Sustainable Scientific Software Development

Europython 2017

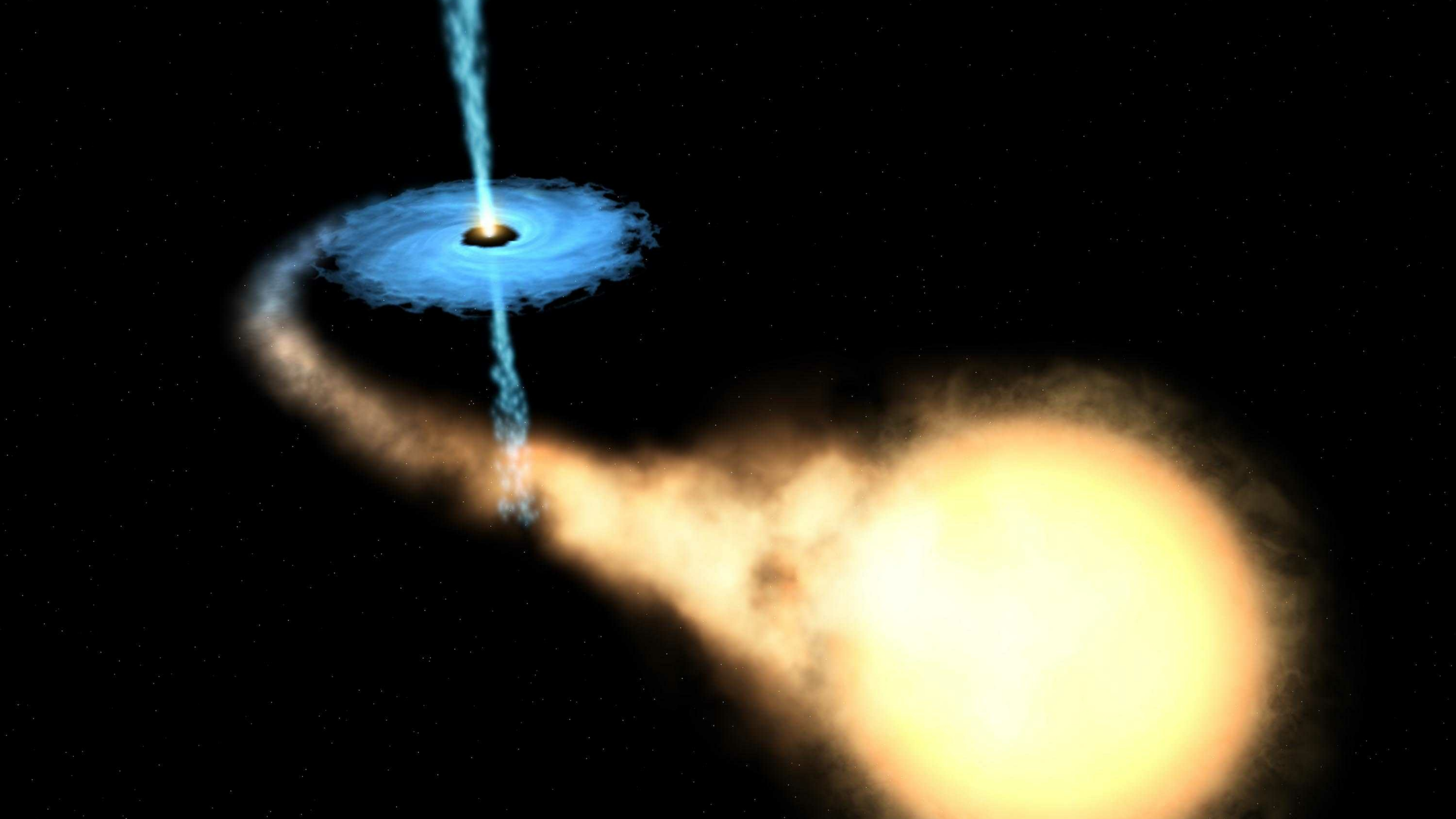
Alice Harpole

Software Sustainability Institute

University of Southampton

Motivation

- I model 'explosions in space'
  - or: *the effects of including general relativity in models of Type I X-ray bursts in neutron star oceans*



Motivation

- Fed up of reading about exciting codes, only to find
  - they're **not** open source
  - they have next to **no** documentation
  - questionable approaches to testing
- This is **not** good science!

