Testing open source Python-centric research software

Hans Fangohr
Max Planck Institute for the Structure and Dynamics of Matter
University of Southampton (UK)
https://fangohr.github.io
@ProfCompMod

2021-09-28

NFDI4Ing Konferenz

Outline

Introduction

Example: Ubermag

Discussion and summary

Introduction

Biography Hans Fangohr / interests

- studies of Physics
- PhD in HPC / Simulation of superconducting material
- Professor of Computational Modelling (2010-)
- Data Analysis at European XFEL GmbH (2017-2020)
- SSU Computational Science at Max-Planck-Institute (now)

Interests shift

- from researching physics problems with computers
- to researching computational methods [which help solve research problems]

Software engineering in Computational Science

Good practice for Computational Research includes

- version control
- tests
- continuous integration

Everybody uses tests

- any software we develop needs testing:
  - often done by executing examples and (manually)
  - checking the results for correctness
- more efficient: make the testing process *automatic*

Example for automatic test (1/2)

```python
def f(n):                          # "production code"
    s = 0
    # Loop from 0 to n:
    for i in range(1, n + 1):
        s = s + i
    return s


def test_f():                      # "test code"
    assert f(3) == 0 + 1 + 2 + 3
    assert f(5) == 15
    assert f(10) == 55
```

Example for automatic test (2/2)

Test by running `py.test` on the source file

```
cd code && py.test -v example1.py
```

```
================= test session starts ================
platform darwin -- Python 3.8.5, pytest-6.1.1, py-1.9.0, pluggy-0.13.1
cachedir: .pytest_cache
rootdir: slides, configfile: pytest.ini
collecting ... collected 1 item

example1.py::test_f PASSED                          [100%]

================= 1 passed in 0.00s ==================
```

Continuous Integration (CI)

Idea is to execute all automatic tests automatically:

- when we push to the git repository
- when a pull request is opened / changed / merged
- periodically
- ....

Typical work flow:

Accept pull requests only when all tests pass.

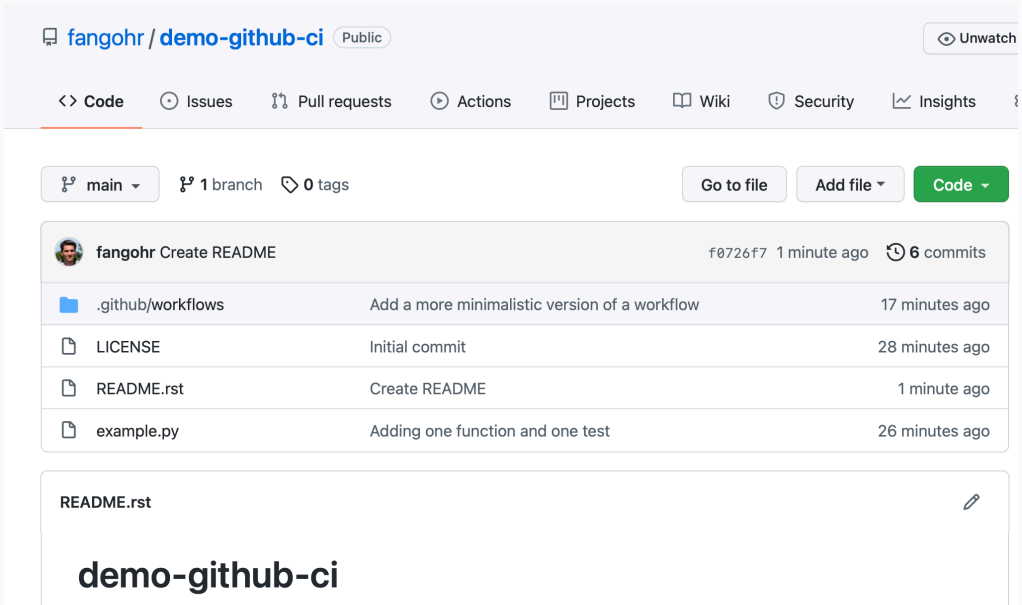
How to run tests & Continuous integration (CI) services

1. Manual, at prompt when required

```
py.test -v .
```

2. Services and tools providing CI include
   - GitHub (workflows), Gitlab (runners)
   - Travis CI, Circle CI
   - Jenkins, Buildbot, ...

