The Rough Road to Real-Life Reproducible Research

Christophe Pouzat, Arnaud Legrand, Konrad Hinsen

3 septembre 2018

Outline

Where are we?

M4-S0 : The Rough Road to Real-Life Reproducible Research

M4-S1 : Data Hell

M4-S2 : Software Hell

M4-S3 : Numerics Hell

M4-S4 : Conclusion

Reproducible Research Hell

Module 4. The Rough Road to Real-Life Reproducible Research

1. Data Hell
2. Software Hell
3. Numerics Hell
4. Conclusion

Where are we?

Two new problems

When we start to work on real data, we typically have to deal with two problems :

- the data are of diverse nature
- the data occupy a lot of memory

Non-homogeneous data

- The influenza-like illness data from module can easily be presented as a table (2 dimensional object)
- Often the table form must be abandoned because
  - the columns don't all have the same length
  - the data can be a time series and a set of images, etc.

Big data

- Text formats are not always appropriate for numbers
- Choice of a binary format because
  - Numbers occupy less memory
  - Numbers in text format must be converted to binary anyway for computation

Text format features we wish to keep : metadata

- Text permits storing the data and all the rest...
- $\Rightarrow$ add information about the data :
  - provenance
  - recording date
  - source
  - etc.
- This information about the data is what is called metadata
- They are vital for doing reproducible research

Text format features we wish to keep : endianness

- Text format is universal
- Binary formats depend on hardware architecture and operating system
- The four-bit sequence 1010 can be read as
  - 1x1 + 0x2 + 1x4 + 0x8 = 5, which is little-endianness
  - 1x8 + 0x4 + 1x2 + 0x1 = 10, which is big-endianness
- A binary storage for reproducible research much specify endianness

Binary formats for composite data allow storing metadata

Wanted : binary formats for

- working with big datasets of diverse nature
- storing metadata along with the data
- having endianness fixed once and for all

‘FITS‘ and ‘HDF5‘

- The Flexible Image Transport System (‘FITS‘), developed in 1981 and still regularly updated
- The Hierarchical Data Format (‘HDF‘), developed at the National Center for Supercomputing Applications, is at its fifth version, ‘HDF5‘

‘FITS‘

- ‘FITS‘ introduced and updates by the astrophysics community
- Format sufficiently general for use in different contexts

The anatomy of a ‘FITS‘ file

- One or mode segments : Header/Data Units (HDUs)
- A HDU is made up of :
  - a header (Header Unit) followed optionally by
  - the data (Data Unit)
- Header = key-value pair $\rightarrow$ metadata
- Data stored as binary tables (one to 999 dimensions) or as tables (text or binary)

Manipulation of ‘FITS‘ files

- The developers of the format offer a ‘C‘ library and associated programs that are easy to use
- ‘PyFITS‘ for Python users
- ‘FITSio‘ for R users

‘HDF5‘

- Hierarchical organization, resembles a filesystem tree
- Structuring element : a group (similar to a directory) contains one of more datasets
- Groups can be nested
- No structure imposed on metadata
- No structure imposed on data - they can be text

Manipulation of ‘HDF5‘ files

- More flexible format $\Rightarrow$ the ‘C‘ library is more complex thatn its ‘FITS‘ equivalent
- The library is distribued with ‘HDFView‘, a powerful tool for exploring and visualizing data
- ‘h5py‘ is a very complete ‘Python‘ interface
- Three ‘R‘ packages : ‘h5‘, ‘hdf5r‘ et ‘rhdf5‘

Archiving

Git (hub, lab, . . .) : not well suited for data storage

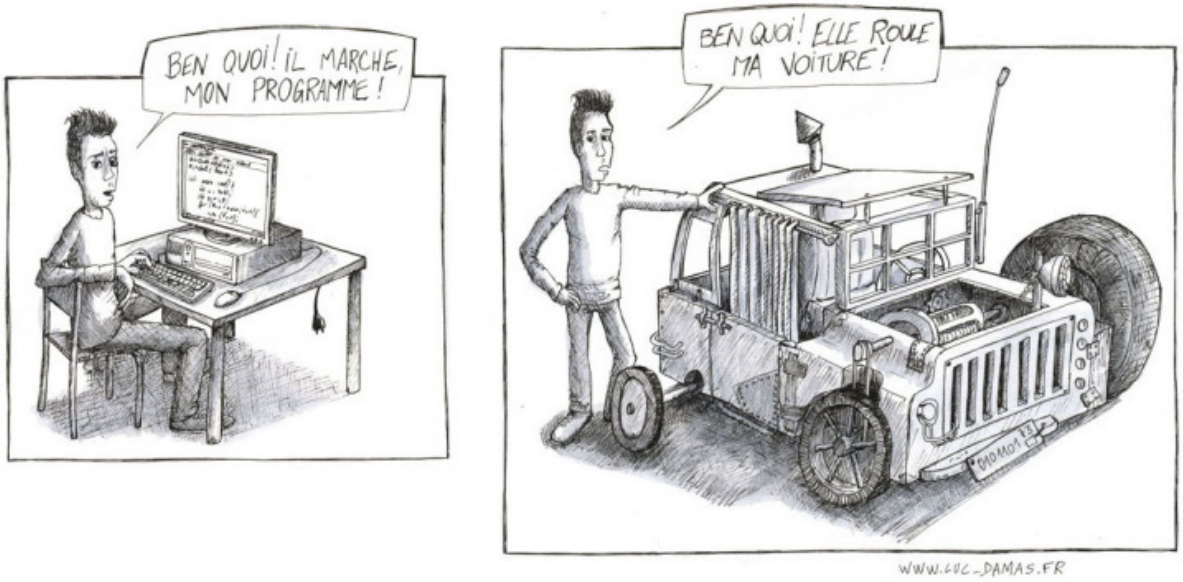
zenodo figshare

Conclusions

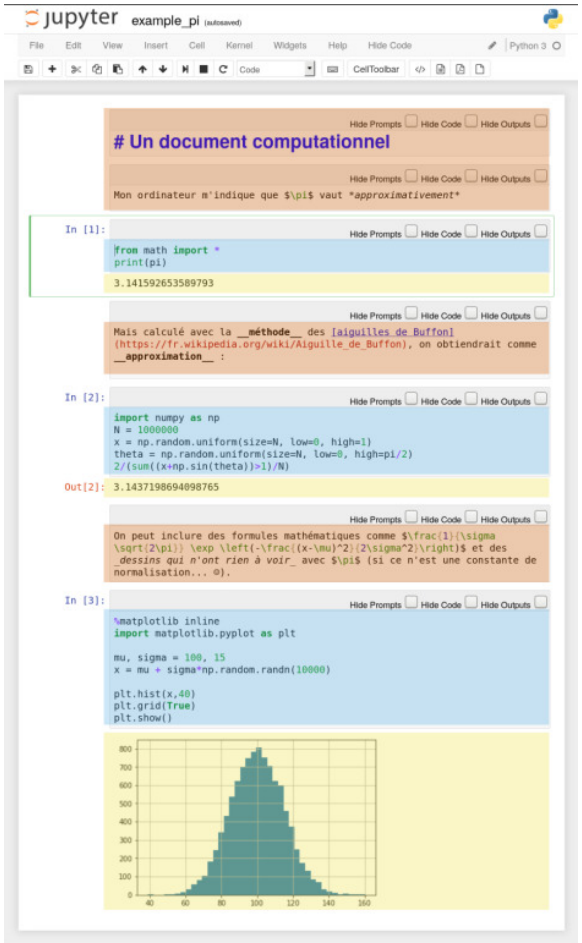
- Real data $\Rightarrow$ size and structure problems
- Read data are complex $\Rightarrow$ metadata
- ‘FITS‘ and ‘HDF5‘ = practical solutions
- In terms of complexity and flexibility : ‘FITS‘ $<$ ‘HDF5‘
- Archiving platforms $\Rightarrow$ persistent storage accessible for everyone

Where are we?

