The Octopus/APE/BerkeleyGW testsuite infrastructure

David A. Strubbe

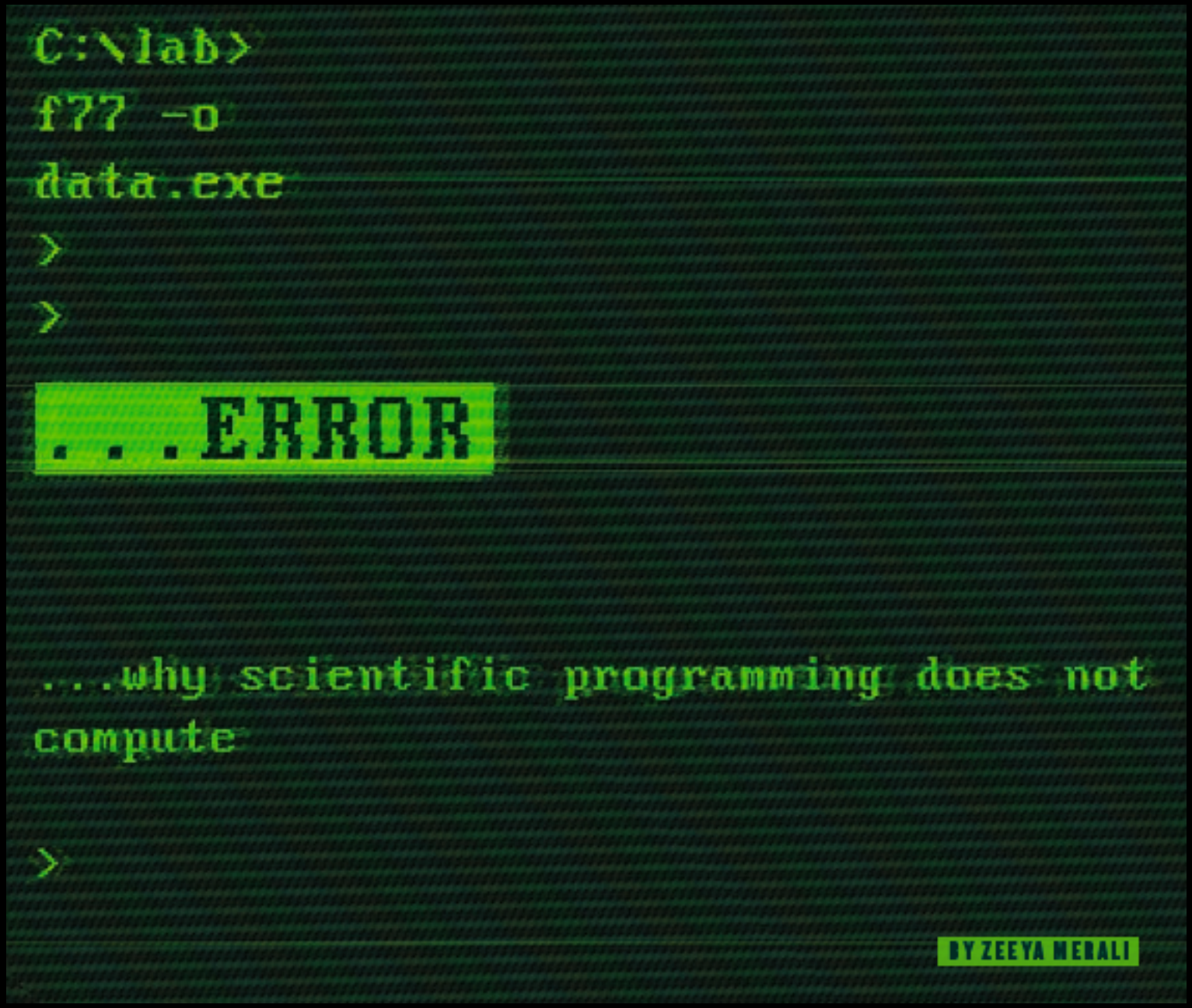
Department of Materials Science and Engineering
Massachusetts Institute of Technology

Heiko Appel, Miguel A. L. Marques, Micael J. T. Oliveira,
Xavier Andrade, Felipe H. da Jornada (UC Berkeley)

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Octopus developers workshop

Jena, 27 January 2015


C:\lab>
f77 -o
data.exe
>
>
....ERROR
....why scientific programming does not
compute
>
BY ZEEYA MERALI

SLAYING THE MONSTER

Problems created by bad documentation are further amplified when successful codes are modified by others to fit new purposes. The result is the bane of many a graduate student or postdoc's life: the 'monster code'. Sometimes decades old, these codes are notoriously messy and become progressively more nightmarish to handle, say computer scientists.

