Python Project Periphery:

All the small stuff they don't teach you

Jacob Wilkins

Scientific Computing Department, STFC

April 29, 2025

Before we start

Before we start

- One of the most common things I hear from scientists is:

  "But my code isn't good enough to publish!"

Before we start

- One of the most common things I hear from scientists is:

  "But my code isn't good enough to publish!"
- After contributing **many** LoC (and hopefully removing a few).

Before we start

- One of the most common things I hear from scientists is:

  "But my code isn't good enough to publish!"
- After contributing **many** LoC (and hopefully removing a few).
- The only **bad** coder is one who can't/doesn't take feedback.

Before we start

- One of the most common things I hear from scientists is:

  "But my code isn't good enough to publish!"
- After contributing **many** LoC (and hopefully removing a few).
- The only **bad** coder is one who can't/doesn't take feedback.
- If:

Before we start

- One of the most common things I hear from scientists is:

  "But my code isn't good enough to publish!"
- After contributing **many** LoC (and hopefully removing a few).
- The only **bad** coder is one who can't/doesn't take feedback.
- If:
  - You've written something useful.

Before we start

- One of the most common things I hear from scientists is:

  "But my code isn't good enough to publish!"
- After contributing **many** LoC (and hopefully removing a few).
- The only **bad** coder is one who can't/doesn't take feedback.
- If:
  - You've written something useful.
  - You can describe what you intended to do.

Before we start

- One of the most common things I hear from scientists is:

  "But my code isn't good enough to publish!"
- After contributing **many** LoC (and hopefully removing a few).
- The only **bad** coder is one who can't/doesn't take feedback.
- If:
  - You've written something useful.
  - You can describe what you intended to do.
  - You're willing to accept outside contributions.

Before we start

- One of the most common things I hear from scientists is:

  "But my code isn't good enough to publish!"
- After contributing **many** LoC (and hopefully removing a few).
- The only **bad** coder is one who can't/doesn't take feedback.
- If:
  - You've written something useful.
  - You can describe what you intended to do.
  - You're willing to accept outside contributions.
  - You're willing to respond to and learn from those.

Before we start

- One of the most common things I hear from scientists is:

  "But my code isn't good enough to publish!"
- After contributing **many** LoC (and hopefully removing a few).
- The only **bad** coder is one who can't/doesn't take feedback.
- If:
  - You've written something useful.
  - You can describe what you intended to do.
  - You're willing to accept outside contributions.
  - You're willing to respond to and learn from those.
- **Your code will be fine and will be useful.**

Before we start

- This is not a guide to programming.
  - There will be references where needed.

Before we start

- This is not a guide to programming.
    - There will be references where needed.
- Instructions are written for a Linux terminal.
    - Works on Windows and GUI tools are available.

Before we start

- This is not a guide to programming.
  - There will be references where needed.
- Instructions are written for a Linux terminal.
  - Works on Windows and GUI tools are available.
- I will be using the emacs editor for most things.
  - Save yourself! Don't fall into this trap.

Before we start

- This is not a guide to programming.
  - There will be references where needed.
- Instructions are written for a Linux terminal.
  - Works on Windows and GUI tools are available.
- I will be using the emacs editor for most things.
  - Save yourself! Don't fall into this trap.
- This is a basic introduction to give you the tools get started and the language to ask questions.

Before we start

- This is not a guide to programming.
  - There will be references where needed.
- Instructions are written for a Linux terminal.
  - Works on Windows and GUI tools are available.
- I will be using the emacs editor for most things.
  - Save yourself! Don't fall into this trap.
- This is a basic introduction to give you the tools get started and the language to ask questions.
  - Sadly, it will not give you domain expertise (yet).

Before we start

- Don't dive immediately into writing a project.

Before we start

- Don't dive immediately into writing a project.
- Don't reinvent the wheel!

Before we start

- Don't dive immediately into writing a project.
- Don't reinvent the wheel!
- Ask/search around for pre-written tools that do what you want.

Before we start

- Don't dive immediately into writing a project.
- Don't reinvent the wheel!
- Ask/search around for pre-written tools that do what you want.
- Libraries have combined experience of $\sim$100+ person-years.

Before we start

- Don't dive immediately into writing a project.
- Don't reinvent the wheel!
- Ask/search around for pre-written tools that do what you want.
- Libraries have combined experience of $\sim$100+ person-years.
- If the library doesn't do **exactly** what you want consider:

Before we start

- Don't dive immediately into writing a project.
- Don't reinvent the wheel!
- Ask/search around for pre-written tools that do what you want.
- Libraries have combined experience of $\sim$100+ person-years.
- If the library doesn't do **exactly** what you want consider:
    - Asking the library if they're willing/plan to implement what you want.

Before we start

- Don't dive immediately into writing a project.
- Don't reinvent the wheel!
- Ask/search around for pre-written tools that do what you want.
- Libraries have combined experience of $\sim$100+ person-years.
- If the library doesn't do **exactly** what you want consider:
    - Asking the library if they're willing/plan to implement what you want.
    - Using the library as a basis (dependency) for your work!

Before we start

- Don't dive immediately into writing a project.
- Don't reinvent the wheel!
- Ask/search around for pre-written tools that do what you want.
- Libraries have combined experience of $\sim$100+ person-years.
- If the library doesn't do **exactly** what you want consider:
  - Asking the library if they're willing/plan to implement what you want.
  - Using the library as a basis (dependency) for your work!
  - Contributing the feature back to the library.
    N.B. Make sure to read their guidelines!

Glossary

- Files
- Folders
- Commands
- Keywords

Before we start

Resources for this talk are available at:
`https://github.com/oerc0122/Python-Project-Periphery`

Before we start

- We need to make GitHub know it's us when we talk.
- Need to generate an ssh key.
- On linux/Gitbash run `ssh-keygen`
- You do not need to add a passphrase for this.
- On GitHub, go to your avatar (top-right)
- Settings → SSH and GPG keys → New SSH key
- Paste contents of `~/.ssh/id.rsa` into box.

The code

Introducing florp

```python
import numpy as np

CBRT_UNITY_IM = np.sqrt(3)/2 * 1j


def florp(a, b, c):
    det = b**2 - (4*a*c)

    return ((-b + np.sqrt(det)) / (2*a),
            (-b - np.sqrt(det)) / (2*a))

def florp2(a, b, c, d):
    q = (3*a*c - b**2) / (9*a**2)
    r = (9*a*b*c - 27*a**2*d - 2*b**3) / (54*a**3)

    s = np.cbrt(r + np.sqrt(q**3 + r**2))
    t = np.cbrt(r - np.sqrt(q**3 + r**2))

    x1 = s + t - (b/3*a)
    x2 = -(s + t)/2 - (b/3*a) + CBRT_UNITY_IM * (s - t)
    x3 = -(s + t)/2 - (b/3*a) - CBRT_UNITY_IM * (s - t)
    return x1, x2, x3
```

Florpulate it

- Florp is a very sophisticated library.

$$
\begin{aligned}
&\text{florp} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \\
&\text{florp2} = \Re(\sqrt[3]{1})(s+t) + \Im(\sqrt[3]{1})(s-t) + p \\
&\text{where} \\
&s = \left[r + \sqrt{q^3 + (r^2)}\right]^{\frac{1}{3}}, t = \left[r - \sqrt{q^3 + (r^2)}\right]^{\frac{1}{3}} \\
&p = \frac{-b}{3a}, q = \frac{3ac - b^2}{9a^2}, r = \frac{9abc - 27a^2 d - 2b^3}{54a^3}.
\end{aligned}
$$

Florpulate it

- Florp is a very sophisticated library.
- It clearly performs florpulation.

$$\text{florp} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$\text{florp2} = \Re(\sqrt[3]{1})(s+t) + \Im(\sqrt[3]{1})(s-t) + p$$

where

$$s = \left[r + \sqrt{q^3 + (r^2)}\right]^{\frac{1}{3}}, t = \left[r - \sqrt{q^3 + (r^2)}\right]^{\frac{1}{3}}$$

$$p = \frac{-b}{3a}, q = \frac{3ac - b^2}{9a^2}, r = \frac{9abc - 27a^2d - 2b^3}{54a^3}.$$

Florpulate it

- Florp is a very sophisticated library.
- It clearly performs florpulation.
- What is florpulation?

$$\mathrm{florp} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$\mathrm{florp2} = \Re(\sqrt[3]{1})(s+t) + \Im(\sqrt[3]{1})(s-t) + p$$

where

$$s = \left[r + \sqrt{q^3 + (r^2)}\right]^{\frac{1}{3}}, t = \left[r - \sqrt{q^3 + (r^2)}\right]^{\frac{1}{3}}$$

$$p = \frac{-b}{3a}, q = \frac{3ac - b^2}{9a^2}, r = \frac{9abc - 27a^2d - 2b^3}{54a^3}.$$

Florpulate it

- Florp is a very sophisticated library.
- It clearly performs florpulation.
- What is florpulation?
- Do you think this is a sensible name for this project?

$$\text{florp} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$\text{florp2} = \Re(\sqrt[3]{1})(s + t) + \Im(\sqrt[3]{1})(s - t) + p$$

where

$$s = \left[r + \sqrt{q^3 + (r^2)} \right]^{\frac{1}{3}}, t = \left[r - \sqrt{q^3 + (r^2)} \right]^{\frac{1}{3}}$$

$$p = \frac{-b}{3a}, q = \frac{3ac - b^2}{9a^2}, r = \frac{9abc - 27a^2 d - 2b^3}{54a^3}.$$

Sensible names for sensible projects

- Before anything else need to give the project usable names!

Sensible names for sensible projects

- Before anything else need to give the project usable names!
- Let's choose `polysolve.py` and rename functions accordingly.

Sensible names for sensible projects

- Before anything else need to give the project usable names!
- Let's choose `polysolve.py` and rename functions accordingly.
- Next thing is to get it saved and tracked.

GitHub

Using GitHub

- This assumes you have some familiarity with git and GitHub.

Adjust to taste

Other repositories do exist, such as GitLab, BitBucket, etc.

Using GitHub

- This assumes you have some familiarity with git and GitHub.
- Also requires you to have a GitHub account.

Adjust to taste

Other repositories do exist, such as GitLab, BitBucket, etc.

Using GitHub

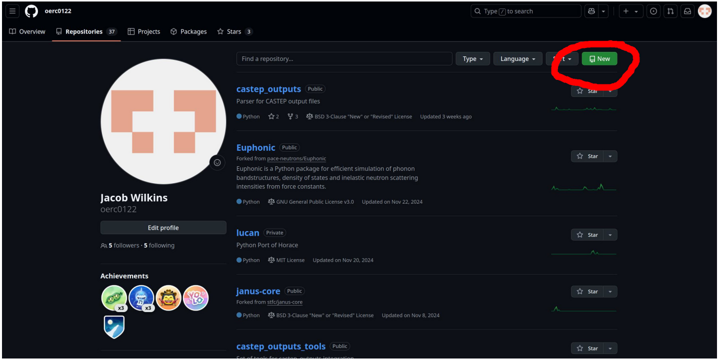
- This assumes you have some familiarity with git and GitHub.
- Also requires you to have a GitHub account.
- Everyone set up?

Adjust to taste

Other repositories do exist, such as GitLab, BitBucket, etc.