Scaling up

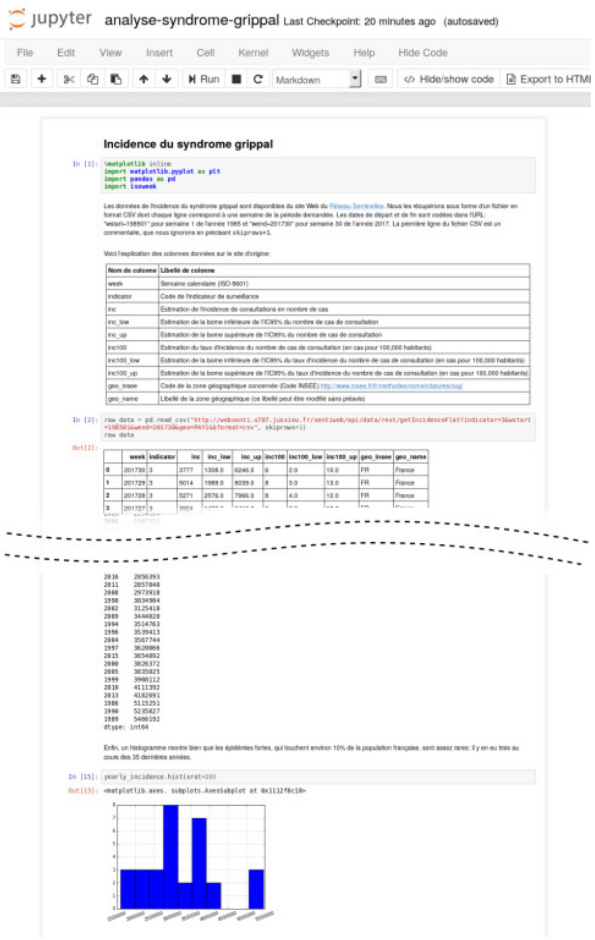

BEN QUOI ! IL MARCHE, MON PROGRAMME !
BEN QUOI ! ELLE ROULE MA VOITURE !
WWW.LUC_DAMAS.FR

Complex code. . .



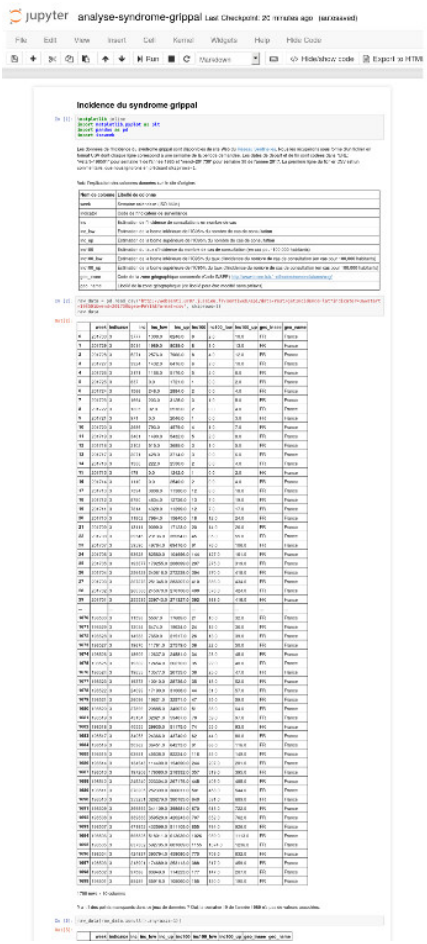
- A real spaghetti bowl
  - No global view
  - Interaction between multiple languages = danger

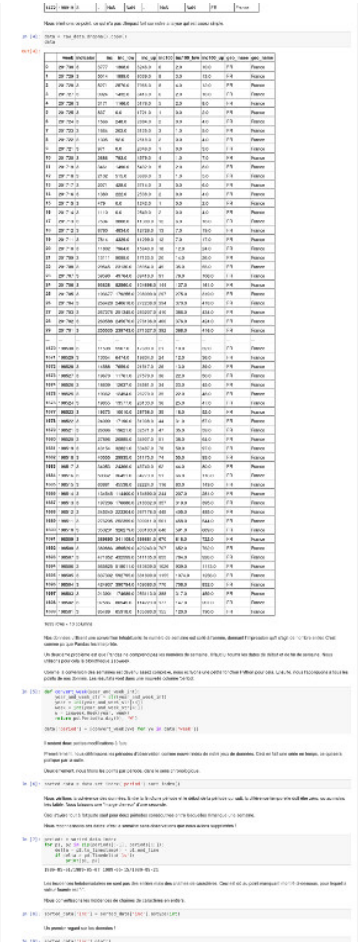
Complex code. . .

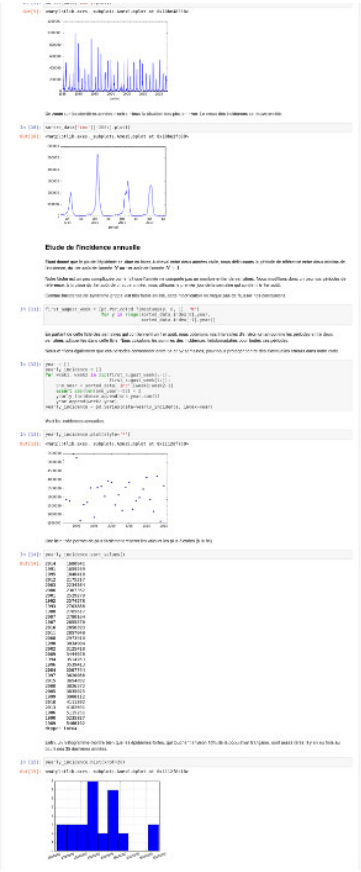


- A real spaghetti bowl
  - No global view
  - Interaction between multiple languages = danger

Complex code. . .





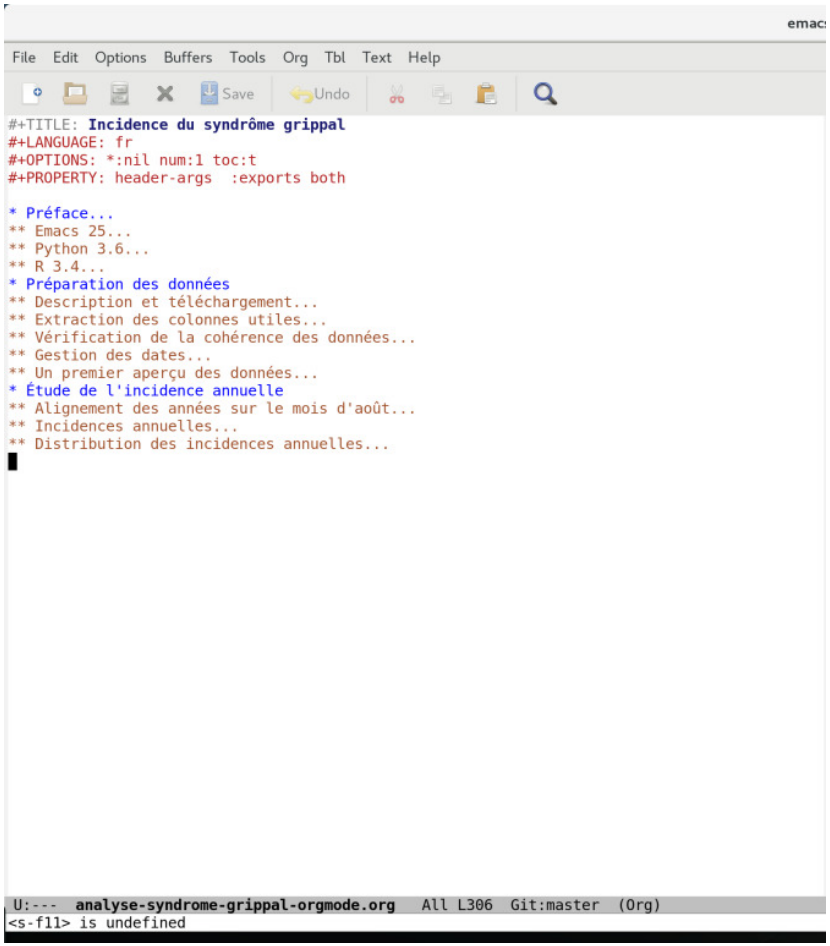


- A real spaghetti bowl
  - No global view
  - Interaction between multiple languages = danger

Complex code...