   Example: `https://github.com/fangohr/demo-github-ci`



Why do we need tests?

Quote from Kent Beck (Test Driven Development)

- *Test-driven development is a way of managing fear during programming [...]*

A test suite

- can help us debug a problem
- will catch future bugs
- helps us embrace change: easier to modify and extend software
- can provide confidence
- serves as living documentation of the code
- makes it easier for others to contribute to the project

Example: Ubermag

Ubermag (https://ubermag.github.io)

What is ubermag?

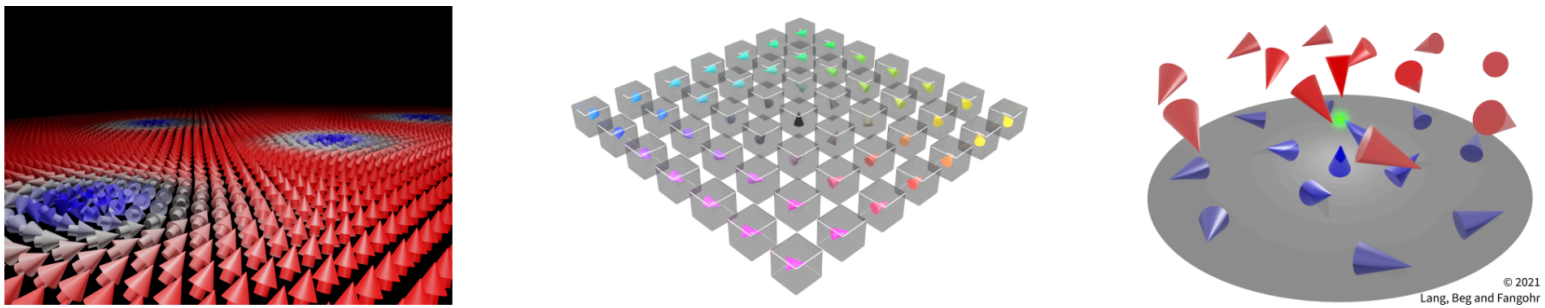
As a whole, ubermag is a level of (Python) utilities sitting above ("*über*") existing micromagnetic simulation packages, such as OOMMF and mumax3. We refer to packages such as OOMMF and mumax3 as the *computational backends*, or as (micromagnetic) *calculators*.

By exposing micromagnetic simulations to the Python ecosystem and integration into Jupyter, all the popular tools used in scientific computing for data analysis, numerical analysis, visualisation, machine learning, etc. become readily available.

The ubermag Python packages allow to specify a micromagnetic problem (without making use of any micromagnetic calculator). Only when the micromagnetic problem needs to be solved, the task is delegated to the computational backend.

Design objectives of Ubermag are:

1. Work towards a common interface (domain spacific language) to multiple simulation packages,
2. Expose micromagnetic simulations to Python ecosystem,
3. Make it easier to compare simulation results obtained with different simulation packages, and
4. Allow execution of micromagnetic simulation in the cloud.



Magnetic systems (left-to-right): Skyrmions, vortex, Bloch point [renderings created with Blender]

- Example: https://ubermag.github.io/quickstart.html

Ubermag Example

- Example: `https://ubermag.github.io/quickstart.html`

[1]:
```python
# Some initial configurations
#%config InlineBackend.figure_formats = ['svg']  # output matplotlib plots as SVG
import pandas as pd
pd.options.display.max_rows = 5
pd.options.display.float_format = '{:,.2e}'.format

import discretisedfield as df
import micromagneticmodel as mm

# Geometry
L = 100e-9  # sample edge length (m)
thickness = 5e-9  # sample thickness (m)

# Material (Permalloy) parameters
Ms = 8e5  # saturation magnetisation (A/m)
A = 13e-12  # exchange energy constant (J/m)

# Dynamics (LLG equation) parameters
gamma0 = 2.211e5  # gyromagnetic ratio (m/As)
alpha = 0.2  # Gilbert damping

system = mm.System(name='vortex_dynamics')

# Energy equation. We omit Zeeman energy term, because H=0.
system.energy = mm.Exchange(A=A) + mm.Demag()

# Dynamics equation
system.dynamics = mm.Precession(gamma0=gamma0) + mm.Damping(alpha=alpha)

# initial magnetisation state
def m_init(point):
    x, y, z = point
    c = 1e9  # (1/m)
    return (-c*y, c*x, 0.1)

# Sample's centre is placed at origin
region = df.Region(p1=(-L/2, -L/2, -thickness/2), p2=(L/2, L/2, thickness/2))
mesh = df.Mesh(region=region, cell=(5e-9, 5e-9, 5e-9))

system.m = df.Field(mesh, dim=3, value=m_init, norm=Ms)
```