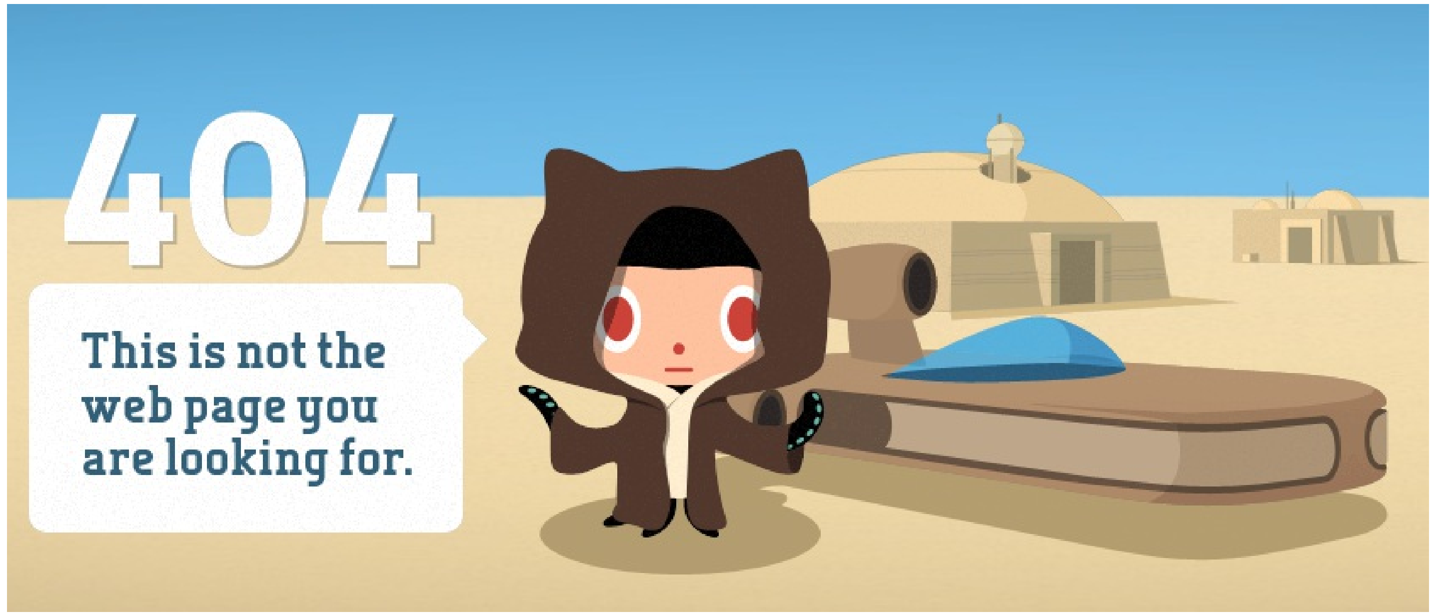
Software
Sustainability
Institute

Overview

- What is software sustainability (and why should I care)?
- Why scientific software is different
- Scientific software development workflow
  - Version control
  - Testing
  - Continuous integration & code coverage
  - Documentation
  - Distribution
- Conclusions

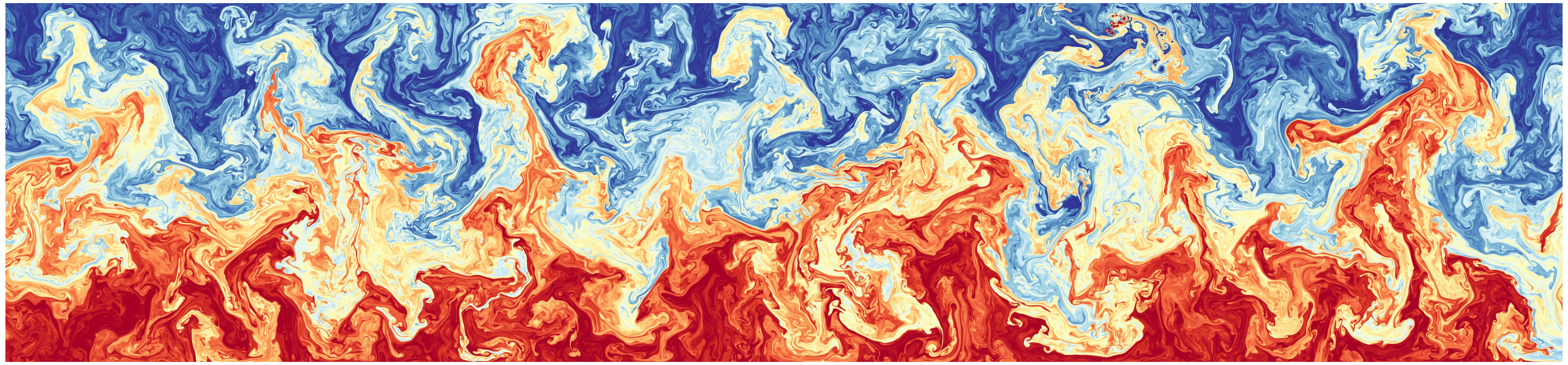
What is software sustainability (and why should I care)?

- Will my code still work in 5/10/20 years' time?
  - Can it be found?
  - Can it be run?
- If not, **harms** future scientific progress



What makes scientific software different?

- Built to investigate **complex**, **unknown** phenomena
- Often developed over long periods of time
- Can involve lots of **collaboration**
- Built by scientists, not software engineers