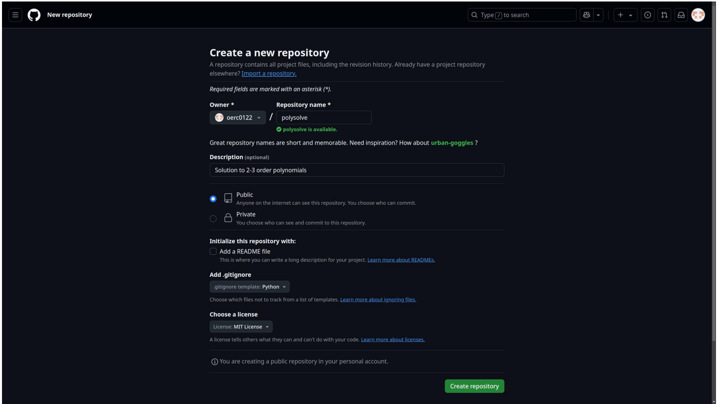
Using GitHub

- Create a new repository with the new (sensible) name!



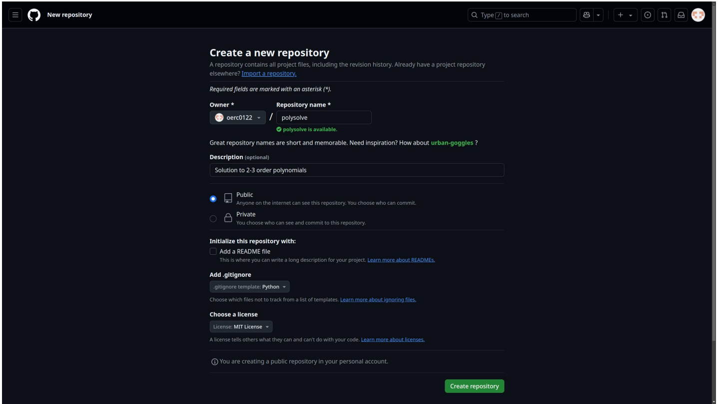
Using GitHub

- Create a new repository with the new (sensible) name!
- Add license, readme and set to ignore "python" extras.



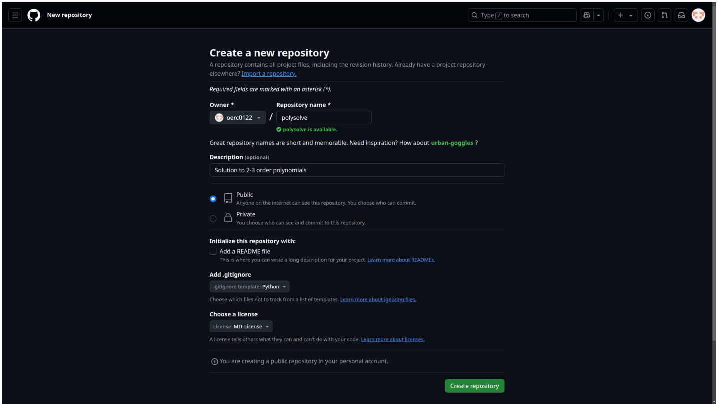
Using GitHub

- Create a new repository with the new (sensible) name!
- Add license, readme and set to ignore “python” extras.
    - The license choice is probably a topic for another talk.



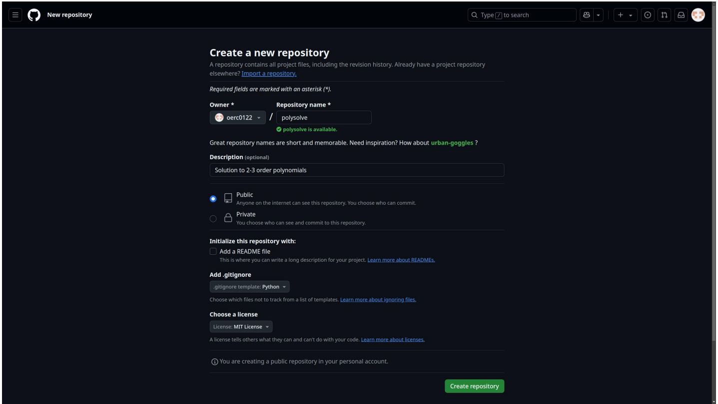
Using GitHub

- Create a new repository with the new (sensible) name!
- Add license, readme and set to ignore “python” extras.
  - The license choice is probably a topic for another talk.
  - The readme is displayed at the bottom of your GitHub page.



Using GitHub

- Create a new repository with the new (sensible) name!
- Add license, readme and set to ignore “python” extras.
  - The license choice is probably a topic for another talk.
  - The readme is displayed at the bottom of your GitHub page.
  - The ignore skips temporary files when using “`git add`”.



Using GitHub

- Create a new repository with the new (sensible) name!
- Add license, readme and set to ignore “python” extras.
  - The license choice is probably a topic for another talk.
  - The readme is displayed at the bottom of your GitHub page.
  - The ignore skips temporary files when using “`git add`”.
- `git clone` the new repo and move `polysolve.py` into it.

Using GitHub

- Create a new repository with the new (sensible) name!
- Add license, readme and set to ignore "python" extras.
  - The license choice is probably a topic for another talk.
  - The readme is displayed at the bottom of your GitHub page.
  - The ignore skips temporary files when using "`git add`".
- `git clone` the new repo and move `polysolve.py` into it.
- `git add polysolve.py`

Using GitHub

- Create a new repository with the new (sensible) name!
- Add license, readme and set to ignore "python" extras.
  - The license choice is probably a topic for another talk.
  - The readme is displayed at the bottom of your GitHub page.
  - The ignore skips temporary files when using "`git add`".
- `git clone` the new repo and move `polysolve.py` into it.
- `git add polysolve.py`
- `git commit -m 'Add initial code'`

Using GitHub

- Create a new repository with the new (sensible) name!
- Add license, readme and set to ignore "python" extras.
  - The license choice is probably a topic for another talk.
  - The readme is displayed at the bottom of your GitHub page.
  - The ignore skips temporary files when using "`git add`".
- `git clone` the new repo and move `polysolve.py` into it.
- `git add polysolve.py`
- `git commit -m 'Add initial code'`
- `git push --set-upstream origin`

Using GitHub

- Double check on GitHub and your files should be on it.

Package

From script to project

- To start a project, we need to define what the project is.

From script to project

- To start a project, we need to define what the project is.
- The first step is to change the structure to that of a project.

From script to project

- To start a project, we need to define what the project is.
- The first step is to change the structure to that of a project.
- Make a new folder "`polysolve`" and `git mv` "`polysolve.py`" into it.

From script to project

- To start a project, we need to define what the project is.
- The first step is to change the structure to that of a project.
- Make a new folder "`polysolve`" and `git mv` "`polysolve.py`" into it.

Layout

This form of putting code in `<project>/...` is called flat-layout.
You can also put code in `src/<project>/...` this is called source-layout.

Making a package

- Now we add a new file called `__init__.py` in `polysolve`.

Making a package

- Now we add a new file called `__init__.py` in `polysolve`.
- The `__init__.py` is a magic file.

Making a package

- Now we add a new file called `__init__.py` in `polysolve`.
- The `__init__.py` is a magic file.
- In Python it makes a folder accessible for `import`.

Making a package

- Now we add a new file called `__init__.py` in `polysolve`.
- The `__init__.py` is a magic file.
- In Python it makes a folder accessible for `import`.

Try it out!

```
#          foldername          filename
>>> from polysolve import polysolve
>>> polysolve.quadratic(1, 2, 3)
#        import.function
```

NOTE: This is only accessible from our project folder, not the system, it's not installed yet.

Making a package

- All files in the folder with the `__init__.py` are accessible.

Making a package

- All files in the folder with the `__init__.py` are accessible.
- Subfolders can be nested, each one needs an `__init__.py`.

Making a package

- All files in the folder with the `__init__.py` are accessible.
- Subfolders can be nested, each one needs an `__init__.py`.
- Code in an `__init__.py` is run **when the module is imported**.

Making a package

- All files in the folder with the `__init__.py` are accessible.
- Subfolders can be nested, each one needs an `__init__.py`.
- Code in an `__init__.py` is run **when the module is imported**.
- This can be used for setup or our package's metadata.

Making a package

- All files in the folder with the `__init__.py` are accessible.
- Subfolders can be nested, each one needs an `__init__.py`.
- Code in an `__init__.py` is run **when the module is imported**.
- This can be used for setup or our package's metadata.

Add some code to __init__.py

```python
"""Module to compute quadratic/cubic roots."""
__author__ = "Me"
__version__ = "0.1"
```

Making a project

- Now that we have a package it's time to make this a project.

[1] For more info on Python packaging take a look on the PyPA at:
https://packaging.python.org/en/latest/tutorials/packaging-projects/

Making a project

- Now that we have a package it's time to make this a project.
- We need a `pyproject.toml`.

TOML History

TOML (Tom's Own Markup Language) is a standardised format designed to replace the non-standardised `.ini` format configurations.

[1] For more info on Python packaging take a look on the PyPA at: `https://packaging.python.org/en/latest/tutorials/packaging-projects/`

Making a project

- Now that we have a package it's time to make this a project.
- We need a `pyproject.toml`.
- `pyproject.toml` defines the metadata our project[1].

TOML History

TOML (Tom's Own Markup Language) is a standardised format designed to replace the non-standardised `.ini` format configurations.

[1] For more info on Python packaging take a look on the PyPA at: `https://packaging.python.org/en/latest/tutorials/packaging-projects/`

Making a project

- Now that we have a package it's time to make this a project.
- We need a `pyproject.toml`.
- `pyproject.toml` defines the metadata our project[1].
- When you `pip install` this reads the `pyproject.toml`.

TOML History

TOML (Tom's Own Markup Language) is a standardised format designed to replace the non-standardised `.ini` format configurations.

[1] For more info on Python packaging take a look on the PyPA at: https://packaging.python.org/en/latest/tutorials/packaging-projects/

Making a project

- Now that we have a package it's time to make this a project.
- We need a `pyproject.toml`.
- `pyproject.toml` defines the metadata our project[1].
- When you `pip install` this reads the `pyproject.toml`.

TOML History

TOML (Tom's Own Markup Language) is a standardised format designed to replace the non-standardised `.ini` format configurations.

Ancient (modern) History

Older projects used to use something called `setup.py`, this is being deprecated except where your project needs e.g. Cython or compiled C++, and even then...

[1] For more info on Python packaging take a look on the PyPA at: https://packaging.python.org/en/latest/tutorials/packaging-projects/

The pyproject.toml

```
[build-system]
requires = ["setuptools >= 61.0.0"]
build-backend = "setuptools.build_meta"

[project]
name = "polysolve"
authors = [{name = "", email = ""}]
requires-python = ">= 3.8"
readme = "README.md"
description = ""
license = {text = "BSD-3-Clause"}

keywords = [...]
dependencies = [...]
classifiers = [...]
dynamic = ["version"]

[project.urls]
Homepage="https://github.com/XXX/polysolve"
Repository="https://github.com/XXX/polysolve.git"

[tool.setuptools.dynamic]
version = {attr = "polysolve.__version__"}
```

Let's look at these individually.

Note

Keywords are arranged into "block"s and are order independent within blocks. Blocks are order independent too.

build-system

```
[build-system]
requires = ["setuptools >= 61.0.0"]
build-backend = "setuptools.build_meta"
```

- These are the Python tools `pip` will use to build your project.

[2] https://packaging.python.org/en/latest/tutorials/packaging-projects/

build-system

```
[build-system]
requires = ["setuptools >= 61.0.0"]
build-backend = "setuptools.build_meta"
```

- These are the Python tools `pip` will use to build your project.
- You may choose something else (info on PyPA[2]), but we'll just stick with `setuptools`.

