Open & reproducible research - What can we do in practice?

Presented by

Felix Z. Hoffmann

@Felix11H

felix11h.github.io/

Resources and links

Open Science Fellows Program: bit.ly/osfprog

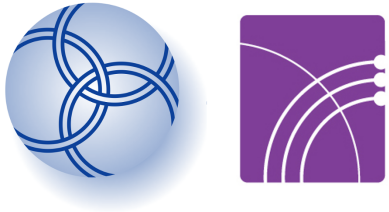
project description: bit.ly/osfproj

prototype: bit.ly/osrep

Who am I?

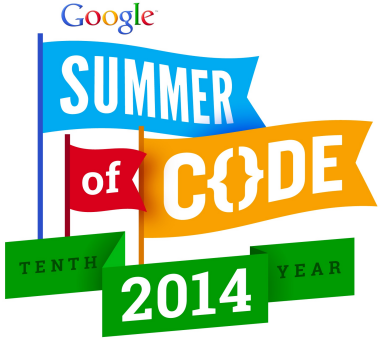
PhD student with Prof. Jochen Triesch

> *Computational models of structural plasticity*



Google Summer of Code 2014

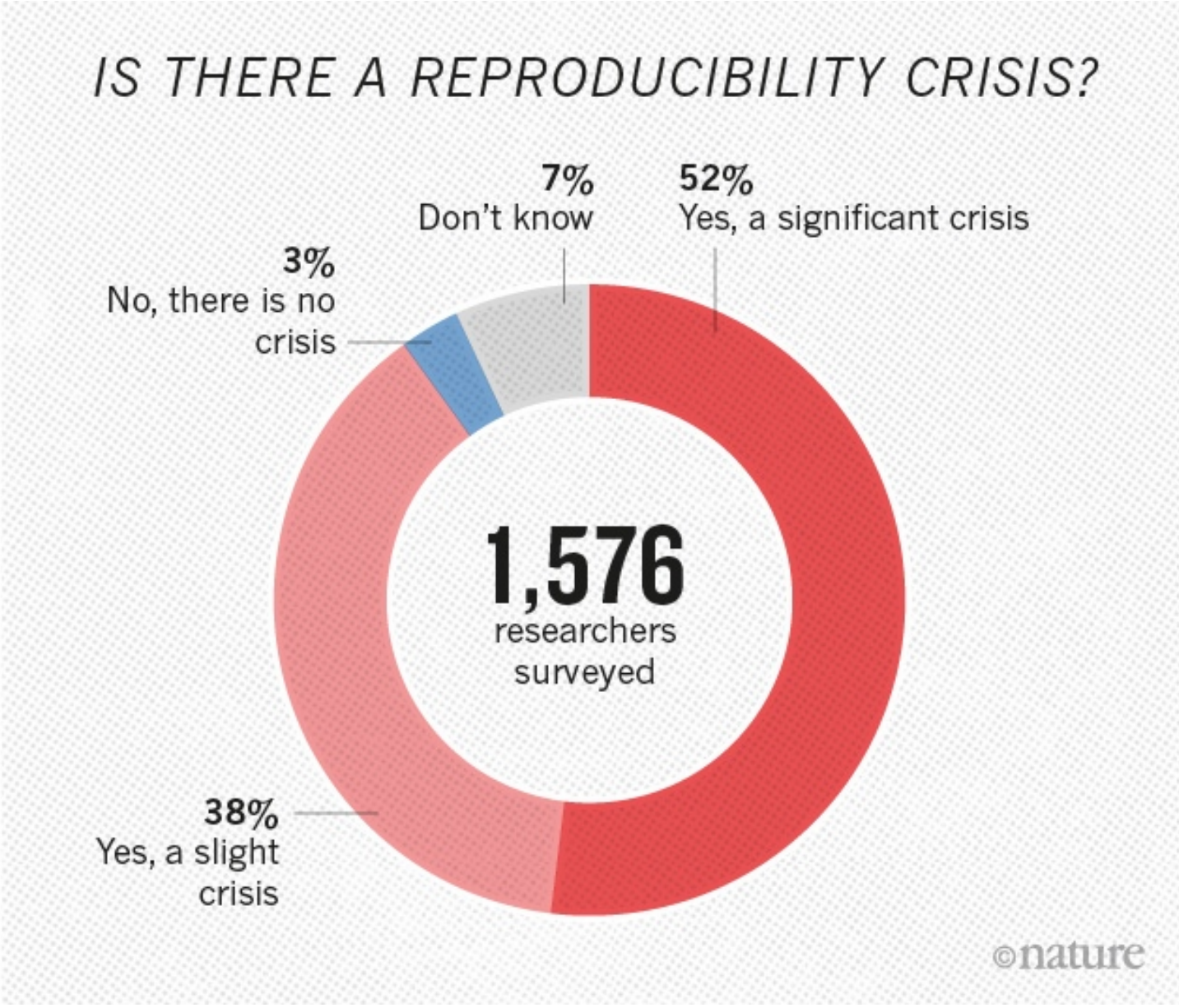
> *Data-centric views in Sumatra*



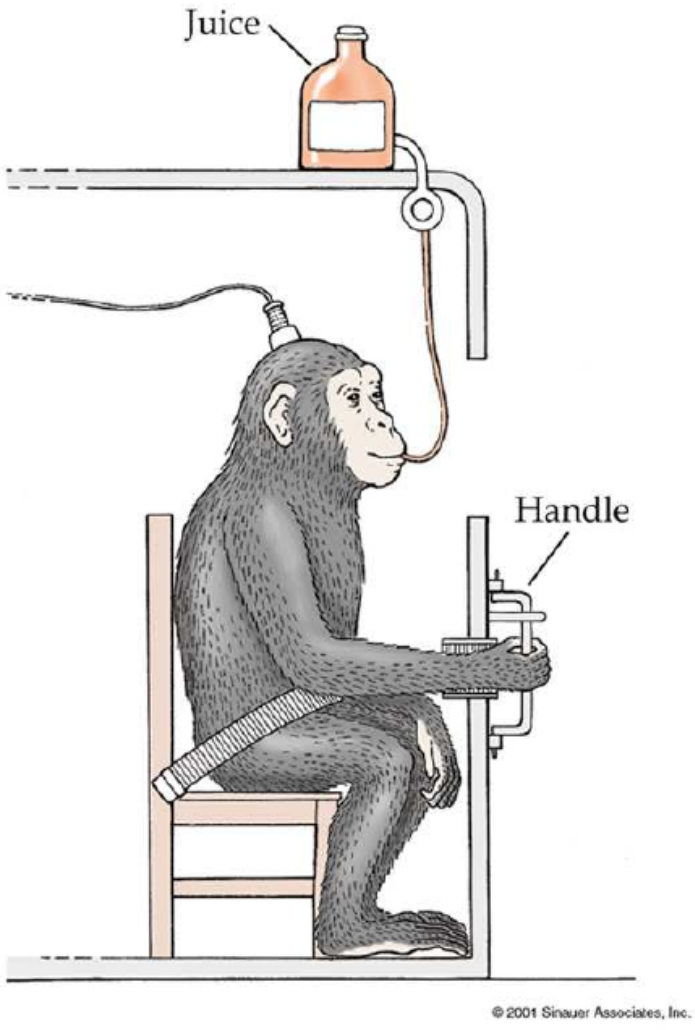
Wikimedia Open Science Fellow 2017/2018

> *Open computational research study*

The reproducibility crisis



Computational reproducibility should be easy...



Hard

Easy?

Computational reproducibility should be easy...

1. cheap and universal access to computers (as opposed to a lab)
2. running code is inexpensive and unproblematic (compared to replicating an experiment)
3. can easily share code & data that allow direction reproduction

Is there a reproducibility crisis in computational research?

Sharing of code & data mandatory in many journals

All data necessary to understand, assess, and extend the conclusions of the manuscript must be available to any reader of Science. All computer codes involved in the creation or analysis of data must also be available to any reader of Science. After publication, all reasonable requests for data and materials must be fulfilled. Any restrictions on the availability of data, codes, or materials, including fees and original data obtained from other sources (Materials Transfer Agreements), must be disclosed to the editors upon submission...

Policy of *Science* since February 11, 2011

Sharing of code & data mandatory in many journals

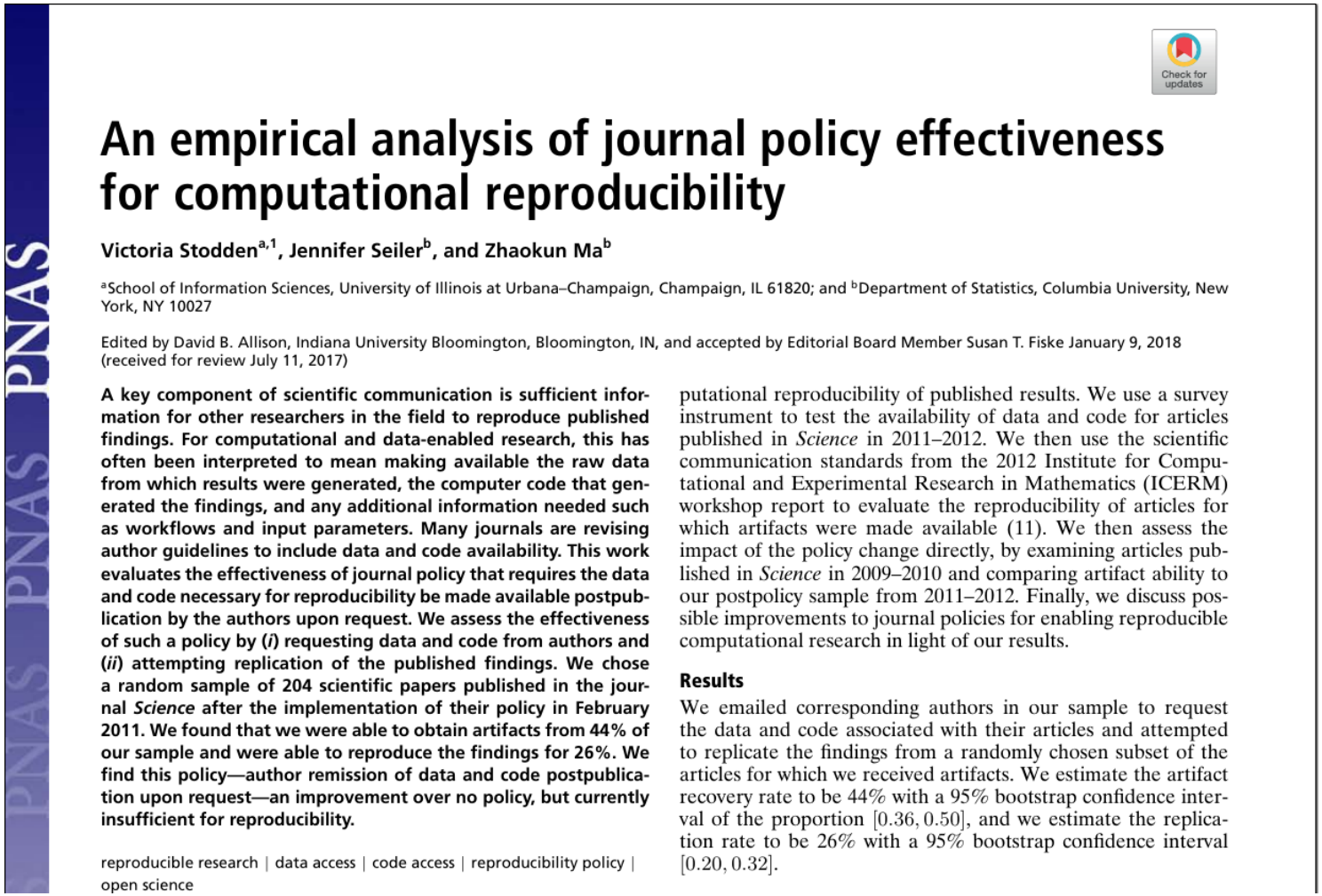
All data necessary to understand, assess, and extend the conclusions of the manuscript must be available to any reader of Science. All computer codes involved in the creation or analysis of data must also be available to any reader of Science. After publication, all reasonable requests for data and materials must be fulfilled. Any restrictions on the availability of data, codes, or materials, including fees and original data obtained from other sources (Materials Transfer Agreements), must be disclosed to the editors upon submission...