Turbulence modelled by Dedalus

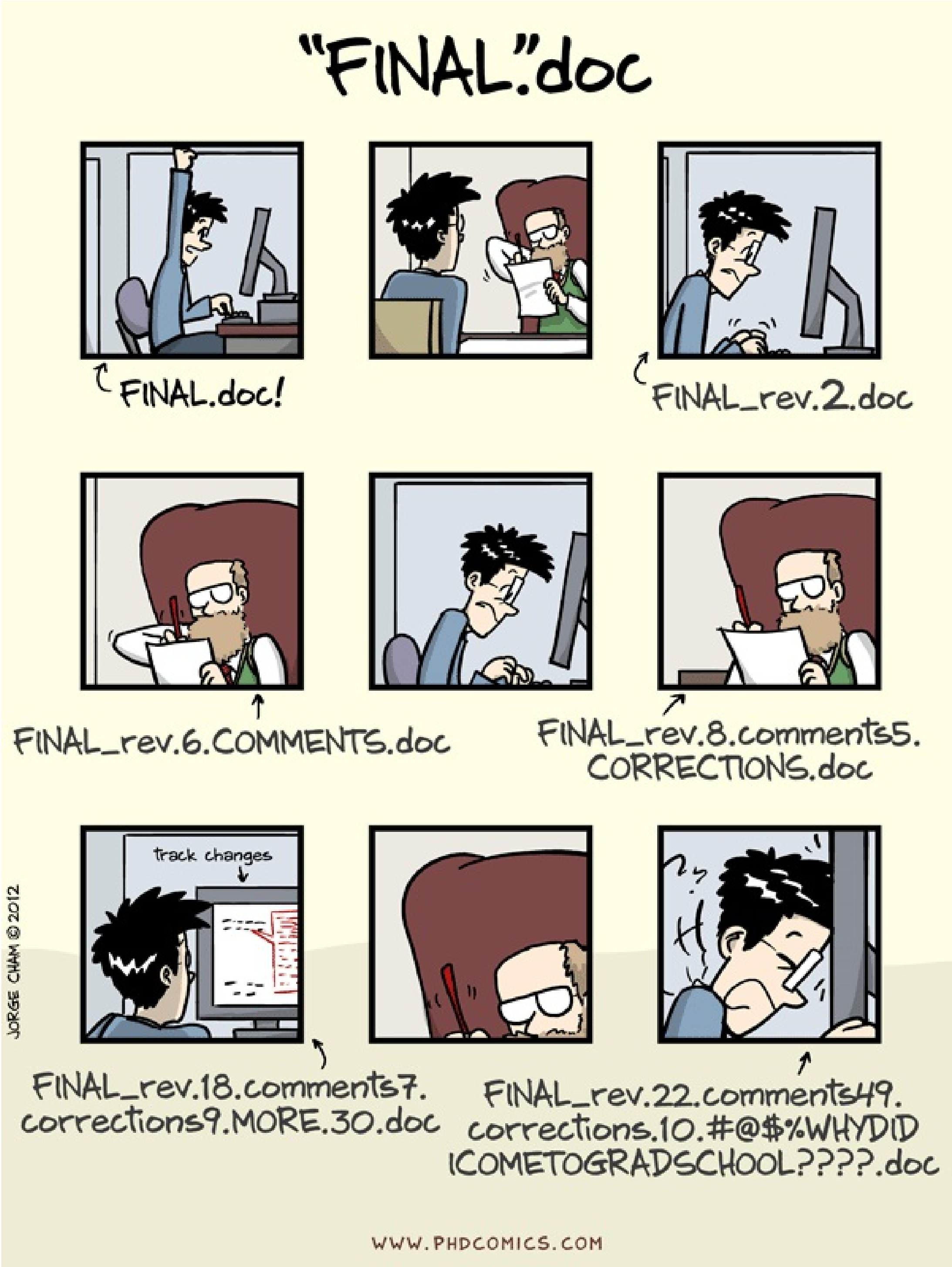
The Scientific Method

- In experimental science, results are not trusted unless follow scientific method:
  - **testing** of apparatus
  - **documentation** of method
- Demonstrate experiment's results are **accurate, reproducible** and **reliable**

The Scientific Method

- In computational science, we are doing experiments with the **computer as our apparatus**
- We should also follow scientific method and ***not*** trust results from codes without proper testing or documentation