[2] https://packaging.python.org/en/latest/tutorials/packaging-projects/

project

```
[project]
name = "polysolve"
authors = [{name = "", email = ""}]
requires-python = ">= 3.8"
readme = "README.md"
description = ""
license = {text = "BSD-3-Clause"}

keywords = [...]
dependencies = [...]
classifiers = [...]
```

- These define the properties which describe your project:
- `name` – The project's installed name.

Note

* PyPI/repository only.

project

```
[project]
name = "polysolve"
authors = [{name = "", email = ""}]
requires-python = ">= 3.8"
readme = "README.md"
description = ""
license = {text = "BSD-3-Clause"}

keywords = [...]
dependencies = [...]
classifiers = [...]
```

- These define the properties which describe your project:
- `authors` – The project's authors.*

Note

* PyPI/repository only.

project

```
[project]
name = "polysolve"
authors = [{name = "", email = ""}]
requires-python = ">= 3.8"
readme = "README.md"
description = ""
license = {text = "BSD-3-Clause"}

keywords = [...]
dependencies = [...]
classifiers = [...]
```

- These define the properties which describe your project:
- `requires-python` – The minimum version of python needed to run the project.

Note

* PyPI/repository only.

project

```
[project]
name = "polysolve"
authors = [{name = "", email = ""}]
requires-python = ">= 3.8"
readme = "README.md"
description = ""
license = {text = "BSD-3-Clause"}

keywords = [...]
dependencies = [...]
classifiers = [...]
```

- These define the properties which describe your project:
- `readme` – The readme file/content.*

Note

* PyPI/repository only.

project

```
[project]
name = "polysolve"
authors = [{name = "", email = ""}]
requires-python = ">= 3.8"
readme = "README.md"
description = ""
license = {text = "BSD-3-Clause"}

keywords = [...]
dependencies = [...]
classifiers = [...]
```

- These define the properties which describe your project:
- `description` – A brief summary of the project.*

Note

* PyPI/repository only.

project

```
[project]
name = "polysolve"
authors = [{name = "", email = ""}]
requires-python = ">= 3.8"
readme = "README.md"
description = ""
license = {text = "BSD-3-Clause"}

keywords = [...]
dependencies = [...]
classifiers = [...]
```

- These define the properties which describe your project:
- `license` – Project license.*

Note

* PyPI/repository only.

project

```toml
[project]
name = "polysolve"
authors = [{name = "", email = ""}]
requires-python = ">= 3.8"
readme = "README.md"
description = ""
license = {text = "BSD-3-Clause"}

keywords = [...]
dependencies = [...]
classifiers = [...]
```

- These define the properties which describe your project:
- `keywords` – Searchable keywords describing project.*

Note

* PyPI/repository only.

project

```
[project]
name = "polysolve"
authors = [{name = "", email = ""}]
requires-python = ">= 3.8"
readme = "README.md"
description = ""
license = {text = "BSD-3-Clause"}

keywords = [...]
dependencies = [...]
classifiers = [...]
```

- These define the properties which describe your project:
- `classifiers` – Set of keyword identifiers (see PyPA).*

Note

* PyPI/repository only.

project

```
[project]
name = "polysolve"
authors = [{name = "", email = ""}]
requires-python = ">= 3.8"
readme = "README.md"
description = ""
license = {text = "BSD-3-Clause"}

keywords = [...]
dependencies = [...]
classifiers = [...]
```

- These define the properties which describe your project:
- `dependencies` – List of project dependencies.

Note

* PyPI/repository only.

Webpages

```
[project.urls]
Homepage="https://github.com/XXX/polysolve"
Repository="https://github.com/XXX/polysolve.
    git"
```

- PyPI will add these links in a sidebar if you upload your project.

Dynamic

```
[project]
dynamic = ["version"]

[tool.setuptools.dynamic]
version = {attr = "polysolve.__version__"}
```

- You may have spotted `dynamic` at the end of the `[project]` block.

Dynamic

```
[project]
dynamic = ["version"]

[tool.setuptools.dynamic]
version = {attr = "polysolve.__version__"}
```

- You may have spotted `dynamic` at the end of the `[project]` block.
- `dynamic` is a special keyword which tells `pip` the variable will come from somewhere else.

Dynamic

```
[project]
dynamic = ["version"]

[tool.setuptools.dynamic]
version = {attr = "polysolve.__version__"}
```

- You may have spotted `dynamic` at the end of the `[project]` block.
- `dynamic` is a special keyword which tells `pip` the variable will come from somewhere else.
- We define our `version` as coming from our package.

Dynamic

```
[project]
dynamic = ["version"]

[tool.setuptools.dynamic]
version = {attr = "polysolve.__version__"}
```

- You may have spotted `dynamic` at the end of the `[project]` block.
- `dynamic` is a special keyword which tells `pip` the variable will come from somewhere else.
- We define our `version` as coming from our package.

Extra dynamicism

We can define several other properties as `dynamic` see PyPA for more info.

Connect the dots

- Copy `pyproject.toml` to your project root directory.

Connect the dots

- Copy `pyproject.toml` to your project root directory.
- We can fill in the gaps in our `pyproject.toml`

Try it out!

```
pip install .
python
>>> from polysolve import polysolve
>>> polysolve.quadratic(3, 1, 2)
```

Connect the dots

- Copy `pyproject.toml` to your project root directory.
- We can fill in the gaps in our `pyproject.toml`
- **Note:** you need to have `dependencies = ["numpy"]`

Try it out!

```
pip install .
python
>>> from polysolve import polysolve
>>> polysolve.quadratic(3, 1, 2)
```

Connect the dots

- Copy `pyproject.toml` to your project root directory.
- We can fill in the gaps in our `pyproject.toml`
- **Note:** you need to have `dependencies = ["numpy"]`
- Then we can see some magic happen.

Try it out!

```
pip install .
python
>>> from polysolve import polysolve
>>> polysolve.quadratic(3, 1, 2)
```

Connect the dots

- Copy `pyproject.toml` to your project root directory.
- We can fill in the gaps in our `pyproject.toml`
- **Note:** you need to have `dependencies = ["numpy"]`
- Then we can see some magic happen.
- `pip` checks we have all the requirements, installs the dependencies, then our project.

Developing

While developing you will want:

```
pip install -e .
```

which will link to the package so as you edit it the system version updates.

Connect the dots

- Copy `pyproject.toml` to your project root directory.
- We can fill in the gaps in our `pyproject.toml`
- **Note:** you need to have `dependencies = ["numpy"]`
- Then we can see some magic happen.
- `pip` checks we have all the requirements, installs the dependencies, then our project.
- **NOTE:** It's now installed system-wide.

Get it gitted

- Now add the `pyproject.toml` to `git`.

Get it gitted

- Now add the `pyproject.toml` to `git`.
- Push it up to GitHub.

Get it gitted

- Now add the `pyproject.toml` to `git`.
- Push it up to GitHub.

More magic!

```
pip install git+https://github.com/<owner_name>/polysolve.git
```

NOTE: PyPI is “easier”, but requires accounts. This is convenient for small stuff.

Great, we have a project

- Now what?

Great, we have a project

- Now what?
- The next step is to make it usable.

Great, we have a project

- Now what?
- The next step is to make it usable.
- That means usable by other people.

Documentation

Documentation, documentation, documentation.

- We're going to begin by looking at documentation.

Documentation, documentation, documentation.

- We're going to begin by looking at documentation.
- Documentation tends to fall by the wayside.

Documentation, documentation, documentation.

- We're going to begin by looking at documentation.
- Documentation tends to fall by the wayside.
- **However**, it's the most important thing in released software.

Documentation, documentation, documentation.

- Let's start with something simple.

Documentation, documentation, documentation.

- Let's start with something simple.
- Our `README.md` basically says the project name.

Documentation, documentation, documentation.

- Let's start with something simple.
- Our `README.md` basically says the project name.
- We know how to install it now, so let's add that.

More magic!

```
pip install git+https://github.com/<owner_name>/polysolve.git
```

Documentation, documentation, documentation.

- Let's start with something simple.
- Our `README.md` basically says the project name.
- We know how to install it now, so let's add that.
- Push it up to GitHub and see the glory of your hard work.

More magic!

```
pip install git+https://github.com/<owner_name>/polysolve.git
```

IDE Ahoy

- Anybody here used VSCode or another IDE[3]?

[3] Interactive Development Environment

IDE Ahoy

- Anybody here used VSCode or another IDE[3]?
- When you start typing a function, it tells you what argument comes next.

[3] Interactive Development Environment

IDE Ahoy

- Anybody here used VSCode or another IDE[3]?
- When you start typing a function, it tells you what argument comes next.
- It also tells you the `type` it should be (`int`, `float`, etc.).

[3] Interactive Development Environment

IDE Ahoy

- The IDE isn't doing any **magic** to find out, we tell it!

IDE Ahoy

- The IDE isn't doing any **magic** to find out, we tell it!
- How do we tell it?

IDE Ahoy

- The IDE isn't doing any **magic** to find out, we tell it!
- How do we tell it?
- We use "type-hints" or "annotations".

```
def quadratic(
        a: float,
        b: float,
        c: float,
) -> tuple[float, float]:
    det = b**2 - (4*a*c)

    return ((-b + np.sqrt(det)) / (2*a),
            (-b - np.sqrt(det)) / (2*a))
```

IDE Ahoy

- The IDE isn't doing any **magic** to find out, we tell it!
- How do we tell it?
- We use "type-hints" or "annotations".

```python
from __future__ import annotations

def quadratic(
        a: float,
        b: float,
        c: float,
) -> tuple[float, float]:
    det = b**2 - (4*a*c)

    return ((-b + np.sqrt(det)) / (2*a),
            (-b - np.sqrt(det)) / (2*a))
```

Note

You may find on older Python versions for complex annotations you need to `import annotations`

Handy dandy

- These type-hints aren't just useful to users.

```
def quadratic(
        a: float,
        b: float,
        c: float,
) -> tuple[float, float]:
    det = b**2 - (4*a*c)

    return ((-b + np.sqrt(det)) / (2*a),
            (-b - np.sqrt(det)) / (2*a))
```

Handy dandy

- These type-hints aren't just useful to users.
- They're useful to us as developers.

```
def quadratic(
        a: float,
        b: float,
        c: float,
) -> tuple[float, float]:
    det = b**2 - (4*a*c)

    return ((-b + np.sqrt(det)) / (2*a),
            (-b - np.sqrt(det)) / (2*a))
```

Handy dandy

- These type-hints aren't just useful to users.
- They're useful to us as developers.
- We know when changing things what we're allowed to do.

```
def quadratic(
        a: float,
        b: float,
        c: float,
) -> tuple[float, float]:
    det = b**2 - (4*a*c)

    return ((-b + np.sqrt(det)) / (2*a),
             (-b - np.sqrt(det)) / (2*a))
```

What are we doing again?

- So we know what we're feeding the black box.

What are we doing again?

- So we know what we're feeding the black box.
- Wouldn't it be nice if the box told us what it did (or is trying to do)?

What are we doing again?

- So we know what we're feeding the black box.
- Wouldn't it be nice if the box told us what it did (or is trying to do)?
- Don't go rushing off to write in the filetREADME.md again!

What are we doing again?

- Python allows us to annotate further!

[4] See quadexm.py

What are we doing again?

- Python allows us to annotate further!
- Introducing the docstring!

```
def quadratic(a: float, b: float, c: float) -> tuple[float
    , float]:
     """
     Solves the roots of a quadratic equation.
    """
```

[4] See quadexm.py

What are we doing again?