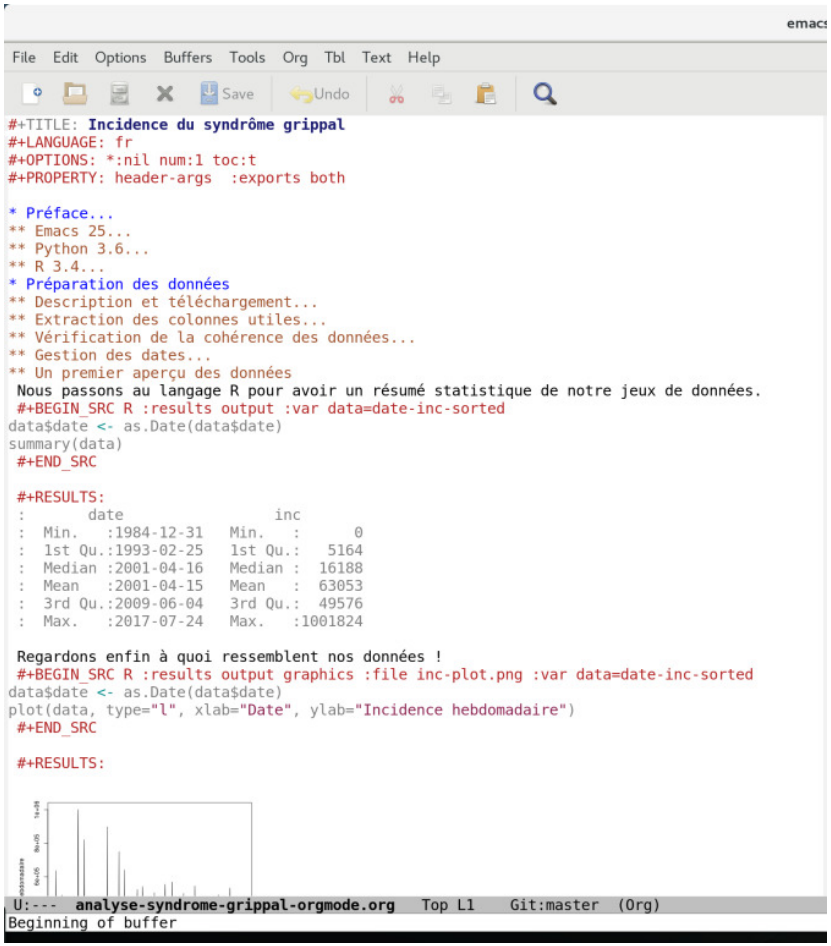
- A real spaghetti bowl
  - No global view
  - Interaction between multiple languages = danger

Complex code. . .



- A real spaghetti bowl
  - No global view
  - Interaction between multiple languages = danger

Complex code. . .

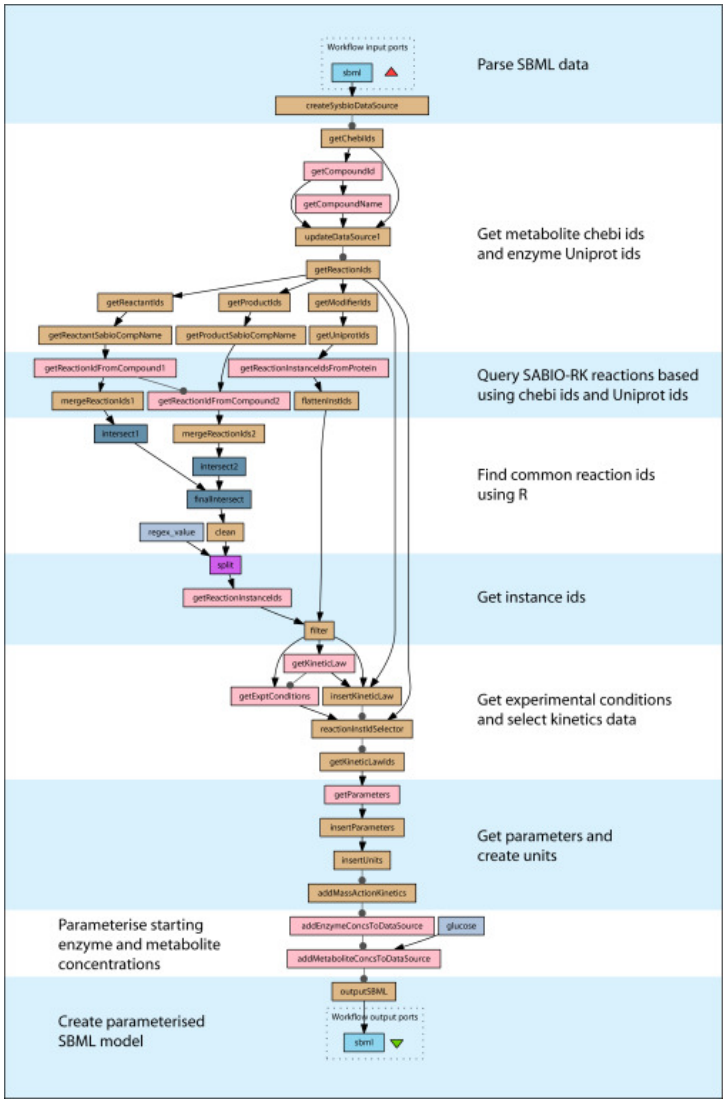


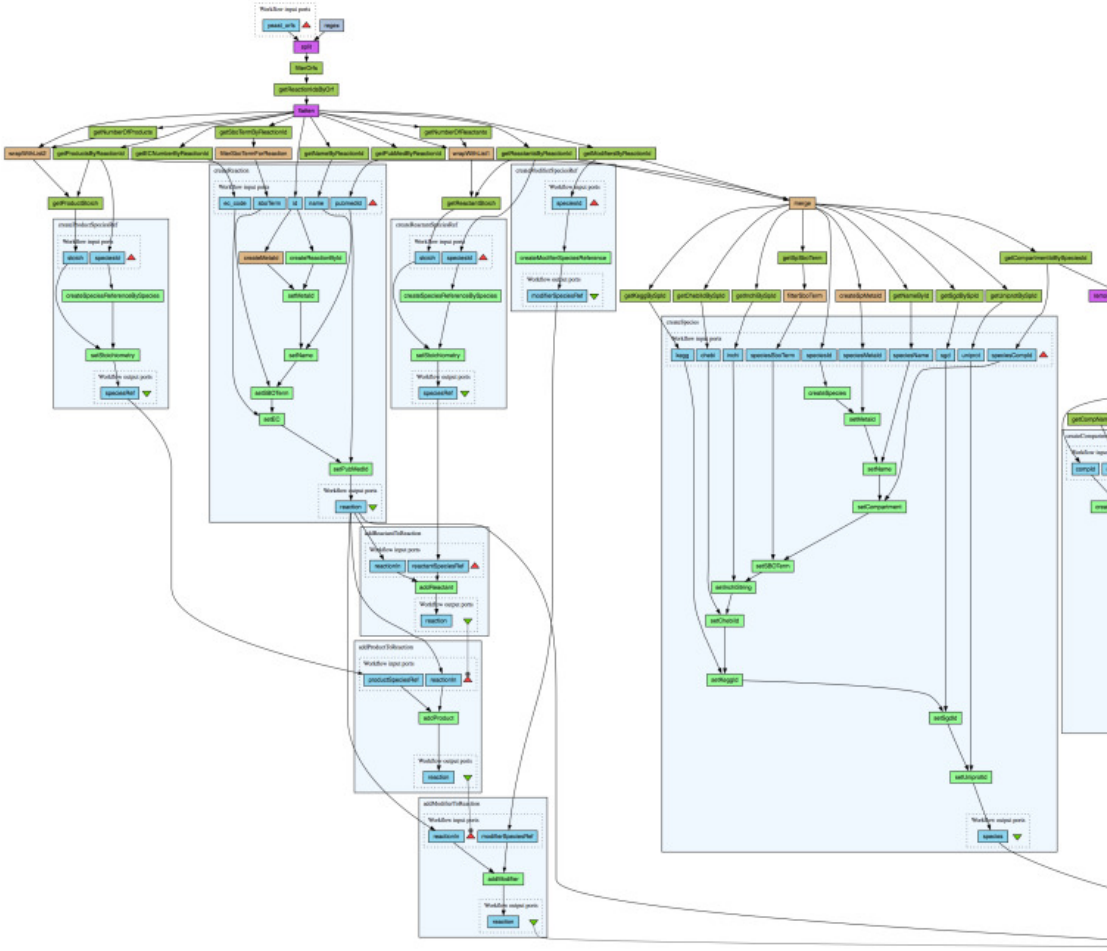
- A real spaghetti bowl
  - No global view
  - Interaction between multiple languages = danger

Complex code...

