# Practices for Scientific Computation

## Principles, Corollaries, and Simple Applications

Patrick Sanan

Institute of Geophysics, ETH Zurich
patrick.sanan@erdw.ethz.ch

https://github.com/psanan/practices_for_scientific_computation

## Setting

## Principles

"You will forget"
"Good practices are good science"
"Numerical codes are special"
"Less."

## Practices

The art of the README
Good login files
Asking good technical questions
Using version control

# Setting

# Motivation

- Scientists rely, increasingly, on computational tools, yet often feel frustrated by them
- People often repeat the same mistakes as they develop workflows based on short-term goals
- Later, I will tell you that you need to learn about version control (git). Why should you bother?

# Assumed Audience

- You're a scientist who works with numerical, computational tools (simulation, data processing and analysis, etc.), beyond the (exclusive) use of commercial packages
- You know what "the command line" is
- You're interested in using the available tools more efficiently

# Outline and Scope

- I will attempt to provide you with a few principles that I feel are often "learned the hard way"
- I will present a few concrete applications of these principles, focusing on common tasks often considered boring.

# Beating the "Greedy Algorithm"

- Smart people working quickly tend to make similar choices when presented with the available tools
- Often, these work well, but not always, because they make "locally optimal", not "globally optimal" choices
- The principles that follow are about adding rules to try to avoid getting trapped in "local minima".

# Principles

## “You will forget”

# Principle: "You will forget"

- After a given project leaves your immediate attention, you will remember few of the technical details
- The Greedy Algorithm:
  - Only consider the exact, current use case
  - Don't document the boring details
- Observations
  - Your own code, from a few months ago, looks like a stranger's code
  - Context switching has large overheads
  - "Getting it to run again" is often a very time-consuming and inspiration-sucking exercise.

# Corollaries: "You will forget"

- State is your enemy
- Encapsulation is essential
- Document, but avoid documentation which you have to remember to update

"Good practices are good science"

# Principle: "Good practices are Good Science"

- Treating computational tools as first-class parts of the scientific process is effective in increasing the quality of scientific output.
- The Greedy Algorithm
  - Treat computational methods as simply another tool in the toolbox, applying when needed
  - When presenting results, consider computational details to be uninteresting and trivial
- Observations
  - The "Third Pillar of Science" concept is a useful idea
  - It can be surprisingly difficult to reproduce computational results (including your own)
  - Applying scientific ethics to the use of computational tools can be effective

# Corollaries: "Good practices are Good Science"

- Challenge your code like you challenge ideas: try to prove it wrong. Get rid of things that don't work.
- The value of clarity, reproducibility and transparency extends to your code
- Document your work as you progress

## “Numerical codes are special”

# Principle: "Numerical codes are special"

- Numerical codes present distinct challenges compared to both general computation / computer science, and general scientific endeavors; these need to be dealt with directly.
- The Greedy Algorithm:
  - If it seems to work, good enough
- Observations
  - Floating-point arithmetic, particularly in parallel, requires special thought
  - Seemingly-simple code can rely on a great deal of mathematical or physical expertise
  - Numerical methods are often not robust to seemingly-minor changes in the problem setting

# Corollaries: "Numerical codes are special"

- "If you see something strange- there is a bug!"[1]
- It pays to learn the "Numerical facts of life"[2], for example the fundamental limits of finite-precision arithmetic.
- Numerical codes need (easy-to-run) tests
- "Don't believe it until you can run it"[3]

[1] Taras Gerya, "Numerical Geodynamic Modelling", Cambridge University Press, 2009
[2] Ed Beuler, "PETSc for Partial Differential Equations" (coming from SIAM Press)
[3] Matt Knepley (?)

## “Less.”

# Principle: "Less."

- Complexity and volume of information inevitably creates work
- The Greedy Algorithm
  - Copy-paste instead of encapsulating and re-using
  - Multiple copies of the same data
  - Leave commented-out code, just in case
- Observations
  - It's easier to write code than read it
  - Unforeseen problems and difficult bugs become much more prevalent, the more complicated a system is
  - Code "rots"
  - Hierarchy, encapsulation and data-hiding make big things small

# Corollaries: "Less."

- Treat code as a liability, as much as an asset
- Look out for ways to delete and encapsulate things
- Try to present work to other people in chunks of less than a "screen".

# Practices

How can we apply our principles?

1. "You will forget"
2. "Good practices are good science"
3. "Numerical codes are special"
4. "Less."

# The art of the README

# Practice: The art of the README

- The common `README` file in the root directory of most software projects is a great example of application of "You will forget", and "Less.".
- This is your one and only change to say something to people when they first come across your project.
- A quickstart is incredibly valuable
  - Get a user to a "working" state as quickly as possible
  - Show, don't explain
  - Have copy-paste-able commands (test them, exactly, on more than one system)
  - Some expected output is very helpful

# Good login files

# Practice: Good login files

- "You will forget":
  - Avoid state
  - Be very careful about adding things to `PATH`, `PYTHONPATH`, `LD_DYNAMIC_LIBRARY_PATH`, etc.
  - Explicit (define helper functions) is better than implicit (hard-coding environment variables)
- "Less.": Use a single set of login files on all systems, for all projects
- "Good practices are good science": login files are very often undocumented state for your experiments.

# Asking good technical questions

# Practice: Asking good technical questions

- "You will forget": You will very likely come back to this question and answer again, later. Write it clearly.
- "Less.": Try to describe the problem as briefly as you can, while still being accurate. This very often leads you to the answer, yourself ("rubber ducking").
- "Numerical code is special":
  - Reproducibility is essential
  - "Bisectability" is important - a similar, working state is invaluable
- "Good practices are good science":
  - State your assumptions to avoid the XY problem
  - Ask your question in public, so that others can learn

For more, see my post, "It doesn't work!", on the EGU blog.

# Using version control

# Practice: Using version control

Good use of version control helps satisfy all of our principles:

- "You will forget": keep track of past states, build repositories with documentation
- "Good practices are good science": Identify exact versions of software, distribute code, collaborate on code.
- "Numerical codes are special": Manage working states effectively, easily move to new machines
- "Less.": maintain a single, canonical history of a project.

For these and many other reasons, there is an entire git tutorial available at github.com/psanan/git_tutorial.