# *Scientific Tests* and Continuous Integration Strategies to Enhance Reproducibility in the Scientific Software Context

Matthew Krafczyk, August Shi, Adhithya Bhaskar, Darko Marinov, Victoria Stodden

**Victoria Stodden**
Associate Professor, School of Information Sciences and NCSA
University of Illinois at Urbana-Champaign

**2nd International Workshop on Practical Reproducible Evaluation of Computer Systems (P-RECS19)**
Co-located with the 28th International Symposium on High-Performance Parallel and Distributed Computing (ACM HPDC 2019)
**Phoenix, Arizona, USA - June 24th, 2019**

# Parsing Reproducibility

- **Empirical Reproducibility:**
  - traditional empirical experiments, e.g. at the bench/lab
- **Statistical Reproducibility**:
  - statistical methodology used permits generalizability of data inferences
- **Computational Reproducibility:**
  - transparency of computational steps that produce scientific findings

**V. Stodden. (2013).** ***Resolving Irreproducibility in Empirical and Computational Research.*** **IMS Bulletin**

# Dissemination of Computational- and Data-enabled Discoveries

***The software contains "ideas that enable biology..."***
*Lior Pachter, Stories from the Supplement, 2013*

CSHL Keynote; Dr. Lior Pachter, UC Berkeley

- Virtually all published discoveries today have **data** and **computational components**
- We see a mismatch between traditional research dissemination standards and modern computational research practice, leading to **reproducibility concerns**

# What Barriers Exist to Computational Reproducibility? Can Software Tests Help?

- In a previous study we attempted to computationally reproduce 55 studies published in the Journal of Computational Physics (none were successful) (V. Stodden, M. Krafczyk, A. Bhaskar, *Enabling the Verification of Computational Results*, P-RECS 2018)

- We evaluated the studies using the *ICERM Criteria* (V. Stodden, D. H. Bailey, J. Borwein, R. J. LeVeque, W. Rider, and W. Stein. (2013). S*etting the Default to Reproducible: Reproducibility in Computational and Experimental Mathematics*, ICERM workshop 2013)

- We chose 2 that appeared most likely to replicate given 40 hours of effort

# The Articles

1. E. Treister and E. Haber. 2016. *A fast marching algorithm for the factored eikonal equation*. J. Comput. Phys. 324 (2016), 210 – 225. https://doi.org/10.1016/j.jcp.2016.08.012

2. M. A. Hernández and M. J. Rubio. 2004. *A modification of Newton's method for nondifferentiable equations*. J. Comput. Appl. Math. 164-165 (2004), 409 – 417. https://doi.org/10.1016/S0377-0427(03)00650-2

For each article, we enumerated a list of figures and tables and used existing code or wrote code to reproduce each, stopping after 40 hours.

# Treister and Haber (T&H)

The released code contained a script `runExperiments.jl` that reproduced Tables 1-3 from the article:

**Table 3**
Results for the 2D Gaussian factor test case (test case 3). The error measures are in the [$l_\infty$, mean $l_2$] norms.

| $h$ | $n$ | $1^{st}$ order | | $2^{nd}$ order | |
|---|---|---|---|---|---|
| | | error in $\tau$ | time (work) | error in $\tau$ | time (work) |
| 1/40 | $161 \times 321$ | [6.15e−03, 3.86e−03] | 0.05 s (205) | [1.60e−04, 5.94e−05] | 0.05 s (236) |
| 1/80 | $321 \times 641$ | [3.07e−03, 1.93e−03] | 0.21 s (221) | [3.85e−05, 1.56e−05] | 0.20 s (206) |
| 1/160 | $641 \times 1281$ | [1.54e−03, 9.67e−04] | 0.87 s (223) | [1.08e−05, 4.03e−06] | 0.86 s (221) |
| 1/320 | $1281 \times 2561$ | [7.68e−04, 4.83e−04] | 3.80 s (259) | [3.18e−06, 1.04e−06] | 3.88 s (265) |
| 1/640 | $2561 \times 5121$ | [3.84e−04, 2.42e−04] | 16.2 s (275) | [9.59e−07, 2.66e−07] | 16.6 s (280) |
| 1/1280 | $5121 \times 10241$ | [1.92e−04, 1.21e−04] | 73.8 s (304) | [2.99e−07, 6.88e−08] | 74.6 s (307) |