- A real spaghetti bowl
  - No global view
  - Interaction between multiple languages = danger

. . . that is difficult to orchestrate





... that is difficult to orchestrate

Workflows :

- Clearer high-level view
- Composition of codes and data movement made explicit
- Safer sharing, reusing, and execution
- Notebooks are a variant that is both impoverished and richer
- No simple/mature path from a notebook to a workflow

Examples :

- Galaxy, Kepler, Taverna, Pegasus, Collective Knowledge, VisTrails
- Light-weight : dask, drake, swift, snakemake, ...
- Hybrids : SOS-notebook, ...

The mess of expensive computations

Long-running computations and big datasets

- JupyterHub and supercomputers : under development
- Checkpoints and caching
- Workflows permit scaling up

Complex ecosystems

What is hiding behind the simple

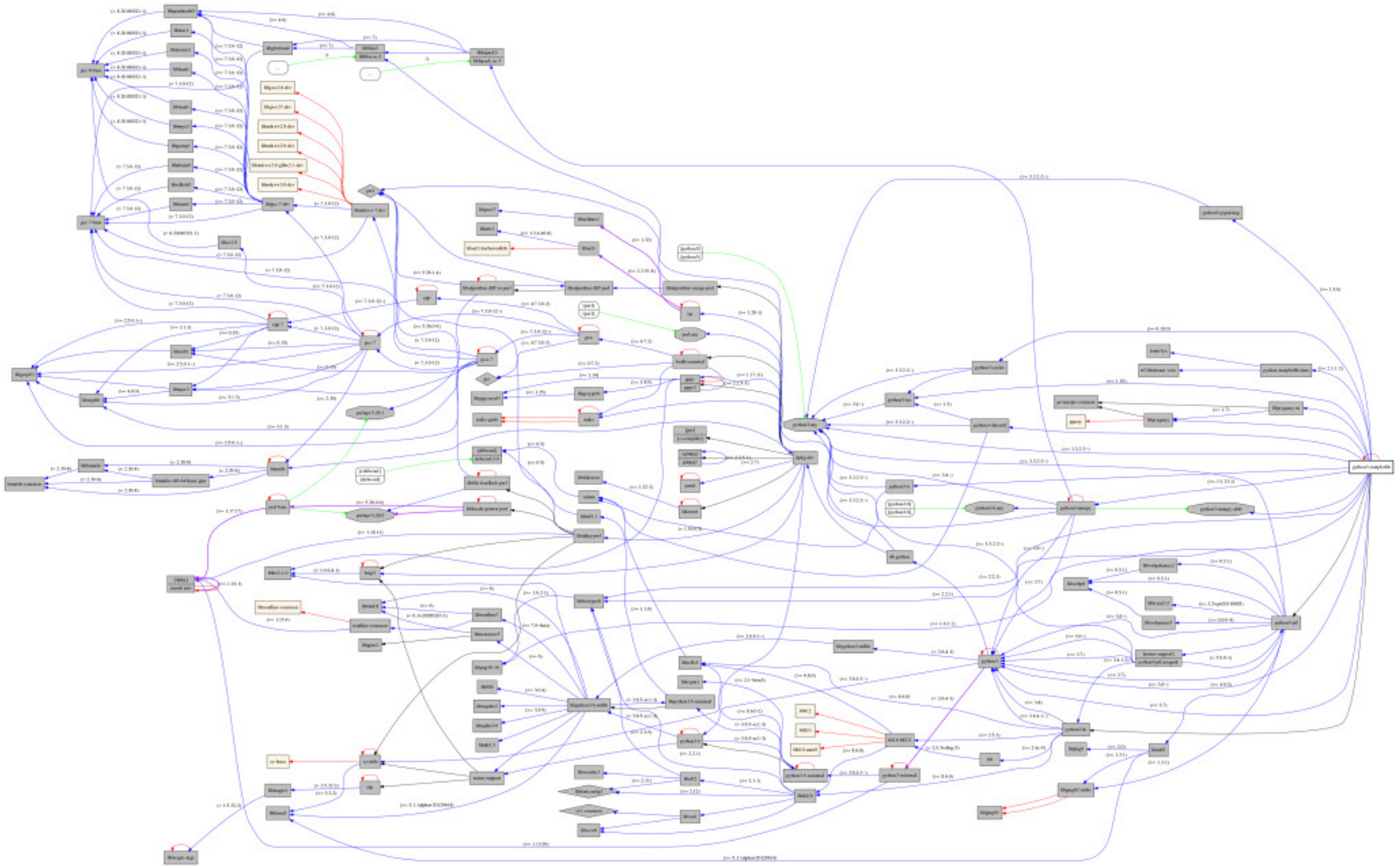
```
import matplotlib
```

```
Package: python3-matplotlib
Version: 2.1.1-2
Depends: python3-dateutil, python-matplotlib-data (>= 2.1.1-2),
python3-pyparsing (>= 1.5.6), python3-six (>= 1.10), python3-tz,
libjs-jquery, libjs-jquery-ui, python3-numpy (>= 1:1.13.1),
python3-numpy-abi9, python3 (<< 3.7), python3 (>= 3.6~),
python3-cycler (>= 0.10.0), python3:any (>= 3.3.2-2~), libc6 (>=
2.14), libfreetype6 (>= 2.2.1), libgcc1 (>= 1:3.0), libpng16-16 (>=
1.6.2-1), libstdc++6 (>= 5.2), zlib1g (>= 1:1.1.4)
```

Complex ecosystems

What is hiding behind the simple

```
import matplotlib
```



Complex ecosystems

No standard :

- Linux (`apt`, `rpm`, `yum`), MacOS X (`brew`, `McPorts`, `Fink`), Windows (?)
- Neither for installation nor for retrieving the information. . . ☹

```python
import sys
print(sys.version)
import matplotlib
print(matplotlib.__version__)
import pandas as pd
print(pd.__version__)
```

```
3.6.3 (default, Oct  3 2017, 21:16:13)
[GCC 7.2.0]
2.1.1
0.20.3
```

```r
library(ggplot2)
sessionInfo()
```

```
R version 3.4.3 (2017-11-30)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Debian GNU/Linux buster/sid

Matrix products: default
BLAS: /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.7.1
LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.7.1

locale:
 [1] LC_CTYPE=fr_FR.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=fr_FR.UTF-8        LC_COLLATE=fr_FR.UTF-8
 [5] LC_MONETARY=fr_FR.UTF-8    LC_MESSAGES=fr_FR.UTF-8

other attached packages:
[1] ggplot2_2.2.1

loaded via a namespace (and not attached):
 [1] colorspace_1.3-2 scales_0.5.0     compiler_3.4.3   laz
 [5] plyr_1.8.4       pillar_1.1.0     gtable_0.2.0     tib
 [9] Rcpp_0.12.15     grid_3.4.3       rlang_0.1.6      mun
```

Controlling one's environment

A controlled environment :

- Work in a virtual machine (heavy) or a Docker container (light)

Preserve the mess

- Automatic capture of the environment
- CDE, ReproZip, CARE

Cleaning up

- Start with a clean environment
- Install only what's strictly necessary (and document it)
- Docker, Singularity, Guix, Nix, ...