"You do have some successes, but you also end up with a huge stinking heap of software that doesn't work very well," says Darling.

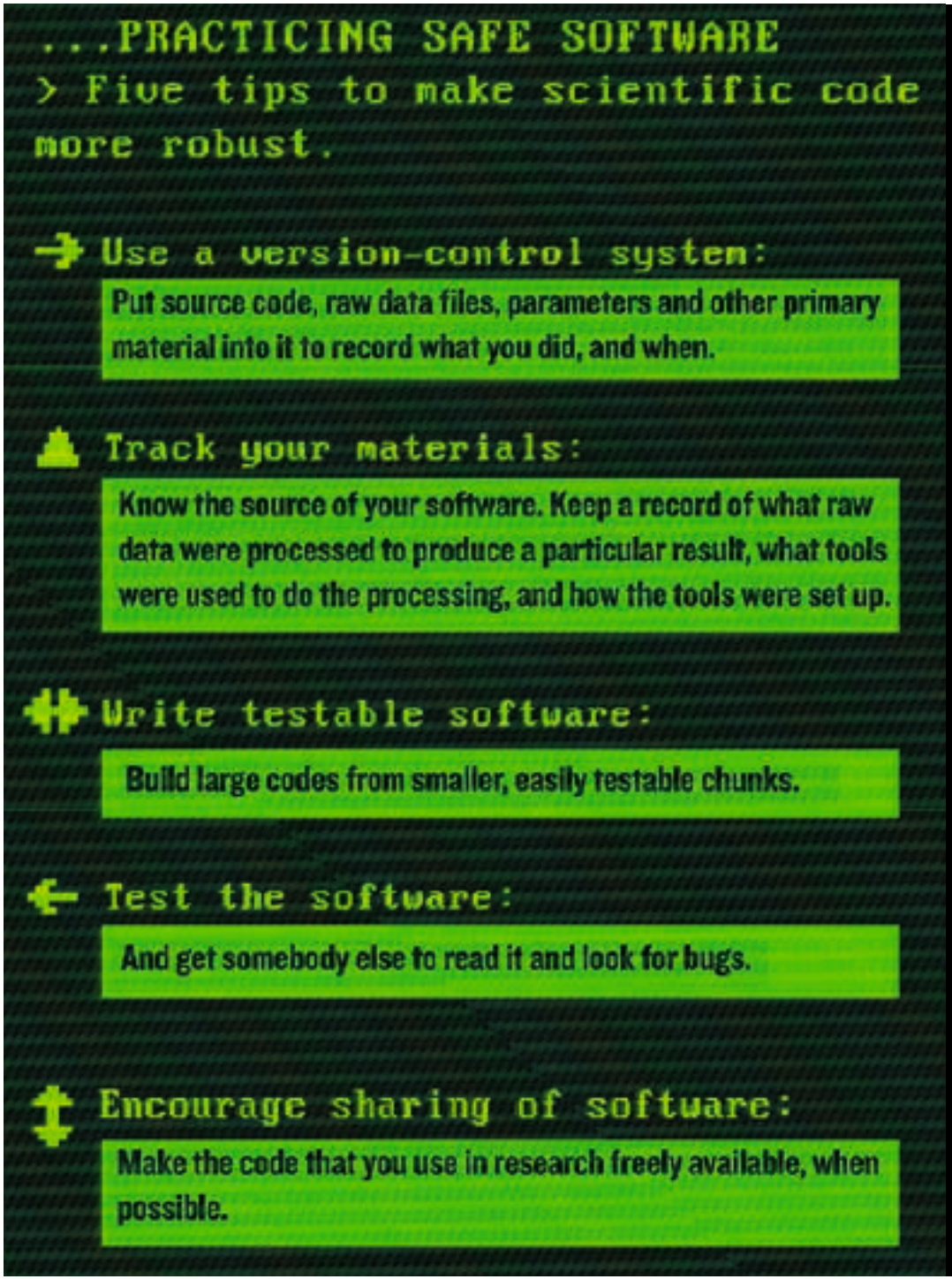
The mangled coding of these monsters can sometimes make it difficult to check for errors.

computer graphics. People can modify the software as they wish, and it is rerun each night on every computing platform that supports it, with the results published on the web. The process ensures that the software will work the same way on different systems.

