 2014

Some wisdom

“A computer is like a mischievous genie.

It will give you exactly what you ask for,

but not always what you want.”

- Joe Sondow

Code & Document

1. Write programs for people
7. Document design and purpose, not mechanics

“Any code of your own that you haven't looked at for six or more months might as well have been written by someone else.”

- Eagleson's law

Real number more likely 3 weeks…

Code & Document

1. Write programs for people
7. Document design and purpose, not mechanics

- Make names consistent, distinctive, and meaningful.
- Make code style, input/output and formatting consistent
- Break programs up into “simple modules”
- Document interfaces and reasons, not implementations (ideally 40% of file content!).

Code & Document, more wisdom…

"Commenting your code is like cleaning your bathroom –
you never want to do it, but it really does create
a more pleasant experience for you and your guests."

- Ryan Campbell

Code & Automatize

2. Let the computer do the work
4. Don’t repeat yourself or others

- never change data manually!
- do not type commands more than once
- script code for a “re-do this” call
- turn scripts into functions, with options/flags/parameters
- modularize code rather than copy-pasting bits.

Code & Automatize

2. Let the computer do the work
4. Don’t repeat yourself or others

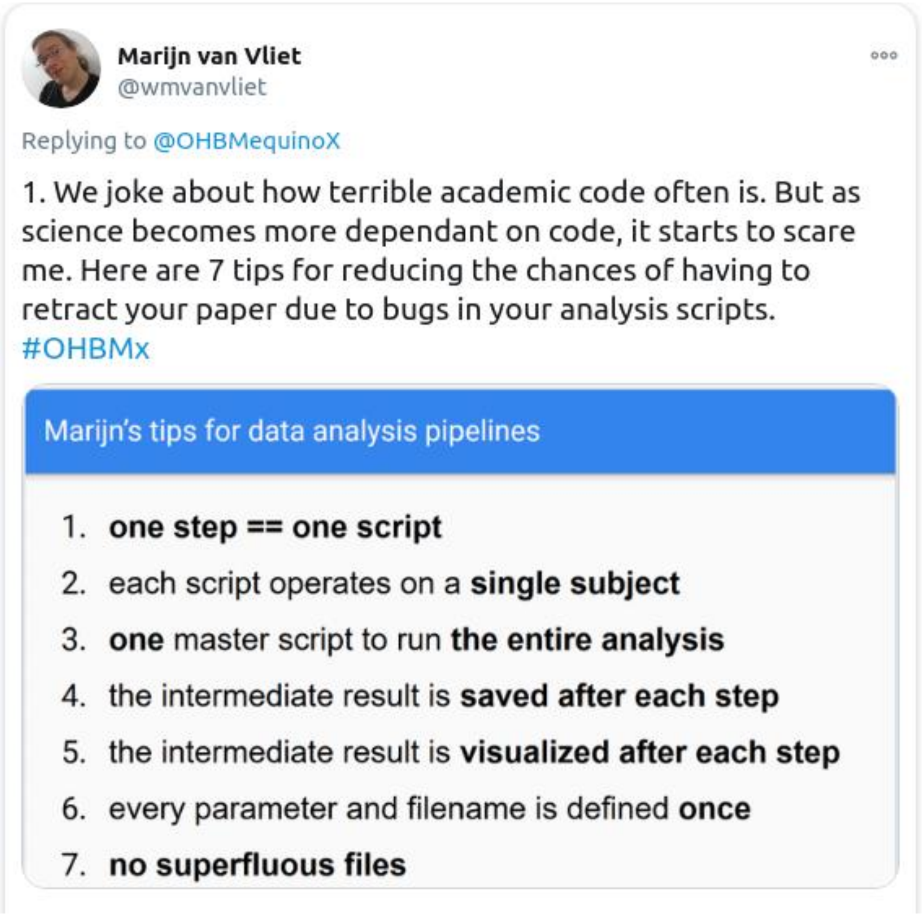
"DRY – Don’t Repeat Yourself
Every piece of knowledge must have a single, unambiguous, authoritative representation within a system."

- Andy Hunt & Dave Thomas

Code & Automatize

2. Let the computer do the work
4. Don’t repeat yourself or others

- never change data manually!
- do not type commands more than once
- script code for a “re-do this” call
- turn scripts into functions, with options/flags/parameters
- modularize code rather than copy-pasting bits.
- **re-use code instead of rewriting it.**



Marijn van Vliet
@wmvanvliet

Replying to @OHBMequinoX

1. We joke about how terrible academic code often is. But as science becomes more dependant on code, it starts to scare me. Here are 7 tips for reducing the chances of having to retract your paper due to bugs in your analysis scripts. #OHBMx

Marijn's tips for data analysis pipelines

1. **one step == one script**
2. each script operates on a **single subject**
3. **one** master script to run **the entire analysis**
4. the intermediate result is **saved after each step**
5. the intermediate result is **visualized after each step**
6. every parameter and filename is defined **once**
7. **no superfluous files**

https://twitter.com/wmvanvliet/status/1240907591791886337
https://arxiv.org/abs/1904.06163

Code & Automatize

2. Let the computer do the work
4. Don’t repeat yourself or others

“[Code reuse] saves a fair amount of coding, but much more important is consistency.”

- Kernighan and Plauger

Code & Develop

3. Make incremental changes
5. Plan for mistakes
8. Collaborate

"Every program has 2 purposes: The one for which it was written and another for which it wasn't."

- Alan J. Perlis

Code & Develop

3. Make incremental changes
5. Plan for mistakes
8. Collaborate

- **use a version control system**.
- put everything that has been created manually in version control.