The test of time

Backwards compatibility

- Python and its rapidly evolving environment

```
python2 -c "print(10/3)"
python3 -c "print(10/3)"
```

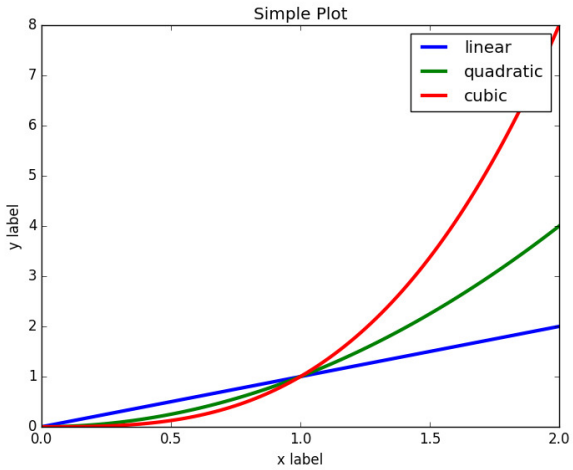
```
3
3.3333333333333335
```

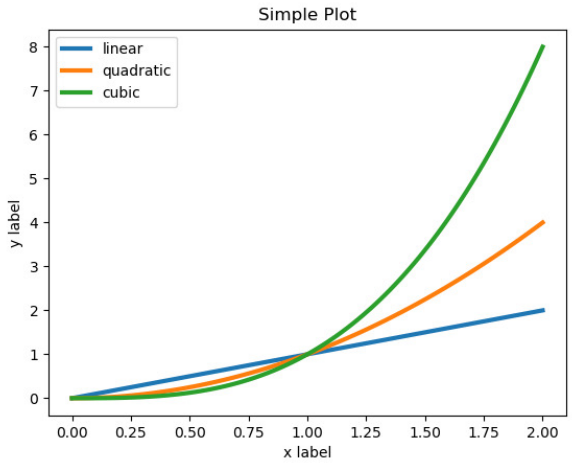
Backwards compatibility

- Python and its rapidly evolving environment

```
python2 -c "print(10/3)"
python3 -c "print(10/3)"
```

```
3
3.3333333333333335
```



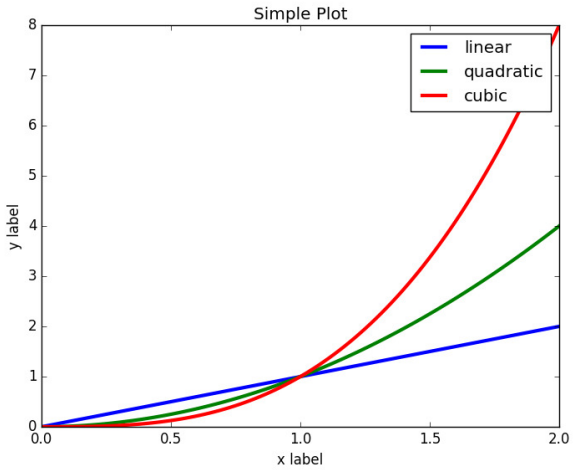


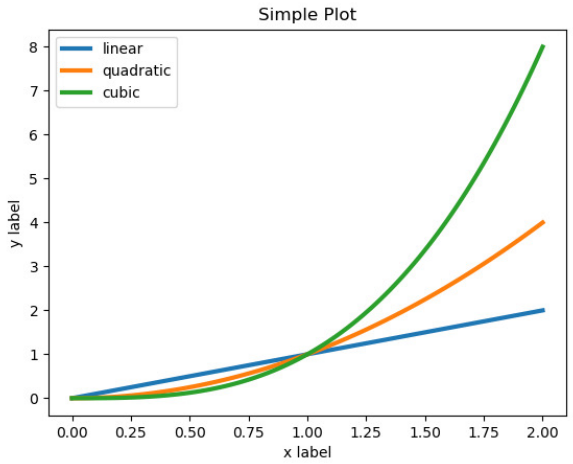
Backwards compatibility

- Python and its rapidly evolving environment

```
python2 -c "print(10/3)"
python3 -c "print(10/3)"
```

```
3
3.3333333333333335
```





- Cortical Thickness Measurements (PLOS ONE, June 2012) : *FreeSurfer : differences were found between the Mac and HP workstations and between Mac OSX 10.5 and OSX 10.6.*
- Format incompatibility between orgmode 7.*, 8.*, 9.*, etc.

Rapid development tools

Rapid but also fragile and unstable :

- Correction or introduction of bugs
- It becomes necessary to check regularly if environments can still be reconstructed and work (continuous integration, regression testing)

Popper : http://falsifiable.us/

Another option :

- Limit onself to what is manageable (C for example)

Archiving

Source code management

- Git (hub, lab, . . .) : stable, open, . . . durable ?
  - ~~Google Code~~, ~~Gitorious~~, ~~Code Spaces~~

Software Heritage

HAL archives-ouvertes.fr

Environment management

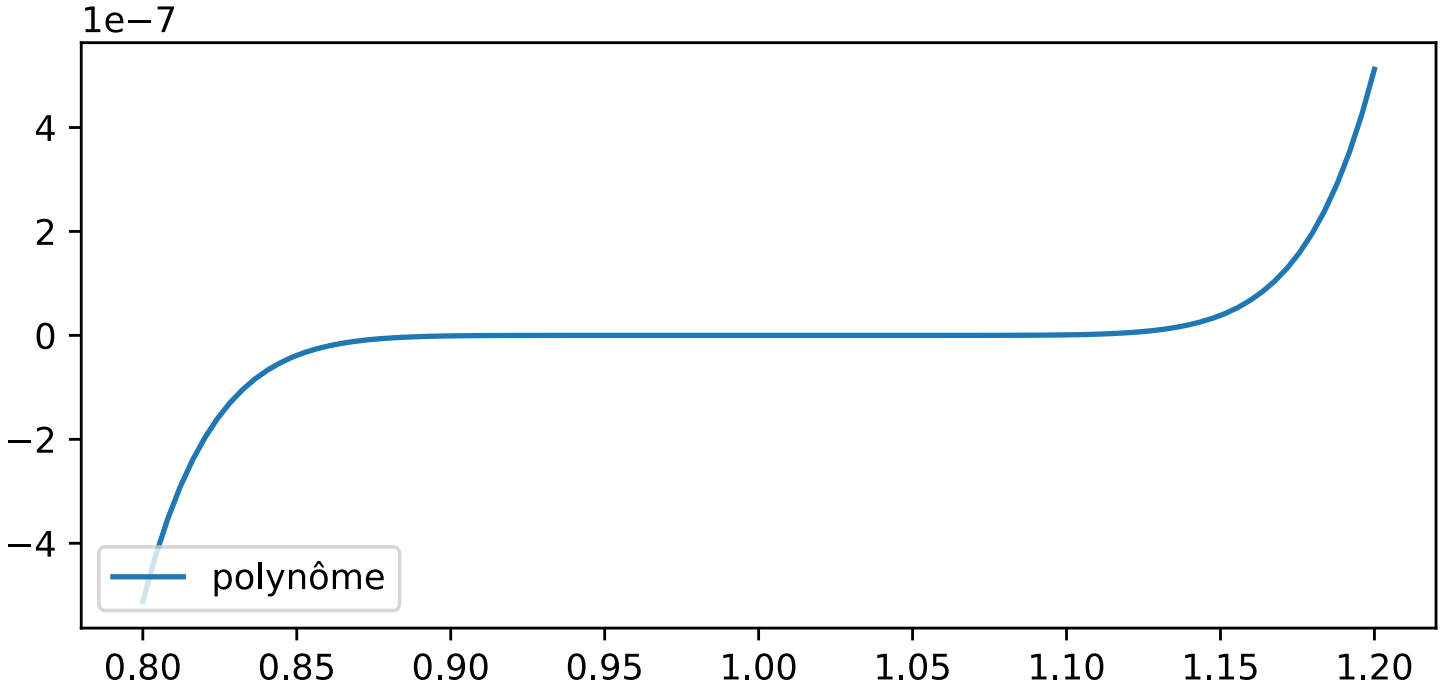
- Longevity of access to Docker Hub, Nix repository, Code Ocean, . . . ?
- Once an environment is frozen, what's the lifetime of a virtual machine, a Docker image, . . . ?

