



USPAS – *Simulation of Beam and Plasma Systems*

Steven M. Lund, Jean-Luc Vay, Remi Lehe, Daniel Winklehner and David L. Bruhwiler

Lecture: **Software Testing**

Instructor: David L. Bruhwiler

Contributors: R. Nagler



U.S. Particle Accelerator School sponsored by **Old Dominion University**

<http://uspas.fnal.gov/programs/2018/odu/courses/beam-plasma-systems.shtml>

January 15-26, 2018 – Hampton, Virginia

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Offices of High Energy Physics and Basic Energy Sciences, under Award Number(s) DE-SC0011237 and DE-SC0011340.



Office of Science

Motivation

- Untested software is broken software
 - look for the tests that are being used to maintain software you're using
- Are there any tests for the software you are using?
 - sometimes there are no tests (or very few)
 - maybe there is a suite of examples
 - this can be very helpful
 - especially if the expected results are included
- Do computational physicists worry about this?
 - we rely (in part) on the reputation of the developers
 - national lab or university team
 - for commercial codes, a company
 - also, we can simulate cases to compare with theory
 - we benchmark with other codes, when possible
 - we trust our own physical intuition to identify problematic results
 - if we see many other users & published results, we feel OK
- This is not a great situation for computational accelerator physics
 - look for tests, ask the developers for tests, complain



D. Bruhwiler – USPAS – January 2018 – Software Testing

2

Validity – this is essential to the issue of reproducibility

- Computational science requires validated software
 - we want our results to be correct
 - need confidence that other published/presented results are correct
- Build environment and computing platform:
 - software is validated on a particular platform, compiler, etc.
 - supporting multiple platforms is possible, but expensive
 - what if the software is built, installed, executed on another platform?
 - can one be sure that it is still valid?
- Versioning:
 - a particular version of software is validated
 - regular use of regression tests can help maintain validity across versions
 - however: tests are never complete, features are added/removed, etc.
 - for multiple dependencies, versioning becomes an N^2 problem
 - how can one be sure of the validity of a particular software version?
 - how can one communicate full versioning information to others?
- Sharing:
 - difficult to share identical build & version(s), including dependencies



Automated software testing

- When you are writing software, you should create tests
 - you should also create tests for simulations that you are doing
- You should also run the tests regularly
- Automated testing frameworks make this possible
 - not easy necessarily, but the effort always pays off
 - there are many frameworks, but we will not review them here
 - look around and find one that you like
- For Python, **pytest** is a good, <https://docs.pytest.org>



About pytest

pytest is a mature full-featured Python testing tool that helps you write better programs.

pytest: helps you write better programs

The **pytest** framework makes it easy to write small tests, yet scales to support complex functional testing for applications and libraries.

An example of a simple test:

```
# content of test_sample.py
def inc(x):
    return x + 1

def test_answer():
    assert inc(3) == 5
```



Getting started with `pytest`

<https://docs.pytest.org/en/latest/getting-started.html#our-first-test-run>

Create a simple test function with just four lines of code:

```
# content of test_sample.py
def func(x):
    return x + 1

def test_answer():
    assert func(3) == 5
```

That's it. You can now execute the test function:

```
$ pytest
----- test session starts -----
platform linux -- Python 3.x.y, pytest-3.x.y, py-1.x.y, pluggy-0.x.y
rootdir: $REGENDOC_TMPDIR, inifile:
collected 1 item

test_sample.py F [100%]

----- FAILURES -----
_____ test_answer _____

    def test_answer():
>     assert func(3) == 5
E       assert 4 == 5
E       + where 4 = func(3)

test_sample.py:5: AssertionError
===== 1 failed in 0.12 seconds =====
```

This test returns a failure report because `func(3)` does not return 5.



`pytest` – naming conventions & test discovery

- by default `pytest` looks in all files & dirs below the current directory
- You can specify one or more paths to override the default:
 - \$ `pytest src/modules/example/test/`
- File names should start or end with "test"
 - for example: `test_example.py` or `example_test.py`
- for tests defined inside a class, the class name should start with "Test"
 - for example: `TestExample`
 - the Python class should **not** have an `__init__` method
- Test method names or function names **must** start with "test_"
 - as in `test_example`
- Learn what tests will be discovered:
 - \$ `pytest --collect-only`



RsBeams – a simple python library for beams

- You previously forked this repo to your own GitHub account
 - for the computer lab this afternoon, you will clone this forked repo
 - or you can clone the original repo, if you prefer
 - \$ git clone https://github.com/radiasoftware/rsbeams.git
 - or maybe you still have it on your class desktop or your own laptop
- heads up regarding the computer lab
 - RsBeams is compatible with Python 3.5 and 2.7
 - you may have to do the following:
 - \$ pip install pykern
 - \$ cd rsbeams/
 - \$ python setup.py install
 - \$ cd test/
 - \$ pytest
- decide what part of the code you would like to test
 - we will now spend some time reviewing the RsBeams source code
 - <https://github.com/radiasoftware/rsbeams>

