Scientific Software Development

Samuel Farrens

ASTRODAT 08/09/2025 Durham

What does the following function compute?

```
def func(a):
    b = e = 0.7
    c = 1.0 - b
    d = 0.0
    g = (1 + a)
    x = b**2
    y = c * g**3 + d * g**2 + e

    return np.sqrt(x * y)
```

What about this function?

```
def hubble(redshift):
    hubble_const = 0.7
    matter = 0.3 * (1 + redshift) ** 3
    curvature = 0.0 * (1 + redshift) ** 2
    dark_energy = 0.7

    return np.sqrt(
        hubble_const**2 * (matter + curvature + dark_energy)
    )
```

(Scientific) Software Design Philosophy

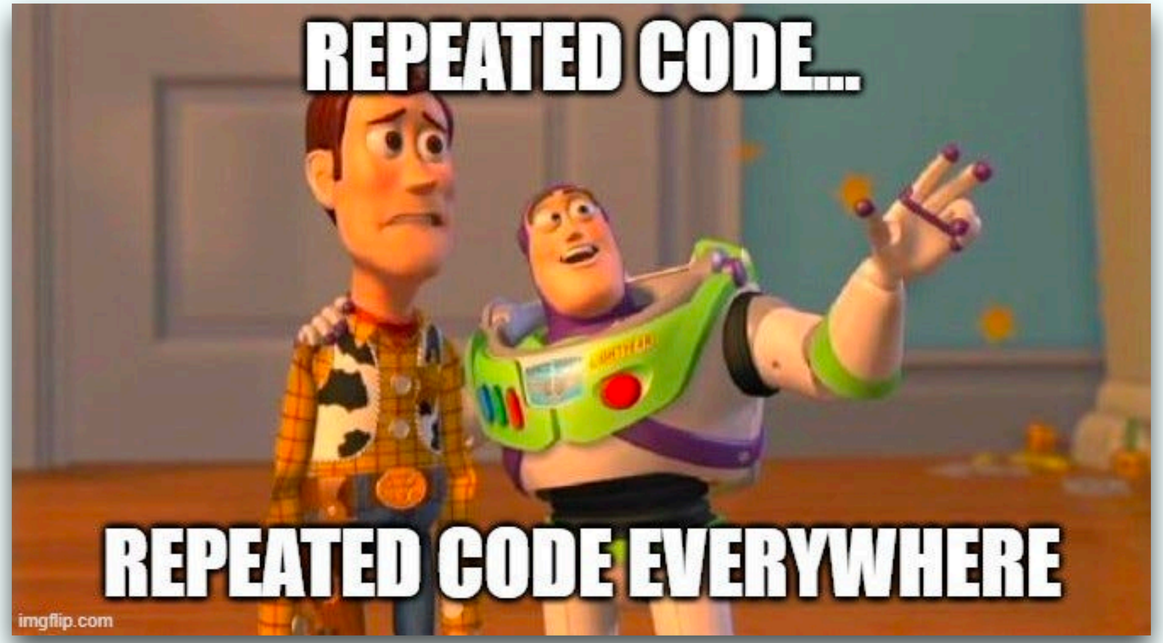
⚠️ THINK BEFORE YOU CODE!

(Scientific) Software Design Philosophy

Some important questions to ask yourself are:

- **Who** is this code for?
- **What** is the scope of this code?
- **How** should this code be used?
- **Where** will this code (likely) be run?
- **How long** will this code (likely) be used?

Is my code…



… a Jupyter Notebook just for me?

… an open source library for the whole world?

Core Concepts

1. Version Control - See Katherine's talk
2. Testing - See James' talk
3. Documentation
4. Continuous Integration/Deployment - See Sarah's talk
5. Optimisation - See Matt's talk
6. Distribution & Reproducible Research - See Bron's talk
7. Project Management & Open-Source Dev - See Alastair's talk

Follow-Up

github.com/sfarrens/Scientific-Software-Dev-Demo

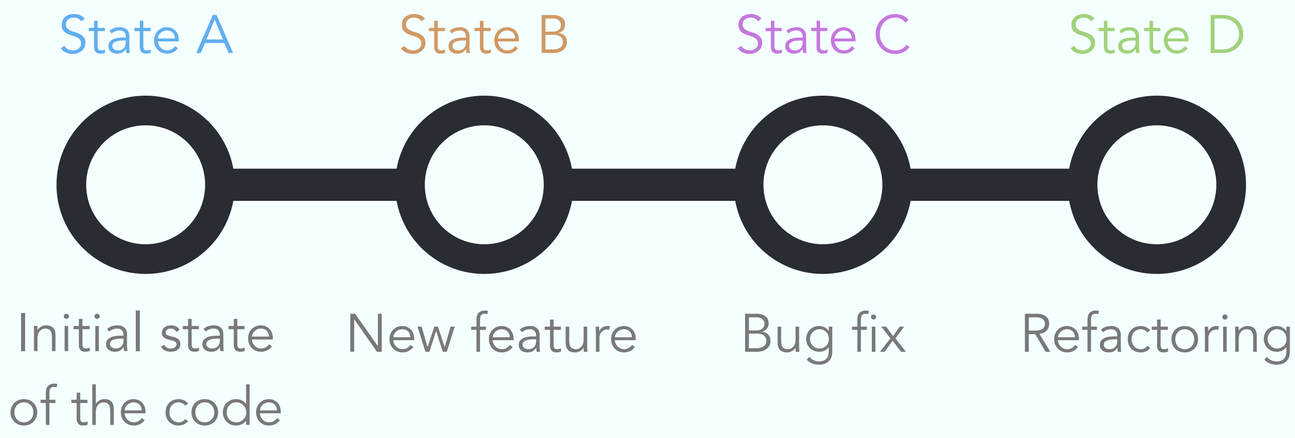
Version Control

Version Control

With *version control* tools like **Git** we can label various *states* of the code so that we can see exactly what has changed and when.