- Python allows us to annotate further!
- Introducing the docstring!
- This is the minimal docstring.

```
def quadratic(a: float, b: float, c: float) -> tuple[float
    , float]:
    """
    Solves the roots of a quadratic equation.
    """
```

[4] See quadexm.py

What are we doing again?

- Python allows us to annotate further!
- Introducing the docstring!
- This is the minimal docstring.
- We can add more![4]

```python
def quadratic(a: float, b: float, c: float) -> tuple[float
    , float]:
    """
    Solves the roots of a quadratic equation.
    """
```

[4] See quadexm.py

What are we doing again?

- We can add more![5]

```
def quadratic(a: float, b: float, c: float) -> tuple[float
    , float]:
     """
     Solves the roots of a quadratic equation.
    """
```

[5] See quadexm.py

What are we doing again?

- We can add more![5]

```
def quadratic(a: float, b: float, c: float) -> tuple[float
    , float]:
    """
    Solves the roots of a quadratic equation.

    Uses the quadratic formula. Result must be real.
    """
```

[5] See quadexm.py

What are we doing again?

- We can add more![5]

```
def quadratic(a: float, b: float, c: float) -> tuple[float
    , float]:
    """
    Solves the roots of a quadratic equation.

    Uses the quadratic formula. Result must be real.

    Parameters
    ----------
    a
        :math:`x^2` coefficient.
    b
        :math:`x` coefficient.
    c
        Constant value.
    """
```

[5] See quadexm.py

What are we doing again?

- We can add more![5]

```python
def quadratic(a: float, b: float, c: float) -> tuple[float
    , float]:
    """
    Solves the roots of a quadratic equation.

    Uses the quadratic formula. Result must be real.

    Parameters
    ----------
    a
        :math:`x^2` coefficient.
    b
        :math:`x` coefficient.
    c
        Constant value.

    Returns
    -------
    tuple[float, float]
        Positive and negative roots of quadratic.
    """
```

[5] See quadexm.py

What are we doing again?

- We can add more![5]

```
Examples
--------
>>> quadratic(1., 0., 0.)
(0.0, -0.0)
>>> quadratic(3., 0., -1.)
(0.5773502691896257, -0.5773502691896257)
```

[5] See quadexm.py

What are we doing again?

- We can add more![5]

```
Raises
------
ValueError
    Discriminant < 0 implying imaginary root.
```

[5] See quadexm.py

What are we doing again?

- We can add more![5]

```
Notes
-----
Equation of the form:

.. math::

    ax^{2} + bx + c
```

[5] See quadexm.py

What are we doing again?

- We can add more![5]

```
See Also
--------
numpy.polyval : Evaluate polynomial at point.
```

[5] See quadexm.py

What are we doing again?

- We can add more![5]

```
References
----------
.. [1] O. McNoleg, "The integration of GIS, remote
   sensing,
      expert systems ...
```

[5] See quadexm.py

Substance over style

- **Note:** what I've been showing you is **one** style of docs.

Substance over style

- **Note:** what I've been showing you is **one** style of docs.
- This style is called `numpydoc` style after the `numpy` library.

Substance over style

The main styles are: numpydoc

numpydoc.readthedocs.io/en/latest/format.html

```python
def quadratic(a: float, b: float, c: float) -> tuple[float, float]:
    """
    Solves the roots of a quadratic equation.

    Uses the quadratic formula. Result must be real.

    Parameters
    ----------
    a
        :math:`x^2` coefficient.
    b
        :math:`x` coefficient.
    c
        Constant value.

    Returns
    -------
    tuple[float, float]
        Positive and negative roots of quadratic.
    """
```

Substance over style

The main styles are: google

google.github.io/styleguide/pyguide.html

```python
def quadratic(a: float, b: float, c: float) -> tuple[float
    , float]:
    """Solves the roots of a quadratic equation.

    Uses the quadratic formula. Result must be real.

    Parameters:
        a: :math:`x^2` coefficient.
        b: :math:`x` coefficient.
        c: Constant value.

    Returns:
        Positive and negative roots of quadratic.
    """
```

Substance over style

The main styles are: sphinx

sphinx-rtd-tutorial.readthedocs.io/en/latest/docstrings.html

```python
def quadratic(a: float, b: float, c: float) -> tuple[float
    , float]:
    """Solves the roots of a quadratic equation.

     Uses the quadratic formula. Result must be real.

    :param a: :math:`x^2` coefficient.
    :param b: :math:`x` coefficient.
    :param c: Constant value.

    :return: Positive and negative roots of quadratic.
    """
```

More magic

- Ok, we've got docstrings. Now time for a callback:

[5] See `quadexm.py`

More magic

- Ok, we've got docstrings. Now time for a callback:
- Remember this?[5]

```
Examples
--------
>>> quadratic(1., 0., 0.)
(0.0, -0.0)
>>> quadratic(3., 0., -1.)
(0.5773502691896257, -0.5773502691896257)
```

[5] See quadexm.py

More magic

- Ok, we've got docstrings. Now time for a callback:
- Remember this?[5]
- What happens if this goes out of date or doesn't work?

```
Examples
--------
>>> quadratic(1., 0., 0.)
(0.0, -0.0)
>>> quadratic(3., 0., -1.)
(0.5773502691896257, -0.5773502691896257)
```

[5] See quadexm.py

More magic

- Thankfully, Python provides a way to use these as tests!

```
Examples
--------
>>> quadratic(1., 0., 0.)
(0.0, -0.0)
>>> quadratic(3., 0., -1.)
(0.5773502691896257, -0.5773502691896257)
```

More magic

- Thankfully, Python provides a way to use these as tests!
- (Already into tests and we're not out of the docs section yet! Sneak peek!)

```
Examples
--------
>>> quadratic(1., 0., 0.)
(0.0, -0.0)
>>> quadratic(3., 0., -1.)
(0.5773502691896257, -0.5773502691896257)
```

More magic

- Thankfully, Python provides a way to use these as tests!
- (Already into tests and we're not out of the docs section yet! Sneak peek!)
- https://docs.python.org/3/library/doctest.html

```
Examples
--------
>>> quadratic(1., 0., 0.)
(0.0, -0.0)
>>> quadratic(3., 0., -1.)
(0.5773502691896257, -0.5773502691896257)
```

More magic

- Thankfully, Python provides a way to use these as tests!
- (Already into tests and we're not out of the docs section yet! Sneak peek!)
- `https://docs.python.org/3/library/doctest.html`

```
Examples
--------
>>> quadratic(1., 0., 0.)
(0.0, -0.0)
>>> quadratic(3., 0., -1.)
(0.5773502691896257, -0.5773502691896257)
```

Try it out!

```
python -m doctest polysolve.py
```

More magic

- Thankfully, Python provides a way to use these as tests!
- (Already into tests and we're not out of the docs section yet! Sneak peek!)
- https://docs.python.org/3/library/doctest.html

```
Examples
--------
>>> quadratic(1., 0., 0.)
(0.0, -0.0)
>>> quadratic(3., 0., -1.)
(0.5773502691896257, -0.5773502691896257)
```

Try it out!

```
if __name__ == "__main__":
    import doctest
    doctest.testmod()

python polysolve.py
```

More magic

- Doctests are designed to imitate Python REPL.

Example

```
>>> my_var = ["hello", "goodbye"]
>>> my_var
['hello', 'goodbye']
>>> for i in range(3):
...     print(i)
0
1
2
```

More magic

- Doctests are designed to imitate Python REPL.
- Designed for copying and pasting from REPL.

Example

```
>>> my_var = ["hello", "goodbye"]
>>> my_var
['hello', 'goodbye']
>>> for i in range(3):
...     print(i)
0
1
2
```

More magic

- Doctests are designed to imitate Python REPL.
- Designed for copying and pasting from REPL.
- Lines starting with “>>>” are run.

Example

```
>>> my_var = ["hello", "goodbye"]
>>> my_var
['hello', 'goodbye']
>>> for i in range(3):
...     print(i)
0
1
2
```

More magic

- Doctests are designed to imitate Python REPL.
- Designed for copying and pasting from REPL.
- Lines starting with ">>>" are run.
- Lines can be continued/indented with "...".

Example

```
>>> my_var = ["hello", "goodbye"]
>>> my_var
['hello', 'goodbye']
>>> for i in range(3):
...     print(i)
0
1
2
```

More magic

- Doctests are designed to imitate Python REPL.
- Designed for copying and pasting from REPL.
- Lines starting with “>>>” are run.
- Lines can be continued/indented with “...”.
- Lines with neither are checked against the result.

Example

```
>>> my_var = ["hello", "goodbye"]
>>> my_var
['hello', 'goodbye']
>>> for i in range(3):
...     print(i)
0
1
2
```

More magic

- Doctests are designed to imitate Python REPL.
- Designed for copying and pasting from REPL.
- Lines starting with ">>>" are run.
- Lines can be continued/indented with "...".
- Lines with neither are checked against the result.
- Need to import libraries if they're needed.

Example

```
>>> import numpy as np
>>> np.array([1, 2, 3])
array([1, 2, 3])
```

More magic

- Doctests are designed to imitate Python REPL.
- Designed for copying and pasting from REPL.
- Lines starting with “>>>” are run.
- Lines can be continued/indented with “...”.
- Lines with neither are checked against the result.
- Need to import libraries if they're needed.

Fun (useful) (magic?) fact

Doctest doesn't care about what strings its reading and will read and run any >>> style stuff even in documentation or text files!

```
python -m doctest my_text.txt
```

Finally getting to docs

- Now after so long, it's time to finally write some docs!

Finally getting to docs

- Now after so long, it's time to finally write some docs!
- (or let the computer write some for us...)

Finally getting to docs

- Now after so long, it's time to finally write some docs!
- (or let the computer write some for us...)

Getting started! (Linux)

```
pip install sphinx sphinx_rtd_theme
mkdir docs; cd docs
sphinx-quickstart
make html
chromium build/html/index.html
```

Keys to the docs

- Key files in the new docs are:

Keys to the docs

- Key files in the new docs are:
  - `conf.py` – Configuration for docs.

Keys to the docs

- Key files in the new docs are:
  - `conf.py` – Configuration for docs.
  - `index.rst` – Main starting file for docs.

