Testing and Continuous Integration

Martin Lüders

Octopus Course 2023, MPSD Hamburg

Motivation

Main use of scientific codes:

- Produce scientific results, often predictions
- Implement new theoretical developments

Both assume and require that the code gives correct results!

But: Scientific codes are extremely complex!

- Easy to make mistakes
- Methods might be numerically unstable
- Theory level might not be adequate

$\Longrightarrow$ All needs to be carefully tested!

Testing is difficult

What we would like to test: **The code gives correct results!**

- The code does what the algorithms promise (no bugs)
    - unit tests
    - test against exact results
- The algorithms are appropriate to represent the theory
    - test against exact results
- The theory is adequate to describe nature
    - test against analytical models

Most of the above need to be done by hand by developers.

Regression testing

The "easy" part:

- Assume the code is correct at some point.
- Make sure future developments don't break it!
  (happens often enough...)

Regression testing:

- set up calculation which tests new development
- record reference values (assumed to be "correct")
- automatically test changes to the code against these reference values.
- Tests should run on different computers and with different compilers
- Tests should probe all parts of the code
- Tests should run in a reasonable time
- Tests to check performance

Continued Integration (CI)

Integrate this testing into the development workflow:

- Tests should be automatically run when changes are done to the code (develop branch)
- Integrated into gitlab (so-called webhooks)
  - certain events (e.g. push, tag) can trigger external actions
  - push to branch with merge request (MR) or main: trigger buildbot
  - push to main rebuilds source code documentation
  - create tag: build distribution tarball, build web-pages
- We use buildbot for triggering these tasks
- We have a number of different computers to run the tests

Continued Integration (CI)

Buildbot:

- main
  - receives requests from gitlab (or web interface)
  - main configuration contains all details
    (e.g. list of workers, schedules, build and run options)
  - sends tasks to the workers
  - report back to gitlab
- workers
  - run tests: (git clone, configure and compile, run custom test script)
  - report results to buildbot master

Our test farm

Range of machines:

- intel x86
- PPC
- intel x86 + NVidia RTX2080 (2 CPU + 10 GPU)

Range of 'toolchains' (i.e. compilers + libraries):

- foss (gnu compilers), fosscuda
- intel, intelcuda
- different combinations with MPI and OpenMP
- several versions of each toolchain
- different optimizations and set of libraries
- valgrind

The Buildbot GUI

- Main views:
  - Waterfall
  - Grid
  - Console
- Pipeline view: Details of the test runs. (also "Rebuild")
  - Details of the run: Look here for error messages
  - Rebuild button
- Other tabs: builders, pending buildrequests, workers
  - Builders: list of pipelines
  - Pending buildrequests: look here to see how long you might have to wait.
  - Workers: list of machines: might indicate is a machine is 'ill'

Test script

Test script (run by buildbot, or locally):

`make check` or `make check-short`

- custom PERL script
- allows for simple if constructions in test files
- schedules tests for multi-processor workers
- handles parallelism

Test files

Example test file:

```
Test       : Crank-Nicolson (SPARSKIT)
Program    : octopus
TestGroups : short-run, real_time
Enabled    : Yes

Processors : 1
Input      : 16-sparskit.01-gs.inp
match ; SCF convergence ; GREPCOUNT(static/info, ’SCF converged’) ; 1
match ; Initial energy ; GREPFIELD(static/info, ’Total       =’, 3) ; -10.60764719

Processors : 4
Input      : 16-sparskit.02-kick.inp
if (available sparskit); then
    match ; Energy [step  1] ; LINEFIELD(td.general/energy, -21, 3) ; -1.058576638440e+01
    match ; Energy [step  5] ; LINEFIELD(td.general/energy, -16, 3) ; -1.043027231981e+01
    match ; Energy [step 10] ; LINEFIELD(td.general/energy, -11, 3) ; -1.043026650500e+01
    match ; Energy [step 15] ; LINEFIELD(td.general/energy, -6, 3) ; -1.043026483491e+01
    match ; Energy [step 20] ; LINEFIELD(td.general/energy, -1, 3) ; -1.043026489604e+01

    match ; Dipole [step  1] ; LINEFIELD(td.general/multipoles, -21, 4) ; 6.723772397619e-13
    match ; Dipole [step  5] ; LINEFIELD(td.general/multipoles, -16, 4) ; -7.295810087049e-01
    match ; Dipole [step 10] ; LINEFIELD(td.general/multipoles, -11, 4) ; -1.339402779435e+00
    match ; Dipole [step 15] ; LINEFIELD(td.general/multipoles, -6, 4) ; -1.833991374772e+00
    match ; Dipole [step 20] ; LINEFIELD(td.general/multipoles, -1, 4) ; -2.215415201335e+00
else
    match; Error missing SPARSKIT; GREPCOUNT(err, ’recompile with SPARSKIT support’) ; 1
endif
```

Writing tests

Having new features tested is essential.
Merge requests might not be accepted without providing a test!!

Guidelines:

- all features should be tested, but not necessarily in one test
- also test error messages
- make calculations as short as possible
- test several relevant quantities (matches are free)

Some remarks

- pushing test results back to gitlab sometimes fails
  ⟶ when in doubt, check on the buildbot GUI.
- We have some random failures
  - Numerical noise (e.g. due to parallelization): increase tolerance of test
  - Possible bugs? We don't know yet.
  - Try to rebuild that pipeline.
    If the failure remains, it's probably a bug!
  - Use the testsuite app to find systematic deviations.

Octopus testsuite web application

- Display results from testsuite
- Helps in adapting failed tests
  - Adjusting tolerance
  - Recentering
- Analyze tests ⟶ improve testsuite
- At `https://octopus-code.org/testsuite/`

Testsuite.app: Implementation

- Python framework: django
- Styling (CSS, JS): bootstrap, custom JS
- Database: postgres
- Plots: bokeh
- Django has a nice ORM
  - Object relational mapper
  - No SQL needed, only dealing with objects
  - But: some tuning needed (how to fetch data etc)

The web interface

- Start at landing page
- Select your branch or commit
  - Failed tests
  - Failed toolchains
- select test:
  - overview of all matchlines
  - results from every builder
- analyze failure:
  - completely wrong result: ⟶ debug
  - value slightly outside tolerance: possibly relax tolerance

General remarks

- The buildbot is usually quite busy
- Do not use the buildbot to test your code. Run the test locally first!
- If you need to push another commit while one pipeline still runs, please kill the old one. (grid view → 'tests' builder → stop)