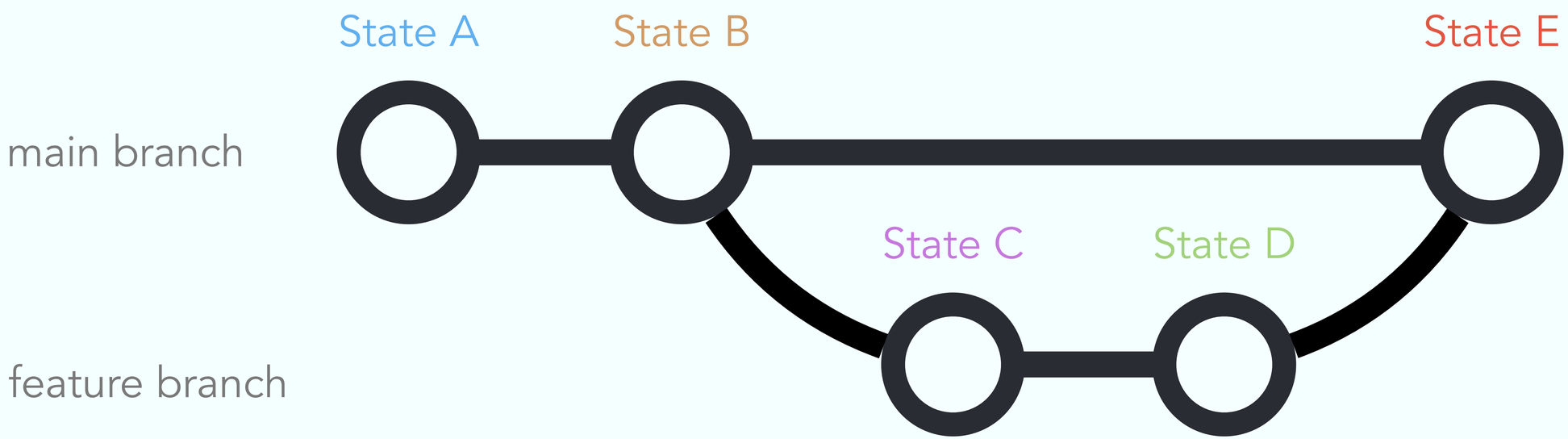
Git is a distributed version control system developed by Linus Torvalds (of the Linux fame) in the mid 2000s.

Version Control



Commits states are easier to manager if they have a particular **scope** and we can always revert to a previous **state**.

Version Control



Branches make it possible to better manage and **refine the scope** of of changes we want to make to the code. This also makes it easier for multiple people to contribute to the same code.

Version Control

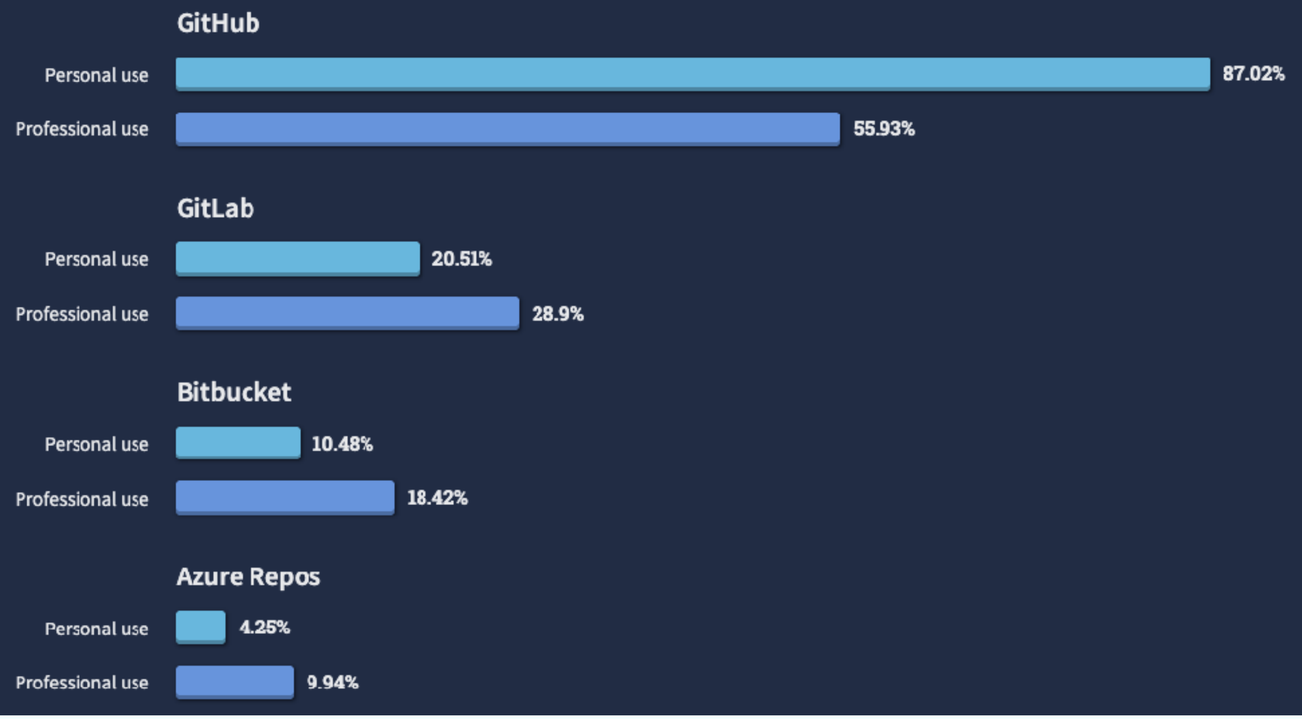
There are various cloud-based platforms for hosting Git repositories. Popular platforms, like **GitHub** and **GitLab**, not only offer a way to get more visibility for your software, but also include various tools that make it much easier to manage and maintain the code.

Version Control

Version control platforms

GitHub is the most popular Version Control for both personal and professional use. GitLab, Bitbucket, and Azure Repos are more likely used for professional purposes instead of personal.

67,035 responses



Git Workflow

Stable

Remote

main

Pull/Merge Request

feature

git pull

git push

Local

main

feature

git add
git commit

git branch
git checkout

Scoped

Testing

Testing

There are many different types of tests we can run on our code:

- **Unit tests:** Make sure all the functions do what they are supposed to
- **Integration tests:** Make sure a collection of functions play well together
- **End-to-end tests:** Make sure the whole code runs from beginning to end
- **Validation/Acceptance tests:** Make sure the code outputs achieve a certain criteria
- **Performance tests:** Make sure it can handle a certain amount of data in a certain amount of time

Unit Testing

Unit tests check that the smallest pieces (i.e. ‘units’) of our software work as intended. We simply check that for a fixed input we get a fixed output.

⚠️ Passing unit tests do not guarantee that our software is working correctly. Running these checks before and after commit states can reliably demonstrate that the individual units of code still work as expected after changes are made.