The system object is now defined and we can investigate some of its properties:

Ubermag Example 2

```
# Energy equation. We omit Zeeman energy term, because H=0.
system.energy = mm.Exchange(A=A) + mm.Demag()

# Dynamics equation
system.dynamics = mm.Precession(gamma0=gamma0) + mm.Damping(alpha=alpha)

# initial magnetisation state
def m_init(point):
    x, y, z = point
    c = 1e9  # (1/m)
    return (-c*y, c*x, 0.1)

# Sample's centre is placed at origin
region = df.Region(p1=(-L/2, -L/2, -thickness/2), p2=(L/2, L/2, thickness/2))
mesh = df.Mesh(region=region, cell=(5e-9, 5e-9, 5e-9))

system.m = df.Field(mesh, dim=3, value=m_init, norm=Ms)
```

The system object is now defined and we can investigate some of its properties:

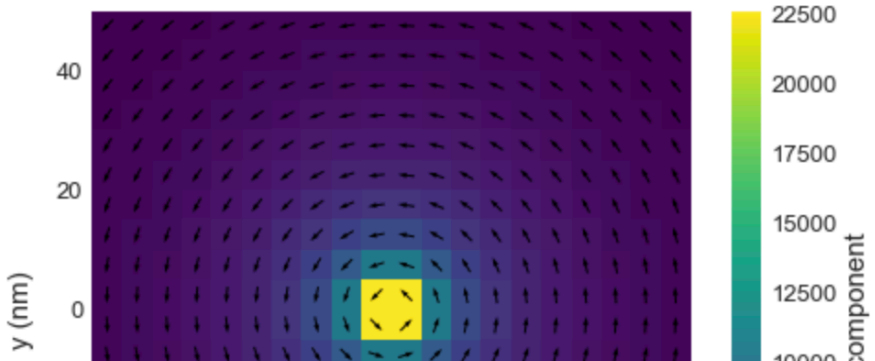
[2]:
```
system.energy
```

[2]: $$-A\mathbf{m}\cdot\nabla^2\mathbf{m} - \frac{1}{2}\mu_0 M_\mathrm{s}\mathbf{m}\cdot\mathbf{H}_\mathrm{d}$$

[3]:
```
system.dynamics
```

[3]: $$-\frac{\gamma_0}{1+\alpha^2}\mathbf{m}\times\mathbf{H}_\mathrm{eff} - \frac{\gamma_0\alpha}{1+\alpha^2}\mathbf{m}\times(\mathbf{m}\times\mathbf{H}_\mathrm{eff})$$

[4]:
```
system.m.plane('z').mpl()
```

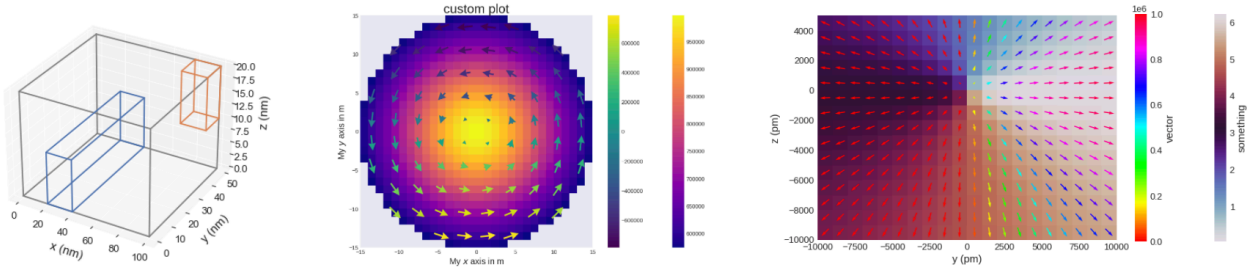


Ubermag sub package: `discretisedfield`

Ubermag - Features and packages

Ubermag is a collection of several independent packages that can be used independently as well as in combination. The whole set of packages – the `ubermag` meta-package – is tight to micromagnetic simulations. The individual sub-packages can be used in other fields, e.g. fluid dynamics.

