Testing and Continuous Integration

Martin Lüders

Octopus Course 2021, MPSD Hamburg
Main use of scientific codes:

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

⇒ All needs to be carefully tested!
Testing is difficult

What we would like to test:
Testing is difficult

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

What we would like to test: **The code gives correct results!**
- The code does what the algorithms promise (no bugs)
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
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
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
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
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
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
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
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.
Regression testing

The "easy" part:

- Assume the code is correct at some point.
- Make sure future developments don’t break it!
Regression testing

The "easy" part:

- Assume the code is correct at some point.
- Make sure future developments don’t break it!
Regression testing

The "easy" part:

- Assume the code is correct at some point.
- Make sure future developments don’t break it!

Regression testing:

- set up calculation which tests new development
Regression testing

The "easy" part:

- Assume the code is correct at some point.
- Make sure future developments don’t break it!

Regression testing:

- set up calculation which tests new development
- record reference values (assumed to be "correct")
Regression testing

The "easy" part:

- Assume the code is correct at some point.
- Make sure future developments don’t break it!

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.
Regression testing

The "easy" part:

- Assume the code is correct at some point.
- Make sure future developments don’t break it!

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
Regression testing

The "easy" part:

- Assume the code is correct at some point.
- Make sure future developments don’t break it!

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
Regression testing

The "easy" part:

- Assume the code is correct at some point.
- Make sure future developments don’t break it!

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
Regression testing

The "easy" part:
- Assume the code is correct at some point.
- Make sure future developments don’t break it!

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 (performance regression tests)
Continued Integration (CI)

Integrate this testing into the development workflow:

- Tests should be automatically run when changes are done to the code (develop branch)
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)
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
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 develop or master: trigger buildbot
Continuous 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 develop or master: trigger buildbot
  - create tag: build distribution tarball, build web-pages
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 develop or master: trigger buildbot
  - create tag: build distribution tarball, build web-pages
- We use buildbot for triggering the test runs
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 develop or master: trigger buildbot
  - create tag: build distribution tarball, build web-pages
- We use buildbot for triggering the test runs
- We have a number of different computers to run the tests
Continued Integration (CI)

Buildbot:
- master
Buildbot:
- master
  - receives requests from gitlab (or web interface)
Continuous Integration (CI)

Buildbot:
- master
  - receives requests from gitlab (or web interface)
  - master configuration contains all details
    (e.g. list of workers, schedules, build and run options)
Continued Integration (CI)

Buildbot:

- master
  - receives requests from gitlab (or web interface)
  - master configuration contains all details
    (e.g. list of workers, schedules, build and run options)
  - sends tasks to the workers
**Continued Integration (CI)**

**Buildbot:**
- master
  - receives requests from gitlab (or web interface)
  - master configuration contains all details
    (e.g. list of workers, schedules, build and run options)
  - sends tasks to the workers
  - report back to gitlab
Buildbot:

- master
  - receives requests from gitlab (or web interface)
  - master configuration contains all details
    (e.g. list of workers, schedules, build and run options)
  - sends tasks to the workers
  - report back to gitlab

- workers
Continued Integration (CI)

Buildbot:

- master
  - receives requests from gitlab (or web interface)
  - master 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)
## Continued Integration (CI)

**Buildbot:**

- **master**
  - receives requests from gitlab (or web interface)
  - master 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 master
Our test farm

Range of machines:

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

Test script (run by buildbot, or locally):

- custom PERL and bash scripts
- allows for simple if constructions in test files
- schedules tests for multi-processor workers
- handles parallelism
Test scripts

Test script (run by buildbot, or locally):
- custom PERL and bash scripts
- allows for simple if constructions in test files
- schedules tests for multi-processor workers
- handles parallelism

make check or make check-short
Test scripts

Test script (run by buildbot, or locally):

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

make check or make check-short

- oct-run_testsuite.sh: run groups of tests and schedule the tests
- oct-run_regression_test.pl: run individual tests

Testfiles live in: testsuite/
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 ; 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 will not be accepted without providing a test!
Writing tests

Having new features tested is essential. Merge requests will 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)
- if possible, provide a unit test (see main/test.F90)
Some remarks

- pushing test results back to gitlab sometimes fails
  \[ \rightarrow \text{when in doubt, check on the buildbot GUI.} \]
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.