Policy of *Science* since February 11, 2011

Sharing of code & data mandatory in many journals

All data necessary to understand, assess, and extend the conclusions of the manuscript must be available to any reader of Science. All computer codes involved in the creation or analysis of data must also be available to any reader of Science. After publication, all reasonable requests for data and materials must be fulfilled. Any restrictions on the availability of data, codes, or materials, including fees and original data obtained from other sources (Materials Transfer Agreements), must be disclosed to the editors upon submission...

Policy of *Science* since February 11, 2011



An empirical analysis of journal policy effectiveness for computational reproducibility

Victoria Stodden[a,1], **Jennifer Seiler**[b], **and Zhaokun Ma**[b]

[a]School of Information Sciences, University of Illinois at Urbana–Champaign, Champaign, IL 61820; and [b]Department of Statistics, Columbia University, New York, NY 10027

Edited by David B. Allison, Indiana University Bloomington, Bloomington, IN, and accepted by Editorial Board Member Susan T. Fiske January 9, 2018 (received for review July 11, 2017)

A key component of scientific communication is sufficient information for other researchers in the field to reproduce published findings. For computational and data-enabled research, this has often been interpreted to mean making available the raw data from which results were generated, the computer code that generated the findings, and any additional information needed such as workflows and input parameters. Many journals are revising author guidelines to include data and code availability. This work evaluates the effectiveness of journal policy that requires the data and code necessary for reproducibility be made available postpublication by the authors upon request. We assess the effectiveness of such a policy by (*i*) requesting data and code from authors and (*ii*) attempting replication of the published findings. We chose a random sample of 204 scientific papers published in the journal *Science* after the implementation of their policy in February 2011. We found that we were able to obtain artifacts from 44% of our sample and were able to reproduce the findings for 26%. We find this policy—author remission of data and code postpublication upon request—an improvement over no policy, but currently insufficient for reproducibility.

reproducible research | data access | code access | reproducibility policy | open science

putational reproducibility of published results. We use a survey instrument to test the availability of data and code for articles published in *Science* in 2011–2012. We then use the scientific communication standards from the 2012 Institute for Computational and Experimental Research in Mathematics (ICERM) workshop report to evaluate the reproducibility of articles for which artifacts were made available (11). We then assess the impact of the policy change directly, by examining articles published in *Science* in 2009–2010 and comparing artifact ability to our postpolicy sample from 2011–2012. Finally, we discuss possible improvements to journal policies for enabling reproducible computational research in light of our results.

Results

We emailed corresponding authors in our sample to request the data and code associated with their articles and attempted to replicate the findings from a randomly chosen subset of the articles for which we received artifacts. We estimate the artifact recovery rate to be 44% with a 95% bootstrap confidence interval of the proportion $[0.36, 0.50]$, and we estimate the replication rate to be 26% with a 95% bootstrap confidence interval $[0.20, 0.32]$.

Out of 206 computational studies in *Science* since 2011,
26 provided code & data directly

Stodden et al. 2018

A few responses...

When you approach a PI for the source codes and raw data, you better explain who you are, whom you work for, why you need the data and what you are going to do with it.

I have to say that this is a very unusual request without any explanation! Please ask your supervisor to send me an email with a detailed, and I mean detailed, explanation.

We do not typically share our internal data or code with people outside our collaboration.

Stodden et al. 2018

A few responses...

When you approach a PI for the source codes and raw data, you better explain who you are, whom you work for, why you need the data and what you are going to do with it.

I have to say that this is a very unusual request without any explanation! Please ask your supervisor to send me an email with a detailed, and I mean detailed, explanation.

We do not typically share our internal data or code with people outside our collaboration.

Stodden et al. 2018

A few responses...

When you approach a PI for the source codes and raw data, you better explain who you are, whom you work for, why you need the data and what you are going to do with it.

I have to say that this is a very unusual request without any explanation! Please ask your supervisor to send me an email with a detailed, and I mean detailed, explanation.

We do not typically share our internal data or code with people outside our collaboration.

Stodden et al. 2018

Of $N = 206$ articles published in *Science* since 2011...

Table 1. Responses to emailed requests (n = 180)

Type of response	Count	Percent, %
Did not share data or code:		
Contact another person	20	11
Asked for reasons	20	11
Refusal to share	12	7
Directed back to supplement	6	3
Unfulfilled promise to follow up	5	3
Impossible to share	3	2
Shared data and code	65	36
Email bounced	3	2
No response	46	26

$\Rightarrow$ Code & data could be retrieved for 91 out of 206 studies.

Stodden et al. 2018

Open Source for Neuroscience

An Open Letter Committing to Open Source in Neuroscience

This Open Letter is the outcome of the Sept 2016 conference at Janelia Research Campus on Collaborative Development of Data-Driven Models of Neural Systems. The signatories of the letter commit to making important parts of their research output open source and will require the same level of openness in work they are asked to review.

This initiative has been described in a recent NeuroView article in Neuron: **A Commitment to Open Source in Neuroscience, Gleeson et al. 2017**.

Sign the Letter | FAQs | Contact

Dear colleagues,

Neuroscientists are increasingly relying on custom built software to help analyze their experimental data and to construct and run models. Packages for extracting, translating, analyzing and visualising data, as well as scripts for modelling and simulating the mechanisms underlying the examined phenomena, are an essential part of the work behind many publications in the field today. These scripts can often be complex and involve many processing steps which can't be fully described in the accompanying publications. Making these publicly available would increase the reproducibility and scientific rigour of the results described. At present though, releasing these is generally not a prerequisite for publication.

A distributed, open and freely available network of tools, databases, and related

Open Source for Neuroscience

Towards that end, we pledge to release promptly, completely, and freely all computer code, model scripts, and parameters necessary to reproduce the analyses and simulations from any of our new publications. We will make all software applications (tools, libraries, etc.) we develop for experimental data analysis or model construction open source at time of publication, whether or not the application is the main subject of the paper.

1. Giorgio A. Ascoli (George Mason University, Fairfax, VA, USA)
2. Andrew P. Davison (CNRS, Gif-sur-Yvette, France)
3. Padraig Gleeson (University College, London, UK)
4. R. Angus Silver (University College, London, UK)
5. Kim T. Blackwell (George Mason University, Fairfax, VA, United States)
6. Sharon Crook (Arizona State University, United States)
7. Ivan Soltesz (Stanford University, United States)
8. Salvador Dura-Bernal (SUNY Downstate, United States)
9. Malin Sandström (INCF, Sweden)

States)

99. Peter Groblewski (Allen Institute for Brain Science, United States)
100. Matthew Valley (Allen Institute for Brain Science, United States)
101. xiaoxuan jia (Allen Institute, United States)
102. Josh Siegle (Allen Institute, United States)
103. Yazan N. Billeh (Allen Institute for Brain Science, United States)
104. Rui Liu (Allen Institute for Brain Science, United States)
105. Anatoly Buchin (ALLEN INSTITUTE FOR BRAIN SCIENCE, United States)

opensourceforneuroscience.org/

Open Source for Neuroscience

Towards that end, we pledge to release promptly, completely, and freely all computer code, model scripts, and parameters necessary to reproduce the analyses and simulations from any of our new publications. We will make all software applications (tools, libraries, etc.) we develop for experimental data analysis or model construction open source at time of publication, whether or not the application is the main subject of the paper.

1. Giorgio A. Ascoli (George Mason University, Fairfax, VA, USA)
2. Andrew P. Davison (CNRS, Gif-sur-Yvette, France)
3. Padraig Gleeson (University College, London, UK)
4. R. Angus Silver (University College, London, UK)
5. Kim T. Blackwell (George Mason University, Fairfax, VA, United States)
6. Sharon Crook (Arizona State University, United States)
7. Ivan Soltesz (Stanford University, United States)
8. Salvador Dura-Bernal (SUNY Downstate, United States)
9. Malin Sandström (INCF, Sweden)

States)

99. Peter Groblewski (Allen Institute for Brain Science, United States)
100. Matthew Valley (Allen Institute for Brain Science, United States)
101. xiaoxuan jia (Allen Institute, United States)
102. Josh Siegle (Allen Institute, United States)
103. Yazan N. Billeh (Allen Institute for Brain Science, United States)
104. Rui Liu (Allen Institute for Brain Science, United States)
105. Anatoly Buchin (ALLEN INSTITUTE FOR BRAIN SCIENCE, United States)

opensourceforneuroscience.org/

Reproducibility when code was available

Tried to reproduce randomly selected 22 studies (out of 56 that were judged as potentially reproducible)

Stodden et al. 2018

Reproducibility when code was available

Tried to reproduce randomly selected 22 studies (out of 56 that were judged as potentially reproducible)

> Even when code was available, more than half of studies were reproducible only with *significant effort!*

Stodden et al. 2018

Reproducibility when code was available

Tried to reproduce randomly selected 22 studies (out of 56 that were judged as potentially reproducible)

> Even when code was available, more than half of studies were reproducible only with *significant effort!*

Problems:

- impossible to reproduce (missing code, data or methodology)

Stodden et al. 2018

Reproducibility when code was available

Tried to reproduce randomly selected 22 studies (out of 56 that were judged as potentially reproducible)

> Even when code was available, more than half of studies were reproducible only with *significant effort!*

Problems:

- impossible to reproduce (missing code, data or methodology)
- required tedious effort (e.g. download large number of individual data sets)

Stodden et al. 2018

Reproducibility when code was available

Tried to reproduce randomly selected 22 studies (out of 56 that were judged as potentially reproducible)

> Even when code was available, more than half of studies were reproducible only with *significant effort!*

Problems:

- impossible to reproduce (missing code, data or methodology)
- required tedious effort (e.g. download large number of individual data sets)
- required intellectual effort (e.g. knowledge of past articles, implementing given pseudo code)

Stodden et al. 2018

Reproducibility when code was available

Tried to reproduce randomly selected 22 studies (out of 56 that were judged as potentially reproducible)

> Even when code was available, more than half of studies were reproducible only with *significant effort!*

Problems:

- impossible to reproduce (missing code, data or methodology)
- required tedious effort (e.g. download large number of individual data sets)
- required intellectual effort (e.g. knowledge of past articles, implementing given pseudo code)
- required tweaking (e.g. mising parameters, minor methods steps)

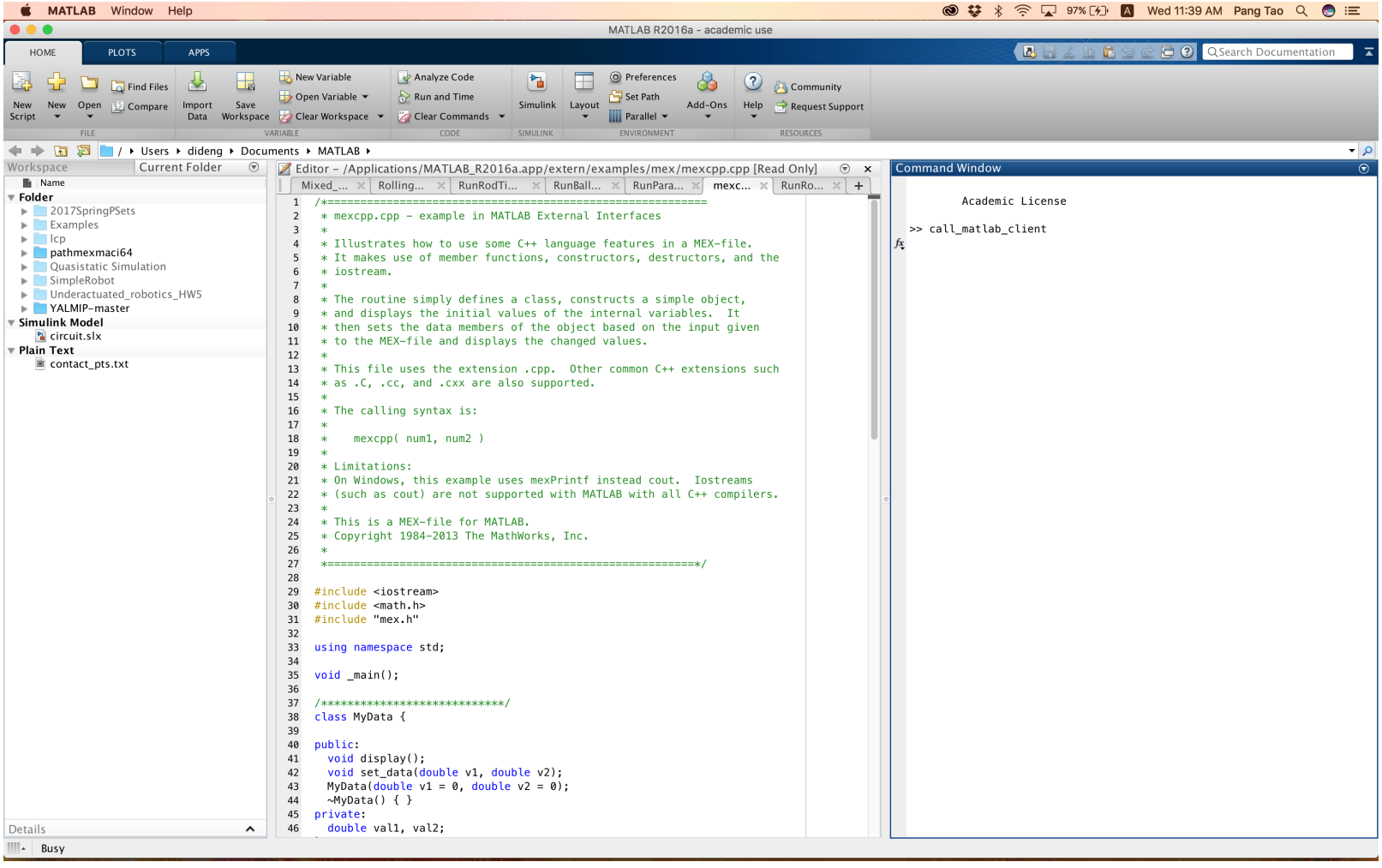
Stodden et al. 2018

Reproducibility when code was available

Table 4. Classification of reproducibility effort (*n* = 22)

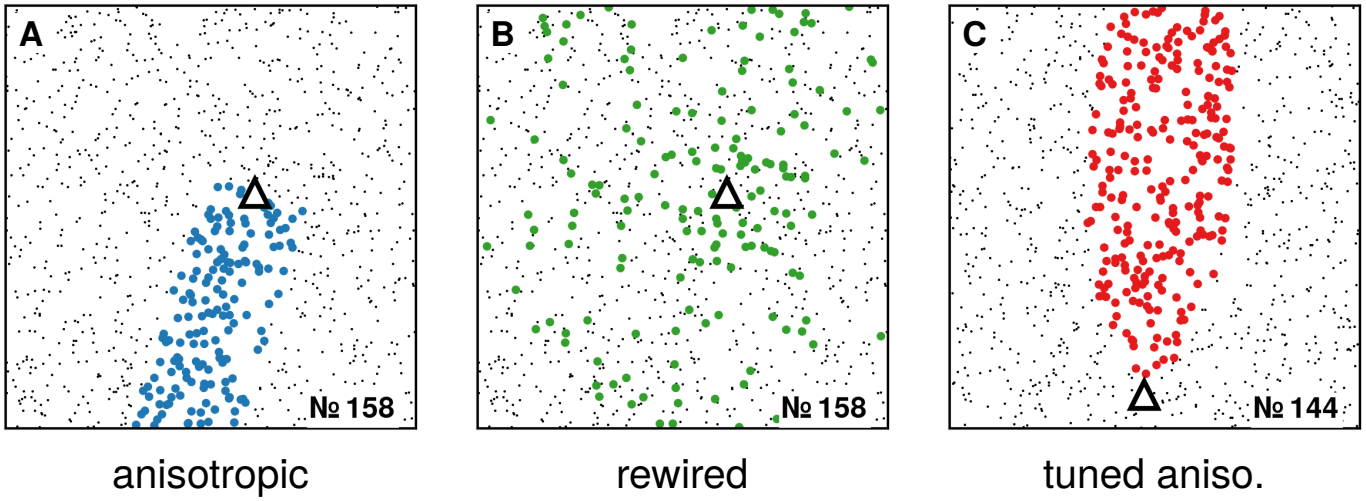
Classification	Percent, %
Reproducible after tweaking (e.g., missing parameters required fiddling to find, missing modified code lines, missing arguments required for differing architecture; missing minor method step)	5
Minor difficulty in reproducing (e.g., installing a specialized library, converting to a different computational system)	18
Straightforward to reproduce with minimal effort	14

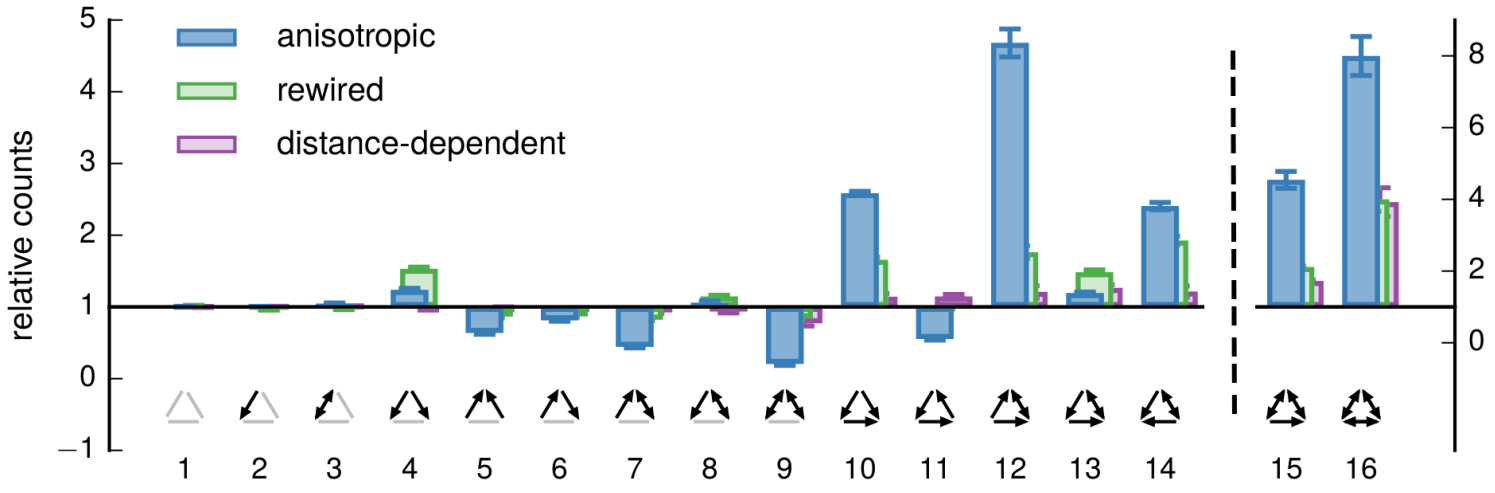
Computational reproducibility remains difficult even when code is available!



How to publish our research so that (computational) results are reproducible?

A complex computational study - Reproducible?





Open Science Fellowship

Open Science Fellowship

- runs from October to June
- fellows are paired with a mentor who supports the progress
- financial support provided
- training seminars & work-shops
- program in German