| $h$ | $n$ | $1^{st}$ order | | $2^{nd}$ order | |
|---|---|---|---|---|---|
| | | error in $\tau$ | time (work) | error in $\tau$ | time (work) |
| **1/40** | $161 \times 321$ | [6.15e-03,3.86e-03] | 0.06s (309.15) | [1.60e-04,5.94e-05] | 0.06s (309.86) |
| **1/80** | $321 \times 641$ | [1.54e-03,9.67e-04] | 0.25s (329.09) | [3.85e-05,1.56e-05] | 0.25s (298.17) |
| **1/160** | $641 \times 1281$ | [1.54e-03,9.67e-04] | 1.02s (329.09) | [1.08e-05,4.03e-06] | 1.09s (351.82) |
| **1/320** | $1281 \times 2561$ | [7.68e-04,4.83e-04] | 4.45s (362.82) | [3.18e-06,1.04e-06] | 4.58s (373.24) |
| **1/640** | $2561 \times 5121$ | [3.84e-04,2.42e-04] | 18.53s (371.54) | [9.59e-07,2.66e-07] | 19.06s (382.16) |
| **1/1280** | $5121 \times 10241$ | [1.92e-04,1.21e-04] | 77.53s (387.00) | [2.99e-07,6.88e-08] | 80.79s (403.26) |

This script crashed during the higher resolution experiments and other figures were missing critical data

# Hernández and Rubio (H&R)

No code was released and their results were produced to machine precision in Octave using code we wrote.

# *Scientific Tests*

- With both articles of interest successfully reproduced, constructed testing scripts that ran the whole procedure:
    - on our local machines and,
    - within a Docker container,
    - and within the Travis CI environment.

- We define these tests as *scientific tests*: tests that produce computational results from a published article.

- Scientific tests apply the notion of the *black-box test* in the scientific context, where the most expansive scientific test is to reproduce all of the results from the article.
- Within Travis CI we only run experiments that can complete in less than 5 minutes.

# The Test Script

- a master script in python, **run.sh**, coordinates all aspects of the experiment from start to finish. It calls subordinate scripts to perform different stages of the work (e.g. running the computational experiments and comparing results).
- we produced a **.travis.yml** file that uses the Ubuntu OS's repositories to install all necessary software, and then executes the run.sh script.

➡ Computational experiments are run, their results are checked, and Travis CI reports success or failure status.

See e.g. (H&R): https://raw.githubusercontent.com/ReproducibilityInPublishing/10.1016_S0377-0427-03-00650-2/master/run.sh and https://raw.githubusercontent.com/ReproducibilityInPublishing/10.1016_S0377-0427-03-00650-2/master/.travis.yml

# Problems Encountered

1. older software versions used at the time of publication, e.g. Julia 0.6

2. scientific variables that differ near machine precision can affect output, so a tolerance used in comparisons

3. when article code was released it was not organized to reproduce results in the articles.

4. For figure comparisons visualization software is often not well documented, and even slight plot color differences can affect comparisons.

In general, code can evolve with time, introducing changes into necessary tools or **dependencies** that can change the output. Operations on **floating-point numbers** can introduce imprecision between different runs. **Multi-threaded algorithms** can be inherently non-deterministic, producing differences between calculated and published results.

# Specialized Hardware and *Reduction Tests*

1. Some articles may use specialized hardware and require an extreme amount of time and resources.

2. In this case, we recommend publishing results from *minimized* versions of their computational experiments together with their main results: e.g. mimic the original work as much as possible but at a smaller scale.

3. Examples: scale down resolution and/or reduce how often snapshots are saved or reduce resolution or time steps.

4. A Reduction Test can permit serious science bugs to be detected quickly.

# *Reduction Test* Example (T&H)

Recall Figure 3 from T&T and our reproduction:

**Table 3**
Results for the 2D Gaussian factor test case (test case 3). The error measures are in the [$l_\infty$, mean $l_2$] norms.

