Sustainable Scientific Software Development

Europython 2017

Alice Harpole
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!
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
"FINAL.doc"

FINAL.doc!

FINAL_rev.2.doc

FINAL_rev.6 COMMENTS.doc

FINAL_rev.8 COMMENTS CORRECTIONS.doc

FINAL_rev.18 COMMENTS.7 CORRECTIONS.9 MORE.30.doc

FINAL_rev.22 COMMENTS.49 CORRECTIONS.10 %%%WHY DID I COMETOSGRAD SCHOOL.doc

WWW.PHDCOMICS.COM
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
March 1876

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

Then started into M the following sentence: “Mr. Watson, come here. I want to see you.” To my delight he came and stood that he had heard and understood what I said. I asked him to repeat the words. He answered, “You said Mr. Watson, come here. I want to see you.” 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 the 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 put out” and “for this” and finally the sentence “Mr. Bell, do you understand what I say?” Do—You—We—Are—Stand—What—Is—S—Say. The sound was audible when the armature S was removed.
### harpolea / r3d2

No description, website, or topics provided.

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

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

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>docs</td>
<td>Merge pull request #4 from harpolea/joss-statement-need-1</td>
<td>a year ago</td>
</tr>
<tr>
<td>paper</td>
<td>Extend the description of the Riemann Problem.</td>
<td>a year ago</td>
</tr>
<tr>
<td>r3d2</td>
<td>Merge branch 'master' of github.com:harpolea/r3d2</td>
<td>a year ago</td>
</tr>
<tr>
<td>tests</td>
<td>Add a subsonic test for find_left.</td>
<td>a year ago</td>
</tr>
<tr>
<td>.coveragerc</td>
<td>Updated .coveragerc</td>
<td>a year ago</td>
</tr>
<tr>
<td>.gitignore</td>
<td>Add sphinx and setup-related things to ignore file.</td>
<td>a year ago</td>
</tr>
<tr>
<td>.travis.yml</td>
<td>Added coverage module</td>
<td>a year ago</td>
</tr>
<tr>
<td>LICENSE</td>
<td>Create LICENSE</td>
<td>a year ago</td>
</tr>
<tr>
<td>Manifest.in</td>
<td>Switch towards a PyPI suitable setup.</td>
<td>a year ago</td>
</tr>
<tr>
<td>README.rst</td>
<td>Update docs flag</td>
<td>8 months ago</td>
</tr>
<tr>
<td>investigate_wave_pattern.py</td>
<td>p_v plotting</td>
<td>a year ago</td>
</tr>
</tbody>
</table>
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
Characterization of transient noise in Advanced LIGO relevant to gravitational wave signal GW150914

B P Abbott¹, R Abbott¹, T D Abbott², M R Abernathy², F Acernese³,⁴, K Ackley⁵, M Adamo⁴,⁶, C Adams⁷, T Adams⁸, P Addesso³  Show full author list

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

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
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
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 random data
plt.imshow(output);  plt.colorbar();   plt.show()
Input is $0 \leq x \leq 1$, so output must be 

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

$f(x) = \int_{0}^{1} f(x) \, dx \approx 0.223$

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')
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
# 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

Current Branches Build History Pull Requests Build #142

✓ 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

Build Jobs

<table>
<thead>
<tr>
<th>Build #</th>
<th>Python</th>
<th>Environment Variables</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>#142.1</td>
<td>2.7</td>
<td>no</td>
<td>2 min 30 sec</td>
</tr>
<tr>
<td>#142.2</td>
<td>3.4</td>
<td>no</td>
<td>2 min 50 sec</td>
</tr>
<tr>
<td>#142.3</td>
<td>3.5</td>
<td>no</td>
<td>2 min 30 sec</td>
</tr>
</tbody>
</table>
## Update docs flag

### Files

<table>
<thead>
<tr>
<th>Files</th>
<th>▪️</th>
<th>▫️</th>
<th>▪️</th>
<th>▪️</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>r3d2/<strong>init</strong>.py</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>100.00%</td>
</tr>
<tr>
<td>r3d2/eos_defns.py</td>
<td>53</td>
<td>51</td>
<td>0</td>
<td>2</td>
<td>96.22%</td>
</tr>
<tr>
<td>r3d2/riemann_problem.py</td>
<td>38</td>
<td>34</td>
<td>0</td>
<td>4</td>
<td>89.47%</td>
</tr>
<tr>
<td>r3d2/state.py</td>
<td>37</td>
<td>37</td>
<td>0</td>
<td>0</td>
<td>100.00%</td>
</tr>
<tr>
<td>r3d2/utils.py</td>
<td>29</td>
<td>24</td>
<td>0</td>
<td>5</td>
<td>82.75%</td>
</tr>
<tr>
<td>r3d2/wave.py</td>
<td>345</td>
<td>320</td>
<td>0</td>
<td>25</td>
<td>92.75%</td>
</tr>
<tr>
<td>tests/test_eos.py</td>
<td>51</td>
<td>51</td>
<td>0</td>
<td>0</td>
<td>100.00%</td>
</tr>
<tr>
<td>tests/test_riemann_problem.py</td>
<td>132</td>
<td>132</td>
<td>0</td>
<td>0</td>
<td>100.00%</td>
</tr>
<tr>
<td>tests/test_state.py</td>
<td>25</td>
<td>25</td>
<td>0</td>
<td>0</td>
<td>100.00%</td>
</tr>
<tr>
<td>tests/test-utils.py</td>
<td>26</td>
<td>26</td>
<td>0</td>
<td>0</td>
<td>100.00%</td>
</tr>
</tbody>
</table>
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

\[
\begin{align*}
\frac{\partial}{\partial t} \begin{pmatrix} \rho \\ u \\ v \\ w \end{pmatrix} + \frac{\partial}{\partial x} \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho w \end{pmatrix} &= 0.
\end{align*}
\]

For further details on this system of equations, see the Living Review of Marti and Müller, particularly section 3.1 for the equations and section 8.6 for the solution of the Riemann Problem.

The initial data is piecewise constant: two states \( w_{L,R} \) are specified, each in terms of \( w = (\rho_0, v_x, v_y, c) \), (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 \( w_X \) 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
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 $w = (\rho, v_x, v_t, e)$ (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)
est_1_U_left = State(10.0, 0.0, 0.0, 2.0, eos, label="L")
est_1_U_right = State(1.0, 0.0, 0.0, 1.5e-6, eos, label="R")
est_1_rp = RiemannProblem(est_1_U_left, est_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](http://www.software.ac.uk) for more