Open Science Fellowship

Photo: Ralf Rebmann, CC BY-SA 4.0

Open Science Fellowship

application for 3rd round open: bit.ly/osfprog

Problems to solve

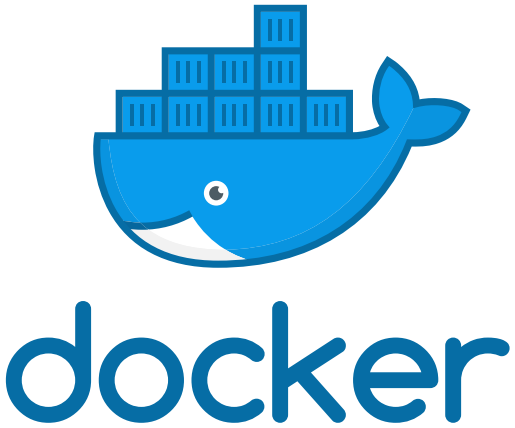
Problem 1

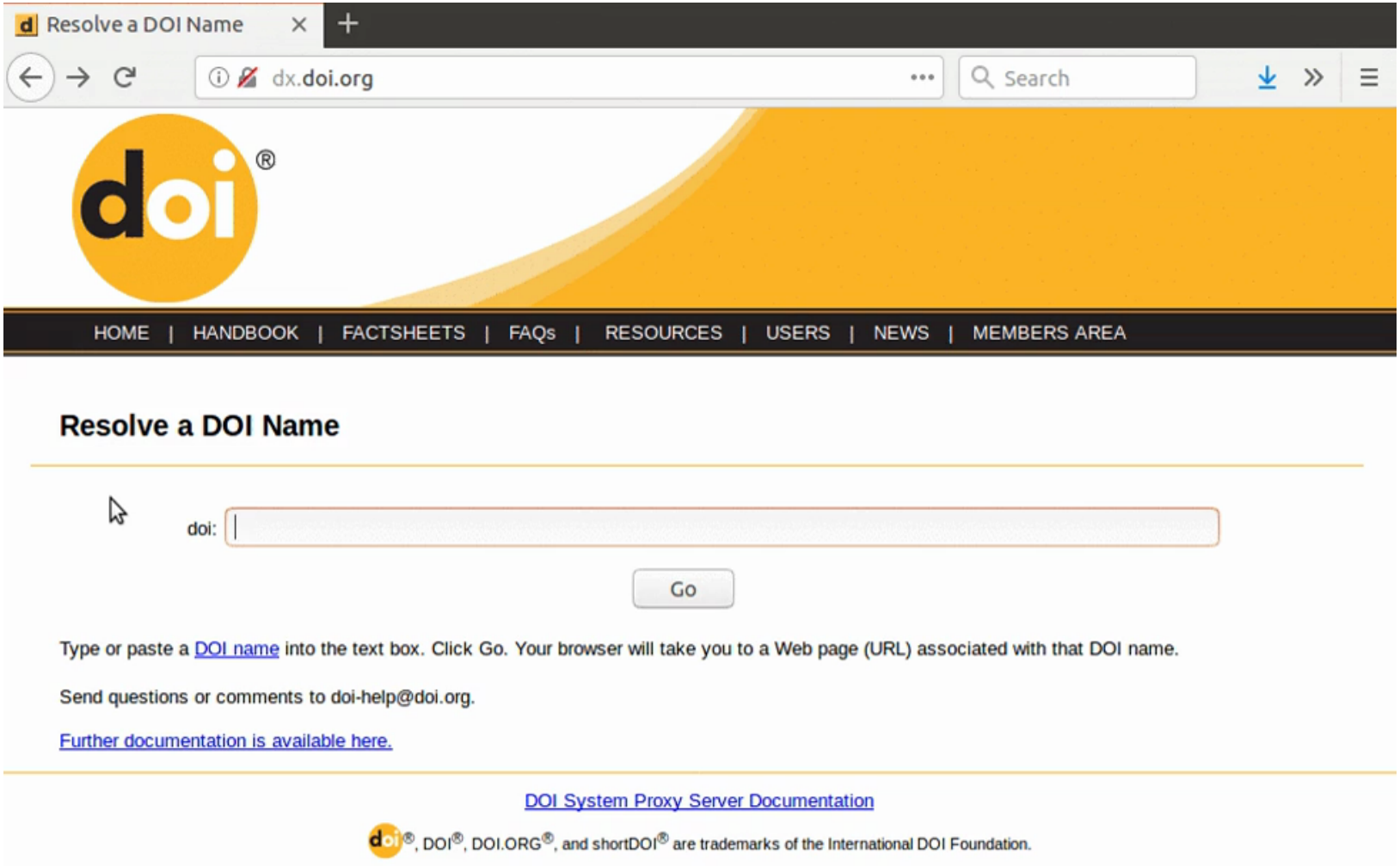
- using of difficult to install computational environment (graph-tool)

Problems to solve

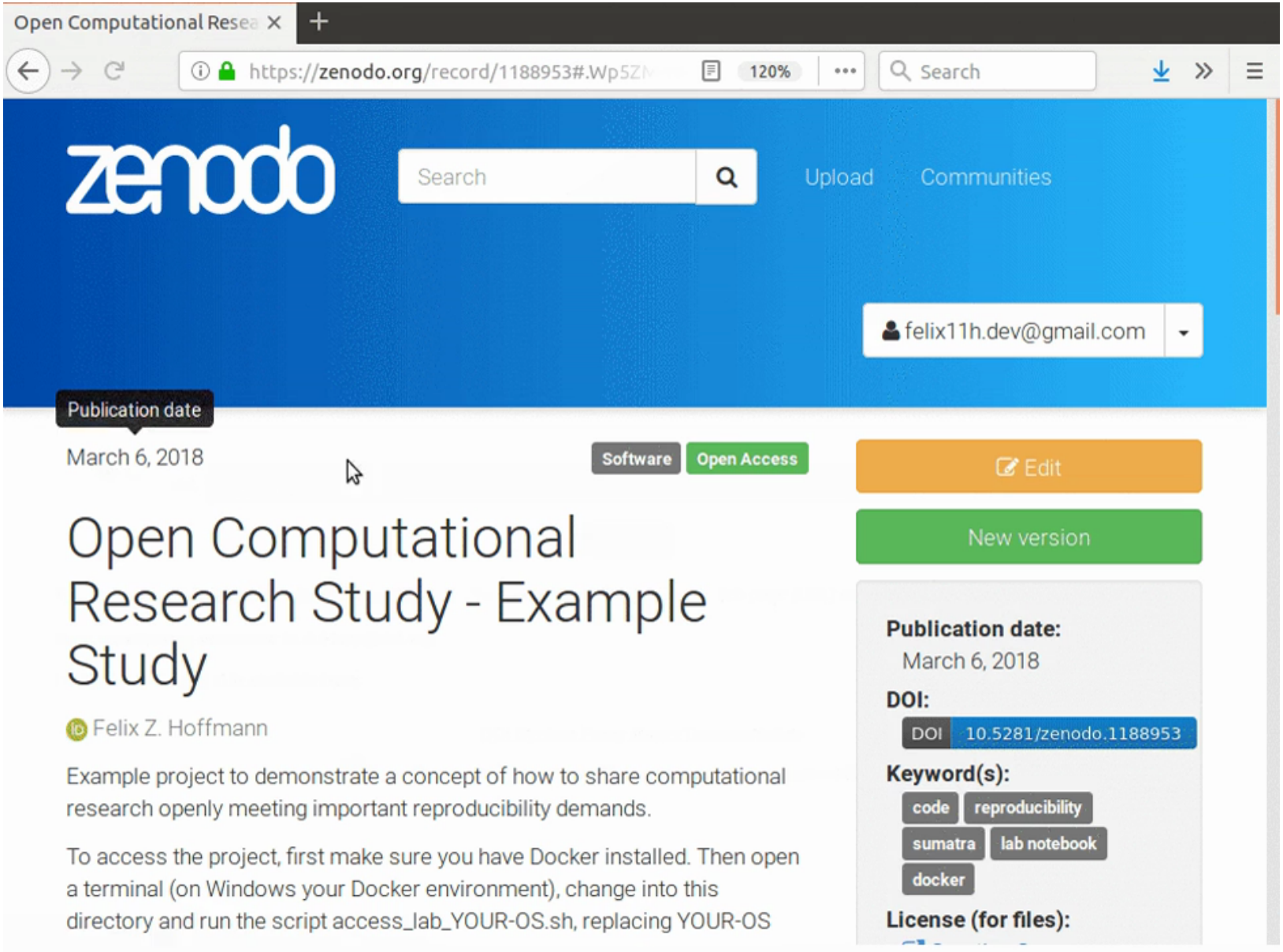
Problem 1

- using of difficult to install computational environment (graph-tool)