Keys to the docs

- Key files in the new docs are:
  - `conf.py` – Configuration for docs.
  - `index.rst` – Main starting file for docs.
- Let's take a look at these.

conf.py

- `conf.py` is an auto-generated Python file with instructions for building the docs.

```
# Configuration file for the Sphinx documentation builder.
#
# For the full list of built-in configuration values, see the documentation:
# https://www.sphinx-doc.org/en/master/usage/configuration.html
```

conf.py

- `conf.py` is an auto-generated Python file with instructions for building the docs.
- It is a full Python file you can run code in, e.g. we can pull out information from our package.

```
# Configuration file for the Sphinx documentation builder.
#
# For the full list of built-in configuration values, see the documentation:
# https://www.sphinx-doc.org/en/master/usage/configuration.html
```

conf.py

- `conf.py` is an auto-generated Python file with instructions for building the docs.
- It is a full Python file you can run code in, e.g. we can pull out information from our package.
- For example, we can use our defined metadata.

```
# -- Project information -----------------------------------------------------
# https://www.sphinx-doc.org/en/master/usage/configuration.html#project-
    information
import polysolve
from datetime import date

project = 'polysolve'
author = polysolve.__author__
copyright = f'{author}, {date.today().year}'
release = polysolve.__version__
```

conf.py

- `conf.py` is an auto-generated Python file with instructions for building the docs.
- It is a full Python file you can run code in, e.g. we can pull out information from our package.
- For example, we can use our defined metadata.
- `sphinx` is a fully extensible package. We'll be using some of these later.

```
# -- General configuration ---------------------------------------------------
# https://www.sphinx-doc.org/en/master/usage/configuration.html#general-
    configuration

extensions = []

templates_path = ['_templates']
exclude_patterns = []
```

conf.py

- `conf.py` is an auto-generated Python file with instructions for building the docs.
- It is a full Python file you can run code in, e.g. we can pull out information from our package.
- For example, we can use our defined metadata.
- `sphinx` is a fully extensible package. We'll be using some of these later.
- `exclude_patterns` allows us to exclude source files from our sphinx build.

```
# -- General configuration ---------------------------------------------------
# https://www.sphinx-doc.org/en/master/usage/configuration.html#general-configuration

extensions = []

templates_path = ['_templates']
exclude_patterns = []
```

conf.py

- `conf.py` is an auto-generated Python file with instructions for building the docs.
- It is a full Python file you can run code in, e.g. we can pull out information from our package.
- For example, we can use our defined metadata.
- `sphinx` is a fully extensible package. We'll be using some of these later.
- `exclude_patterns` allows us to exclude source files from our sphinx build.
- We can change the docs theme to render them differently.

```
# -- Options for HTML output -------------------------------------------------
# https://www.sphinx-doc.org/en/master/usage/configuration.html#options-for-
    html-output

html_theme = 'sphinx_rtd_theme'
html_static_path = ['_static']
```

conf.py

- `conf.py` is an auto-generated Python file with instructions for building the docs.
- It is a full Python file you can run code in, e.g. we can pull out information from our package.
- For example, we can use our defined metadata.
- `sphinx` is a fully extensible package. We'll be using some of these later.
- `exclude_patterns` allows us to exclude source files from our sphinx build.
- We can change the docs theme to render them differently.
- Since we installed `sphinx_rtd_theme` we can try that.

```
# -- Options for HTML output -------------------------------------------------
# https://www.sphinx-doc.org/en/master/usage/configuration.html#options-for-html-output

html_theme = 'sphinx_rtd_theme'
html_static_path = ['_static']
```

- sphinx docs are written in REStructured Text (ReST/rst)[5].

```
.. polysolve documentation master file, created by
   sphinx-quickstart on Mon Oct 14 21:27:04 2024.
   You can adapt this file completely to your liking, but it should at least
   contain the root `toctree` directive.

polysolve documentation
=======================

Add your content using ``reStructuredText`` syntax. See the
`reStructuredText <https://www.sphinx-doc.org/en/master/usage/restructuredtext
     /index.html>`_
documentation for details.


.. toctree::
   :maxdepth: 2
   :caption: Contents:
```

[5] www.sphinx-doc.org/en/master/usage/restructuredtext/index.html

index.rst

- sphinx docs are written in REStructured Text (ReST/rst)[5].
- Text "marked-up" with formatting (like LaTeX or HTML).

```
.. polysolve documentation master file, created by
   sphinx-quickstart on Mon Oct 14 21:27:04 2024.
   You can adapt this file completely to your liking, but it should at least
   contain the root `toctree` directive.

polysolve documentation
=======================

Add your content using ``reStructuredText`` syntax. See the
`reStructuredText <https://www.sphinx-doc.org/en/master/usage/restructuredtext
     /index.html>`_
documentation for details.


.. toctree::
   :maxdepth: 2
   :caption: Contents:
```

[5] www.sphinx-doc.org/en/master/usage/restructuredtext/index.html

Writing our docs

- Now we need some files to actually to actually fill with docs!

Writing our docs

- Now we need some files to actually to actually fill with docs!
- Create a file called `usage.rst`.

Writing our docs

- Now we need some files to actually to actually fill with docs!
- Create a file called `usage.rst`.
- Write some documentation.

```
Usage
=====

Here's how to use polysolve!
```

Writing our docs

- Now we need some files to actually to actually fill with docs!
- Create a file called `usage.rst`.
- Write some documentation.
- Add it to our "table of contents tree" (`toctree`).

```
.. toctree::
   :maxdepth: 2
   :caption: Contents:
```

Writing our docs

- Now we need some files to actually to actually fill with docs!
- Create a file called `usage.rst`.
- Write some documentation.
- Add it to our “table of contents tree” (`toctree`).

```
.. toctree::
   :maxdepth: 2
   :caption: Contents:

   usage
```

Writing our docs

- Now we need some files to actually to actually fill with docs!
- Create a file called `usage.rst`.
- Write some documentation.
- Add it to our "table of contents tree" (`toctree`).
- Build our docs!

Make it so!

```
make html
```

Docstring Magic

- So now we can write about every single function in our project.

Docstring Magic

- So now we can write about every single function in our project.
  - How many could there be?

Docstring Magic

- So now we can write about every single function in our project.
    - How many could there be?
    - What do you mean not every project has 20 lines?

Docstring Magic

- So now we can write about every single function in our project.
  - How many could there be?
  - What do you mean not every project has 20 lines?
- Remember our docstrings?

Docstring Magic

- So now we can write about every single function in our project.
  - How many could there be?
  - What do you mean not every project has 20 lines?
- Remember our docstrings?
- Maybe there's a way to avoid writing everything twice.

Docstring magic

- What if we could extract all the docstrings we've already written?

Docstring magic

- What if we could extract all the docstrings we've already written?
- We're going to need to do a couple of things.

Docstring magic

- What if we could extract all the docstrings we've already written?
- We're going to need to do a couple of things.
- Time to use some extensions.

```
extensions = [    ]
```

Docstring magic

- What if we could extract all the docstrings we've already written?
- We're going to need to do a couple of things.
- Time to use some extensions.
- `sphinx.ext.autodoc` extracts docstrings from functions.

```
extensions = [
"sphinx.ext.autodoc",
]
```

Docstring magic

- What if we could extract all the docstrings we've already written?
- We're going to need to do a couple of things.
- Time to use some extensions.
- `sphinx.ext.autodoc` extracts docstrings from functions.
- `sphinx.ext.napoleon` converts our `numpydoc` to `sphinx`

```
extensions = [
"sphinx.ext.autodoc",
"sphinx.ext.napoleon",
]
```

Docstring magic

- What if we could extract all the docstrings we've already written?
- We're going to need to do a couple of things.
- Time to use some extensions.
- `sphinx.ext.autodoc` extracts docstrings from functions.
- `sphinx.ext.napoleon` converts our `numpydoc` to `sphinx`
- `sphinx.ext.autosummary` adds a summary to each page.

```
extensions = [
"sphinx.ext.autodoc",
"sphinx.ext.napoleon",
"sphinx.ext.autosummary",
]
```

Docstring magic

- We need to create all the infrastructure to extract our info.

Docstring magic

- We need to create all the infrastructure to extract our info.
- Just kidding, there's a tool for that!

It's magic!

```
sphinx-apidoc -o docs/source/api polysolve
```

Docstring magic

- We need to create all the infrastructure to extract our info.
- Just kidding, there's a tool for that!
- Just add it to the `toctree` and we're set.

```
.. toctree::
   :maxdepth: 2
   :caption: Contents:

   usage
   api/modules
```

Docstring magic

- We need to create all the infrastructure to extract our info.
- Just kidding, there's a tool for that!
- Just add it to the `toctree` and we're set.
- after we `make html`

Type-hints

- But our typehints aren't with our params...

Type-hints

- But our typehints aren't with our params...
- If only there were some tool to extract those too.

Type-hints

- But our typehints aren't with our params...
- If only there were some tool to extract those too.
- Alas, it's surely impossible...

Type-hints

- But our typehints aren't with our params...
- If only there were some tool to extract those too.
- Alas, it's surely impossible...

Da-da-da-daaaa

```
pip install sphinx-autodoc-typehints
```

Type-hints

- But our typehints aren't with our params...
- If only there were some tool to extract those too.
- Alas, it's surely impossible...

Da-da-da-daaaa

```
extensions = [...,
              "sphinx_autodoc_typehints",
              ]
```

But what is a float really?

- But now I'm unhappy because when I click `float` it doesn't take me to the documentation of `float`.

But what is a float really?

- But now I'm unhappy because when I click `float` it doesn't take me to the documentation of `float`.
- Some people, honestly.

But what is a float really?

- But now I'm unhappy because when I click `float` it doesn't take me to the documentation of `float`.
- Some people, honestly.
- Introducing "`intersphinx`".

Ta-da

```
extensions = [...,
              "sphinx.ext.intersphinx",
              ]
              ...
intersphinx_mapping = {'python': ('https://docs.python.org/3/', None),
                       'numpy': ('https://docs.scipy.org/doc/numpy/', None),
                       ...}
```

But what is a float really?

- But now I'm unhappy because when I click `float` it doesn't take me to the documentation of `float`.
- Some people, honestly.
- Introducing "`intersphinx`".
- Links your documentation against other `sphinx` documentation sites automatically.

Ta-da

```
extensions = [...,
              "sphinx.ext.intersphinx",
              ]
              ...
intersphinx_mapping = {'python': ('https://docs.python.org/3/', None),
                       'numpy': ('https://docs.scipy.org/doc/numpy/', None),
                       ...}
```

Adding it to the project

- Now that we know what we need, we can add these to our `pyproject.toml`.

Adding it to the project

- Now that we know what we need, we can add these to our `pyproject.toml`.
- Not everyone needs to install them, so let's add them as an optional dependency.

Adding it to the project

- Now that we know what we need, we can add these to our `pyproject.toml`.
- Not everyone needs to install them, so let's add them as an optional dependency.

Adding it in

```
[project.optional-dependencies]
docs = ["sphinx",
        "sphinx_rtd_theme",
        "sphinx_autodoc_typehints"]
```

Adding it to the project

- Now that we know what we need, we can add these to our `pyproject.toml`.
- Not everyone needs to install them, so let's add them as an optional dependency.

Adding it in

To install:

```
pip install -e ".[docs]"
```

Done with docs

- More extensions and tools are available for building docs.

Done with docs

- More extensions and tools are available for building docs.
- In particular things like:

Done with docs

- More extensions and tools are available for building docs.
- In particular things like:
  - Integrated Jupyter tutorials (`nbsphinx`).

Done with docs

- More extensions and tools are available for building docs.
- In particular things like:
  - Integrated Jupyter tutorials (`nbsphinx`).
  - Testing within documentation (`sphinx.ext.doctest`).

Done with docs

- More extensions and tools are available for building docs.
- In particular things like:
  - Integrated Jupyter tutorials (`nbsphinx`).
  - Testing within documentation (`sphinx.ext.doctest`).
  - and many more...

Tests

Done with docs

- Now we have some documentation to back up our code.

Done with docs

- Now we have some documentation to back up our code.
- Now we're ready to check it works.

Done with docs

- Now we have some documentation to back up our code.
- Now we're ready to check it works.
- We already have `doctest`s, which are good, but incomplete.

Types of tests

- We generally break testing down into 3–4 main types:

Types of tests

- We generally break testing down into 3–4 main types:
  - Unit tests – Tests of each function.

Types of tests

- We generally break testing down into 3–4 main types:
  - Unit tests – Tests of each function.
  - System tests / End-to-end tests – Small tests of the whole programs.

Types of tests

- We generally break testing down into 3–4 main types:
  - Unit tests – Tests of each function.
  - System tests / End-to-end tests – Small tests of the whole programs.
  - Benchmark tests – Tests real world cases.

Types of tests

- We generally break testing down into 3–4 main types:
  - Unit tests – Tests of each function.
  - System tests / End-to-end tests – Small tests of the whole programs.
  - Benchmark tests – Tests real world cases.
  - Integration tests – Tests interfaces between program components.

