Testing

Good research code

Patrick Mineault

Bulding around testing

Most scientists who write software constantly test their code. That is, if you are a scientist writing software, I am sure that you have tried to see how well your code works by running every new function you write, examining the inputs and the outputs of the function, to see if the code runs properly (without error), and to see whether the results make sense. Automated code testing takes this informal practice, makes it formal, and automates it, so that you can make sure that your code does what it is supposed to do, even as you go about making changes around it. –Ariel Rokem, Shablona README

Open discussion

- Let's test `fib.py`
- What can we test?

What can we test about `fib`?

- Correctness, e.g. $F(4) = 5$
- Edge cases, e.g. $F(0) = 1$, $F(-1) \rightarrow \textit{error}$
- Functional goals are achieved, e.g. caching works
- It's much easier to test decoupled code with no side effects
  - Forces you to write modular decoupled code

How can you decide what to test?

- If something caused a bug, test it
  - 70% of bugs will be old bugs that keep reappearing
- If you manually checked if procedure X yielded something reasonable results, write a test for it.

How can we test?

- `assert`
- Hide code behind `if __name__ == '__main__'`
- Test suite

`assert`

- `assert` throws an error if the assertion is False

```
assert -(7 // 2) == (-7 // 2)
```

- Great for inline tests
- e.g. check whether the shape of a matrix is correct after a permute op

Hide code behind `if __name__ == '__main__'`

- Code behind `__name__ == '__main__'` is only run if you run the file as a script directly.
- Use this for lightweight tests in combination with `assert`.

```
if __name__ == '__main__':
    assert fib(4) == 5
```

Use a test suite

- Create a specialized file with tests that run with the help of a runner.
- There's `pytest` and `unittest`.
- I use `unittest` because that's what I learned, and it's built-in.

Basic template

```python
# test_something.py
import unittest

class MyTest(unittest.TestCase):
    def sample_test(self):
        self.assertTrue(True)

if __name__ == '__main__':
    unittest.main()
```

Run it

```
$ python test_something.py
```

To run all tests within a directory:

```
$ nosetests
```

Live coding

Let's code up `fib.py` tests!

Points from live coding example

- Paths!
  - Sometimes you can get away with hacking `sys.path`
  - Ideally, set up a package with `pip install -e .`
- There's a lot of cruft in writing tests: no shame in copy and paste!

A hierarchy of tests can be run with a runner

- Static tests (literally your editor parsing your code to figure if it will crash)
- Asserts
- Unit tests (test one function = one unit; what we just saw)
- Integration tests
- Smoke tests
- Regression tests
- E2E (literally a robot clicking buttons)

Write lots of tiny unit tests that run very quickly

- Goal: each unit test should run in 1 ms.
- The faster you iterate, the better for your WM.
  - If your test suite takes more than 5 seconds to run, you will be tempted to go do something else.

Open discussion

Q: what do you think is the ratio of test code to real code in a real codebase?

Open discussion

A: 1:1 to 3:1, but can be many, many times that in safety critical applications

e.g. the aviation standard DO-178C requires 100% code coverage at its third highest safety level (Level C).

Demo

Let's code CKA tests. We will turn properties of CKA listed in the paper into tests.

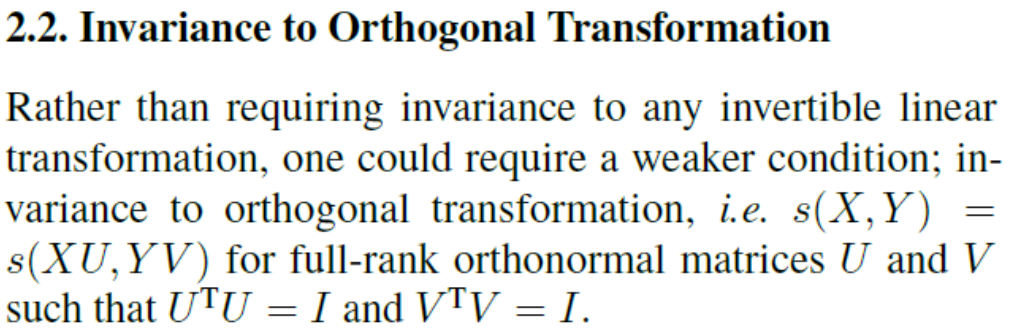
What we know about CKA

- Only makes sense if two matrices are the same size along the first dimension
- Pearson correlation: If $\mathbf{X}$ and $\mathbf{Y}$ are one-dimensional, then $CKA = \rho(\mathbf{X}, \mathbf{Y})^2$.
- $CKA(\mathbf{X}, \mathbf{X}) = 1$

Live coding

What else can we know about CKA? Let's read the paper!

- 2.1 *not* invariant to non-isotropic scaling
- 2.2 invariant to rotations, $CKA(\alpha\mathbf{XU}, \beta\mathbf{YV}) = CKA(\mathbf{X}, \mathbf{Y})$



2.2. Invariance to Orthogonal Transformation

Rather than requiring invariance to any invertible linear transformation, one could require a weaker condition; invariance to orthogonal transformation, *i.e.* $s(X, Y) = s(XU, YV)$ for full-rank orthonormal matrices U and V such that $U^{\mathrm{T}}U = I$ and $V^{\mathrm{T}}V = I$.

Figure 1: Invariance to rotation

- 2.3 invariant to isotropic scaling, $CKA(\alpha\mathbf{X}, \beta\mathbf{Y}) = CKA(\mathbf{X}, \mathbf{Y})$

Live coding (II)

Points from live coding example

- Your test code can be ugly, as long as it's functional!
- Define boundary conditions, pathological examples
  - Test that bad inputs indeed raise errors! Your code should yell when you feed it bad inputs.
- Lock in current behaviour for regression testing

Refactoring with confidence

- Your code is ugly: time to refactor!
  1. Your code is ugly, tests pass
  2. Rewrite the code
  3. Your code is clean, tests don't pass
  4. Rewrite the code
  5. Iterate until tests pass again
- Much less stressful with tests and git
- Focus on one test at a time with `python test_cka.py TestCka.test_same`
  - Don't forget to run the whole suite at the end!

Advanced topics!

Testing deterministic side-effect free computational code has a very high returns:effort ratio, but...

- You can also test data loaders for correctness.
- You can also test data for correctness
- You can also test notebooks for correctness
- You can integrate your tests into Github
- You can test stochastic functions

Lesson 3

- Test your code
- Free your WM from having to consider that a piece of code unrelated to the thing you care about is broken
- From lesson 1: much simpler to refactor code to make it tidy when you know you have a test scaffold which catch mistakes
- From lesson 2: you will have to decouple code to write tests
- Your 5-minute assignment: find a commented-out print statement in your code and replace it with `assert`