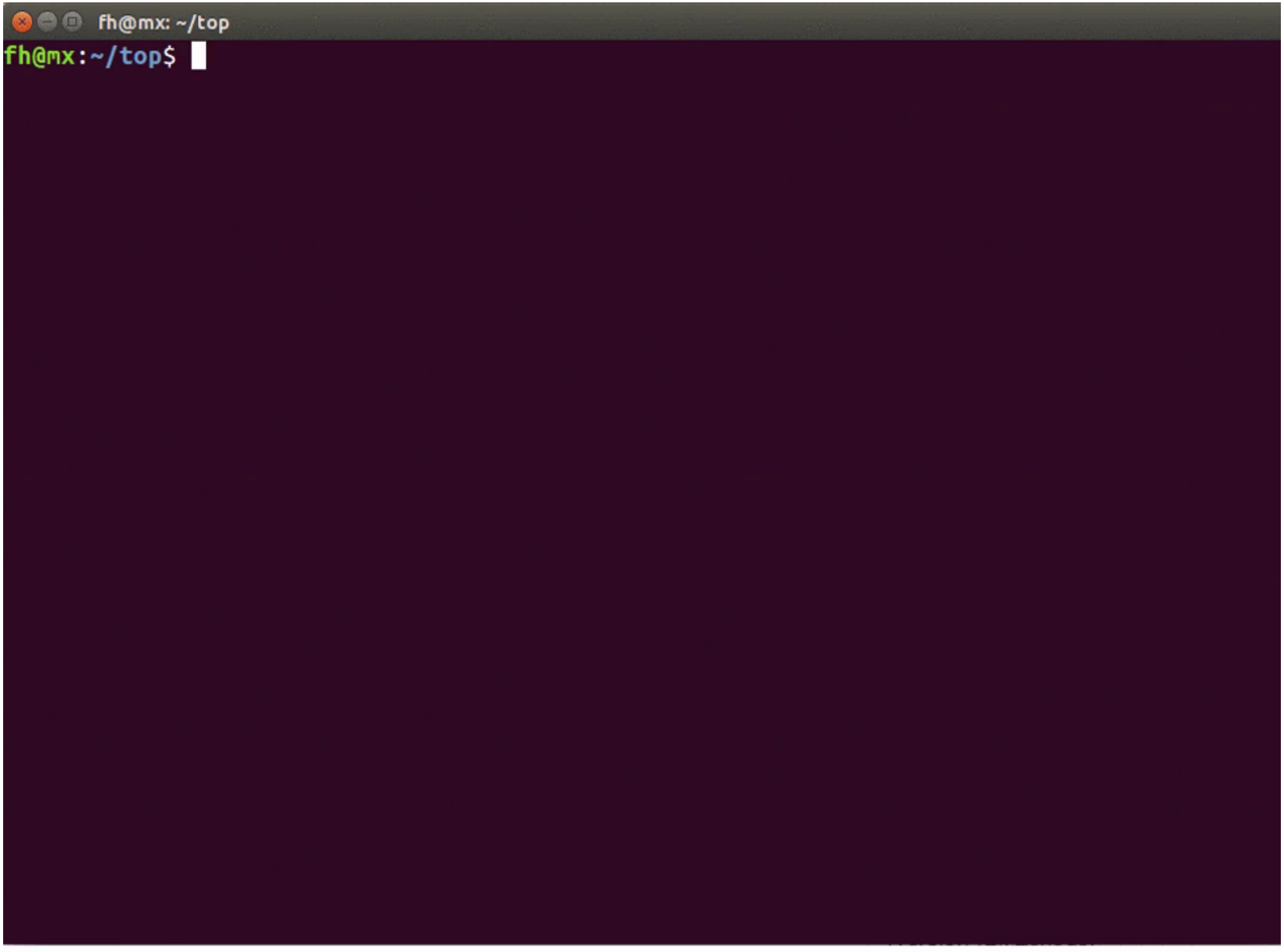




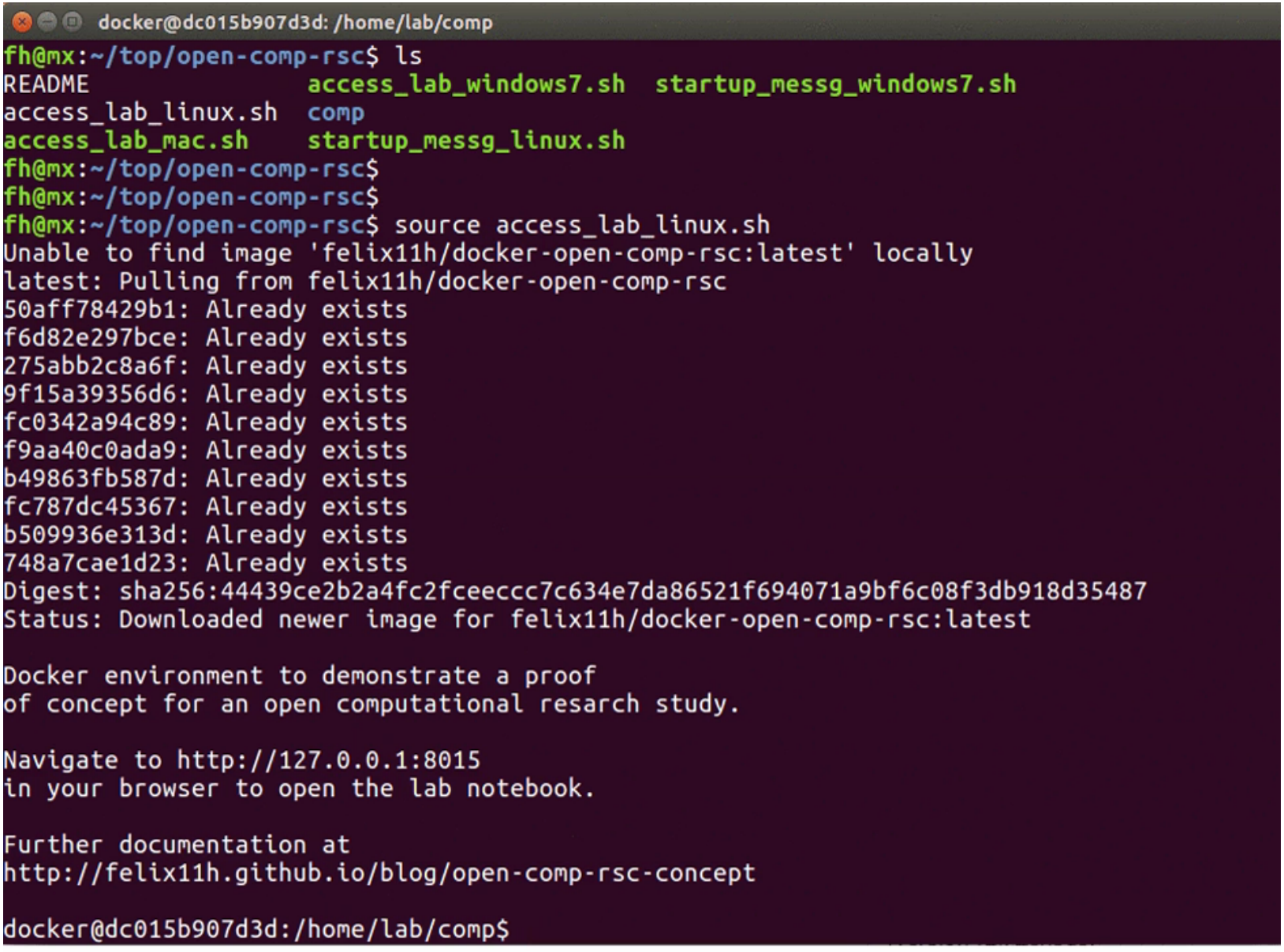
1. Find the archived code by DOI from publication



2. Download code from Zenodo



3. Unpack the downloaded archive

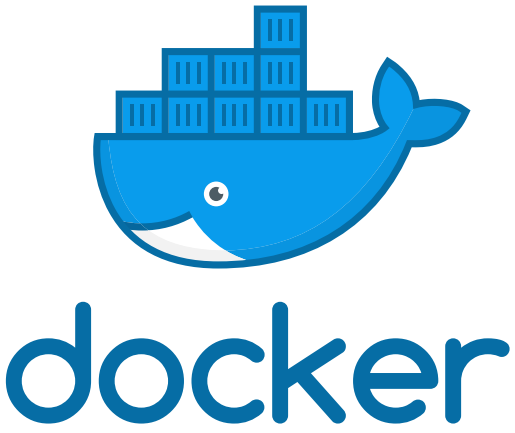


4. Access the environment - image is automatically downloaded from Docker Hub

Problems to solve

Problem 1

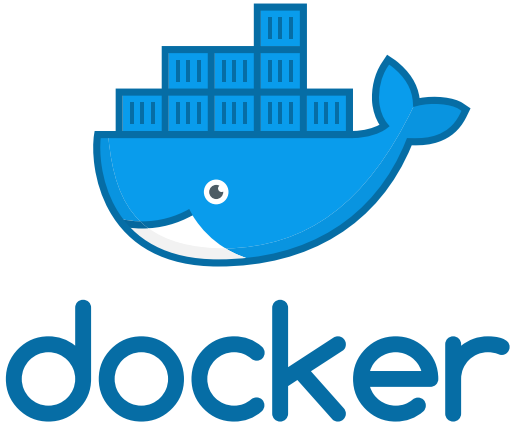
- using of difficult to install computational environment (graph-tool)



Problems to solve

Problem 1 ✓

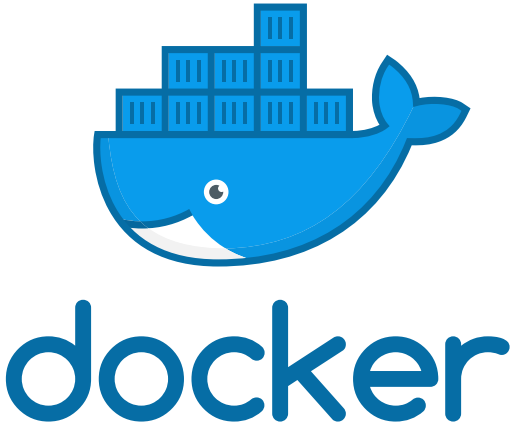
- using of difficult to install computational environment (graph-tool)



Problems to solve

Problem 1 ✓

- using of difficult to install computational environment (graph-tool)



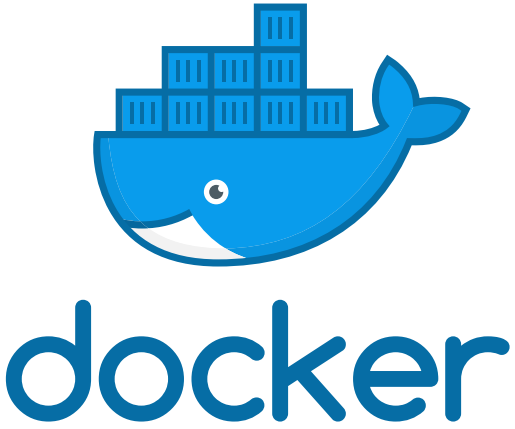
Problem 2

- long & resource demanding computations

Problems to solve

Problem 1 ✓

- using of difficult to install computational environment (graph-tool)



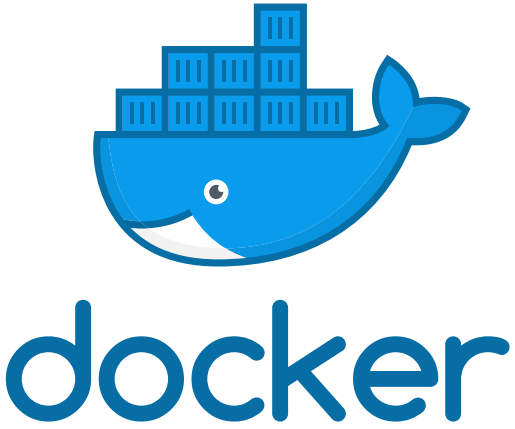
Problem 2

- long & resource demanding computations
- subsequent analysis require output of previous computations

Problems to solve

Problem 1 ✓

- using of difficult to install computational environment (graph-tool)



Problem 2

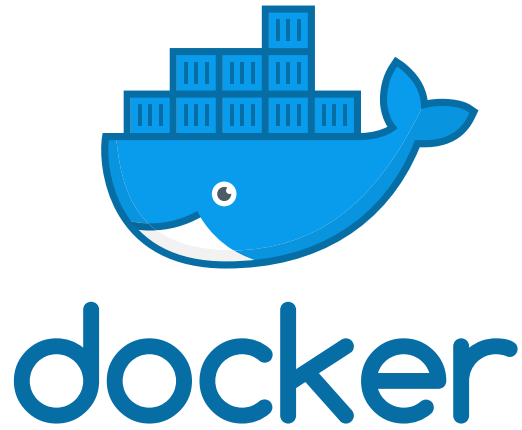
- long & resource demanding computations
- subsequent analysis require output of previous computations

difficult to understand what is required to reproduce a single output (1 figure)

Problems to solve

Problem 1 ✓

- using of difficult to install computational environment (graph-tool)

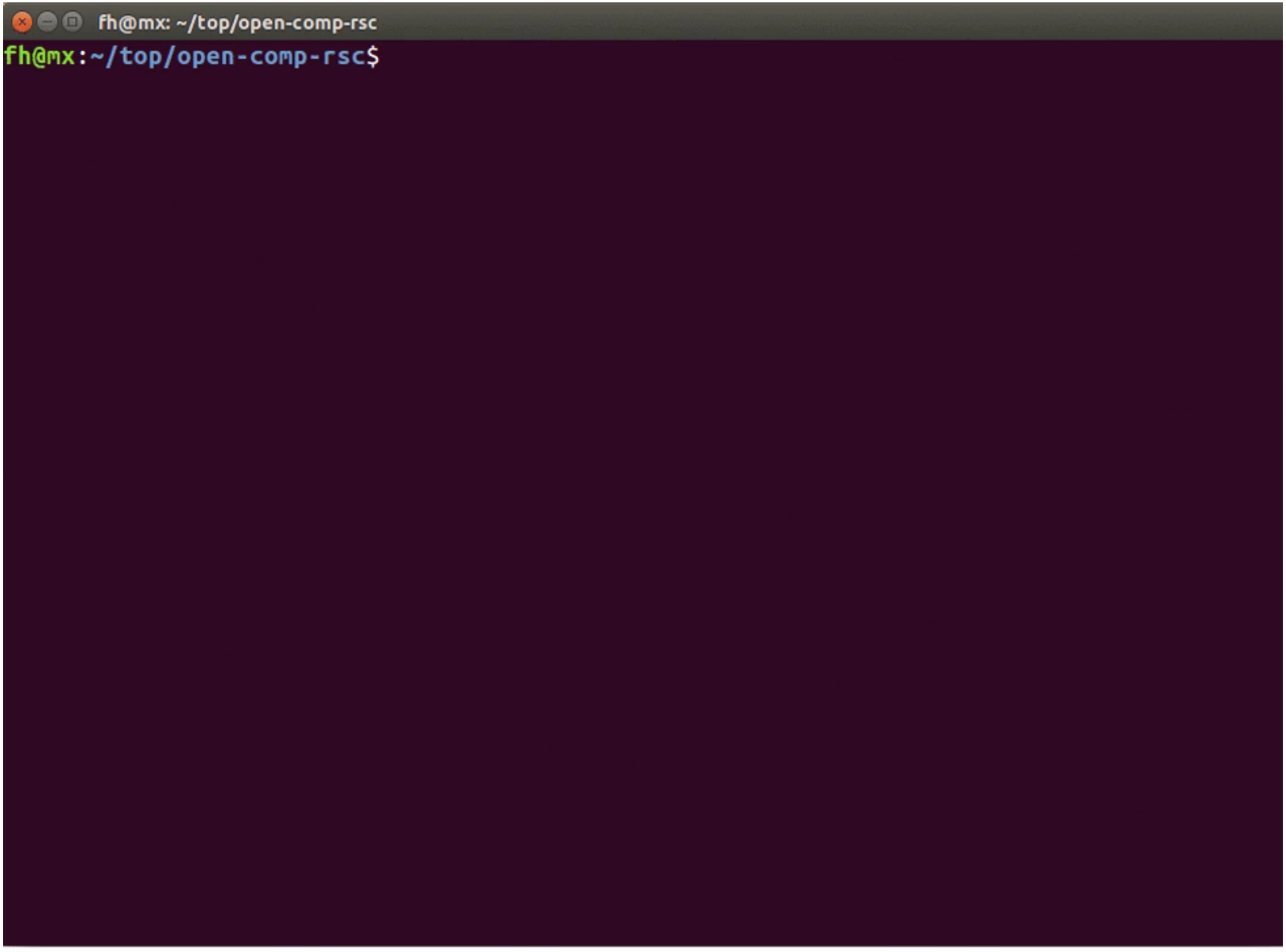


Problem 2

- long & resource demanding computations
- subsequent analysis require output of previous computations

difficult to understand what is required to reproduce a single output (1 figure)