Finite-difference fields – `discretisedfield`



- Three-dimensional finite-different scalar or vector fields independent of the underlying physics
- It provides common field operations such as
  - Mathematical operations (standard operations, derivatives, integration, ...)
  - Specific tools, e.g. topological charge density (`discretisedfield.tools`)
  - Visualisation with `matplotlib` and `k3d`
  - Reading and writing `vtk`, `ovf`, and `hdf5` files
- Possible applications are amongs others
  - Micromagnetics
  - Fluid dynamics

Source code /home page on github

`https://github.com/ubermag/discretisedfield`

Discretisedfield running tests

- In source directory:
  `py.test -v`
- from Python prompt:
  `import discretisedfield as df; df.test()`
- through continuous integration
  - CI through Github actions:

    `https://github.com/ubermag/discretisedfield/actions`
  - Set up of one workflow: `https://github.com/ubermag/discretisedfield/blob/master/.github/workflows/workflow.yml`
  - Example output from running tests on Ubuntu:

    `https://github.com/ubermag/discretisedfield/runs/3673411912?check_suite_focus=true`

Github workflow 1/2

```yaml
name: workflow

on:
  push:
    branches-ignore:
      - 'stable'
  schedule:
    - cron: '0 0 * * 1'  # run every Monday

jobs:
  workflow:
    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        os: [ubuntu-latest, macos-latest, windows-latest]
    defaults:
      run:
        shell: bash -l {0}

    steps:
    - name: Initialisation
      uses: actions/checkout@v2

    - name: Set up conda
      uses: conda-incubator/setup-miniconda@v2
      with:
        auto-update-conda: true
        activate-environment: conda-environment
        environment-file: .github/environment.yml
```

Github workflow 2/2

```
- name: Install testing packages
  run: python -m pip install --upgrade pip pytest-cov nbval pycodestyle

- name: Install package
  run: python -m pip install .

- name: Unit tests
  run: make test-unittests
  if: matrix.os != 'ubuntu-latest'

- name: Unit tests with coverage
  run: make test-coverage
  if: matrix.os == 'ubuntu-latest'

- name: Documentation tests
  run: make test-docs

- name: Jupyter notebook tests
  run: make test-ipynb

- name: Python code style
  run: make test-pycodestyle

- name: Upload coverage to Codecov.io
  uses: codecov/codecov-action@v1
  if: matrix.os == 'ubuntu-latest'
```

Actual execution targets in in `Makefile`

Advantage of testing steps in Makefile (or other script):

- easier to execute the workflow steps locally

Makefile contents:

```
PROJECT=discretisedfield
IPYNBPATH=docs/ipynb/*.ipynb
PYTHON?=python

test-unittests:
	$(PYTHON) -c "import sys; import $(PROJECT); sys.exit($(PROJECT).test())"

test-coverage:
	$(PYTHON) -m pytest -v --cov=$(PROJECT) --cov-report=xml

test-docs:
	$(PYTHON) -m pytest -v --doctest-modules --ignore=$(PROJECT)/tests $(PROJECT)

test-ipynb:
	$(PYTHON) -m pytest -v --nbval $(IPYNBPATH)

test-pycodestyle:
	$(PYTHON) -m pycodestyle --filename=*.py .

test-all: test-unittests test-docs test-ipynb test-pycodestyle
```

Testing examples in documentation strings (doctests) 1/2

```
def f(n):
    """Given a positive integer n, return the
    sum of numbers from 1 up to n.
    Examples:
    >>> f(1)
    1
    >>> f(3)
    6
    """
    s = 0
    for i in range(1, n + 1):
        s = s + i
    return s
```

Testing examples in documentation strings (doctests) 2/2

Test by running `py.test` on the source file

```
cd code && py.test --doctest-modules -v example_doctest.py
```

```
============================= test session starts ======================
platform darwin -- Python 3.8.5, pytest-6.2.4, py-1.9.0, pluggy-0.13.1
cachedir: .pytest_cache
rootdir: /Users/fangohr/git/talks-fangohr/2021-09-28-nfdi-testing-talk
collecting ... collected 1 item

example_doctest.py::example_doctest.f PASSED

============================== 1 passed in 0.05s =======================
```