BuildBot

More-rigorous testing could help. Diane Kelly, a computer scientist at the Royal Military College of Canada in Kingston, Ontario, says the problem is that scientists rely on "validation testing" — looking to see whether the answer that the code produces roughly matches what the scientists expect — and this can miss important errors[2]. The software industry relies on a different approach: breaking codes into manageable chunks and testing each piece individually, then visually inspecting the lines of code that stitch these chunks together (see 'Practicing safe software').

testsuite



SVN repository

Builds and runs identify version and SVN revision

subroutines, modules

testsuite, BuildBot

releases, and having current svn version available

Outline

Motivation

How to run the testsuite

How the testsuite works

Automatic use by BuildBot

Use in development cycle

Format of a test file

Options to testsuite scripts

Use in parallel or with a scheduler

Job parallelism

Creating tests

Making the code testable

Tools: buildbot_query.pl and fix_testsuite.py

Adaptations to APE and BerkeleyGW

Purposes of the testsuite

For developers:

- Prevent unintentional changes (e.g. other parts of code; or when optimizing)
- “Regression” testing (prevent problems from recurring)
- Identify random variation between runs
- Find differences between architectures
- Find differences between compilers
- Find differences between libraries
- Find differences due to compiler optimization levels
- Find differences for different numbers of processors
- Find differences in performance over time, or compare machines
- Quick way to test anything: does this new algorithm give same result?
- With BuildBot can even check machine status; check compiler warnings

For ordinary users: Check success of new build before doing production runs!

Running the testsuite: by hand

```
$ ../oct-run_regression_test.pl  -D ../../bin/ -f 01-quadratic_box.test

***** Quadratic Box *****

Using workdir    : /tmp/octopus.ia3KsY
Using executable : /Users/dstrubbe/Software/octopus_hack/testsuite/finite_systems_2d/../../bin//
octopus
Using test file  : 01-quadratic_box.test


Using input file : ./01-quadratic_box.01-ground_state.inp

Starting test run ...
Executing: cd /tmp/octopus.ia3KsY;  /Users/dstrubbe/Software/octopus_hack/testsuite/
finite_systems_2d/../../bin//octopus > out

** Warning:
**   You have specified a large number of eigensolver iterations ( 250).
**   This is not a good idea as it might slow down convergence, even for
**   independent particles, as subspace diagonalization will not be used
**   often enough.

       Elapsed time:      2.2 s

Finished test run.

 Execution                           :       [   OK   ]
 Eigenvalue                          :       [   OK   ]       (Calculated value = 0.999921)
 Occupation                          :       [   OK   ]       (Calculated value = 1.000000)
```

Or run short tests with `make check`, or all tests with `make check-full`

What the testsuite is doing

`make check` calls `testsuite/oct-run_testsuite.sh`
Finds `.test` files in `testsuite/` in requested group
For each file, call `testsuite/oct-run_regression_test.pl`
Create a temporary working directory (random name `octopus.XXXXXX`)
Copy input file from testsuite directory to working directory
Run code in working directory
Show any warnings or errors from code
Show exit status (not always useful!) and elapsed time
Run match commands on output files
Copy directory to subdirectory, and run next input file for this test
Standard output is saved as `.log` files in `testsuite/` directory
Finally, remove working directory unless test failed or asked to save it
`make clean` removes working directories from `/tmp` if older than 10 hours

```
Passed:  108 / 116
Skipped: 6 / 116
Failed:  2 / 116

testfile                                                    # failed testcases
-------------------------------------------------------------------------------
functionals/06-rdmft.test                                   2
linear_response/01-casida.test                              21


Total run-time of the testsuite: 00:39:38
```

The Buildbot

Widely used open-source testing framework: `www.buildbot.net`
Build master running on tddft.org server
Each SVN commit to trunk or latest branch (e.g. 4.1.x) triggers compile and test
Build slaves: hbar and tigger (Berkeley), lascar (San Sebastián), babbage and mauchly (Coimbra), chum (Louvain-la-Neuve)
Status viewed on website and by emails sent for each failure
Note: hbar is cluster with scheduler; need to use `queue_monitor.pl`