Unit Testing

```python
import numpy as np
import numpy.testing as npt

from cosmo import hubble

class TestCosmo:
    fid_cosmo = {
        "H0": 70,
        "omega_m_0": 0.3,
        "omega_k_0": 0.0,
        "omega_lambda_0": 0.7,
    }
    H_tolerance = 0.01
    z_range = np.array([0.0, 0.5, 1.0])
    H_expect = np.array([70, 91.60, 123.24])

    def test_hubble(self):
        H_vals = hubble(self.z_range, self.fid_cosmo)

        npt.assert_allclose(
            H_vals,
            self.H_expect,
            atol=self.H_tolerance,
            err_msg=(
                "The H(z) differs from expected values by more than "
                f"{self.H_tolerance} decimal places."
            ),
        )
```

Unit Testing

We can execute out unit tests using tools such as **pytest**.

```
pytest test_cosmo.py
```

```
============================================= test session starts ==============================================
platform darwin -- Python 3.11.5, pytest-7.4.2, pluggy-1.3.0 -- …/bin/python3.11
cachedir: .pytest_cache
rootdir: …/demo
collected 1 item

test_cosmo.py::TestCosmo::test_hubble PASSED                                                          [100%]

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

Unit Testing

```python
def hubble(redshift, cosmo_dict):
    hubble_const = cosmo_dict["H0"]
    matter = cosmo_dict["omega_m_0"] * (1 + redshift) ** 4
    curvature = cosmo_dict["omega_k_0"] * (1 + redshift) ** 2
    dark_energy = cosmo_dict["omega_lambda_0"]

    return np.sqrt(
        hubble_const**2 * (matter + curvature + dark_energy)
    )
```

```
============================================= test session starts ==============================================
platform darwin -- Python 3.11.5, pytest-7.4.2, pluggy-1.3.0 -- …/bin/python3.11
cachedir: .pytest_cache
rootdir: …/demo
collected 1 item

test_cosmo.py::TestCosmo::test_hubble FAILED                                                          [100%]

================================================== FAILURES ==================================================
```

Coverage

Coverage is a measure of the percentage of lines of code that have been explicitly tested (i.e. ‘covered’) by a unit test. The higher the coverage the more oversight we have on the robustness of our software.

⚠️ 100% coverage does not equate to perfect performance. This simply informs us that we have enough tests to account every line of code in our software.

Coverage

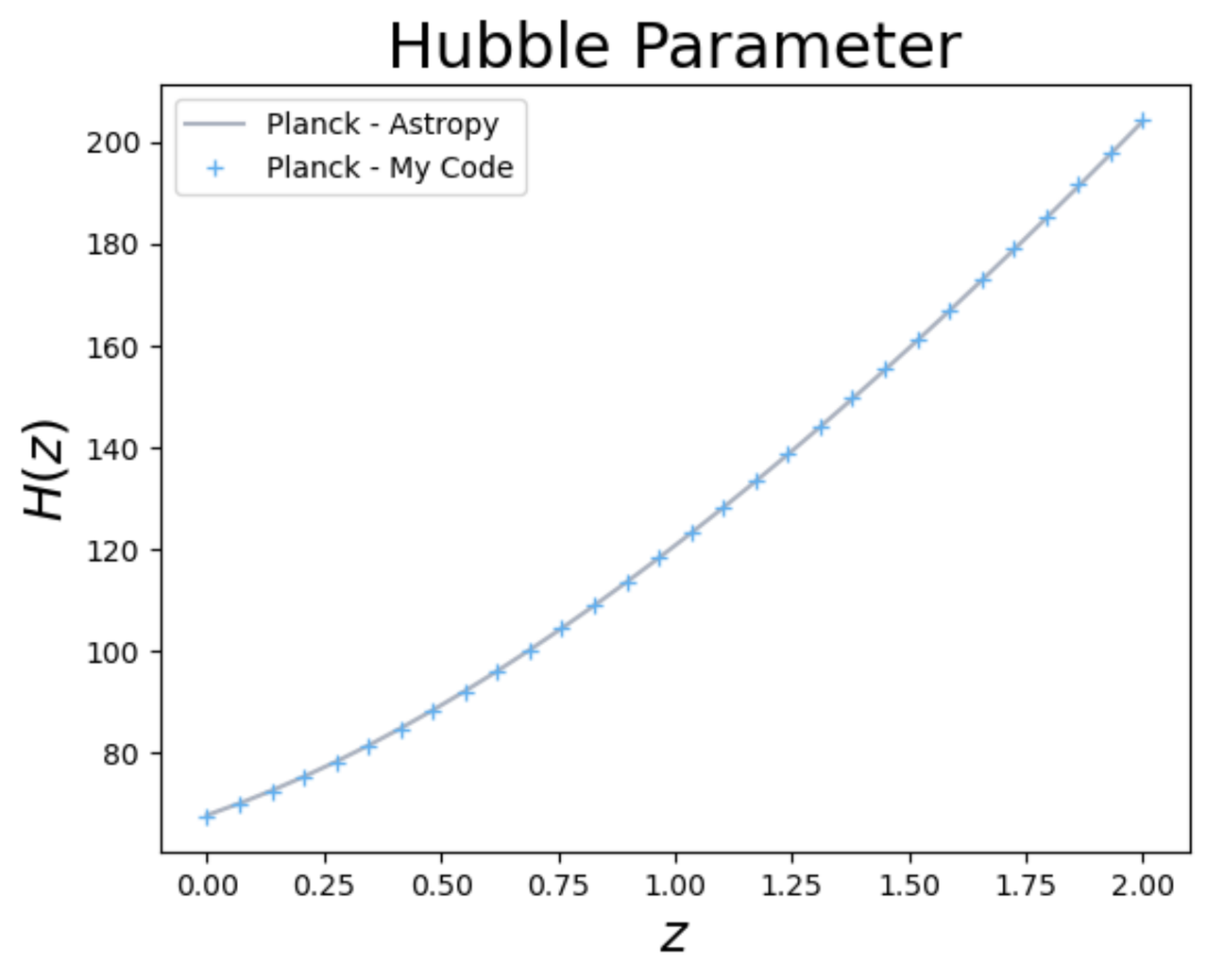
```
============================================================ test session starts =============================================================
platform darwin -- Python 3.13.5, pytest-8.4.1, pluggy-1.6.0 -- /Users/farrens/Documents/Library/mambaforge/envs/mycosmo/bin/
python3.13
cachedir: .pytest_cache
rootdir: /Users/farrens/Documents/Codes/github/Scientific-Software-Dev-Demo
configfile: pyproject.toml
testpaths: src/mycosmo
plugins: emoji-0.2.0, cov-6.2.1, pydocstyle-2.4.0
collected 5 items

src/mycosmo/__init__.py::PYDOCSTYLE PASSED 😃                                                                                          [ 20%]
src/mycosmo/constants.py::PYDOCSTYLE PASSED 😃                                                                                         [ 40%]
src/mycosmo/cosmology.py::PYDOCSTYLE PASSED 😃                                                                                         [ 60%]
src/mycosmo/tests/test_cosmology.py::TestCosmology::test_hubble PASSED 😃                                                              [ 80%]
src/mycosmo/tests/test_cosmology.py::TestCosmology::test_critical_density PASSED 😃                                                    [100%]