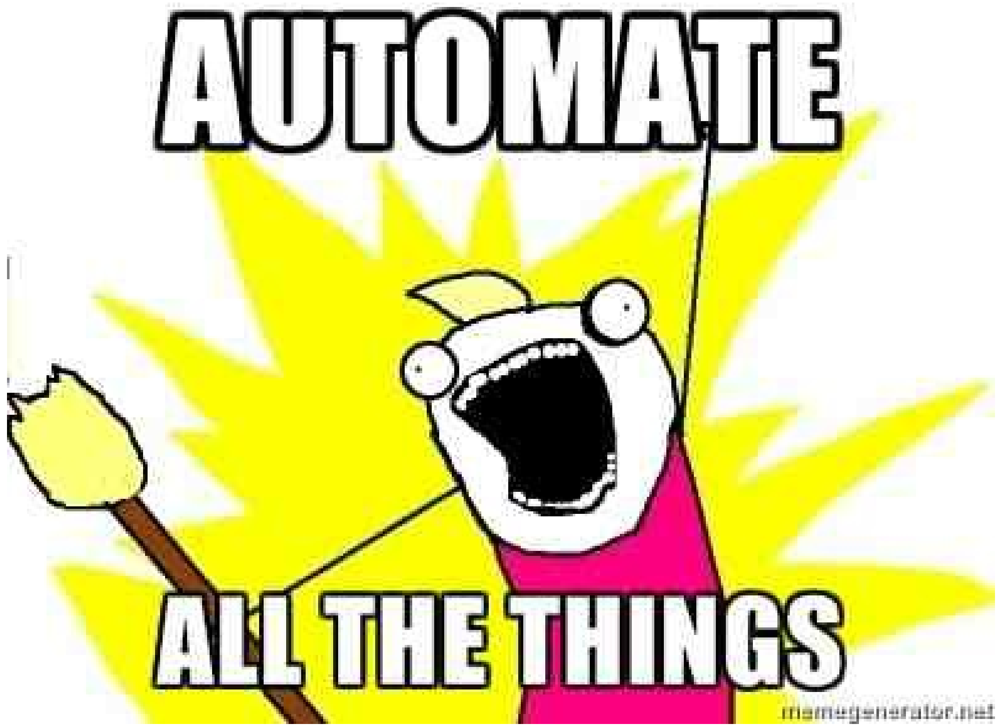
Source


"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.
CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.
corrections9.MORE.30.doc
FINAL_rev.22.comments49.
corrections.10.#@$%WHYDID
ICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012
WWW.PHDCOMICS.COM

PhD Comics

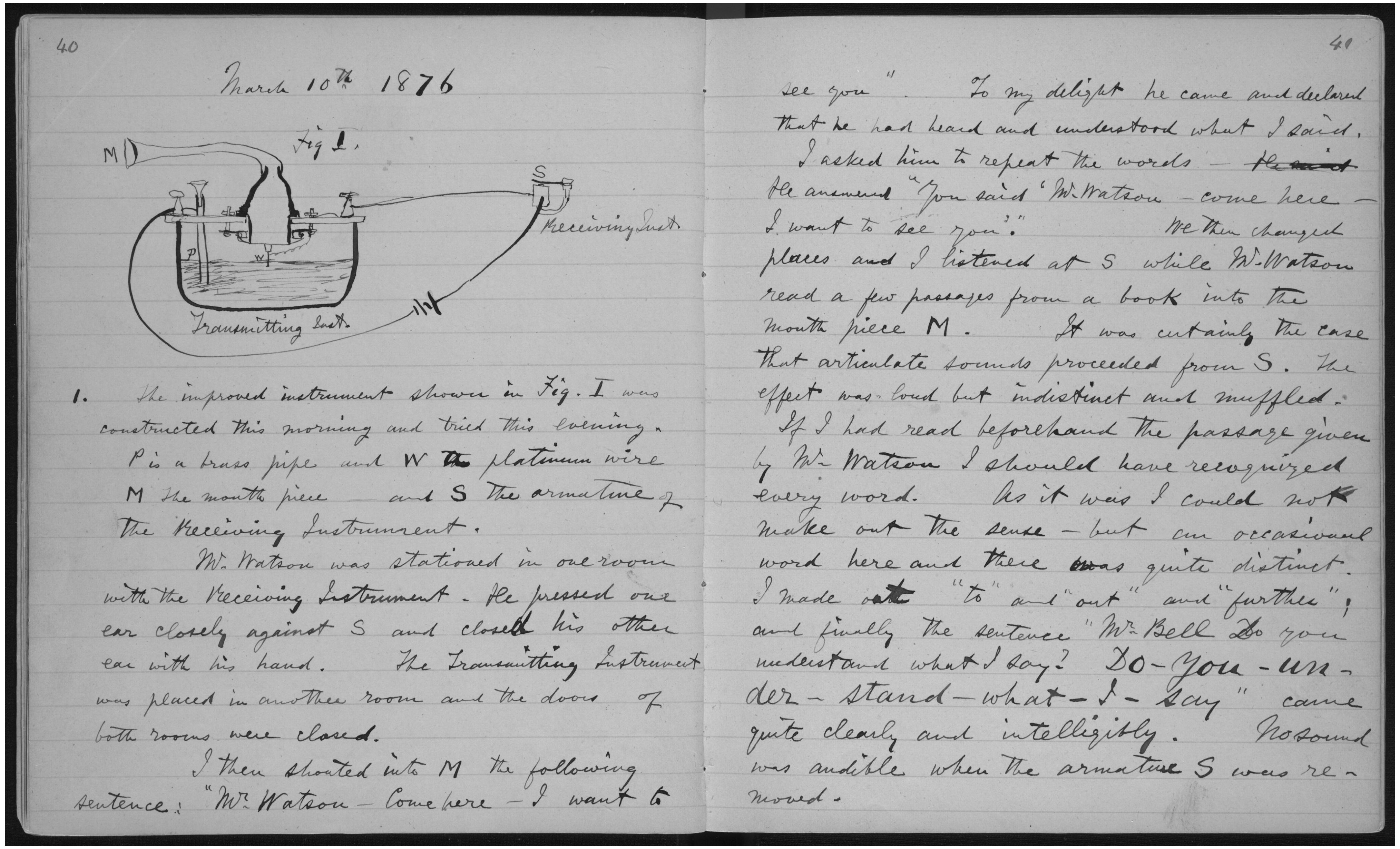
Development workflow

- Goal: implement sustainable practices throughout development
- Fortunately, there are lots of tools that will help us **automate** things!



Version control

- Keeps a **log** of all changes to code
- Computational science version of a lab book



40

March 10th 1876

Fig. 1.