Preserve as much information as possible automatically

- Software, versions, installation procedures

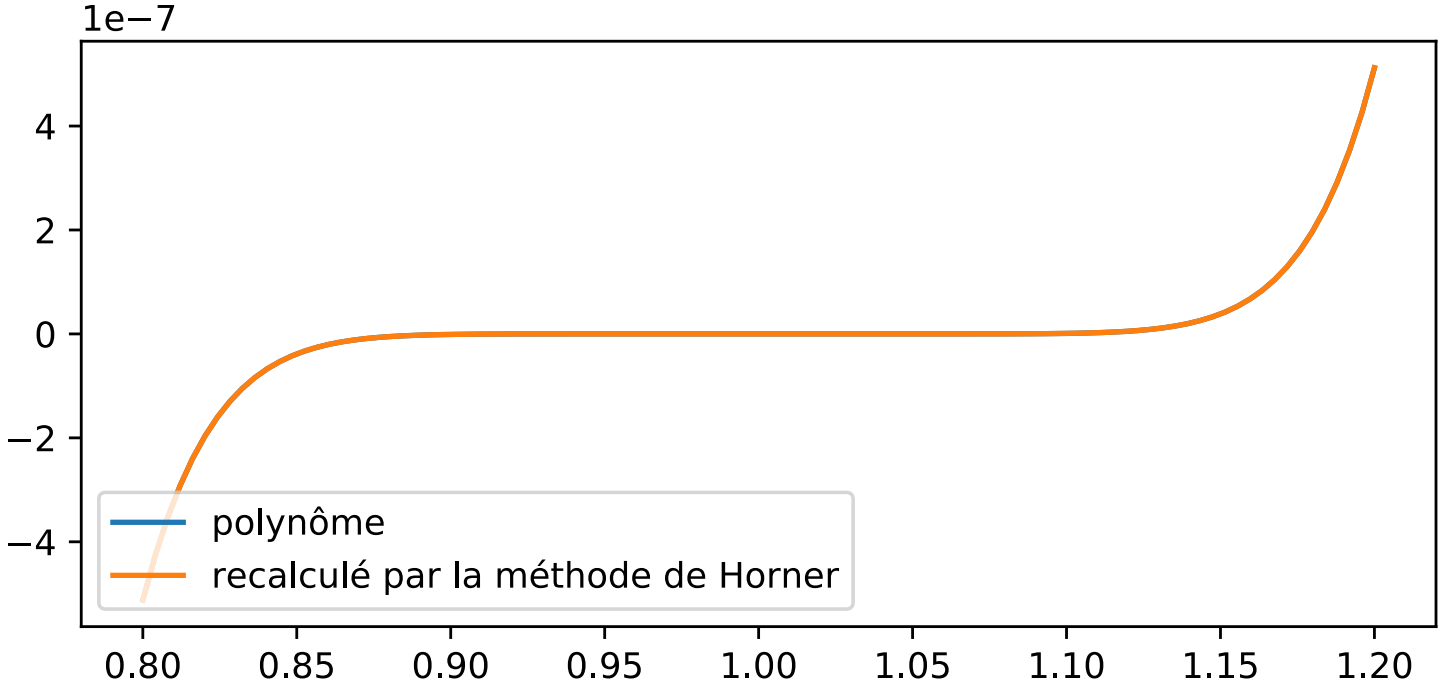
Where are we?

Floating-point arithmetic



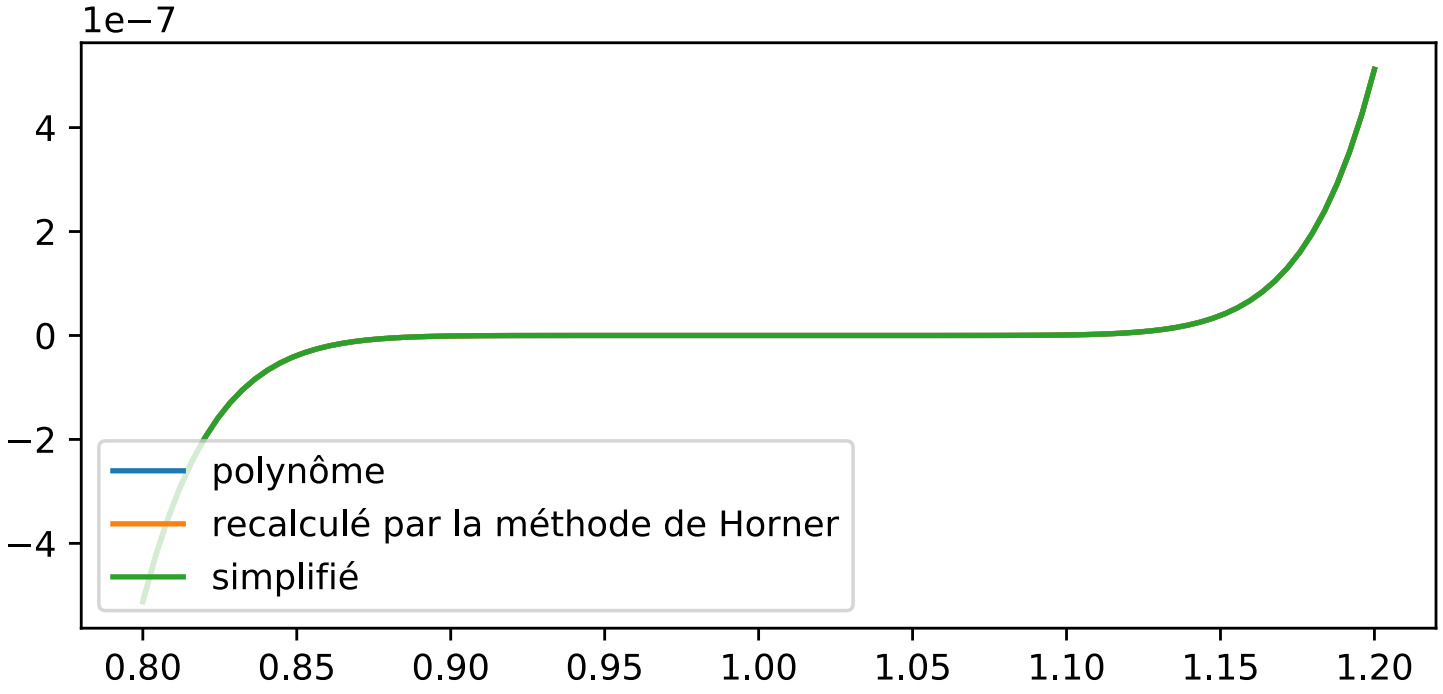

```
def polynome(x):
    return x**9 - 9.*x**8 + 36.*x**7 - 84.*x**6 + 126.*x**5 \
           - 126.*x**4 + 84.*x**3 - 36.*x**2 + 9.*x - 1.
```