============================================================ tests coverage ============================================================
_____________________________________ coverage: platform darwin, python 3.13.5-final-0 ______________________________________

Name                                                                                                                    Stmts   Miss  Cover
-------------------------------------------------------------------------------------------------------------------------------------------
/Users/farrens/Documents/Library/mambaforge/envs/mycosmo/lib/python3.13/site-packages/mycosmo/__init__.py                   0      0   100%
/Users/farrens/Documents/Library/mambaforge/envs/mycosmo/lib/python3.13/site-packages/mycosmo/constants.py                  2      0   100%
/Users/farrens/Documents/Library/mambaforge/envs/mycosmo/lib/python3.13/site-packages/mycosmo/cosmology.py                 11      0   100%
-------------------------------------------------------------------------------------------------------------------------------------------
TOTAL                                                                                                                      13      0   100%
============================================================ 5 passed in 0.13s =============================================================
```

Validation Testing

Validation/acceptance tests compare the global output of our software against some predefined requirements. This is something to consider when deciding on the overall scope of the project.

⚠️ Passing validation/acceptance tests does not necessarily mean that everything is working perfectly. Beware of confirmation bias!

Validation Testing



Documentation

Documentation

There are also many different ways to document our code:

- **README:** The TL;DR of the code, what the code is and how use it
- **API Documentation:** More comprehensive information about the *Application Programming Interface (API)*, in other words how all the functions, classes, etc. work
- **Examples/Demos:** Concrete examples of how to use the code in practice

API Documentation

```python
def hubble(redshift, cosmo_dict):
    """Hubble Parameter.

    Calculate the Hubble parameter at a given redshift using the cosmological parameter values provided.

    """
    hubble_const = cosmo_dict["H0"]
    matter = cosmo_dict["omega_m_0"] * (1 + redshift) ** 3
    curvature = cosmo_dict["omega_k_0"] * (1 + redshift) ** 2
    dark_energy = cosmo_dict["omega_lambda_0"]

    return np.sqrt(
        hubble_const**2 * (matter + curvature + dark_energy)
    )
```

API Documentation

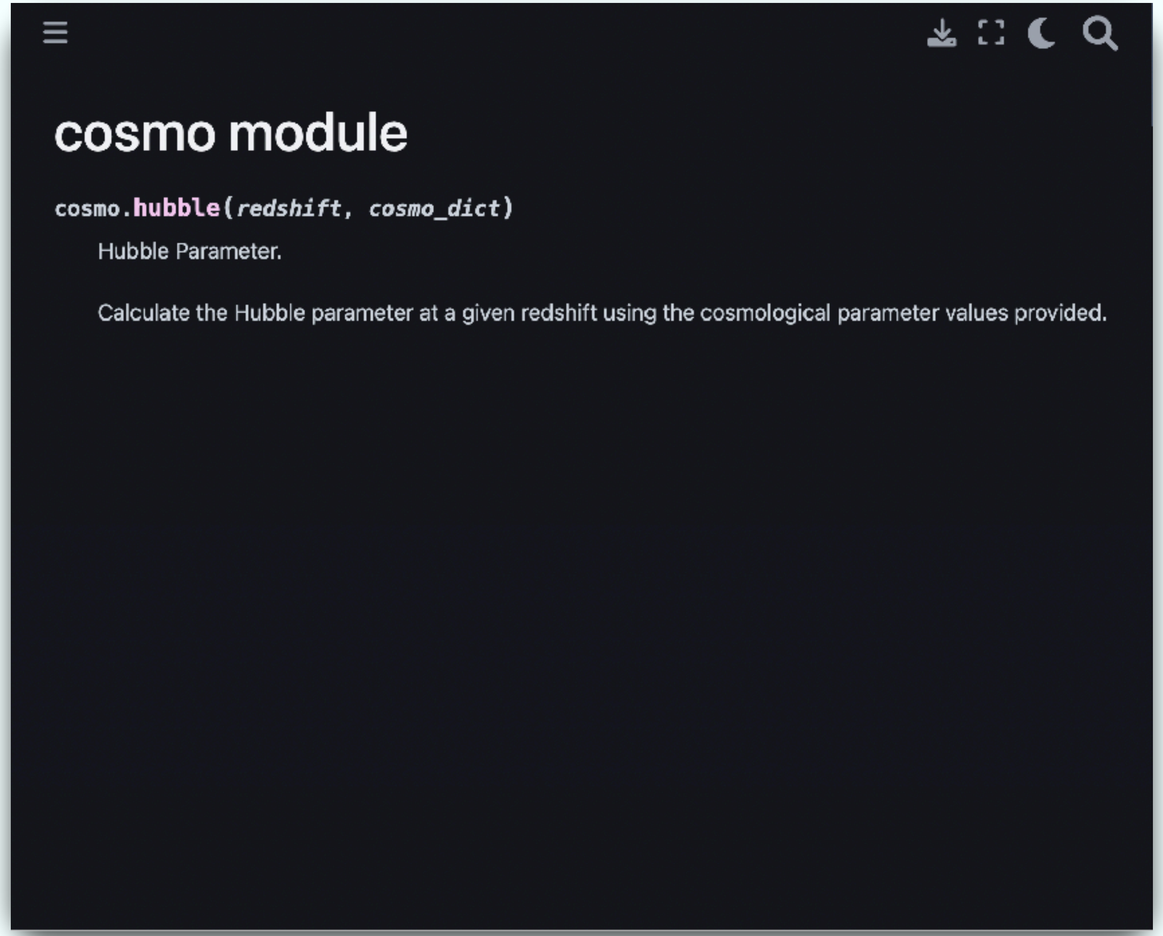
Sphinx is a tool for automatically sourcing the API of a Python package and converting the *docstrings* into HTML pages.