Types of tests

- We generally break testing down into 3–4 main types:
  - Unit tests – Tests of each function.
  - System tests / End-to-end tests – Small tests of the whole programs.
  - Benchmark tests – Tests real world cases.
  - Integration tests – Tests interfaces between program components.
- We split these into three types:

Types of tests

- We generally break testing down into 3–4 main types:
  - Unit tests – Tests of each function.
  - System tests / End-to-end tests – Small tests of the whole programs.
  - Benchmark tests – Tests real world cases.
  - Integration tests – Tests interfaces between program components.
- We split these into three types:
  - Science tests – Check the validity against a known result.

Types of tests

- We generally break testing down into 3–4 main types:
  - Unit tests – Tests of each function.
  - System tests / End-to-end tests – Small tests of the whole programs.
  - Benchmark tests – Tests real world cases.
  - Integration tests – Tests interfaces between program components.
- We split these into three types:
  - Science tests – Check the validity against a known result.
  - Regression tests – Check values haven't changed.

Types of tests

- We generally break testing down into 3–4 main types:
  - Unit tests – Tests of each function.
  - System tests / End-to-end tests – Small tests of the whole programs.
  - Benchmark tests – Tests real world cases.
  - Integration tests – Tests interfaces between program components.
- We split these into three types:
  - Science tests – Check the validity against a known result.
  - Regression tests – Check values haven't changed.
  - Fail-state tests – Intentionally check failure states.

Testing

- Our `doctests` go some of the way towards unit-tests.

Testing

- Our `doctests` go some of the way towards unit-tests.
- But they aren't the be all and end all.

Testing

- Our `doctests` go some of the way towards unit-tests.
- But they aren't the be all and end all.
- Let's see how to do proper tests.

- First let's install the `pytest`[6] library.

Let's get started!

```
pip install pytest
```

[6] **NOTE:** Python ships with the `unittest` library, but rather than teaching two methods and confusing things, I'm sticking with one.

Testing

- First let's install the `pytest`[6] library.
- Let's create a `tests` folder.

[6] **NOTE:** Python ships with the `unittest` library, but rather than teaching two methods and confusing things, I'm sticking with one.

Testing

- First let's install the pytest[6] library.
- Let's create a tests folder.
- In that folder, let's create a test_quadratic.py.

```
import pytest
import numpy as np
from polysolve.polysolve import quadratic

def test_quadratic():
    """Tests that quadratic finds the root for a known problem."""
    params = [3., 0., -1.]
    roots = quadratic(*params)
    assert all(np.isclose(np.polyval(params, root), 0.) for root in roots)
```

[6] **NOTE:** Python ships with the unittest library, but rather than teaching two methods and confusing things, I'm sticking with one.

Testing

- First let's install the pytest[6] library.
- Let's create a tests folder.
- In that folder, let's create a test_quadratic.py.
- What type of test is this?

```
import pytest
import numpy as np
from polysolve.polysolve import quadratic

def test_quadratic():
    """Tests that quadratic finds the root for a known problem."""
    params = [3., 0., -1.]
    roots = quadratic(*params)
    assert all(np.isclose(np.polyval(params, root), 0.) for root in roots)
```

[6] **NOTE:** Python ships with the unittest library, but rather than teaching two methods and confusing things, I'm sticking with one.

Testing

- First let's install the `pytest`[6] library.
- Let's create a `tests` folder.
- In that folder, let's create a `test_quadratic.py`.
- What type of test is this?
- Run it!

Try it out!

```
pytest
```

[6] **NOTE:** Python ships with the `unittest` library, but rather than teaching two methods and confusing things, I'm sticking with one.

Testing

- First let's install the `pytest`[6] library.
- Let's create a `tests` folder.
- In that folder, let's create a `test_quadratic.py`.
- What type of test is this?
- Run it!
- But what about our doctests?

Try it out!

```
pytest --doctest-modules
```

[6] **NOTE:** Python ships with the `unittest` library, but rather than teaching two methods and confusing things, I'm sticking with one.

Testing

- `pytest` picks up files called `test_*`

Testing

- `pytest` picks up files called `test_*`
- Runs all functions starting with `test_`.

Testing

- `pytest` picks up files called `test_*`
- Runs all functions starting with `test_`.
- (and as mentioned with the `--doctest-modules` flag, runs those too)

Testing

- `pytest` picks up files called `test_*`
- Runs all functions starting with `test_`.
- (and as mentioned with the `--doctest-modules` flag, runs those too)
- Collates them all and runs them together.

Multiple-Testing

- So we have our first test, but solving $3x^2 - 1$ shouldn't be all we try.

Multiple-Testing

- So we have our first test, but solving $3x^2 - 1$ shouldn't be all we try.
- Try to think of all the common cases...

Multiple-Testing

- So we have our first test, but solving $3x^2 - 1$ shouldn't be all we try.
- Try to think of all the common cases...
  1. What other common cases might we try?

Multiple-Testing

- So we have our first test, but solving $3x^2 - 1$ shouldn't be all we try.
- Try to think of all the common cases...
  1. What other common cases might we try?
  2. What happens if $a = 0$?

Multiple-Testing

- So we have our first test, but solving $3x^2 - 1$ shouldn't be all we try.
- Try to think of all the common cases...
  1. What other common cases might we try?
  2. What happens if $a = 0$?
  3. What happens if $b^2 - 4ac < 0$?

Multiple-Testing

- So we have our first test, but solving $3x^2 - 1$ shouldn't be all we try.
- Try to think of all the common cases...
  1. What other common cases might we try?
  2. What happens if $a = 0$?
  3. What happens if $b^2 - 4ac < 0$?
  4. What happens if I pass `int`s?

Multiple-Testing

- So we have our first test, but solving $3x^2 - 1$ shouldn't be all we try.
- Try to think of all the common cases...
  1. What other common cases might we try?
  2. What happens if $a = 0$?
  3. What happens if $b^2 - 4ac < 0$?
  4. What happens if I pass `int`s?

     ⋮ ...

Multiple-Testing

- Focussing on question 1...

Multiple-Testing

- Focussing on question 1...
- We want to run say: x^2, $x^2 + 14x + 49$, $3x^2 + 2x + 1$, ...

- Focussing on question 1...
- We want to run say: x^2, $x^2 + 14x + 49$, $3x^2 + 2x + 1$, ...
- Do we need to create a function for each one?

Multiple-Testing

- Focussing on question 1...
- We want to run say: x^2, $x^2 + 14x + 49$, $3x^2 + 2x + 1$, ...
- Do we need to create a function for each one? No.

```
@pytest.mark.parametrize('params, expected',
                         [([1., 0., 0.], [0., 0.]),
                          ([1., 14., 49.], [7., 7.]),
                          ([3., 2., 1.], [1/3, -1.])])
def test_quadratic(params, expected):
    """Test quadratic meets expectations."""
    assert np.isclose(quadratic(*params), expected)
```

Multiple-Testing

- Focussing on question 1...
- We want to run say: x^2, $x^2 + 14x + 49$, $3x^2 + 2x + 1$, ...
- Do we need to create a function for each one? No.

```
@pytest.mark.parametrize('a', [1, 2, 3])
@pytest.mark.parametrize('b', [1, 2, 3])
def test_example(a, b):
    """Example function taking 2 arguments."""
    assert np.product([a, b]) == a*b
```

Note

Stacked `pytest.mark.parametrize`s give Cartesian product.

Testing failures

- Now we need to fail spectacularly.

[7] HCF - Halt and Catch Fire – Genuine assembly instruction

Testing failures

- Now we need to fail spectacularly.
- Usually, providing a wrong answer is worse than exploding[7].

[7] HCF - Halt and Catch Fire – Genuine assembly instruction

Testing failures

- Now we need to fail spectacularly.
- Usually, providing a wrong answer is worse than exploding[7].
- It's good to make sure our failures fail and are helpful.

```
def test_quadratic_fails():
    """Check bad quadratic raises error."""
    with pytest.raises(ValueError,
                       match="negative discriminant"):
        # There are infinite roots on this flat line.
        quadratic(0., 0., 0.)
```

[7] HCF - Halt and Catch Fire – Genuine assembly instruction

Testing failures

- Now we need to fail spectacularly.
- Usually, providing a wrong answer is worse than exploding[7].
- It's good to make sure our failures fail and are helpful.
- Does it fail?

```
def test_quadratic_fails():
    """Check bad quadratic raises error."""
    with pytest.raises(ValueError,
                       match="negative discriminant"):
        # There are infinite roots on this flat line.
        quadratic(0., 0., 0.)
```

[7] HCF - Halt and Catch Fire – Genuine assembly instruction

Testing as design

- We should know what we want our code to do before we write it.

Testing as design

- We should know what we want our code to do before we write it.
- One way of writing software is:

Testing as design

- We should know what we want our code to do before we write it.
- One way of writing software is:
  - Define tests which describe desired functionality.

Testing as design

- We should know what we want our code to do before we write it.
- One way of writing software is:
  - Define tests which describe desired functionality.
  - Develop until tests pass.

Testing as design

- We should know what we want our code to do before we write it.
- One way of writing software is:
  - Define tests which describe desired functionality.
  - Develop until tests pass.
- This can be useful for known problems.

Testing as design

- We should know what we want our code to do before we write it.
- One way of writing software is:
  - Define tests which describe desired functionality.
  - Develop until tests pass.
- This can be useful for known problems.
- Roughly describing something called Behaviour-Driven Development.

Testing

- Ok, I've written a test which doesn't work (yet).

Testing

- Ok, I've written a test which doesn't work (yet).
- We can skip the test if we know it doesn't work (yet).

```
@pytest.mark.skip(reason="Beyond maths as we know it.")
def test_quintic():
    """Test quintic meets expectations."""
    assert np.isclose(quintic(1., 0., 0., 0., 0., 0., 0.),
        expected)
```

Testing

- Ok, I've written a test which doesn't work (yet).
- We can skip the test if we know it doesn't work (yet).
- It is bad practice to skip tests because they don't work.

```
@pytest.mark.skip(reason="Beyond maths as we know it.")
def test_quintic():
    """Test quintic meets expectations."""
    assert np.isclose(quintic(1., 0., 0., 0., 0., 0., 0.),
          expected)
```

Testing

- Ok, I've written a test which doesn't work (yet).
- We can skip the test if we know it doesn't work (yet).
- It is bad practice to skip tests because they don't work.
- It is worse practice to remove tests because they don't work.

```python
@pytest.mark.skip(reason="Beyond maths as we know it.")
def test_quintic():
    """Test quintic meets expectations."""
    assert np.isclose(quintic(1., 0., 0., 0., 0., 0., 0.),
        expected)
```

Testing

- Ok, I've written a test which doesn't work (yet).
- We can skip the test if we know it doesn't work (yet).
- It is bad practice to skip tests because they don't work.
- It is worse practice to remove tests because they don't work.
- Remove tests only if they don't fit the design.

```python
@pytest.mark.skip(reason="Beyond maths as we know it.")
def test_quintic():
    """Test quintic meets expectations."""
    assert np.isclose(quintic(1., 0., 0., 0., 0., 0., 0.),
        expected)
```

Testing

- Testing helps you:

Testing

- Testing helps you:
  - Prove the efficacy of your code.

Testing

- Testing helps you:
  - Prove the efficacy of your code.
  - Develop functionality defined by requirements.

Testing

- Testing helps you:
  - Prove the efficacy of your code.
  - Develop functionality defined by requirements.
  - Identify exactly when something went wrong.