Sumatra

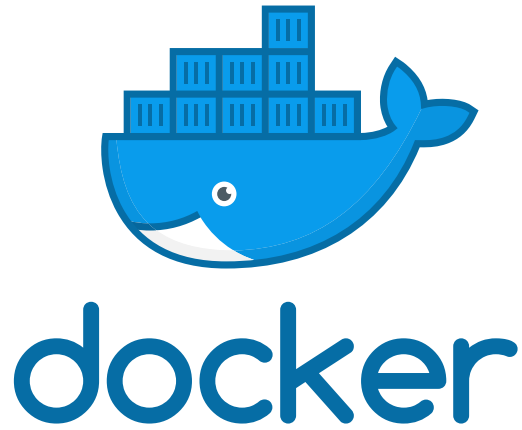


1. Accessing the environment will host the lab notebook locally

Problems to solve

Problem 1 ✓

- using of difficult to install computational environment (graph-tool)



Problem 2

- long & resource demanding computations
- subsequent analysis require output of previous computations

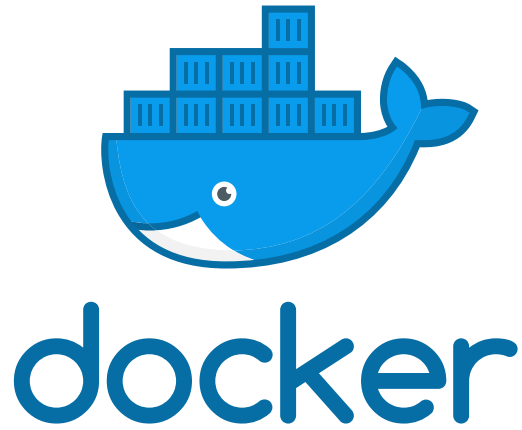
difficult to understand what is required to reproduce a single output (1 figure)

Sumatra

Problems to solve

Problem 1 ✓

- using of difficult to install computational environment (graph-tool)



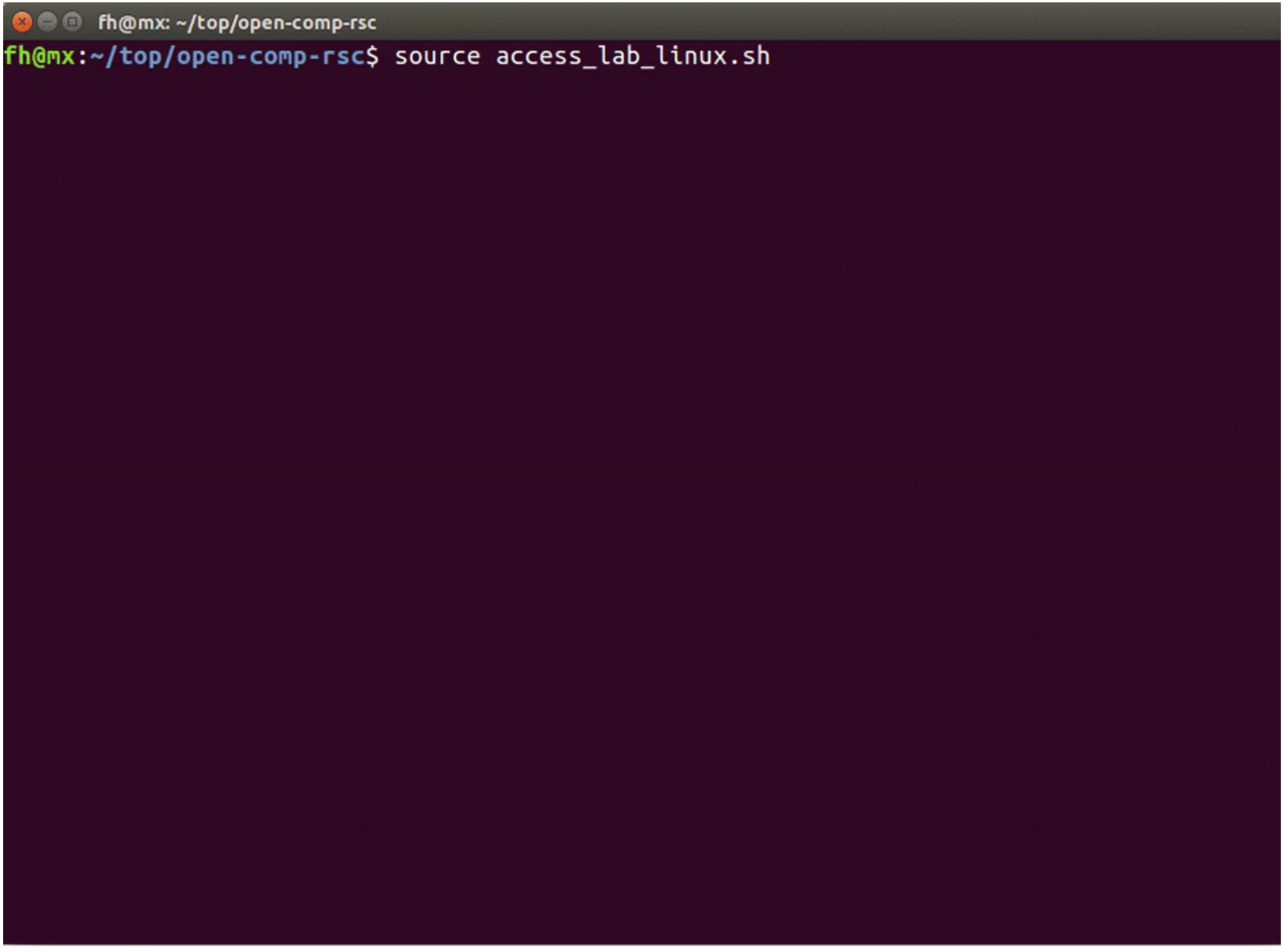
Problem 2 ✓

- long & resource demanding computations
- subsequent analysis require output of previous computations

difficult to understand what is required to reproduce a single output (1 figure)

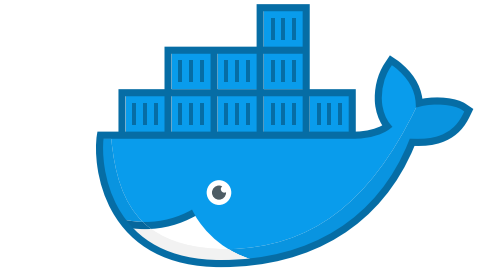
Sumatra

Davison 2012



1. In the environment, the "repeat all" command reproduces all computations in chronological order ...

Computational reproducibility in a prototype



docker + *Sumatra*

=

computational reproducibility (in a prototype)

Example study:
http://bit.ly/osproj

Documentation:
http://bit.ly/osrep

Five *R*s for reproducible scientific code

R[1] *Re-runnable*: can be run again when needed

Benureau and Rougier 2018

Five *R*s for reproducible scientific code

R[1] *Re-runnable*: can be run again when needed → document dependencies necessary to run code

Benureau and Rougier 2018

Five *R*s for reproducible scientific code

R^{1} *Re-runnable*: can be run again when needed $\rightarrow$ document dependencies necessary to run code

R^{2} *Repeatable*: program is deterministic, produces repeatable output

Benureau and Rougier 2018

Five *R*s for reproducible scientific code

R^{1} *Re-runnable*: can be run again when needed $\rightarrow$ document dependencies necessary to run code

R^{2} *Repeatable*: program is deterministic, produces repeatable output $\rightarrow$ add seeds for random number generators

Benureau and Rougier 2018

Five *R*s for reproducible scientific code

R[1] *Re-runnable*: can be run again when needed → document dependencies necessary to run code

R[2] *Repeatable*: program is deterministic, produces repeatable output → add seeds for random number generators

R[3] *Reproducible*: another researcher can take code & input data, execute code, and re-obtain same results