```
mkdir docs
sphinx-quickstart docs
sphinx-apidoc -Mfeo docs/source .
```

API Documentation

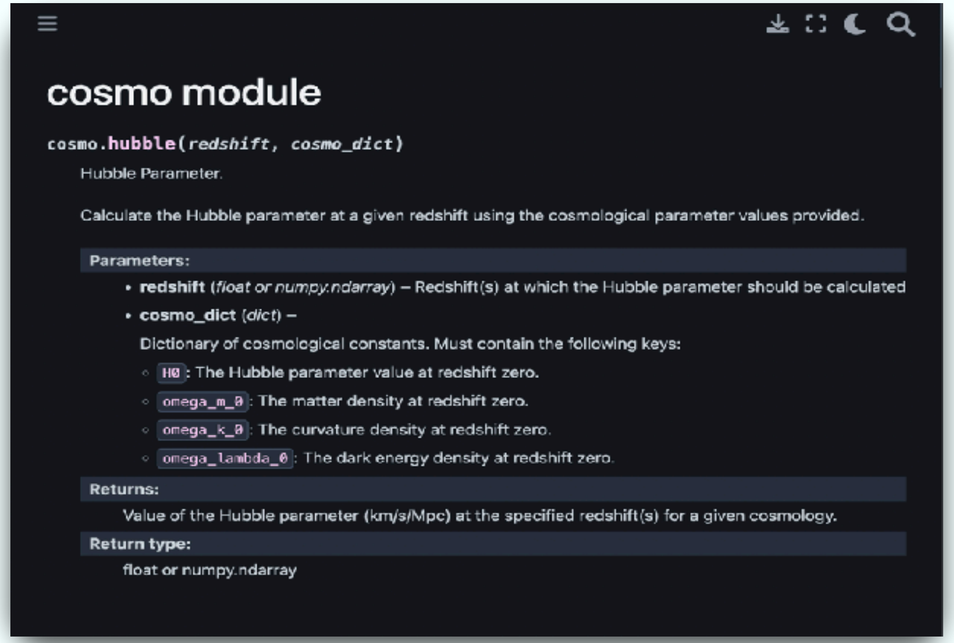
```python
def hubble(redshift, cosmo_dict):
    """Hubble Parameter.

    Calculate the Hubble parameter at a given redshift using the cosmological parameter values provided.

    """
    hubble_const = cosmo_dict["H0"]
    matter = cosmo_dict["omega_m_0"] * (1 + redshift) ** 3
    curvature = cosmo_dict["omega_k_0"] * (1 + redshift) ** 2
    dark_energy = cosmo_dict["omega_lambda_0"]

    return np.sqrt(
        hubble_const**2 * (matter + curvature + dark_energy)
    )
```

```
sphinx-build docs/source docs/build
```



API Documentation

```
"""Hubble Parameter.

Calculate the Hubble parameter at a given redshift using the cosmological parameter values
provided.

Parameters
----------
redshift : float or numpy.ndarray
    Redshift(s) at which the Hubble parameter should be calculated
cosmo_dict : dict
    Dictionary of cosmological constants. Must contain the following keys:

    * ``H0``: The Hubble parameter value at redshift zero.
    * ``omega_m_0``: The matter density at redshift zero.
    * ``omega_k_0``: The curvature density at redshift zero.
    * ``omega_lambda_0``: The dark energy density at redshift zero.

Returns
-------
float or numpy.ndarray
    Value of the Hubble parameter (km/s/Mpc) at the specified redshift(s) for a given cosmology.

"""
```



cosmo module

cosmo.hubble(*redshift*, *cosmo_dict*)

Hubble Parameter.

Calculate the Hubble parameter at a given redshift using the cosmological parameter values provided.

Parameters:

- **redshift** (*float or numpy.ndarray*) – Redshift(s) at which the Hubble parameter should be calculated
- **cosmo_dict** (*dict*) –

  Dictionary of cosmological constants. Must contain the following keys:

  - `H0`: The Hubble parameter value at redshift zero.
  - `omega_m_0`: The matter density at redshift zero.
  - `omega_k_0`: The curvature density at redshift zero.
  - `omega_lambda_0`: The dark energy density at redshift zero.

Returns:

Value of the Hubble parameter (km/s/Mpc) at the specified redshift(s) for a given cosmology.

Return type:

float or numpy.ndarray

API Documentation

```
"""Hubble Parameter.

Calculate the Hubble parameter at a given redshift using the cosmological parameter values
provided.

Parameters
----------
redshift : float or numpy.ndarray
    Redshift(s) at which the Hubble parameter should be calculated
cosmo_dict : dict
    Dictionary of cosmological constants. Must contain the following keys:

    * ``H0``: The Hubble parameter value at redshift zero.
    * ``omega_m_0``: The matter density at redshift zero.
    * ``omega_k_0``: The curvature density at redshift zero.
    * ``omega_lambda_0``: The dark energy density at redshift zero.

Returns
-------
float or numpy.ndarray
    Value of the Hubble parameter (km/s/Mpc) at the specified redshift(s) for a given cosmology.

Notes
-----
This function implements the calculation of the Hubble parameter as follows:

.. math::
    H(z) = \sqrt{H_0^2 (\Omega_{m,0}(1+z)^3 + \Omega_{k,0}(1+z)^2 +
        \Omega_{\Lambda,0})}


"""
```

cosmo module

cosmo.hubble(*redshift*, *cosmo_dict*)

Hubble Parameter.

Calculate the Hubble parameter at a given redshift using the cosmological parameter values provided.

Parameters:

- **redshift** (*float or numpy.ndarray*) – Redshift(s) at which the Hubble parameter should be calculated
- **cosmo_dict** (*dict*) –

  Dictionary of cosmological constants. Must contain the following keys:

  - `H0`: The Hubble parameter value at redshift zero.
  - `omega_m_0`: The matter density at redshift zero.
  - `omega_k_0`: The curvature density at redshift zero.
  - `omega_lambda_0`: The dark energy density at redshift zero.

Returns:

Value of the Hubble parameter (km/s/Mpc) at the specified redshift(s) for a given cosmology.

Return type:

float or numpy.ndarray

Notes

This function implements the calculation of the Hubble parameter as follows:

$$H(z) = \sqrt{H_0^2(\Omega_{m,0}(1+z)^3 - \Omega_{k,0}(1+z)^2 + \Omega_{\Lambda,0})}$$

Scientific software tip!

Put equations and/or cite papers

API Documentation

```
"""Hubble Parameter.

Calculate the Hubble parameter at a given redshift using the cosmological parameter values
provided.

Parameters
----------
redshift : float or numpy.ndarray
    Redshift(s) at which the Hubble parameter should be calculated
cosmo_dict : dict
    Dictionary of cosmological constants. Must contain the following keys:

    * ``H0``: The Hubble parameter value at redshift zero.
    * ``omega_m_0``: The matter density at redshift zero.
    * ``omega_k_0``: The curvature density at redshift zero.
    * ``omega_lambda_0``: The dark energy density at redshift zero.