Floating-point arithmetic



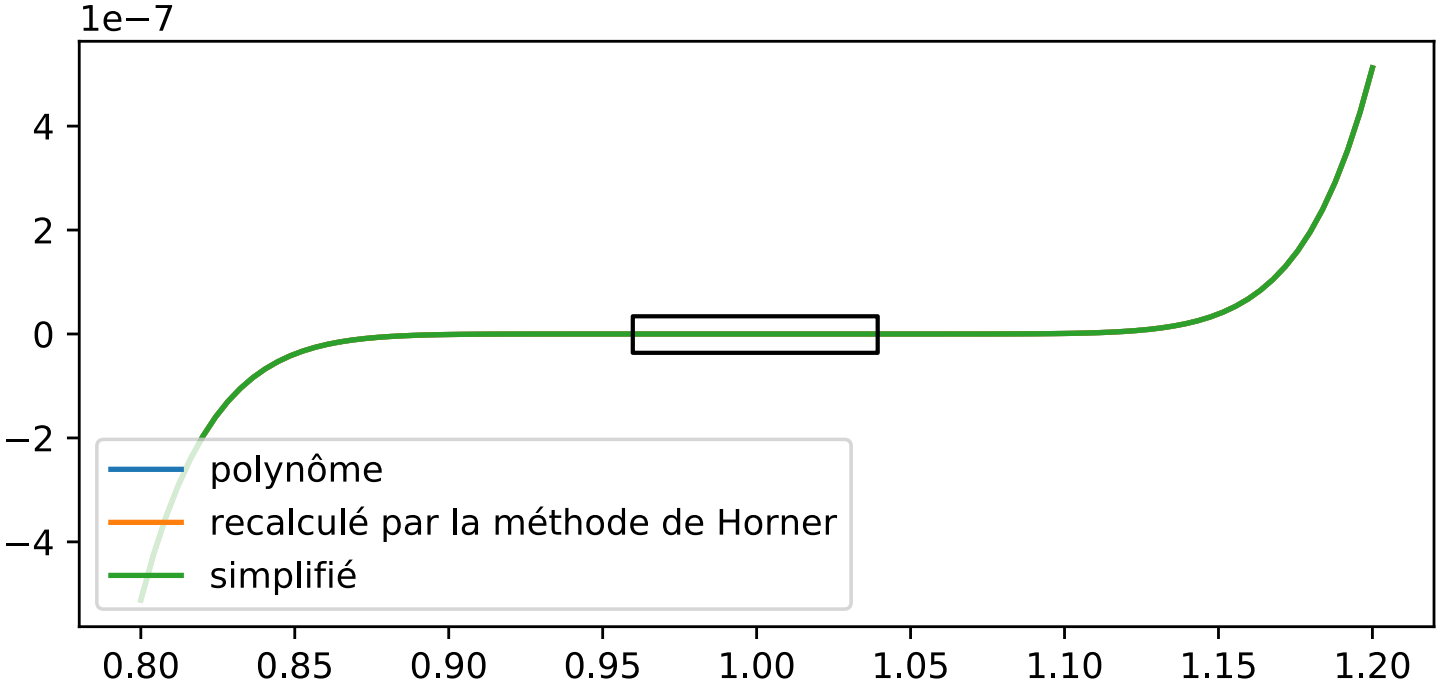

```
def horner(x):
    return x*(x*(x*(x*(x*(x*(x*(x*(x - 9.) + 36.) - 84.) + 126.) \
           - 126.) + 84.) - 36.) + 9.) - 1.
```

Floating-point arithmetic

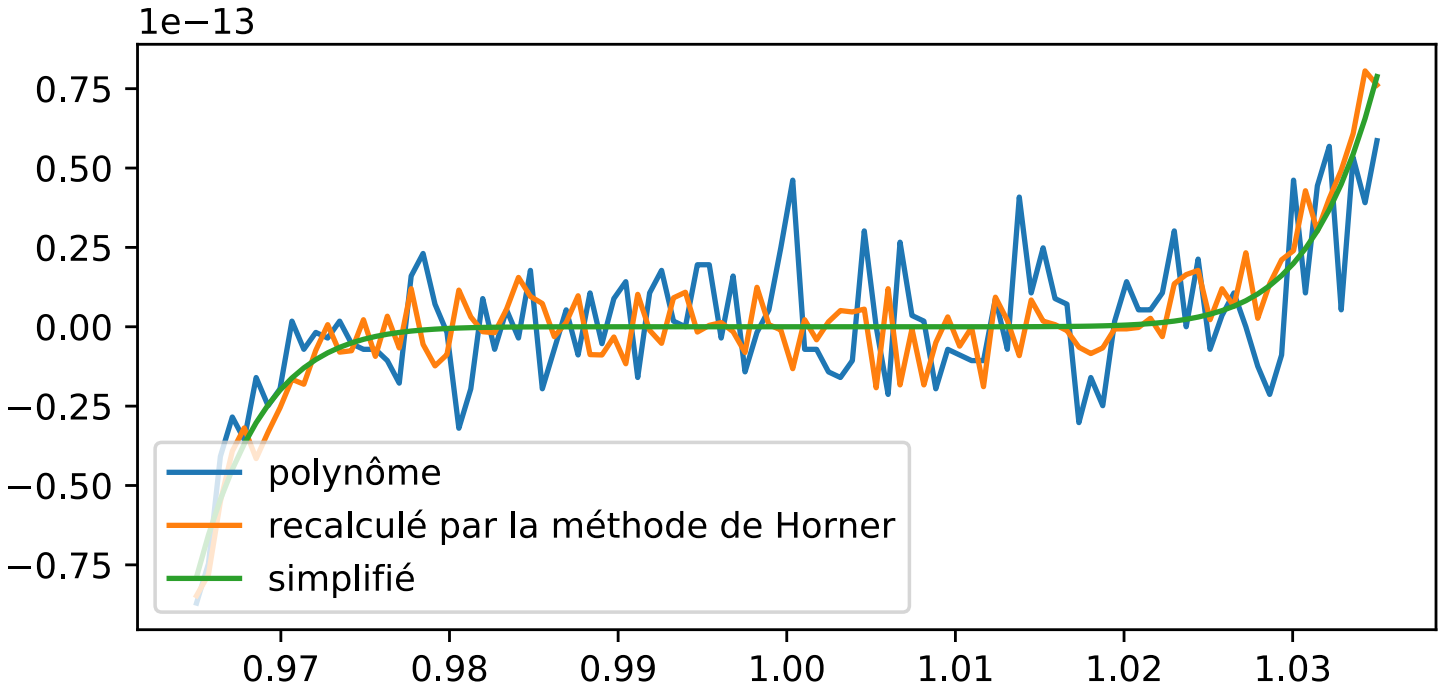



```
def simple(x):
    return (x-1.)**9
# trop facile!
```

Floating-point arithmetic



Floating-point arithmetic



Rounding

- Every operation includes implicit rounding.
- a+b is actually `arrondi`(a+b).
- Unfortunately :

  `arrondi`(`arrondi`(a+b)+c) $\neq$ `arrondi`(a+`arrondi`(b+c)).
- Operation order therefore matters.

For a reproducible computation, operation order must be preserved ! ! !

How to explain it to my compiler ?

To speed up computations, compilers may change operation order, and thus results.
Two options for computing reproducibly :

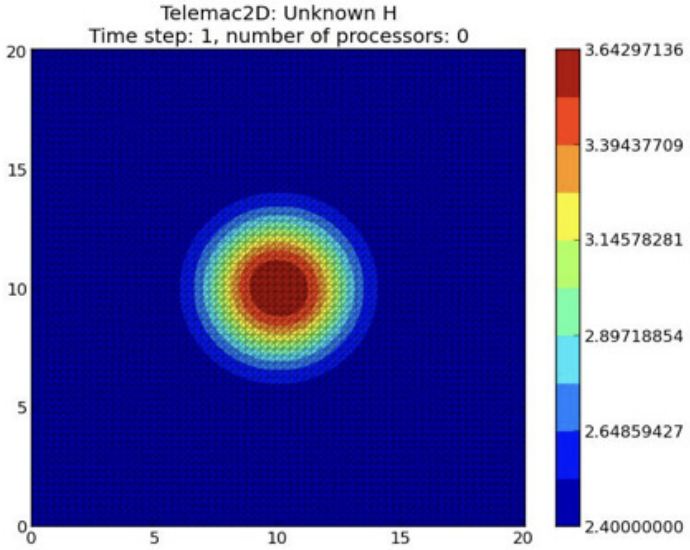
- Insist on the preservation of operation order,
  - if the language permits it.
  - Example : Module ‘ieee_arithmetic‘ in Fortran 2003
- Make compilation reproducible :
  - Record the precise compiler version
  - Record all compilation options

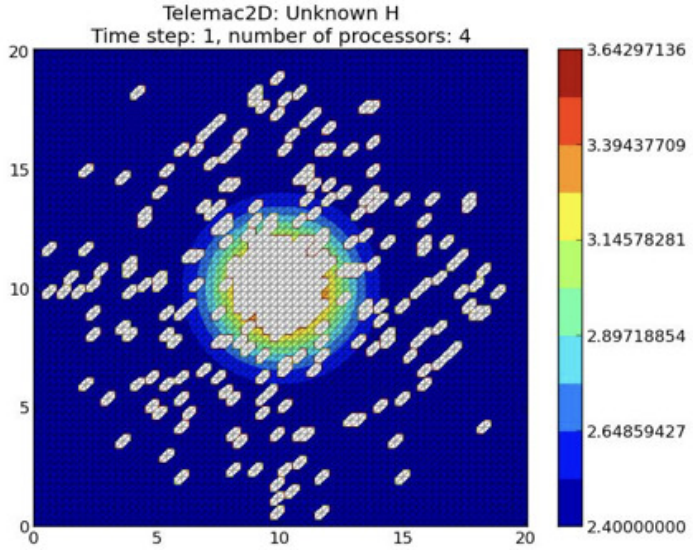
Parallel computation

- Goal : get results sooner $\rightarrow$ Minimize communications between processosrs $\rightarrow$ Adapt data distribution $\rightarrow$ ... hence the operation order ☹
- Consequence : results depends on the number of processors !