1. The improved instrument shown in Fig. I was constructed this morning and tried this evening.

P is a brass pipe and W the platinum wire M the mouth piece and S the armature of the Receiving Instrument.

Mr. Watson was stationed in one room with the Receiving Instrument. He pressed one ear closely against S and closed his other ear with his hand. The Transmitting Instrument was placed in another room and the doors of both rooms were closed.

I then shouted into M the following sentence: "Mr. Watson — Come here — I want to

41

see you." To my delight he came and declared that he had heard and understood what I said.

I asked him to repeat the words — ~~He said~~ He answered "You said 'Mr. Watson — come here — I want to see you.'" We then changed places and I listened at S while Mr. Watson read a few passages from a book into the mouth piece M. It was certainly the case that articulate sounds proceeded from S. The effect was loud but indistinct and muffled.

If I had read beforehand the passage given by Mr. Watson I should have recognized every word. As it was I could not make out the sense — but an occasional word here and there ~~was~~ quite distinct.

I made out "to" and "out" and "further"; and finally the sentence "Mr. Bell Do you understand what I say? Do-you-un-der-stand-what-I-say" came quite clearly and intelligibly. No sound was audible when the armature S was removed.

Alexander Graham Bell's lab book - Wikimedia harpolea / r3d2

Unwatch 3 | Star 1 | Fork 1

<> Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Settings | Insights

No description, website, or topics provided.

Edit

Add topics

213 commits | 7 branches | 2 releases | 3 contributors | MIT

Branch: **master** | New pull request | Create new file | Upload files | Find file | Clone or download

harpolea committed on **GitHub** Update docs flag — Latest commit 6e9afbd on 24 Oct 2016

docs	Merge pull request #4 from harpolea/joss-statement-need-1	a year ago
paper	Extend the description of the Riemann Problem.	a year ago
r3d2	Merge branch 'master' of github.com:harpolea/r3d2	a year ago
tests	Add a subsonic test for find_left.	a year ago
.coveragerc	Updated .coveragerc	a year ago
.gitignore	Add sphinx and setup-related things to ignore file.	a year ago
.travis.yml	Added coverage module	a year ago
LICENSE	Create LICENSE	a year ago
Manifest.in	Switch towards a PyPI suitable setup.	a year ago
README.rst	Update docs flag	8 months ago
investigate_wave_pattern.py	p_v plotting	a year ago

Version control

- Aids **collaboration** - no overwriting each other's changes
- Can hack without fear - develop on a **branch**, so no danger of irreversibly breaking everything

Testing

- Should not trust results unless
  - apparatus & method (i.e. the software) that produced them has been **demonstrated to work**
  - any **limitations** (e.g. numerical error, algorithm choice) are understood and quantified

Classical and Quantum Gravity

PAPER • OPEN ACCESS

Characterization of transient noise in Advanced LIGO relevant to gravitational wave signal GW150914

B P Abbott[1], R Abbott[1], T D Abbott[2], M R Abernathy[1], F Acernese[3,4], K Ackley[5], M Adamo[4,6], C Adams[7], T Adams[8], P Addesso[3] Show full author list

Published 6 June 2016 • © 2016 IOP Publishing Ltd
Classical and Quantum Gravity, Volume 33, Number 13
Focus Issue: Gravitational Waves