Doctest documentation:

`https://docs.pytest.org/en/latest/how-to/doctest.html`

Jupyter notebook documentation: Example

jupyter jupyterdemo2 Last Checkpoint: 2 minutes ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [1]:
```
1 + 2
```

Out[1]:
```
3
```

Cells can contain text and latex equations such as $f(x) = \sin(2\pi\omega t^2)$ and $\omega = 220$ Hz. We can use code to define the corresponding functions:

In [2]:
```
import numpy as np
def f(t):
    omega = 220
    return np.sin(2 * np.pi * omega * t**2)
```

Let's compute the data and plot the beginning of it:

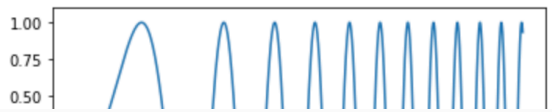
In [3]:
```
t = np.linspace(0, 2, 44100)
y = f(t)
```

In [4]:
```
y[:5], y[-5:]
```

Out[4]:
```
(array([0.00000000e+00, 2.84318460e-06, 1.13727384e-05, 2.55886614e-05,
        4.54909535e-05]),
 array([-8.43092023e-01, -6.83293553e-01, -4.80754541e-01, -2.48140602e-01,
         1.25522677e-13]))
```

In [5]:
```
#NBVAL_IGNORE_OUTPUT
%matplotlib inline
import pylab
pylab.plot(t[0:5000], y[0:5000])
```

Out[5]:
```
[<matplotlib.lines.Line2D at 0x7fdb186cbfa0>]
```



- Idea: use notebook as regression test
- Question: is NoteBook still VALid? (`nbval`)

Jupyter NoteBook VALidation (nbval):

- Notebook validation: execute input cell and compare computed output with data in file
- available as ”nbval” plugin for py.test
  `https://github.com/computationalmodelling/nbval`

```
fangohr@iMac-Hans code % py.test -v --nbval jupyterdemo2.ipynb
============================= test session starts ==============================
platform darwin -- Python 3.9.0, pytest-6.2.5, py-1.10.0, pluggy-1.0.0 -- /Users
/fangohr/.pyenv/versions/3.9.0/envs/tmp-discretisedfield/bin/python3.9
cachedir: .pytest_cache
rootdir: /Users/fangohr/git/talks-fangohr/2021-09-28-nfdi-testing-talk2/slides,
configfile: pytest.ini
plugins: anyio-3.3.1, nbval-0.9.6
collected 6 items

jupyterdemo2.ipynb::Cell 0 PASSED                                        [ 16%]
jupyterdemo2.ipynb::Cell 1 PASSED                                        [ 33%]
jupyterdemo2.ipynb::Cell 2 PASSED                                        [ 50%]
jupyterdemo2.ipynb::Cell 3 PASSED                                        [ 66%]
jupyterdemo2.ipynb::Cell 4 PASSED                                        [ 83%]
jupyterdemo2.ipynb::Cell 5 PASSED                                        [100%]

============================== 6 passed in 1.77s ===============================
```

- → treat documentation as regression test

Code style testing

- try stick to conventions:
  - PEP8
    `https://www.python.org/dev/peps/pep-0008/`
- tools to help with this
  - pycodestyle (used to be `pep8`)
  - flake8, pylint, ..., black

Example for style violation

```
def f(n):
    s = 0
    # Loop from 0 to n:
    for i in range(1,n+1):
        s = s + i
    return s
```

Test by running `pycodestyle` on the source file, and ask for return code `$?`

```
cd code && pycodestyle example_style.py
echo "exit code: $?"
```

```
example_style.py:4:21: E231 missing whitespace after ','
Return code: 1
```

Sidenote: Black

```
cd code && black --check --diff example_style.py
```

```
fangohr@iMac-Hans code % black --check --diff example_style.py
--- example_style.py     2021-09-23 09:35:36.680117 +0000
+++ example_style.py    2021-09-23 09:55:37.476536 +0000
@@ -1,6 +1,6 @@
 def f(n):
     s = 0
     # Loop from 0 to n:
-    for i in range(1,n+1):
+    for i in range(1, n + 1):
         s = s + i
     return s
would reformat example_style.py
Oh no! 💥 💔 💥
1 file would be reformatted.
fangohr@iMac-Hans code % echo $?
1
```