Returns
-------
float or numpy.ndarray
    Value of the Hubble parameter (km/s/Mpc) at the specified redshift(s) for a given cosmology.

Notes
-----
This function implements the calculation of the Hubble parameter as follows:

.. math::
    H(z) = \sqrt{H_0^2 (\Omega_{m,0}(1+z)^3 + \Omega_{k,0}(1+z)^2 +
        \Omega_{\Lambda,0})}

Example
-------
>>> from mycosmo.cosmology import hubble
>>> cosmo_dict = {"H0": 70, "omega_m_0": 0.3, "omega_k_0": 0.0, "omega_lambda_0": 0.7}
>>> hubble(0.0, cosmo_dict)
70.0

"""
```

cosmo module

cosmo.hubble(*redshift*, *cosmo_dict*)

Hubble Parameter.

Calculate the Hubble parameter at a given redshift using the cosmological parameter values provided.

Parameters:

- **redshift** (*float or numpy.ndarray*) – Redshift(s) at which the Hubble parameter should be calculated
- **cosmo_dict** (*dict*) –

  Dictionary of cosmological constants. Must contain the following keys:

  - `H0` : The Hubble parameter value at redshift zero.
  - `omega_m_0` : The matter density at redshift zero.
  - `omega_k_0` : The curvature density at redshift zero.
  - `omega_lambda_0` : The dark energy density at redshift zero.

Returns:

Value of the Hubble parameter (km/s/Mpc) at the specified redshift(s) for a given cosmology.

Return type:

float or numpy.ndarray

Notes

This function implements the calculation of the Hubble parameter as follows:

$$H(z) = \sqrt{H_0^2(\Omega_{m,0}(1+z)^3 + \Omega_{k,0}(1+z)^2 + \Omega_{\Lambda,0})}$$

Example

```
>>> from mycosmo.cosmology import hubble
>>> cosmo_dict = {"H0": 70, "omega_m_0": 0.3, "omega_k_0": 0.0, "omega_lambda_0": 0.7}
>>> hubble(0.0, cosmo_dict)
70.0
```

Continuous Integration/Deployment

Continuous Integration/Deployment

Platforms like GitHub, GitLab, etc. offer free servers and tools for performing:

- **Continuous integration:** Automated running of unit/integration tests every time the code is changed
- **Continuous deployment:** Automated building and deployment of e.g. API documentation

GitHub

Continuous Integration

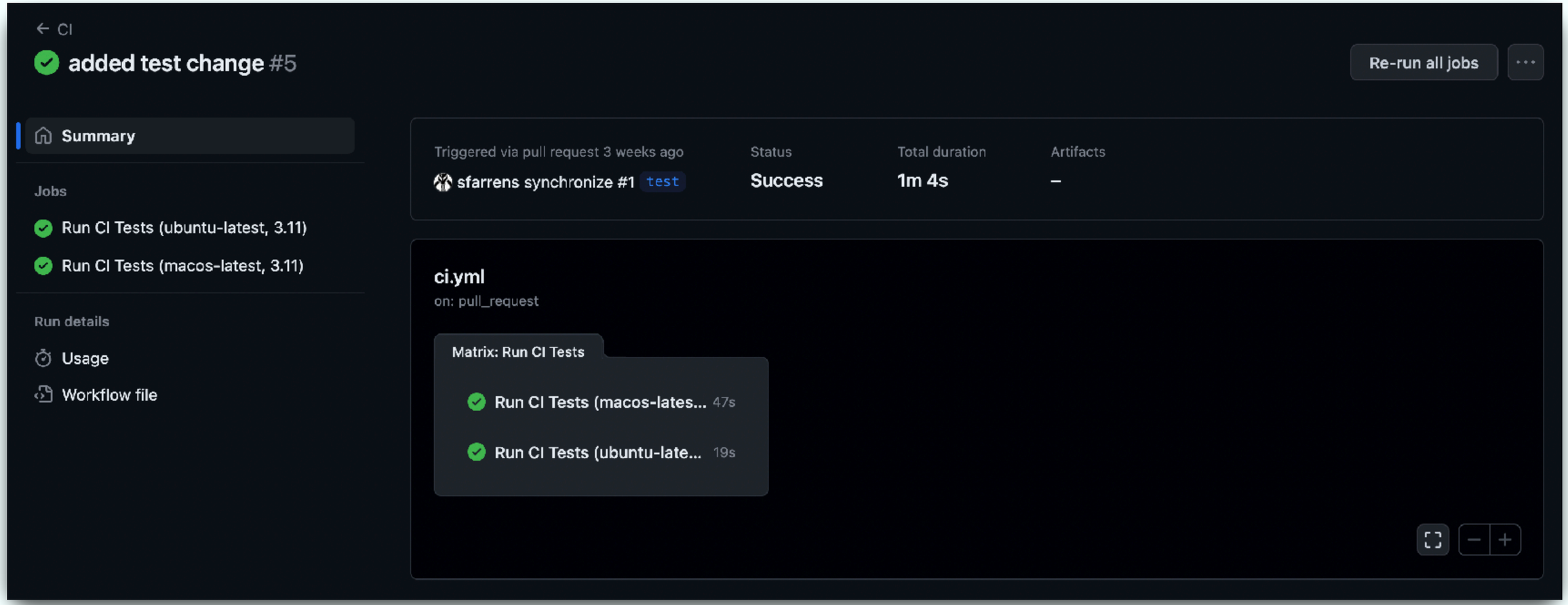
```yaml
name: CI

on:
  pull_request:
    branches:
     - main

jobs:
  test-full:
    name: Run CI Tests
    runs-on: ${{ matrix.os }}

    strategy:
      fail-fast: false
      matrix:
        os: [ubuntu-latest, macos-latest]
        python-version: ["3.11"]

    steps:
      - name: Checkout
        uses: actions/checkout@v3

      - name: Set up Python ${{ matrix.python-version }}
        uses: actions/setup-python@v4
        with:
          python-version: ${{ matrix.python-version }}

      - name: Check Python version
        run: python --version

      - name: Install dependencies
        run: python -m pip install ".[test]"