  $\rightarrow$ keep track of changes: what, when, who & why!

Remember last week?

Code & Develop

3. Make incremental changes
5. Plan for mistakes
8. Collaborate

“If it hasn’t been tested, it doesn’t work.”

- Eric Mason

Code & Develop

3. Make incremental changes
5. Plan for mistakes
8. Collaborate

- automated testing of the code, in part or whole (unit, integration, regression tests)
- like manuscript writing, have colleagues review the code and/or write the code together

Code & Develop

Errors come in (at least) 2 forms:

- "code crash" $\rightarrow$ obvious & can be caught
- "wrong results" $\rightarrow$ difficult to spot!

"Debugging time increases as a square of the program's size."

- Chris Wenham

"Debugging is like being the detective in a crime movie where you are also the murderer."

- Filipe Fortes

Code & Optimization

6. Optimize software only *after* it works correctly.

- Use a profiler to identify bottlenecks.
- Write code in the highest-level language possible.

"Make it correct, make it clear, make it concise, make it fast. In that order."
– Wes Dyer

Scientific computing best practices

1. Write programs for people, not computers
2. Let the computer do the work
3. Make incremental changes
4. Don’t repeat yourself or others
5. Plan for mistakes
6. Optimize software only after it works correctly
7. Document design and purpose, not mechanics
8. Collaborate

G. Wilson et al., “Best Practices for Scientific Computing”, PLOS Biology, 12:e1001745, 2014

What about the data?

“I will sort and clean this data (code) right before we submit. Or for sure once this is published.”

= “I only clean my teeth right before my dentist appointment.”

Haste makes waste

Things we want to spend time on: **answering our scientific question.**

Things we don't want to spend time on:

- figure out how a dataset should be organized
- organize the dataset
- rewrite code because the data structure changed
- digging into our data to write our methods section

… and so, we rush !

"I have a grant proposal / grant report / PhD to finish!"

Issues & solutions

- "My data organization is as good as yours"
- Only the main author(s) know(s) where is what, what is useful (or not), etc. until…
- Not all the information gathered in one place

→ No error checking neither "memory"

→ Need specific batch/code to process the data

→ Not easy to re-use data or share with others

Issues & solutions

- “My data organization is as good as yours”
- Only the main author(s) know(s) where is what, what is useful (or not), etc. until…
- Not all the information gathered in one place

→ **Be complete**

→ **Be consistent**

→ **Be careful**

Ask yourself…

- Bus factor of a project:

  “If you were hit by a bus, can one of your lab-mates resume your research where you left off with less than a week delay?”

- Technical debt:

  “You’re taking a time-loan that you will have to pay back later. And you are not going to like the interest rate!”

Safety in 3 steps

- "Backup" your computer
- Archive your data (or ensure they are…)
- Version your code

$\rightarrow$ Be able to reproduce your results from scratch !

Safety in 3 steps

- “Backup” your computer $\rightarrow$ Dox
- Archive your data (or ensure they are…) $\rightarrow$ Mass-storage
- Version your code $\rightarrow$ GitLab

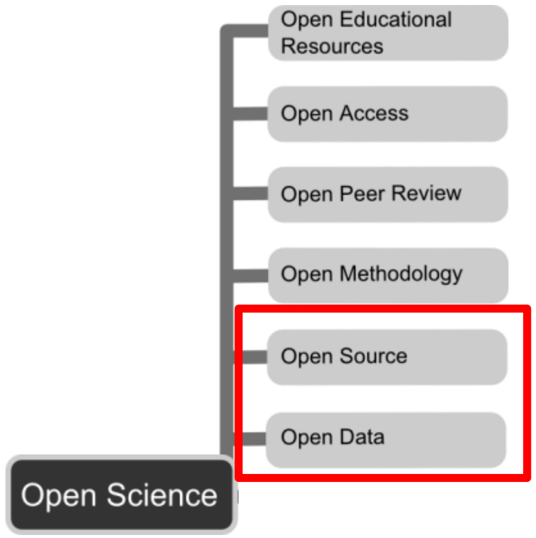
$\rightarrow$ Be able to reproduce your results from scratch !

Reproducibility & similar notions

		DATA	
		Same	Different
CODE	Same	Reproducible	Replicable
	Different	Robust	Generalisable

Open science

- Use open tools and format
- When you publish your results, do not be afraid to
  - share your **data**
  - share your **code**
  - share your **methodology**
  - share your **paper**
- Helping and being helped



Goals

Improve

- productivity of scientific programming and
- reliability of the resulting code

→ speed up result production

→ boost confidence in results

→ ensure results reproducibility

→ increase your scientific impact

Conclusion

- Looks scary… but just recommendations

  → adopt them incrementally

- Do not be afraid, try and follow these tips

  → for EVERY bit of code written, data bit, and analysis done.

- **Invest some time NOW**

  → gain in the long term!

References

- G. Wilson et al., “Best Practices for Scientific Computing”, PLOS Biology, 12:e1001745, 2014 https://doi.org/10.1371/journal.pbio.1001745
- https://en.wikipedia.org/wiki/Scrum_%28software_development%29
- https://en.wikipedia.org/wiki/Agile_software_development
- https://en.wikipedia.org/wiki/Version_control
- https://en.wikipedia.org/wiki/Open_science
- https://uclouvain.be/fr/universite-numerique/rdm

Finally

“Code is like humor.

When you have to explain it, it’s bad.”

– Cory House

“The first 90% of the code accounts for the first 90% of the development time. The remaining 10% of the code accounts for the other 90% of the development time.”

- Tom Cargill

LIÈGE université
GIGA

Thank you for your attention!

LIÈGE université
GIGA