Writing regression tests for PLUMED

Giovanni Bussi

Regression testing

https://en.wikipedia.org/wiki/Regression_testing

Regression testing […] is re-running functional and non-functional tests to ensure that previously developed and tested software still performs as expected after a change. If not, that would be called a regression. […] Defects found at this stage are the most costly to fix. This problem is being addressed by the rise of unit testing.

Unit testing

https://en.wikipedia.org/wiki/Unit_testing

In computer programming, unit testing is a software testing method by which individual units of source code—sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures—are tested to determine whether they are fit for use

What is a “unit” in PLUMED?

Any C++ class? E.g. class `Pbc`

An Action? E.g. class `Colvar::Distance`

An Action created with a one-line plumed.dat file?
Or combined with PRINT?

There’s a spectrum of possible choices.

- Simple tasks might be tested “in context”.
- Tricky tasks (e.g. Pbc) might benefit a specific test, to avoid widespread unexplained failures.

In PLUMED, we only have “regtest” (but you can implement *de facto* unit tests there 😛)

Where and how are regtests located

Behavior of functions might change with PLUMED versions → within repository → Directory: plumed2/regtest

When implementing changes, you want to run different versions against the same test → not within code

Production packages must be testable → Home-made scripts, no additional dependency

How to run them

From plumed2/regtest (or subdirs): make
(assume executable named `plumed`)

From plumed2 : make check
(use the non-installed `plumed2/src/lib/plumed`)

From plumed2 : make installcheck
(use an installed plumed with different program name)

Which types

See regtest/script/run:

- Check which options were enabled
- Run the test (or compile + run, see below)
- Check diff

plumed: run plumed
driver: run plumed driver # (subcase of plumed)
sum_hills: run plumed sum_hills # (subclass of plumed)

make: compile a test program (C++, C, FORTRAN)

python: run the correct python interpreter

Anatomy of a regtest

Open and show:

basic/rt0

basic/rt-make-0

basic/rt-make-exceptions

Remember to discuss:

- Reference files, comparison of zeros
- How to reset

Other stuff

Pre/post scripts
(plumed_regtest_before, plumed_regtest_after)
E.g.: basic/rt-fix-371a/config

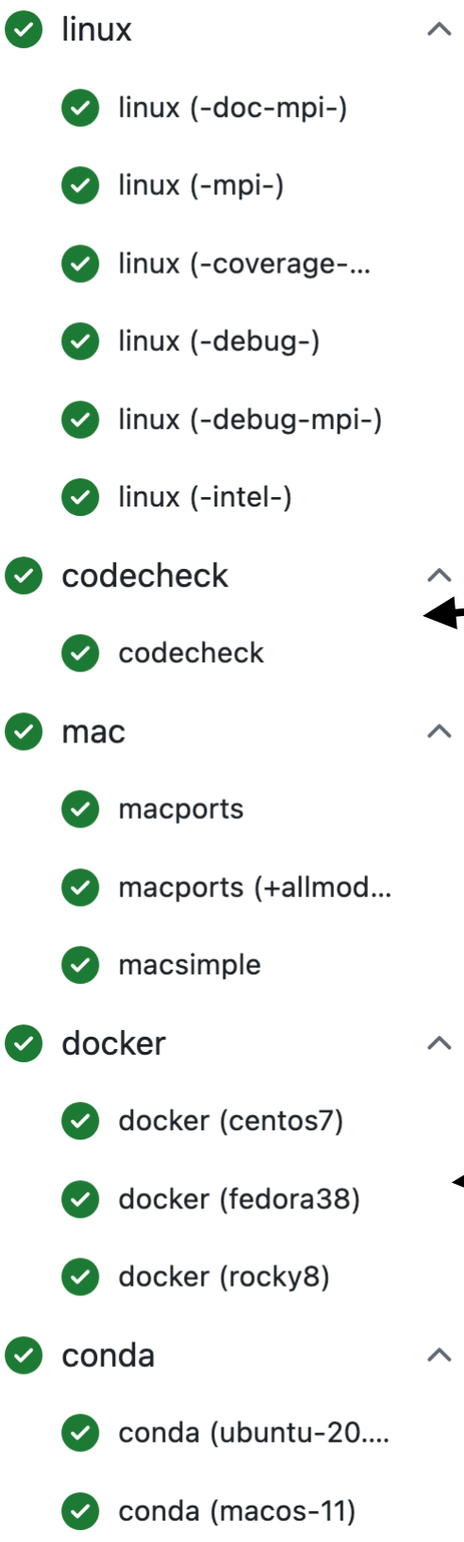
Custom skips (plumed_custom_skip)
E.g.: basic/rt32b/config

MPI (mpiprocs)
(note: set MPIEXEC in config or PLUMED_MPIRUN at runtime)
E.g. basic/rt-multi-1/config

Modules: (plumed_modules)
E.g. crystallization/rt-average-vec/config

Libraries (plumed_needs)
E.g. analysis/rt-fftw/config

Testing on GitHub Actions



Doc built and uploaded

Debug flags

Enforce coding rules
Cppcheck for bug hunting

Macports / Homebrew

Very old / very new compilers

Conda

Every commit is tested, takes ~ 1h

Coverage

Test:	plumed test coverage	**Lines:**	49801	58457	85.2 %
Date:	2023-06-28 11:44:29	**Functions:**	5549	6964	79.7 %

Directory	Line Coverage			Functions	
adjmat		88.6 %	1038 / 1171	82.1 %	183 / 223
analysis		88.5 %	859 / 971	77.2 %	156 / 202
annfunc		97.1 %	134 / 138	88.9 %	8 / 9
bias		91.4 %	2393 / 2617	85.6 %	143 / 167
cltools		80.0 %	1358 / 1698	92.0 %	103 / 112
colvar		94.0 %	2099 / 2233	82.5 %	184 / 223
config		45.3 %	58 / 128	45.8 %	22 / 48
core		85.7 %	3220 / 3757	80.3 %	428 / 533
crystallization		83.8 %	1015 / 1211	77.0 %	144 / 187
dimred		84.8 %	556 / 656	73.1 %	79 / 108
drr		94.3 %	1194 / 1266	94.8 %	92 / 97
eds		92.6 %	452 / 488	87.0 %	20 / 23
fisst		79.6 %	319 / 401	82.8 %	24 / 29
function		85.9 %	909 / 1058	84.0 %	84 / 100
funnel		98.0 %	248 / 253	86.7 %	13 / 15
generic		94.4 %	1204 / 1275	79.7 %	177 / 222
gridtools		93.6 %	1183 / 1264	85.5 %	171 / 200
isdb		81.6 %	8645 / 10595	81.5 %	198 / 243
logmfd		87.6 %	326 / 372	93.8 %	15 / 16
main		80.0 %	12 / 15	100.0 %	1 / 1
manyrestraints		81.5 %	75 / 92	69.2 %	18 / 26
mapping		93.7 %	758 / 809	89.0 %	89 / 100
maze		85.3 %	661 / 775	84.3 %	86 / 102
membranefusion		88.2 %	456 / 517	85.7 %	18 / 21

Possible exercises

Implement or fix something in PLUMED, then add a regression test:
- Make sure that code without the implementation fails the test
- Make sure the test covers relevant use cases

Look at current coverage scan (plumed.org/coverage-master), find untested features and add a test. Also test for errors are useful (i.e., make sure error is reported, see e.g. this file)

These additions are **very useful**, we are happy if you contribute them to the official GitHub repository! You will end up here: github.com/plumed/plumed2/graphs/contributors 👏