Testing

- Testing helps you:
  - Prove the efficacy of your code.
  - Develop functionality defined by requirements.
  - Identify exactly when something went wrong.
  - Avoid adding broken code.

Testing

- Testing helps you:
  - Prove the efficacy of your code.
  - Develop functionality defined by requirements.
  - Identify exactly when something went wrong.
  - Avoid adding broken code.
- **Hint:** Testing is good.

Testing

- Testing helps you:
  - Prove the efficacy of your code.
  - Develop functionality defined by requirements.
  - Identify exactly when something went wrong.
  - Avoid adding broken code.
- **Hint:** Testing is good.
- Code without tests can be considered worthless.

Testing

- Testing helps you:
  - Prove the efficacy of your code.
  - Develop functionality defined by requirements.
  - Identify exactly when something went wrong.
  - Avoid adding broken code.
- **Hint:** Testing is good.
- Code without tests can be considered worthless.
- So let's add it to our `pyproject.toml`

Adding it in

```
[project.optional-dependencies]
docs = ["sphinx",
        "sphinx-rtd-theme",
        "sphinx_autodoc_typehints"]
test = ["pytest"]
```

Testing

- As discussed a few other times `pytest` is one of many testing frameworks. Others include:

Testing

- As discussed a few other times `pytest` is one of many testing frameworks. Others include:
- `unittest` – Basic test harness installed with `Python`.

Testing

- As discussed a few other times `pytest` is one of many testing frameworks. Others include:
- `unittest` – Basic test harness installed with `Python`.
- `cucumber` – Tests written in "English" rather than code.

Testing

- As discussed a few other times `pytest` is one of many testing frameworks. Others include:
- `unittest` – Basic test harness installed with `Python`.
- `cucumber` – Tests written in "English" rather than code.
- `hypothesis` – Tests with randomly generated values meeting requirements.

CI/CD

Automagic

- But running tests isn't fun.

Automagic

- But running tests isn't fun.
- It'd be boring if we had to do it **every** time.

Automagic

- But running tests isn't fun.
- It'd be boring if we had to do it **every** time.
- If only there were a better way...

Automagic

- But running tests isn't fun.
- It'd be boring if we had to do it **every** time.
- If only there were a better way...
- CI/CD (continuous integration/continuous deployment)

Automagic

- But running tests isn't fun.
- It'd be boring if we had to do it **every** time.
- If only there were a better way...
- CI/CD (continuous integration/continuous deployment)
- Fancy name which means automated testing & building.

Automagic

- GitHub lets us run tests on their machines.

Automagic

- GitHub lets us run tests on their machines.
- with some minor caveats

Automagic

- GitHub lets us run tests on their machines.
- with some minor caveats
- We just need to tell them what they need to do.

Automagic

- GitHub lets us run tests on their machines.
- with some minor caveats
- We just need to tell them what they need to do.
- We do this by adding a `yaml` file in the right place.

Adding it in!

```
mkdir -P .github/workflows
cp <pres>/test_florp.yml .github/workflows
```

Automagic

- GitHub lets us run tests on their machines.
- with some minor caveats
- We just need to tell them what they need to do.
- We do this by adding a `yaml` file in the right place.
- GitHub provides actions where we just need to fill in values.

Anatomy of the YAML

- Display name and script permissions.

```yaml
name: Python application

permissions:
  contents: read
```

Anatomy of the YAML

- Display name and script permissions.
- What will trigger the run.
  - When `main` changes.
  - When a pull request is opened or changes.

```
on:
  push:
    branches: [ "main" ]
  pull_request:
    types:
      - opened
      - synchronize
      - reopened
```

Anatomy of the YAML

- Display name and script permissions.
- What will trigger the run.
- Main job description.

```yaml
jobs:
  build:

    runs-on: ubuntu-latest

    strategy:
      matrix:
        python-version: ["3.8", "3.9", "3.10"]

    steps:
    - uses: actions/checkout@v3
    - name: Set up Python ${{ matrix.python-version }}
      uses: actions/setup-python@v3
      with:
        python-version: ${{ matrix.python-version }}
    - name: Install
      run: |
        python -m pip install --upgrade pip
        pip install .
    - name: Run tests
      run: |
        pytest --doctest-modules
```

Anatomy of the YAML

- Display name and script permissions.
- What will trigger the run.
- Main job description.
- Run on Ubuntu with each of the python versions

```
runs-on: ubuntu-latest

strategy:
  matrix:
    python-version: ["3.8", "3.9", "3.10"]
```

Anatomy of the YAML

- Display name and script permissions.
- What will trigger the run.
- Main job description.
- Run on Ubuntu with each of the python versions
- Job stages using matrix defined previously.

```yaml
 steps:
- uses: actions/checkout@v3
- name: Set up Python ${{ matrix.python-version }}
  uses: actions/setup-python@v3
  with:
    python-version: ${{ matrix.python-version }}
- name: Install
  run: |
    python -m pip install --upgrade pip
    pip install .
- name: Run tests
  run: |
    pytest --doctest-modules
```

Anatomy of the YAML

- Display name and script permissions.
- What will trigger the run.
- Main job description.
- Run on Ubuntu with each of the python versions
- Job stages using matrix defined previously.
  - Download the repo using git
  - Install Python

```
- uses: actions/checkout@v3
- name: Set up Python ${{ matrix.python-version }}
  uses: actions/setup-python@v3
  with:
    python-version: ${{ matrix.python-version }}
```

Anatomy of the YAML

- Display name and script permissions.
- What will trigger the run.
- Main job description.
- Run on Ubuntu with each of the python versions
- Job stages using matrix defined previously.
    - Download the repo using git
    - Install Python
    - Install the project

```
run: |
  python -m pip install --upgrade pip
  pip install .
```

Anatomy of the YAML

- Display name and script permissions.
- What will trigger the run.
- Main job description.
- Run on Ubuntu with each of the python versions
- Job stages using matrix defined previously.
    - Download the repo using git
    - Install Python
    - Install the project
    - Run the tests.

```
run: |
  pytest --doctest-modules
```

Testing

- For more info see
  `https://docs.github.com/en/actions`

Testing

- For more info see `https://docs.github.com/en/actions`
- There is a bit of magic, using other people's scripts.

Testing

- For more info see
  `https://docs.github.com/en/actions`
- There is a bit of magic, using other people's scripts.
  - (The `actions/...@v3`)

Testing

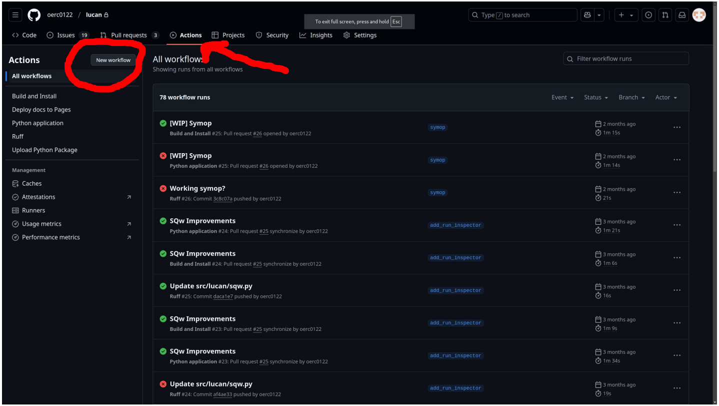
- For more info see `https://docs.github.com/en/actions`
- There is a bit of magic, using other people's scripts.
  - (The `actions/...@v3`)
- Besides that, it's just the commands you would run.

Testing

- For more info see
  `https://docs.github.com/en/actions`
- There is a bit of magic, using other people's scripts.
  - (The `actions/...@v3`)
- Besides that, it's just the commands you would run.
- GitHub offers Windows/Mac machines too!

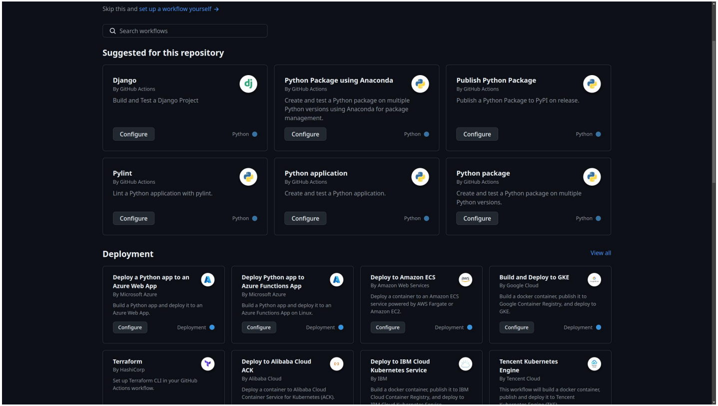
Action Economy

- GitHub contains a number of pre-written scripts for doing common jobs.



Action Economy

- GitHub contains a number of pre-written scripts for doing common jobs.
- These can be useful starting points for writing more complex scripts yourself.



Documentation in Action(s)

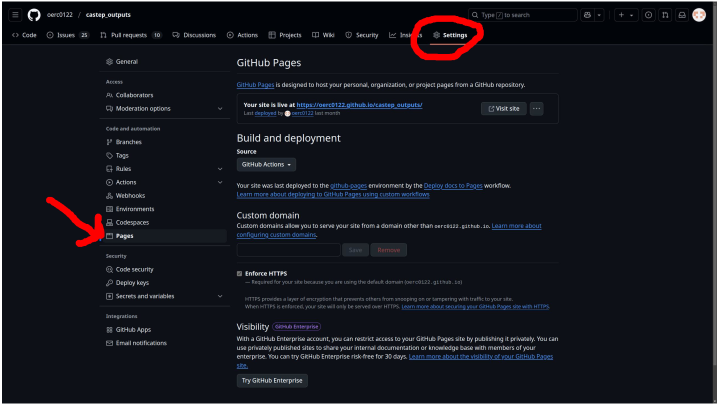
- But what about all the docs we've written?

Documentation in Action(s)

- But what about all the docs we've written?
- I don't want to host a website (but you can)!

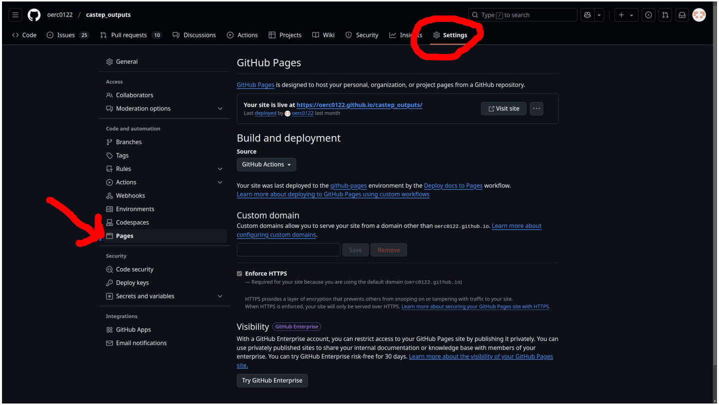
Documentation in Action(s)

- But what about all the docs we've written?
- I don't want to host a website (but you can)!
- GitHub provides "GitHub pages"; sites for projects.