Black (https://github.com/psf/black)

- Opinionated pep8-compatible style checker
- can modify source automatically
- use before committing code

Test coverage

Given some Python code:

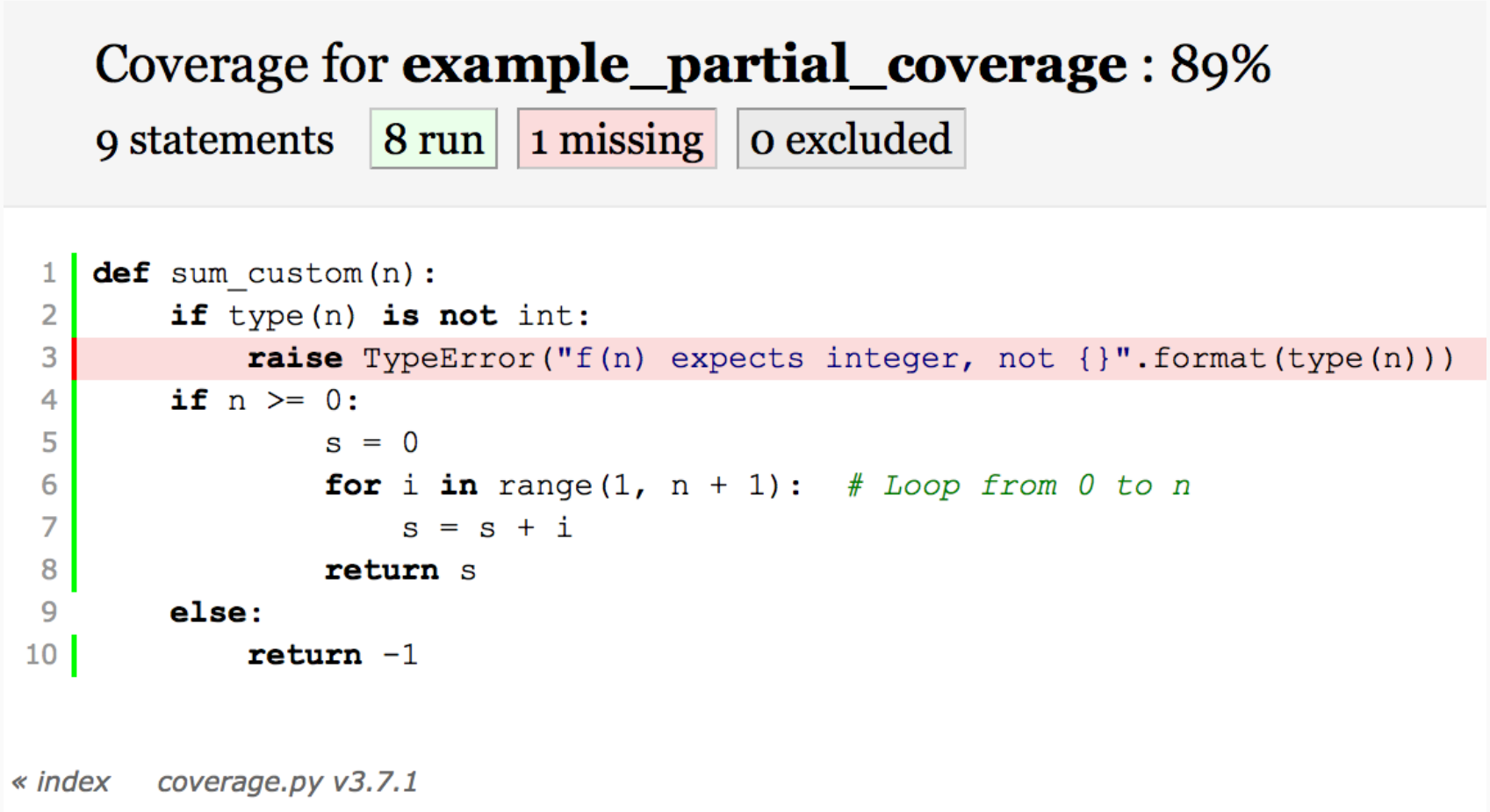
```
def sum_custom(n):
    if type(n) is not int:
        raise TypeError("f(n) expects integer, not {}".format(type(n)))
    if n >= 0:
            s = 0
            for i in range(1, n + 1):  # Loop from 0 to n
                s = s + i
            return s
    else:
        return -1
```