Article PDF

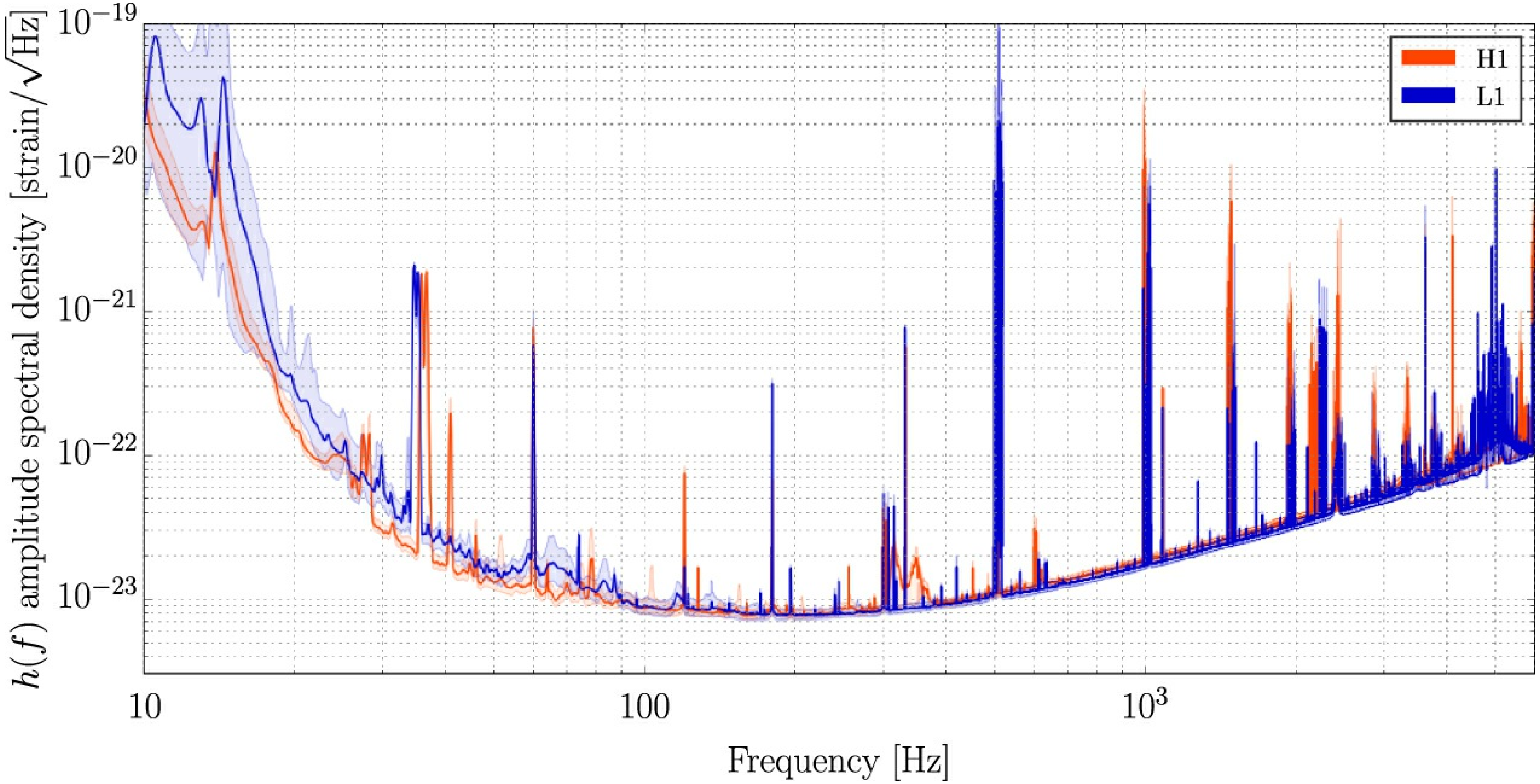
Figures ▾ References ▾ Citations ▾

+ Article information

Abstract
1. Introduction
2. Identifying noise sources
3. Potential noise sources
4. Mitigating noise sources
5. Transient search backgrounds

Abstract

On 14 September 2015, a gravitational wave signal from a coalescing black hole binary system was observed by the Advanced LIGO detectors. This paper describes the transient noise backgrounds used to determine the significance of the event (designated GW150914) and presents the results of investigations into potential correlated or uncorrelated sources of transient noise in the detectors around the time of the event. The


H1
L1
10^{-19}
10^{-20}
10^{-21}
10^{-22}
10^{-23}
$h(f)$ amplitude spectral density [strain/$\sqrt{\text{Hz}}$]
10
100
10^3
Frequency [Hz]

Testing

- Scientific codes can be hard to test as they are
  - often **complex**
  - investigate **unknowns**
- Does not mean we should give up!

Testing: Step 1

- Break it down with **unit tests**
  - Can't trust the sum if the parts don't work
  - Makes testing complex codes more manageable
  - Make sure these cover entire parameter space and check code breaks when it should

```python
import unittest

def squared(x):
        return x*x

class test_units(unittest.TestCase):

    def test_squared(self):
        self.assertTrue(squared(-5) == 25)
        self.assertTrue(squared(1e5) == 1e10)
        self.assertRaises(TypeError, squared, "A string")
```

Testing: Step 2

- Build it back up with **integration tests**
  - Need to check all parts work together
  - Can get more difficult here

Testing: Step 3

- Monitor development with **regression tests**
  - Check versions against each other
  - Performance should improve (or at least not get worse)
  - Bonus! Helps enforce **backwards compatibility** for users