      - name: Run tests
        run: python -m pytest
```

`.github/workflows/ci.yml`

This defines a GitHub Actions CI workflow that will run with Python 3.11 on Ubuntu and macOS. Unit tests are run using pytest.

GitHub

Continuous Integration



GitHub

Continuous Deployment

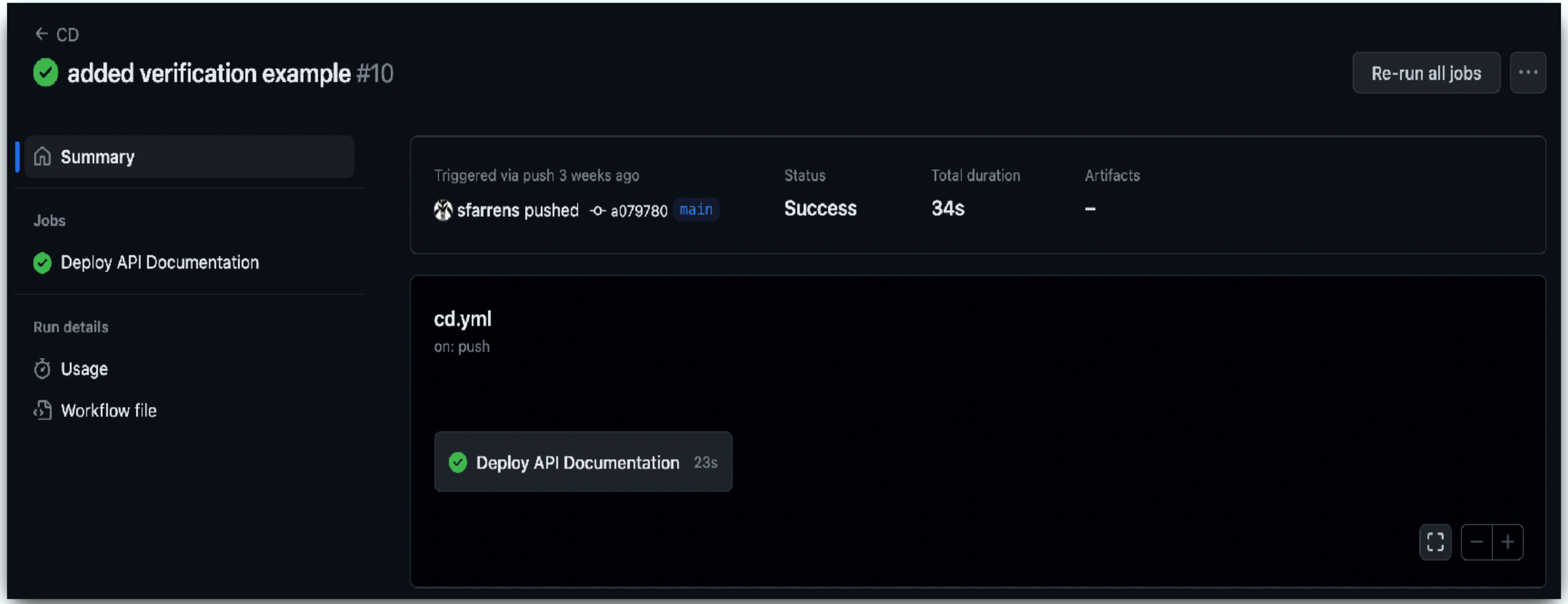
```yaml
name: CD

on:
  push:
    branches:
     - main

jobs:
  docs:
    name: Deploy API Documentation
    runs-on: ubuntu-latest

    steps:
      - name: Checkout
        uses: actions/checkout@v3

      - name: Set up Python ${{ matrix.python-version }}
        uses: actions/setup-python@v4
        with:
          python-version: 3.11

      - name: Check Python version
        run: python --version

      - name: Install dependencies
        run: python -m pip install ".[docs]"

      - name: Build API documentation
        run: |
          sphinx-apidoc -Mfeo docs/source mycosmo
          sphinx-build docs/source docs/build

      - name: Deploy API documentation
        uses: peaceiris/actions-gh-pages@v3.5.9
        with:
          github_token: ${{ secrets.GITHUB_TOKEN }}
          publish_dir: docs/build
```

`.github/workflows/cd.yml`

This defines a GitHub Actions CI workflow that will run with Python 3.11 on Ubuntu. API HTML is built using Sphinx and then deployed as a website.

GitHub

Continuous Deployment



Optimisation

Optimisation

Everyone wants their code to be as fast and scalable as possible and there are may ways you can go about this:

- **Vectorisation:** e.g. Numpy
- **Multiprocessing/Multithreading:** e.g. Joblib, mpi4py
- **JIT (Just-In-Time) compilation:** e.g. Numba, Jax
- **GPU (Graphics Processing Unit) computation:** e.g. Jax, TensorFlow

However, we should investigate where our *bottlenecks* are before investing time in optimising the code.

⚠️ Slow down before you speed up!

Profiling

```python
#! /usr/bin/env python

from sys import exit
import numpy as np

from mycosmo.cosmo import hubble


def main():

    cosmo_dict = {"H0": 70, "omega_m_0": 0.3, "omega_k_0": 0.0, "omega_lambda_0": 0.7}
    z_range = np.linspace(0, 1)
    print(hubble(z_range, cosmo_dict))


if __name__ == "__main__":
    exit(main())
```

scripts/calc_hubble.py

```
python -m cProfile -o mycosmo.pstats scripts/calc_hubble.py
```

Profiling

```
snakeviz mycosmo.pstats
```

Distribution
&
Reproducible Research

Distribution

We can share our code with the rest of the world in the following ways:

- **Cloud repo:** Simply put it on GitHub, GitLab, etc. and let people download/clone it
- **Package manager:** Upload it to a package managing repository like PyPI or Conda

Distribution

```
[project]
name = "mycosmo"
readme = "README.md"
requires-python = ">=3.11"
authors = [{ "name" = "Samuel Farrens", "email" = "samuel.farrens@cea.fr" }]
maintainers = [{ "name" = "Samuel Farrens", "email" = "samuel.farrens@cea.fr" }]
description = 'This is an example cosmology package.'
dependencies = ["numpy"]
version = "0.0.1"

[project.optional-dependencies]
docs = ["numpydoc", "sphinx", "sphinx-book-theme"]
release = ["build", "twine"]
test = ["pytest"]

[tool.pytest.ini_options]
addopts = ["--verbose"]
testpaths = ["mycosmo"]
```

Distribution

```
python -m build
```

```
dist
├── mycosmo-0.0.1-py3-none-any.whl
└── mycosmo-0.0.1.tar.gz
```

Distribution

The Python Package Index (PyPI) is the main repository for distributing Python packages. Once your package has been uploaded, anyone in the world can install it with `pip`.