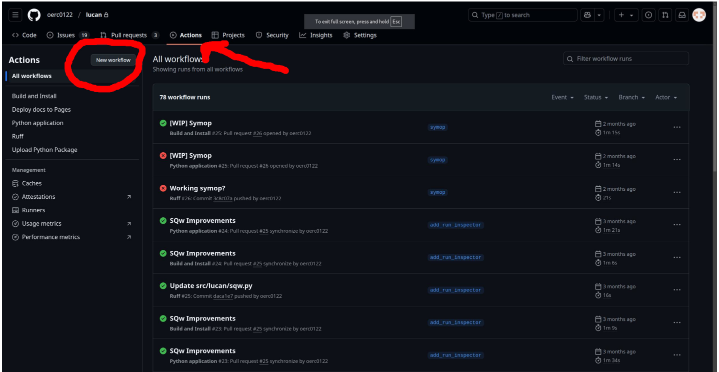
Documentation in Action(s)

- But what about all the docs we've written?
- I don't want to host a website (but you can)!
- GitHub provides "GitHub pages"; sites for projects.
- These can point to a branch or be managed by actions.



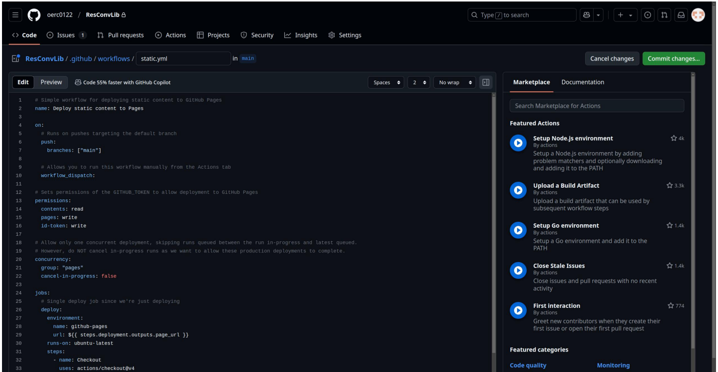
Documentation in Actions(s)

- So let's give it a go!



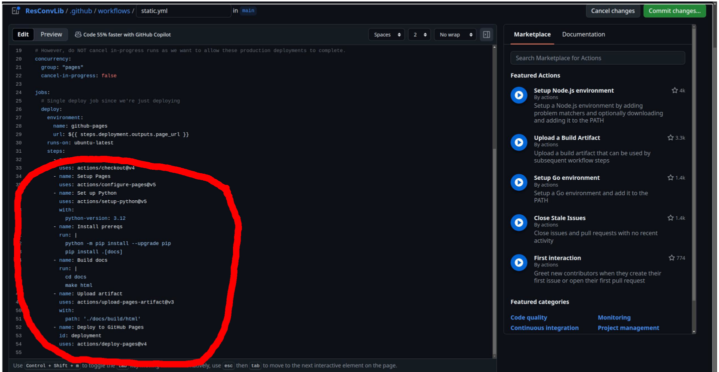
Documentation in Actions(s)

- So let's give it a go!
- Go to the marketplace and find an action doing (almost) what we want.
  Static Pages.



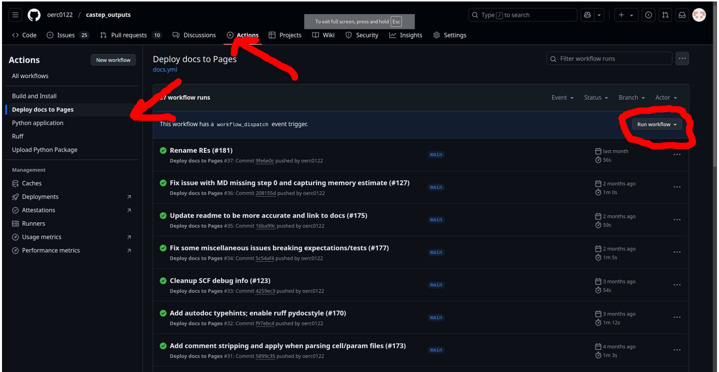
Documentation in Actions(s)

- So let's give it a go!
- Go to the marketplace and find an action doing (almost) what we want.
  Static Pages.
- Add a bit of script to make it do (exactly) what we want.
  Build the docs and upload those.



Documentation in Actions(s)

- So let's give it a go!
- Go to the marketplace and find an action doing (almost) what we want.
  Static Pages.
- Add a bit of script to make it do (exactly) what we want.
  Build the docs and upload those.
- Run it!



Documentation in Actions(s)

- So let's give it a go!
- Go to the marketplace and find an action doing (almost) what we want.
  Static Pages.
- Add a bit of script to make it do (exactly) what we want.
  Build the docs and upload those.
- Run it!
- Go to `https://<username>.github.io/polysolve`!

Bonus

CITATION.cff

- Ok, so you've written the best project ever.

CITATION.cff

- Ok, so you've written the best project ever.
- At what point does the fame and glory start rolling in?

CITATION.cff

- Ok, so you've written the best project ever.
- At what point does the fame and glory start rolling in?
- Until people know how great you are that's not going to happen.

CITATION.cff

- Ok, so you've written the best project ever.
- At what point does the fame and glory start rolling in?
- Until people know how great you are that's not going to happen.
- `CITATION.cff` is a standard format for code attribution.

```
cff-version: 1.2.0
message: "If you use this software, please cite it as below."
authors:
  - family-names: Example
    given-names: Stephen
    orcid: https://orcid.org/1234-5678-9101-1121
title: "My Research Software"
version: 2.0.4
identifiers:
  - type: doi
    value: 10.5281/zenodo.1234
date-released: 2021-08-11
```

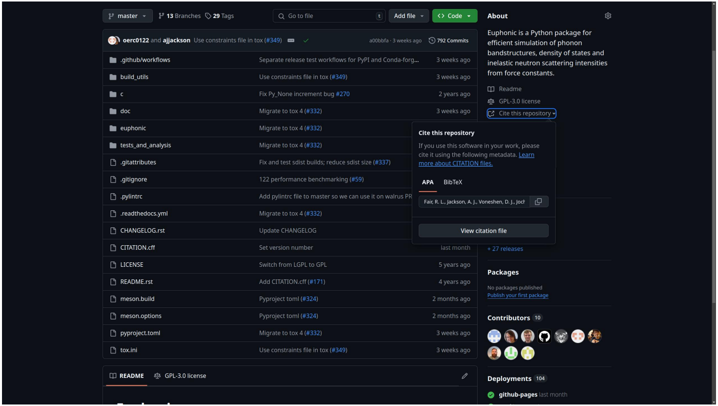
CITATION.cff

- Ok, so you've written the best project ever.
- At what point does the fame and glory start rolling in?
- Until people know how great you are that's not going to happen.
- CITATION.cff is a standard format for code attribution.
- https://citation-file-format.github.io/


```
cff-version: 1.2.0
message: "If you use this software, please cite it as below."
authors:
  - family-names: Example
    given-names: Stephen
    orcid: https://orcid.org/1234-5678-9101-1121
title: "My Research Software"
version: 2.0.4
identifiers:
  - type: doi
    value: 10.5281/zenodo.1234
date-released: 2021-08-11
```

CITATION.cff

- Ok, so you've written the best project ever.
- At what point does the fame and glory start rolling in?
- Until people know how great you are that's not going to happen.
- `CITATION.cff` is a standard format for code attribution.
- `https://citation-file-format.github.io/`



DOIs

- Going a step further than just adding the citation file.

DOIs

- Going a step further than just adding the citation file.
- Mint a DOI for a version of the software allowing it to be cited.

Extra tools

- Remember we mentioned an IDE easlier in the talk?

Extra tools

- Remember we mentioned an IDE easlier in the talk?
- An IDE is just one of many tools which can be helpful in development.

Extra tools

- Remember we mentioned an IDE easlier in the talk?
- An IDE is just one of many tools which can be helpful in development.
  - IDE – Specialised editor to make development easier
    - (`VSCode`, `Spyder`).

Extra tools

- Remember we mentioned an IDE easlier in the talk?
- An IDE is just one of many tools which can be helpful in development.
  - IDE – Specialised editor to make development easier
    - (`VSCode`, `Spyder`).
  - Linting – Checks for stylistic errors
    - (`ruff`, `flake8`, `pylint`)

Extra tools

- Remember we mentioned an IDE easlier in the talk?
- An IDE is just one of many tools which can be helpful in development.
  - IDE – Specialised editor to make development easier
    - (`VSCode`, `Spyder`).
  - Linting – Checks for stylistic errors
    - (`ruff`, `flake8`, `pylint`)
  - Type checking – Checks for passing the wrong data through
    - (`mypy`, `pydantic`, `beartype`)

Ruff times

- Ruff[8] is a tool to encourage (enforce) code standards.

Adding it in

```
[project.optional-dependencies]
docs = ["sphinx",
        "sphinx-rtd-theme",
        "sphinx_autodoc_typehints"]
test = ["pytest"]
lint = ["ruff==0.8.0"]
```

[8] https://docs.astral.sh/ruff/

Ruff times

- Ruff[8] is a tool to encourage (enforce) code standards.
- Modern replacement for half a dozen prior tools.

[8] `https://docs.astral.sh/ruff/`

Ruff times

- Ruff[8] is a tool to encourage (enforce) code standards.
- Modern replacement for half a dozen prior tools.
- Flags violations of code standards.

Try it out!

```
ruff check
```

[8] https://docs.astral.sh/ruff/

Ruff times

- Ruff[8] is a tool to encourage (enforce) code standards.
- Modern replacement for half a dozen prior tools.
- Flags violations of code standards.
- Also handles formatting code

Try it out!

```
ruff format
```

[8] https://docs.astral.sh/ruff/

Ruff times

- Ruff[8] is a tool to encourage (enforce) code standards.
- Modern replacement for half a dozen prior tools.
- Flags violations of code standards.
- Also handles formatting code
- Customisable through `pyproject.toml`

```
[tool.ruff.lint]
select = [
    "PL", # Pylint errors
    "E",   # Pycodestyle
    "W",   # Pycodestyle warnings
    "F",   # Pyflakes
    "B",   # Flake8 bugbear
]
```

[8] https://docs.astral.sh/ruff/

Ruff times

- Ruff[8] is a tool to encourage (enforce) code standards.
- Modern replacement for half a dozen prior tools.
- Flags violations of code standards.
- Also handles formatting code
- Customisable through `pyproject.toml`

```
[tool.ruff]
line-length = 100
indent-width = 4
target-version = "py38"

[tool.isort]
profile = "black"
```

[8] https://docs.astral.sh/ruff/

Ruff times[9]

```yaml
name: Lint

on:
  push:
  pull_request:

lint_check_ruff:
  runs-on: ubuntu-latest
  steps:
    - uses: actions/checkout@v4
    - uses: astral-sh/ruff-action@v3
      with:
        args: "check"

lint_format_ruff:
  runs-on: ubuntu-latest
  steps:
    - uses: actions/checkout@v4
    - uses: astral-sh/ruff-action@v3
      with:
        args: "format --check"
```

[9] lint.yml

cookiecutter

- Possible to take out some of the boilerplate of setup.

cookiecutter

- Possible to take out some of the boilerplate of setup.
- `cookiecutter.io` is a set of pre-configured project folders.

cookiecutter

- Possible to take out some of the boilerplate of setup.
- `cookiecutter.io` is a set of pre-configured project folders.
- Modules for many languages (including Python)

Checklist

- ☐ Sensible names
- ☐ On GitHub
- ☐ Project layout
  - ☐ `__init__.py`
  - ☐ `pyproject.toml`
- ☐ Documentation
  - ☐ Docstrings
  - ☐ Doctests
  - ☐ Typehints
  - ☐ `sphinx-quickstart`
  - ☐ `sphinx-apidoc`
- ☐ Tests
- ☐ CI/CD
  - ☐ Automated testing
  - ☐ Automatic documentation
- ☐ `CITATION.cff`