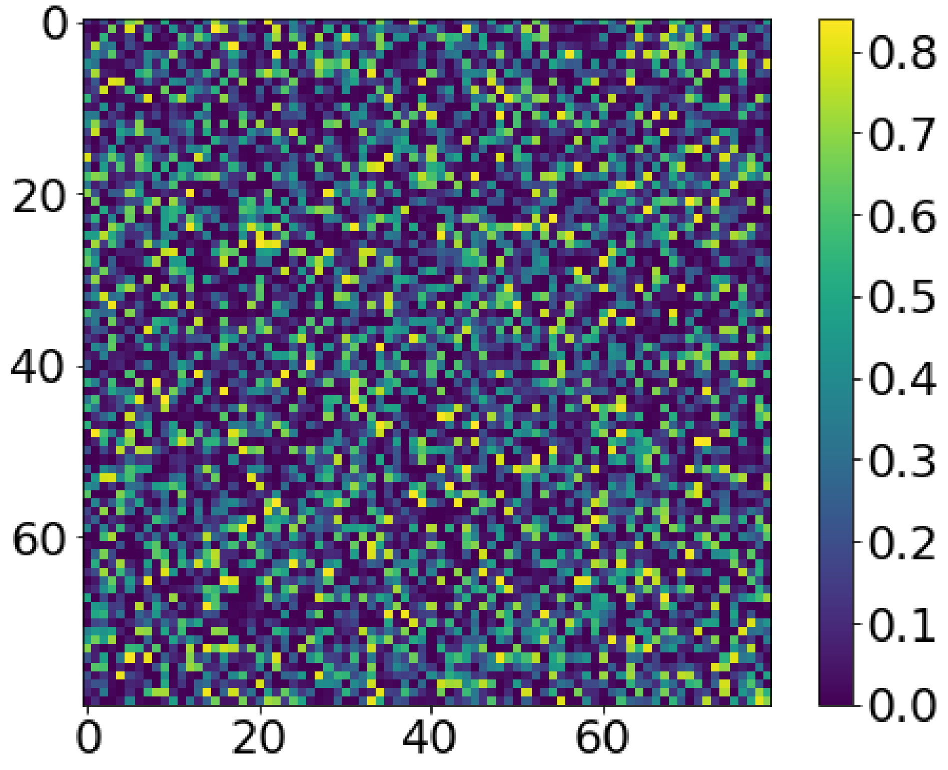
Science-specific issues

- Unknown behaviour
  - Use **controls** - simple input data with known solution
- Randomness
  - isolate random parts
  - test averages, check limits, conservation of physical quantities

```
data = rand(80,80)                       # declare some random data

def func(a):                              # function to apply to data
    return a**2 * numpy.sin(a)

output = func(data)                      # calculate & plot some function of ra
plt.imshow(output);  plt.colorbar();   plt.show()
```



Input is $0 \leq x \leq 1$, so output must be

$$0 \leq f(x) \leq \sin(1) \simeq 0.841$$

$$\overline{f(x)} = \int_0^1 f(x)\,dx \simeq 0.223$$

```python
def test_limits(a):
    if numpy.all(a >= 0.) and numpy.all(a <= 0.842): return True
    return False

def test_average(a):
    if numpy.isclose(numpy.average(a), 0.223, rtol=5.e-2): return True
    return False

if test_limits(output):
        print('Function output within correct limits')
else:
        print('Function output is not within correct limits')
if test_average(output):
        print('Function output has correct average')
else:
        print('Function output does not have correct average')
```

```
Function output within correct limits
Function output has correct average
```

Science-specific issues

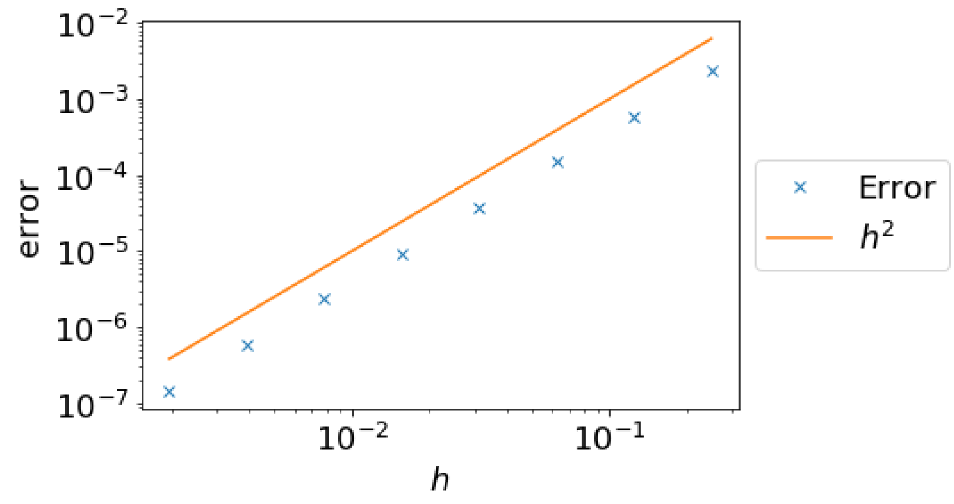
- Simulations
  - **convergence tests** - does accuracy of solution improve with order of algorithm used?
  - if not, algorithm may not be implemented correctly
- Numerical error
  - use numpy.isclose & numpy.allclose

```python
# use trapezium rule to find integral of sin x between 0,1
hs = numpy.array([1. / (4. * 2.**n) for n in range(8)])
errors = numpy.zeros_like(hs)

for i, h in enumerate(hs):
    xs = numpy.arange(0., 1.+h, h)
    ys = numpy.sin(xs)

    # use trapezium rule to approximate integral of sin(x)
    integral_approx = sum((xs[1:] - xs[:-1]) *
                          0.5 * (ys[1:] + ys[:-1]))
    errors[i] = -numpy.cos(1) + numpy.cos(0) - integral_approx

plt.loglog(hs, errors, 'x', label='Error')
plt.plot(hs, 0.1*hs**2, label=r'$h^2$')
plt.xlabel(r'$h$'); plt.ylabel('error')
```



Continuous integration & code coverage

- **Continuous integration** tools regularly run tests for you & report back results
  - Travis CI & CircleCI
- Find out when bugs occur much sooner - much easier to fix!
- **Danger**: outdated tests almost as useless as no tests
- If tests only cover 20% of code, why should you trust the other 80%?
  - **Code coverage**! Codecov

harpolea / r3d2

build passing

Current · Branches · Build History · Pull Requests › Build #142 · More options

✓ master Update docs flag

Commit 6e9afbd
Compare b1b2549..6e9afbd
Branch master

Alice Harpole authored · GitHub committed

#142 passed

Ran for 3 min 4 sec
Total time 7 min 50 sec

8 months ago

Restart build

Build Jobs

✓	# 142.1	Python: 2.7	no environment variables set	2 min 30 sec
✓	# 142.2	Python: 3.4	no environment variables set	2 min 50 sec
✓	# 142.3	Python: 3.5	no environment variables set	2 min 30 sec