```
twine upload dist/*
pip install mycosmo
```

Reproducible Research

Reproducing scientific results is fundamental to our confidence and understanding. There are various steps we can take to try to ensure that our software provides consistent results over appropriate timescales.

- **Open-source development:** Ensuring that others can find and see our code is fundamental to reproducible research
- **Software environments:** Tools such as **Conda** or **virtualenv** can be used to ensure consistency and version control of packages
- **Container systems:** Tools such as **Docker** and **Singularity** can be used to provide virtual operating systems where the whole system can be controlled

Reproducible Research

Docker is a platform that provides OS-level virtualisation via a *container* system. This allows users to define a dedicated virtual operating system with all the required dependencies pre-installed.

Docker

```
FROM python:3.11

WORKDIR /home

COPY . .

RUN pip install .
```

Docker

```
FROM python:3.11

WORKDIR /home

COPY . .

RUN pip install .
```

Dockerfile

```
docker build -t mycosmo .
```

Docker

```
docker run --rm -it mycosmo
```

```
Python 3.11.5 (main, Sep  7 2023, 18:53:16) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from mycosmo.cosmo import hubble
>>> cosmo_dict = {"H0": 70, "omega_m_0": 0.3, "omega_k_0": 0.0, "omega_lambda_0": 0.7}
>>> hubble(0.0, cosmo_dict)
70.0
```

Project Management
&
Open-Souce Development

Lessons

- Think about **roles for managing projects**
  - Maintainers - Oversee the global management, integrity and direction of the project
  - Reviewers - Review pull/merge requests and ensure the integrity of the codebase
  - Developers - Add new features, fix bugs, and other improvements
- Pull/Merge requests are **conversations not examinations**
  - Don’t take things personally
  - Ask questions if things are not clear
  - **Be polite!**

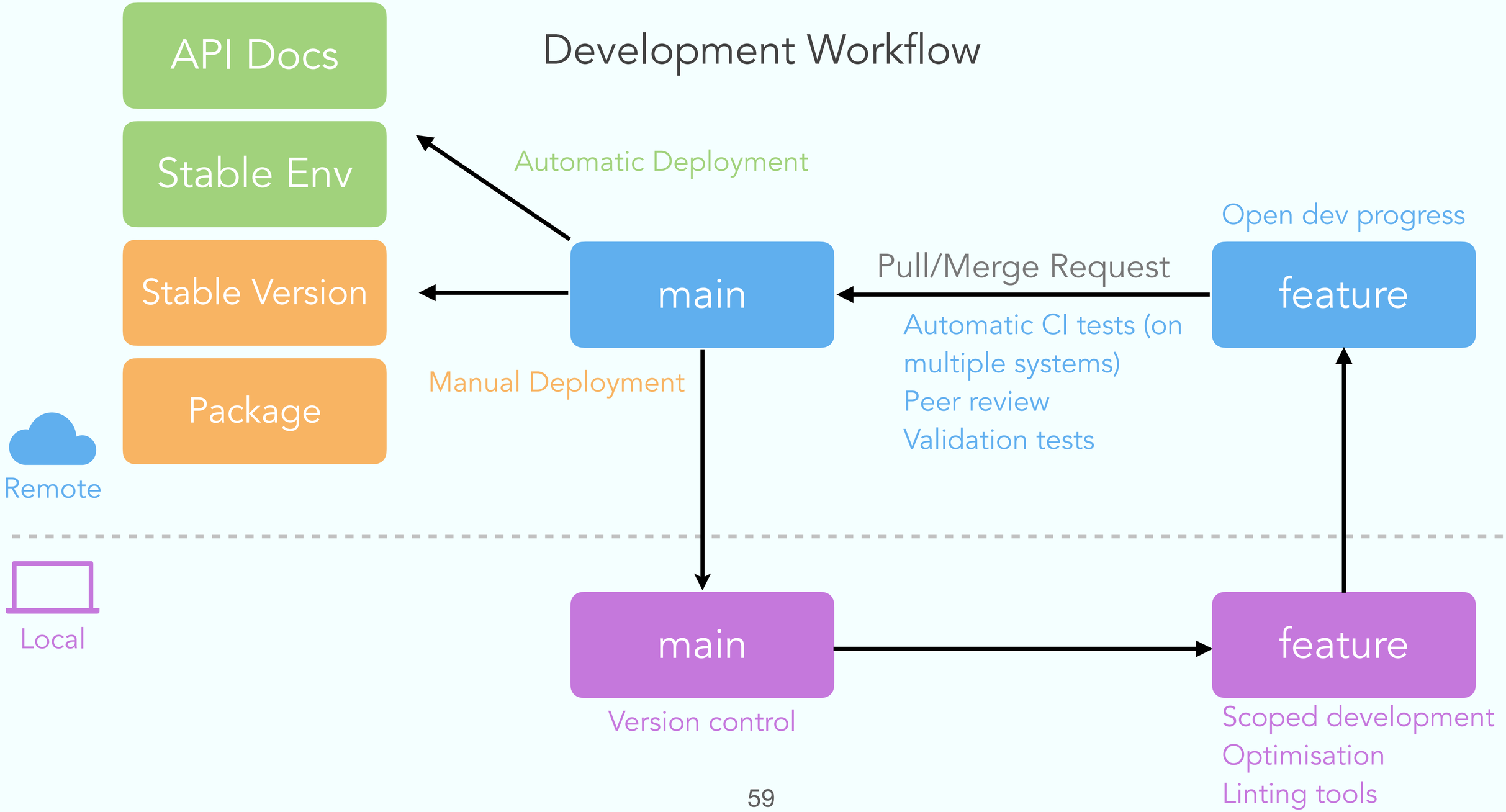
Lessons

- Choose **the right license**
  - Your institute/collaboration may have requirements
  - Generally less restrictive is better for the community
- Try to set **milestones**
  - Set a timeline for a given release
  - Identify critical bug fixes and features to be added
  - Review the status of issues and pull/merge requests

Lessons

- **Avoid re-inventing the *wheel***
  - Don’t be afraid to contribute to other open-source projects
  - Look at existing tools and think about what is really missing
- **Think about the users!**
  - Make sure your code runs on the systems people need to us (Linux, macOS, etc.)
  - Write good documentation
  - Think about how people will actually use the code

Development Workflow



Conclusions

- *Version control* is fundamental for **maintaining** your software
- Platforms like *GitHub/GitLab* will help you **manage** the development of your software (automatic *CI/CD*) and give it more **visibility**
- *Tests* make your code more **robust and reliable**
- Good *documentation* will make your code easier to **use and interpret**
- *Packaging* your code will allow you to *distribute* it to the **whole world**
- *Profile* your code before trying to *optimise* it!
- You can make your **scientific results** more *reproducible* by thinking about how you manage (i.e. *open-source*) and distribute it (e.g. *environments* and *containers*)