And some tests

```
from example_partial_coverage import sum_custom as f

def test_sum_custom():          # partitioning n
    assert f(3) == 0 + 1 + 2 + 3
    assert f(5) == 15
    assert f(10) == 55
    assert f(-1) == -1
    assert f(0) == 0
```

Test coverage output



Coverage for **example_partial_coverage** : 89%

9 statements | 8 run | 1 missing | 0 excluded

```
1  def sum_custom(n):
2      if type(n) is not int:
3          raise TypeError("f(n) expects integer, not {}".format(type(n)))
4      if n >= 0:
5              s = 0
6              for i in range(1, n + 1):  # Loop from 0 to n
7                  s = s + i
8              return s
9      else:
10         return -1
```

« index coverage.py v3.7.1

```
--cov-report=html
```

coverage example, terminal output

py.test can capture code coverage of tests and output in different formats:

- html - as on last slide
- term - text based output (as below)
- xml - can be used in other services to analyses the data further

Example for terminal output:

```
cd code && py.test --cov --cov-report=term test_example_partial_coverage.py
```

```
============================= test session starts ==============================
platform darwin -- Python 3.4.3 -- py-1.4.27 -- pytest-2.7.1
rootdir: /Users/fangohr/gitdocs/teaching-software-engineering/slides, inifile: pytest.ini
plugins: hypothesis, cov

test_example_partial_coverage.py .
--------------- coverage: platform darwin, python 3.4.3-final-0 ---------------
Name                                 Stmts   Miss  Cover
------------------------------------------------------
example_partial_coverage.py              9      1    89%
test_example_partial_coverage.py         7      0   100%
========================== 1 passed in 0.01 seconds ===========================
```

Discussion and summary

What can we test?

- installation
- unit tests
- system tests
- documentation strings
- additional documentation (Jupyter Notebooks, sphinx)
- code style
- test coverage
- complexity
- performance

Testing scientific research software is difficult

because not knowing the exact answer is the reason the code was written.

Options:

- reduce to situation with known answer:
  - conservation laws
  - trivial cases for which analytical solution is known
- compare with software using a different numerical method for the same system
- compare with software written by somebody else
- compare with experimental data
- compare with previous result

Testing for data-heavy projects

Assume we need to analyse / process large data files

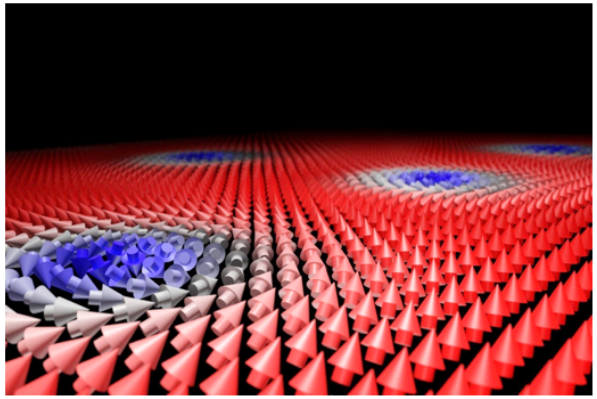
- find small test cases
- worth creating mock data
- Open Science COVID Analysis (OSCOVIDA)

  (`https://oscovida.github.io`)

  - Issues with data from RKI/JHU:
    - server not there
    - file missing
    - file incomplete (no rows of data, but header there)
    - suprising changes in file content/structure
    - changes of content in one data set but not in a linked data set (”Städteregion Aachen” changes to ”StadtRegion Aachen”)

Summary & Closing

- Reported from testing of Python-based open source projects (slides available)
- Acknowledgements
  - Marijan Beg, Martin Lang, Sam Holt (Ubermag)
  - Horizon 2020 European Research Projects OpenDreamKit (676541) and PaNOSC (823852)
  - EPSRC Programme grant on Skyrmionics (EP/N032128/1)
- We are *recruiting* (research software engineering / computational science) - get in touch!


Contact: hans.fangohr@mpsd.mpg.de, https://fangohr.github.io