Update docs flag

harpolea 8 months ago ✔ CI Passed

6e9afbd master b1b2549

95.14% | ø | ø

Diff | Files | Build | Graphs

Files	≡	● (green)	● (yellow)	● (red)	Coverage
r3d2/__init__.py	5	5	0	0	100.00%
r3d2/eos_defns.py	53	51	0	2	96.22%
r3d2/riemann_problem.py	38	34	0	4	89.47%
r3d2/state.py	37	37	0	0	100.00%
r3d2/utils.py	29	24	0	5	82.75%
r3d2/wave.py	345	320	0	25	92.75%
tests/test_eos.py	51	51	0	0	100.00%
tests/test_riemann_problem.py	132	132	0	0	100.00%
tests/test_state.py	25	25	0	0	100.00%
tests/test_utils.py	26	26	0	0	100.00%

Documentation

- Ideal: someone else in your field should be able to set up and use your code without extra help from you
- Include comprehensive installation instructions
- Document the code itself (sensible function & variable names, comments)
- User guide with **examples** to demonstrate usage
  - jupyter notebooks great for this
- Automate with Sphinx, host at Read the Docs

Riemann Problems

The code solves Riemann Problems for the relativistic Euler equations

$$\partial_t \begin{pmatrix} D \\ S_x \\ S_t \\ \tau \end{pmatrix} + \partial_x \begin{pmatrix} S_x \\ S_x v_x + p \\ S_t v_x \\ (\tau + p) v_x \end{pmatrix} = 0.$$

For further details on this system of equations, see the Living Review of Martí and Müller, particularly section 3.1 for the equations and section 8.5 for the solution of the Riemann Problem.

The initial data is piecewise constant: two states $w_{L,R}$ are specified, each in terms of $w = (\rho_0, v_x, v_t, \epsilon)$, (the specific rest mass density, normal (x) and tangential (t) velocity components, and the specific internal energy). At $t = 0$ the data is set by w_L for $x < 0$ and w_R for $x > 0$. Each state has associated with it an equation of state (EOS) to close the set of equations: the EOS does not need to be the same for each state.

Code

To set up a Riemann problem, first set up a left and right state. Each state has its own equation of state. Here we use the first test from the Test Bench section of the Living Review:

In [1]:
```
from r3d2 import eos_defns, State, RiemannProblem
```

In [2]:
```
eos = eos_defns.eos_gamma_law(5.0/3.0)
test_1_U_left = State(10.0, 0.0, 0.0, 2.0, eos, label="L")
test_1_U_right = State(1.0, 0.0, 0.0, 1.5e-6, eos, label="R")
test_1_rp = RiemannProblem(test_1_U_left, test_1_U_right)
```

The Riemann Problem will produce output directly in the notebook:

In [3]:
```
test_1_rp
```

Table Of Contents

Riemann Problems
- Code
  - Changing equation of state
  - Reactive problems

This Page

Show Source

Quick search

Go

Enter search terms or a module, class or function name.

Riemann Problems

The code solves Riemann Problems for the relativistic Euler equations

$$\partial_t \begin{pmatrix} D \\ S_x \\ S_t \\ \tau \end{pmatrix} + \partial_x \begin{pmatrix} S_x \\ S_x v_x + p \\ S_t v_x \\ (\tau + p) v_x \end{pmatrix} = 0.$$

For further details on this system of equations, see the Living Review of Martí and Müller, particularly section 3.1 for the equations and section 8.5 for the solution of the Riemann Problem.

The initial data is piecewise constant: two states $w_{L,R}$ are specified, each in terms of $w = (\rho_0, v_x, v_t, \epsilon)$, (the specific rest mass density, normal (x) and tangential (t) velocity components, and the specific internal energy). At $t = 0$ the data is set by w_L for $x < 0$ and w_R for $x > 0$. Each state has associated with it an equation of state (EOS) to close the set of equations: the EOS does not need to be the same for each state.

Code

To set up a Riemann problem, first set up a left and right state. Each state has its own equation of state. Here we use the first test from the Test Bench section of the Living Review:

In [1]:
```
from r3d2 import eos_defns, State, RiemannProblem
```

In [2]:
```
eos = eos_defns.eos_gamma_law(5.0/3.0)
test_1_U_left = State(10.0, 0.0, 0.0, 2.0, eos, label="L")
test_1_U_right = State(1.0, 0.0, 0.0, 1.5e-6, eos, label="R")
test_1_rp = RiemannProblem(test_1_U_left, test_1_U_right)
```

Distribution

- Make it findable
  - Open source! (where possible)
  - **DOI** e.g. from zenodo
- Reproducible results require a **reproducible runtime environment**
  - package code in e.g. docker container, conda environment, PyPI
- Installation should be as painless as possible
  - makefiles, try to limit reliance on non-open source libraries/material

Conclusions

- We need to **future-proof** our software
- Apply the **scientific method** to software development
- Only trust results from codes that are
  - **reproducible** (open source!)
  - **tested**
  - **documented**
- Check out the SSI website www.software.ac.uk for more