Benureau and Rougier 2018

Five *R*s for reproducible scientific code

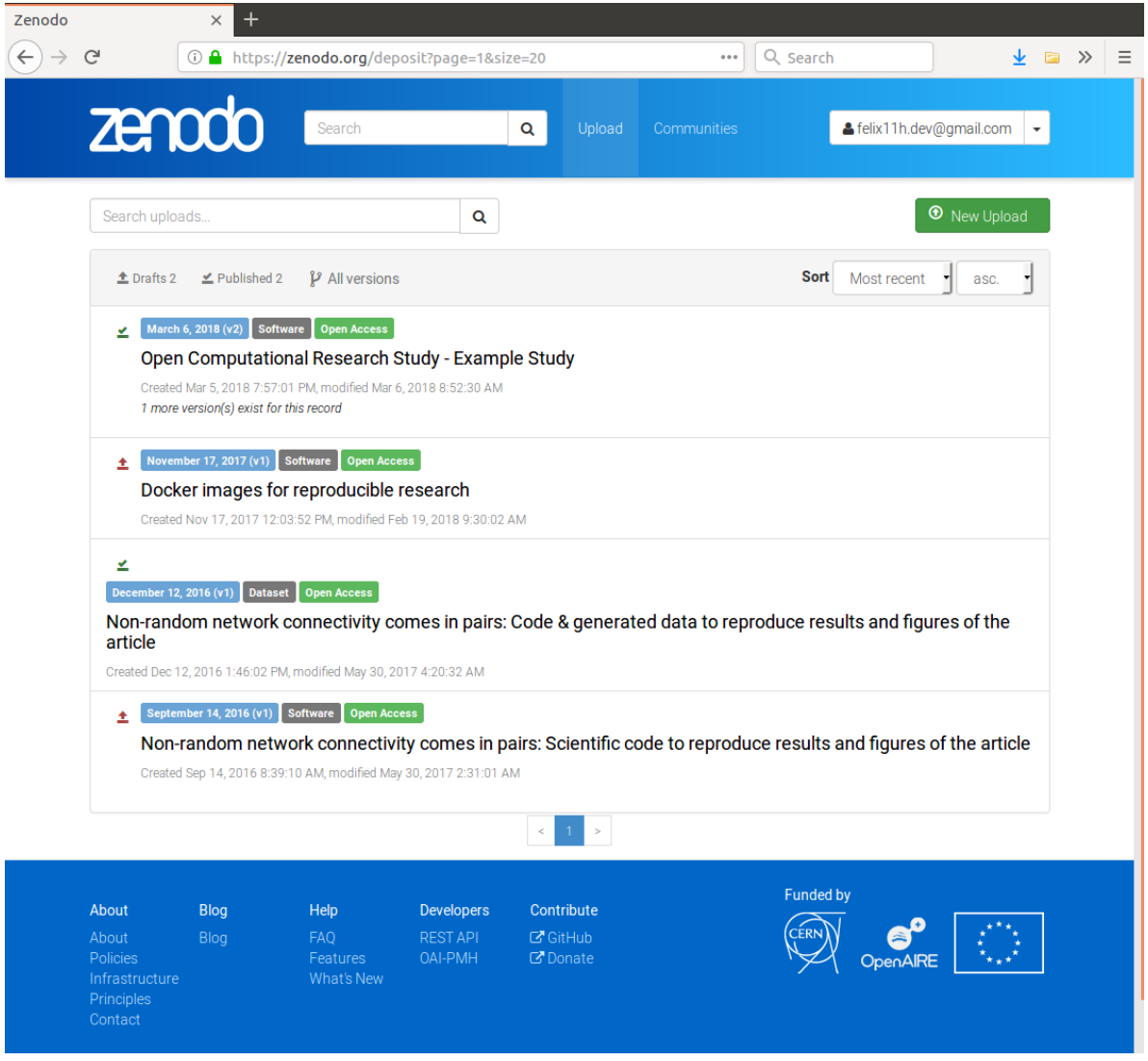
R^{1} *Re-runnable*: can be run again when needed → document dependencies necessary to run code

R^{2} *Repeatable*: program is deterministic, produces repeatable output → add seeds for random number generators

R^{3} *Reproducible*: another researcher can take code & input data, execute code, and re-obtain same results → detailed versions of dependencies, version of code, availability

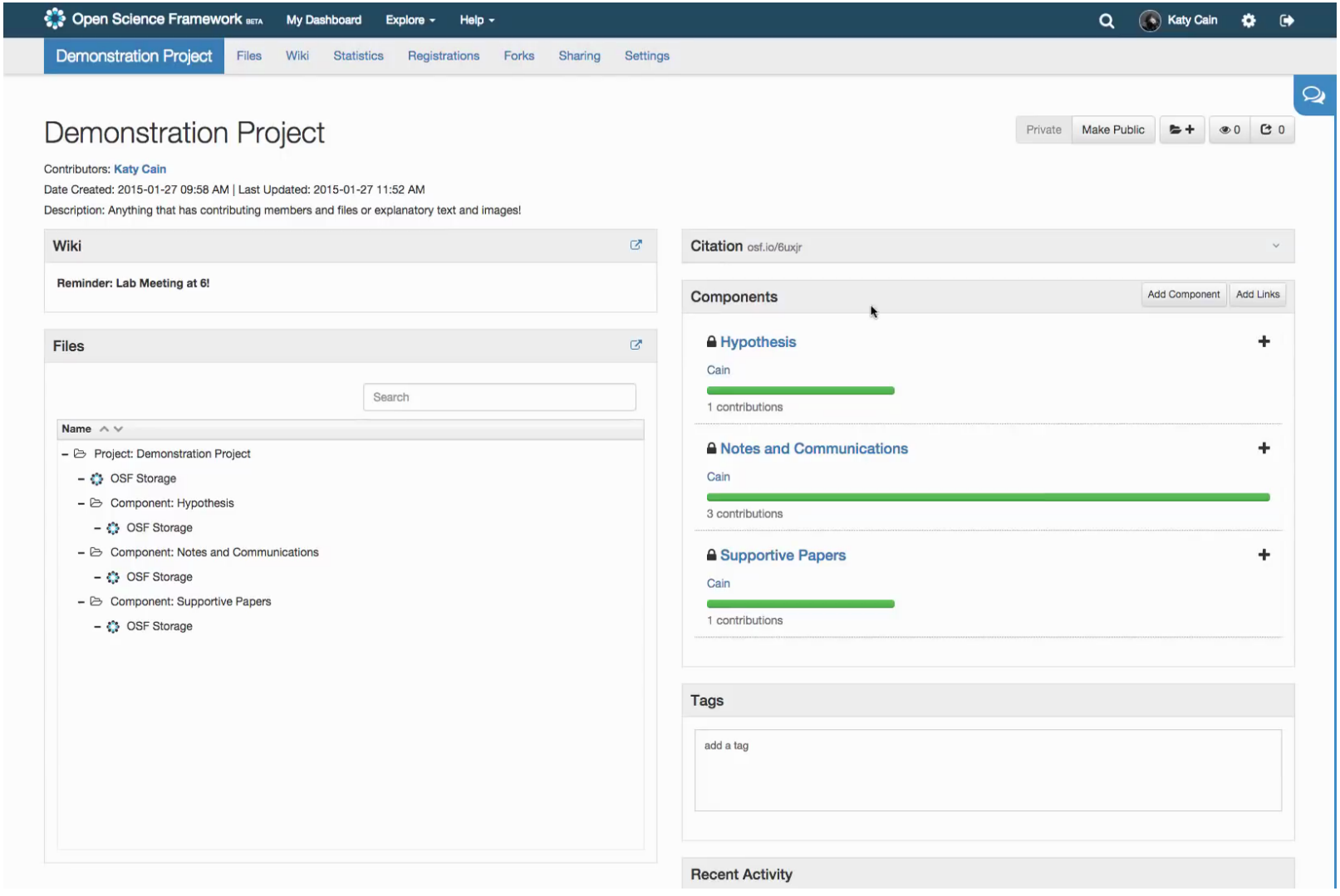
Benureau and Rougier 2018

Where to publish code



https://zenodo.org/

Where to publish code



https://osf.io/

Five *R*s for reproducible scientific code

R^{1} *Re-runnable*: can be run again when needed → document dependencies necessary to run code

R^{2} *Repeatable*: program is deterministic, produces repeatable output → add seeds for random number generators

R^{3} *Reproducible*: another researcher can take code & input data, execute code, and re-obtain same results → detailed versions of dependencies, version of code, availability

Benureau and Rougier 2018

Five *R*s for reproducible scientific code

R^{1} *Re-runnable*: can be run again when needed → document dependencies necessary to run code

R^{2} *Repeatable*: program is deterministic, produces repeatable output → add seeds for random number generators

R^{3} *Reproducible*: another researcher can take code & input data, execute code, and re-obtain same results → detailed versions of dependencies, version of code, availability

R^{4} *Reusable*: program can be easily used, and modified, by you and other people, inside & outside own lab

Benureau and Rougier 2018

Five *R*s for reproducible scientific code

R[1] *Re-runnable*: can be run again when needed → document dependencies necessary to run code

R[2] *Repeatable*: program is deterministic, produces repeatable output → add seeds for random number generators

R[3] *Reproducible*: another researcher can take code & input data, execute code, and re-obtain same results → detailed versions of dependencies, version of code, availability

R[4] *Reusable*: program can be easily used, and modified, by you and other people, inside & outside own lab → avoid hard coded numbers, write documentation

Benureau and Rougier 2018

Five *R*s for reproducible scientific code

R^{1} *Re-runnable*: can be run again when needed → document dependencies necessary to run code

R^{2} *Repeatable*: program is deterministic, produces repeatable output → add seeds for random number generators

R^{3} *Reproducible*: another researcher can take code & input data, execute code, and re-obtain same results → detailed versions of dependencies, version of code, availability

R^{4} *Reusable*: program can be easily used, and modified, by you and other people, inside & outside own lab → avoid hard coded numbers, write documentation

R^{5} *Replicable*: program can be re-implemented by another research to re-obtain results

Benureau and Rougier 2018

Five *R*s for reproducible scientific code

R^{1} *Re-runnable*: can be run again when needed → document dependencies necessary to run code

R^{2} *Repeatable*: program is deterministic, produces repeatable output → add seeds for random number generators

R^{3} *Reproducible*: another researcher can take code & input data, execute code, and re-obtain same results → detailed versions of dependencies, version of code, availability

R^{4} *Reusable*: program can be easily used, and modified, by you and other people, inside & outside own lab → avoid hard coded numbers, write documentation

R^{5} *Replicable*: program can be re-implemented by another research to re-obtain results → see for example ReScience journal

Benureau and Rougier 2018

References

Benureau, Fabien C. Y. and Nicolas P. Rougier (2018). Re-Run, Repeat, Reproduce, Reuse, Replicate: Transforming Code into Scientific Contributions. In: *Frontiers in Neuroinformatics* 11.

Collberg, Christian, Todd Proebsting, Gina Moraila, Akash Shankaran, Shi Zuoming, and Alex M Warren (2013). Measuring Reproducibility in Computer Systems Research. In:

Davison, Andrew (2012). Automated Capture of Experiment Context for Easier Reproducibility in Computational Research. In: *Computing in Science & Engineering* 14.4, pp. 48–56.

Rougier, Nicolas P. et al. (2017). Sustainable Computational Science: The ReScience Initiative. In:

Stodden, Victoria, Jennifer Seiler, and Zhaokun Ma (2018). An Empirical Analysis of Journal Policy Effectiveness for Computational Reproducibility. In: *Proceedings of the National Academy of Sciences* 115.11, pp. 2584–2589.

Thank you!