Minimizing the impact of parallelism is an active research topic.

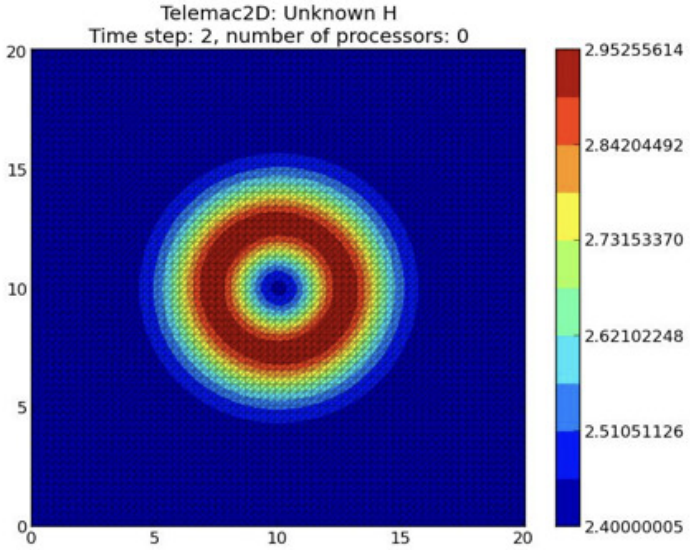
Parallel computation : example

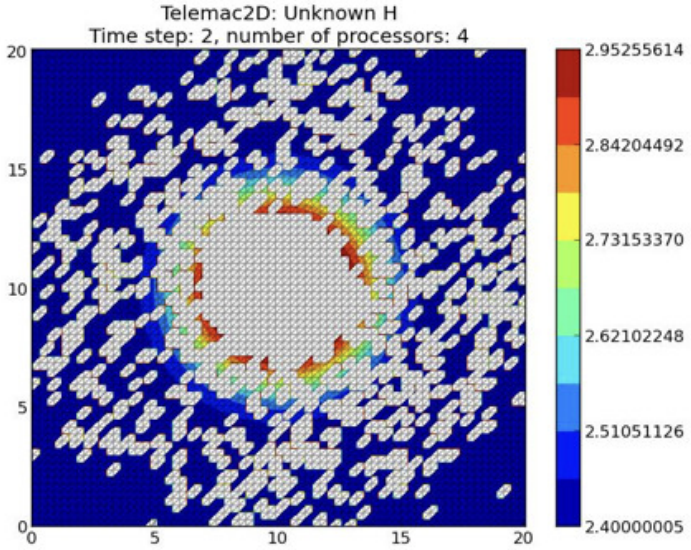




Source : Rafife Nheili, PhD. Thesis, University of Perpignan, 2016

Parallel computation : example





Source : Rafife Nheili, PhD. Thesis, University of Perpignan, 2016

Computing platforms

- Computing platform : hardware + infrastructure software
- Computation = platform + software + data
- The platform defines the interpretation of the software.
- Platform and software define the interpretation of the data.
- Other platform-defined aspects :
  - integer representation (16/32/64 bits)
  - error handling

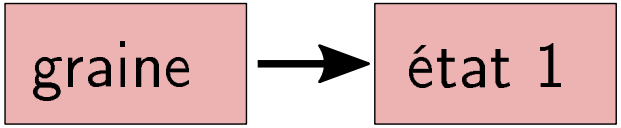
Random numbers

- Used to simulate stochastic processes.
- In practice : pseudo- random numbers.
- Series of numbers that appear to be random...
- ... but are generated by a deterministic algorithm.

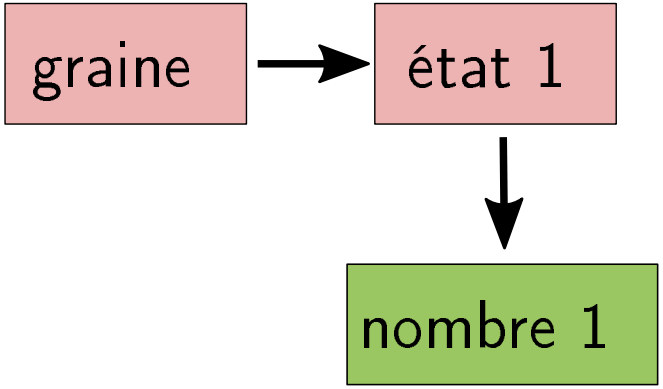
Pseudo-random number generators

graine

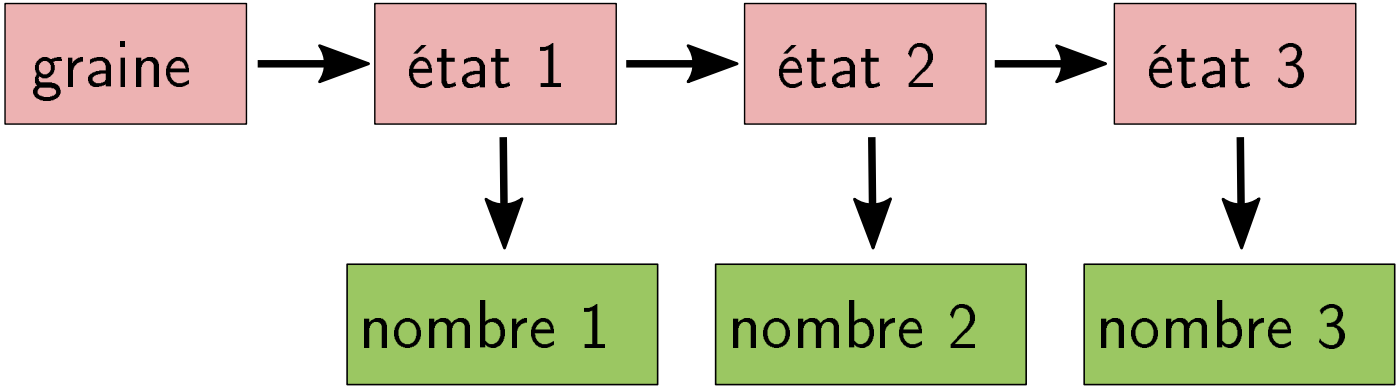
Pseudo-random number generators



Pseudo-random number generators



Pseudo-random number generators



Reproducibility in theory

- Principle : same seed + same algorithm $\rightarrow$ same series
- The seed is often chosen as a function of time
- It must be defined in the application code

Reproductibility in practice

- Same seed + same algorithm $\rightarrow$ same series :
  not obvious with floating-point arithmetic !
- A simple trick to permit verification :
  test the first values of the series.

Example : the Python language

```python
import random

random.seed(123)
for i in range(5):
    print(random.random())
```

```
0.0523635988509
0.0871866775226
0.40724176367
0.107700234938
0.901198877952
```

Example : the Python language

```python
import random

random.seed(123)
assert random.random() == 0.052363598850944326
assert random.random() == 0.08718667752263232
assert random.random() == 0.4072417636703983
```

Take-home message

- The results of a numerical computation depend on
  - the software
  - the input data
  - the computing platform : hardware, compilers, ...
- Platform influence is important for floating-point arithmetic. Record all parameters on which your results may depend :
  - compiler version, compilation options
  - hardware (processor type, GPU, ...)
  - number of processors
- When using a random number generator, define your own seed and verify the first elements of the series.

Where are we?

The take-home message of the MOOC

A major concern

- Scientific method
- Inspectability and reusability

Tools exist

- Computational documents and workflows, version control and archives, software environments, continuous integration, ...
- These tools evolve constantly
  - Choose those that are best adapted to your context
  - Find a compromise between modern and durable tools

Use in practice, don't get discouraged !

- Takes notes rigorously
- Make information useable and accessible
- Improve in small steps