| $h$ | $n$ | $1^{st}$ order | | $2^{nd}$ order | |
|---|---|---|---|---|---|
| | | error in $\tau$ | time (work) | error in $\tau$ | time (work) |
| 1/40 | $161 \times 321$ | [6.15e−03, 3.86e−03] | 0.05 s (205) | [1.60e−04, 5.94e−05] | 0.05 s (236) |
| 1/80 | $321 \times 641$ | [3.07e−03, 1.93e−03] | 0.21 s (221) | [3.85e−05, 1.56e−05] | 0.20 s (206) |
| 1/160 | $641 \times 1281$ | [1.54e−03, 9.67e−04] | 0.87 s (223) | [1.08e−05, 4.03e−06] | 0.86 s (221) |
| 1/320 | $1281 \times 2561$ | [7.68e−04, 4.83e−04] | 3.80 s (259) | [3.18e−06, 1.04e−06] | 3.88 s (265) |
| 1/640 | $2561 \times 5121$ | [3.84e−04, 2.42e−04] | 16.2 s (275) | [9.59e−07, 2.66e−07] | 16.6 s (280) |
| 1/1280 | $5121 \times 10241$ | [1.92e−04, 1.21e−04] | 73.8 s (304) | [2.99e−07, 6.88e−08] | 74.6 s (307) |

| $h$ | $n$ | $1^{st}$ order | | $2^{nd}$ order | |
|---|---|---|---|---|---|
| | | error in $\tau$ | time (work) | error in $\tau$ | time (work) |
| **1/40** | $161 \times 321$ | [6.15e-03,3.86e-03] | 0.06s (309.15) | [1.60e-04,5.94e-05] | 0.06s (309.86) |
| **1/80** | $321 \times 641$ | [1.54e-03,9.67e-04] | 0.25s (329.09) | [3.85e-05,1.56e-05] | 0.25s (298.17) |
| **1/160** | $641 \times 1281$ | [1.54e-03,9.67e-04] | 1.02s (329.09) | [1.08e-05,4.03e-06] | 1.09s (351.82) |
| **1/320** | $1281 \times 2561$ | [7.68e-04,4.83e-04] | 4.45s (362.82) | [3.18e-06,1.04e-06] | 4.58s (373.24) |
| **1/640** | $2561 \times 5121$ | [3.84e-04,2.42e-04] | 18.53s (371.54) | [9.59e-07,2.66e-07] | 19.06s (382.16) |
| **1/1280** | $5121 \times 10241$ | [1.92e-04,1.21e-04] | 77.53s (387.00) | [2.99e-07,6.88e-08] | 80.79s (403.26) |

Notice our computational costs increases for each experiment at a faster rate than that reported in Table 3.

# *Reduction Test* Example (T&H)

Travis CI uses `stages` to define different test categories, executed in sequence.

Certain stages can be flagged to not trigger a build failure when they do fail. This technique can be harnessed to create two test groups:

1. A first group completes quickly, letting the authors know about obvious bugs, and reduction tests can be fit here.

2. A second group of tests can take much longer to complete performing more thorough analysis. Scientific tests that require a large amount of time, but for which authors do not want to wait for their completion during the software development cycle, are good candidates here.

We grouped reproduction of all but the last two rows of Tables 1–6 as the *reduction tests* in this paradigm, and the last two rows of each table as the set of long-running *scientific tests*.

# *Reduction Test* Example (T&H)

Travis CI uses `stages` to define different test categories, executed in sequence.

Certain stages can be flagged to not trigger a build failure when they do fail. This technique can be harnessed to create two test groups:

1. A first group completes quickly, letting the authors know about obvious bugs, and reduction tests can be fit here.

2. A second group of tests can take much longer to complete performing more thorough analysis. Scientific tests that require a large amount of time, but for which authors do not want to wait for their completion during the software development cycle, are good candidates here.

We grouped reproduction of all but the last two rows of Tables 1–6 as the *reduction tests* in this paradigm, and the last two rows of each table as the set of long-running *scientific tests*.

# Thank you and Links

**T&H:**

– Code and data: https://github.com/ReproducibilityInPublishing/j.jcp.2016.08.012. (Commit ba16911 at time of publication)

– TravisCI: https://travis-ci.org/ReproducibilityInPublishing/j.jcp.2016.08.012

**H&R:**

– Code and data: https://github.com/ReproducibilityInPublishing 10.1016_S0377-0427-03-00650-2. (Commit 227b842 at time of publication)

– TravisCI: https://travis-ci.org/ReproducibilityInPublishing/ 10.1016_S0377-0427-03-00650-2