R^0 – runnable code

LISTING 0: Random walk (R^0) raw code, archive

```
import random

x = 0
for i in xrange(10):
    step = random.choice([-1,+1])
    x += step
    print x,
```

Output

```
-1, 0, -1, 0, -1, 0, -1, 0, 1, 2
# with the steps being -1,+1,-1,+1,-1,+1,-1,+1,+1,+1
```

Benureau and Rougier 2018

R^1 – Re-runnable

LISTING 1: Re-runnable random walk (R^1) raw code, archive

```
# Tested with Python 3
import random

x = 0
walk = []
for i in range(10):
    step = random.choice([-1,+1])
    x += step
    walk.append(x)

print(walk)
```

R^2 – Repeatable

LISTING 2: Re-runnable, repeatable random walk (R^2) raw code, archive

```
# Tested with Python 3
import random

random.seed(1) # RNG initialization

x = 0
walk = []
for i in range(10):
    step = random.choice([-1,+1])
    x += step
    walk.append(x)

print(walk)
# Saving output to disk
with open('results-R2.txt', 'w') as fd:
    fd.write(str(walk))
```

R^3 – Reproducible

LISTING 3: Re-runnable, repeatable, reproducible random walk (R^3)

raw code, archive

```
# Copyright (c) 2017 N.P. Rougier and F.C.Y. Benureau
# Release under the BSD 2-clause license
# Tested with 64-bit CPython 3.6.2 / macOS 10.12.6
import sys, subprocess, datetime, random

def compute_walk():
    x = 0
    walk = []
    for i in range(10):
        if random.uniform(-1, +1) > 0:
            x += 1
        else:
            x -= 1
        walk.append(x)
    return walk

# If repository is dirty, don't run anything
if subprocess.call(("git", "diff-index",
                    "--quiet", "HEAD")):
    print("Repository is dirty, please commit first")
    sys.exit(1)
```

R^3 – Reproducible

```
# Get git hash if any
hash_cmd = ("git", "rev-parse", "HEAD")
revision = subprocess.check_output(hash_cmd)

# Unit test
random.seed(42)
assert compute_walk() == [1,0,-1,-2,-1,0,1,0,-1,-2]

# Random walk for 10 steps
seed = 1
random.seed(seed)
walk = compute_walk()

# Display & save results
print(walk)
results = {
    "data"     : walk,
    "seed"     : seed,
    "timestamp": str(datetime.datetime.utcnow()),
    "revision" : revision,
    "system"   : sys.version}
with open("results-R3.txt", "w") as fd:
    fd.write(str(results))
```

R^4 – Reusable

LISTING 4: Re-runnable, repeatable, reproducible, reusable random walk (R^4)

raw code, archive

```
# Copyright (c) 2017 N.P. Rougier and F.C.Y. Benureau
# Release under the BSD 2-clause license
# Tested with 64-bit CPython 3.6.2 / macOS 10.12.6
import sys, subprocess, datetime, random

def compute_walk(count, x0=0, step=1, seed=0):
    """Random walk
       count: number of steps
       x0   : initial position (default 0)
       step : step size (default 1)
       seed : seed for the initialization of the
          random generator (default 0)
    """
    random.seed(seed)
    x = x0
    walk = []
    for i in range(count):
        if random.uniform(-1, +1) > 0:
            x += 1
        else:
            x -= 1
        walk.append(x)
    return walk
```

R^4 – Reusable

```
def compute_results(count, x0=0, step=1, seed=0):
    """Compute a walk and return it with context"""
    # If repository is dirty, don't do anything
    if subprocess.call(("git", "diff-index",
                        "--quiet", "HEAD")):
        print("Repository is dirty, please commit")
        sys.exit(1)

    # Get git hash if any
    hash_cmd = ("git", "rev-parse", "HEAD")
    revision = subprocess.check_output(hash_cmd)

    # Compute results
    walk = compute_walk(count=count, x0=x0,
                        step=step, seed=seed)
    return {
        "data"      : walk,
        "parameters": {"count": count, "x0": x0,
                       "step": step, "seed": seed},
        "timestamp" : str(datetime.datetime.utcnow()),
        "revision"  : revision,
        "system"    : sys.version}
```

R^4 – Reusable

```
if __name__ == "__main__":
    # Unit test checking reproducibility
    # (will fail with Python<=3.2)
    assert (compute_walk(10, 0, 1, 42) ==
              [1,0,-1,-2,-1,0,1,0,-1,-2])

    # Simulation parameters
    count, x0, seed = 10, 0, 1
    results = compute_results(count, x0=x0, seed=seed)

    # Save & display results
    with open("results-R4.txt", "w") as fd:
        fd.write(str(results))
    print(results["data"])
```

R^5 – Replicable

LISTING 5: Replicated random walk (R^5) raw code, archive

```
# Copyright (c) 2017 N.P. Rougier and F.C.Y. Benureau
# Release under the BSD 2-clause license
# 64-bit CPython 3.6.2 / NumPy 1.12.0 / macOS 10.12.6
import random
import numpy as np

def _rng(seed):
    """Return a numpy random number generator
       initialized with seed as it would be with
       a python random generator.
    """
    rng = random.Random()
    rng.seed(seed)
    _, keys, _ = rng.getstate()
    rng = np.random.RandomState()
    state = rng.get_state()
    rng.set_state((state[0], keys[:-1], state[2],
                   state[3], state[4]))
    return rng
```

R^5 – Replicable

```
def walk(n, seed):
    """Random walk for n steps"""
    rng = _rng(seed)
    steps = 2 * (rng.uniform(-1, +1, n) > 0) - 1
    return steps.cumsum().tolist()

if __name__ == "__main__":
    # Unit test
    assert (walk(n=10, seed=42) ==
            [1,0,-1,-2,-1,0,1,0,-1,-2])

    # Random walk for 10 steps, with seed=1
    seed = 1
    path = walk(n=10, seed=seed)

    # Save & display results
    results = {"data": path, "seed": seed}
    with open("results-R5.txt", "w") as fd:
        fd.write(str(results))
    print(path)
```

But also...

Please find attached a .zip file called [redacted].zip that has the custom MATLAB [redacted] analysis code. If you run Masterrunfigureone.m this will generate several panels from the paper.

In the next email I will enclose the custom image analysis software. This can also be accessed from [URL redacted] where there is a manual and tutorial.

Please let me know if you have any troubles, or if there is anything else I can help with.

Stodden et al. 2018

Reproducibility when code was available

56 out of 91 studies were judged as *potentially reproducible*

Table 4. Classification of reproducibility effort (*n* = 22)

Classification	Percent, %
Impossible to reproduce (missing essential code, data, or methodology)	5
Nearly impossible to reproduce (specialized hardware, intense computation requirements, sensitive data, human study, or other unavoidable reasons)	14
Difficult to reproduce because of unavoidable inherent complexity (e.g., requiring 300 million Markov chain Monte Carlo steps on each dataset, or needing months to do runs)	14
Reproducible with substantial tedious effort (e.g., individual download of a large number of datasets, hand coding of data into a new format, i.e., from an image, many archiving steps required)	5

Stodden et al. 2018

Reproducibility when code was available

56 out of 91 studies were judged as *potentially reproducible*

Table 4. Classification of reproducibility effort (*n* = 22)

Classification	Percent, %
Reproducible with substantial intellectual effort (e.g., methods well defined but required some knowledge of jargon or understanding of the field; or down the rabbit hole references to past articles required to reproduce; etc.)	5
Could reproduce with fairly substantial skill and knowledge (e.g., required GPU programing abilities to run code that wasn't given; translating complex models into MATLAB code; pseudo code with functions not detailed described in text into code; missing scripts)	23

Stodden et al. 2018

Reproducibility when code was available

56 out of 91 studies were judged as *potentially reproducible*

Table 4. Classification of reproducibility effort (*n* = 22)

Classification	Percent, %
Reproducible after tweaking (e.g., missing parameters required fiddling to find, missing modified code lines, missing arguments required for differing architecture; missing minor method step)	5
Minor difficulty in reproducing (e.g., installing a specialized library, converting to a different computational system)	18
Straightforward to reproduce with minimal effort	14

Stodden et al. 2018

Reproducibility when code was available

56 out of 91 studies were judged as *potentially reproducible*

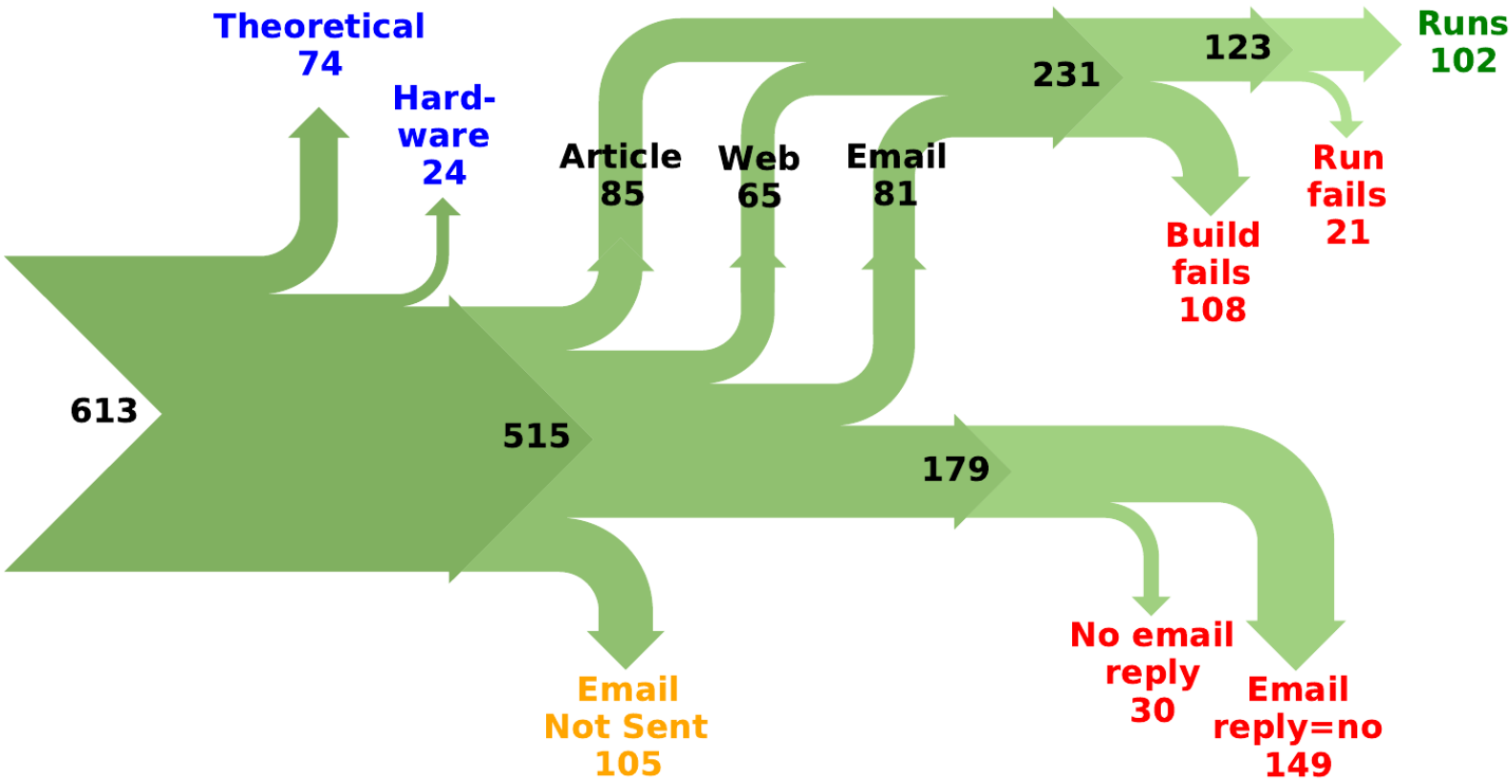
Table 4. Classification of reproducibility effort (*n* = 22)

Classification	Percent, %
Reproducible after tweaking (e.g., missing parameters required fiddling to find, missing modified code lines, missing arguments required for differing architecture; missing minor method step)	5
Minor difficulty in reproducing (e.g., installing a specialized library, converting to a different computational system)	18
Straightforward to reproduce with minimal effort	14

Even when code was available, more than half of studies were reproducible only with *significant effort!*

Stodden et al. 2018

Measuring Reproducibility in Computer Systems Research



Collberg et al. 2013