Grid View

Octopus	12700	12701	12702	12703	12704	12705	12706	12707	12708	12709	12710	12711	12712	12713	12714
babbage_x86_64_gfortran		OK	OK	OK		OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
babbage_x86_64_gfortran_dim4		OK	OK	OK		OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
babbage_x86_64_gfortran_dim4_test (waiting)	failed shell_1 shell_4		failed shell_4					failed shell_4				failed shell_4			
babbage_x86_64_gfortran_dist		OK	OK	OK		OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
babbage_x86_64_gfortran_test (waiting)	failed shell_1 shell_4		failed shell_4					failed shell_4				failed shell_4			
chum_x86_64_intel	OK	OK	OK	OK		OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
chum_x86_64_intel_mpich2	OK	OK	failed shell_4			failed shell_4		failed shell_4		failed shell_4		failed shell_4	OK		OK
lascar_x86_64_g95_openmpi	failed shell_4	failed shell_4	failed shell_4	failed shell_4		failed shell_4		failed shell_4		failed shell_4	failed shell_4	failed shell_4	failed shell_4		failed shell_4
lascar_x86_64_gfortran_cl_amd (building)	OK	OK	OK			OK		OK		OK		OK		OK	building

http://www.tddft.org/programs/octopus/buildbot/grid

Use in development cycle

- Absolutely check that code compiles on your machine before commit
- Compile or run on multiple machines if doing something likely to vary (*e.g.* optional libraries, unusual constructs, MPI/serial)
- Run at least some test before making any but the most trivial commits.
- Run tests that relate to any parts you know you are affecting
- Run tests for any part you are concerned you might have affected
- Run whole testsuite when changing core functionalities
- Commit frequently so BuildBot (and community) checks your work
- Before release: run full testsuite on diverse set of machines

- Check for failures after a commit
- Try to fix promptly to keep code working, not inconvenience others, and make sure further problems are noticed if they arise
- If you can’t solve the problem, ask for help
- If necessary, run on the specific machine to reproduce
- If really stuck, revert the change and recommit when you find solution

The test file: parsed by `oct-run_regression_test.pl`

A comment line. Will be ignored.
Test : title
  Write a title to output to identify the test. Should be the first tag in the file.
Programs :
  Which main executable(s) will be used. Should be either `octopus; octopus_mpi` for normal runs or `oct-test; oct-test_mpi` for special test runs. Anything that does not run properly for both serial and parallel executables is not acceptable.
Options :
  Specify that this test can only be run if the code was compiled with certain options, e.g. scalapack, netcdf, etc. The code is run with "-c" argument to query for options.
TestGroups : group-name, [group-name2, [...]]
  The oct-run_testsuite.sh script can be run with argument "-g" and a list of groups. Then tests will only be run if there is a match between the argument list and the list in TestGroups. Current groups: each directory ("components", "finite_systems_1d", etc.), "long-run" (included only "make check-full"), and "short-run" (everything that is not "long-run". [This tag is actually read by oct-run_testsuite.sh rather than oct-run_regression_test.pl.]
Enabled : Yes/No
  If Yes, will be run; if No, will not be run. Use to turn off a test without deleting it from the repository.
Processors : integer
  Number of processors to use. Default is 2. Ignored if mpiexec is not available. Should not be more than 5 under current guidelines.
Precision : 1e-4
  A floating point number, the tolerance for testing whether a match has passed or failed. Persists until next Precision tag. Default is 1e-4.
Util : util-name
  Perform a run of a utility in serial (e.g. oct-dielectric_function).
Not_Util
  Unsets 'Util' and returns to using exectable specified by Programs.
Input : file-name
  Perform a run, after copying the input file to working directory.

The test file: match commands

match ; name ; COMMAND(..); reference-value

Extracts a calculated number from a run and tests it against the reference value. The name is an identifier printed to output. The number is extracted as the standard output from the listed COMMAND, after filtering to remove non-numeric results. The COMMAND is one of this set:

. GREP(filename, 'search-regex', column, offset)

Finds the first line in file containing 'search-regex' (a regular expression passed to the 'grep' command, and which MAY NOT contain a comma), and returns the specified column of that line. The optional 'offset' directs the use of that many lines after the line containing 'search-regex'. No offset specified is equivalent to offset = 0. If there are multiple occurrences of 'search-regex', the first one will be used. This is the most robust of the commands to changes in output formatting, and should be used when possible in preference to the others.

. GREPFIELD(filename, 'search-text', field, offset)

Like GREP except returns the specified field of that line. "Field" is meant in the sense used by 'awk', i.e. the line is parsed into white-space separated groups, indexed starting with 1.

. SIZE(filename)

Returns the size of the specified file via 'ls -lt'. Useful for binary files whose contents cannot easily be checked.

. LINE(filename, line, column)

Returns the specified column of the specified line number from the file. Negative number means from the end of the file. Use GREP instead if possible.

. LINEFIELD(filename, line, field)

Returns the specified field of the specified line number from the file. Negative number means from the end of the file. Use GREPFIELD instead if possible.

. SHELL(shell-command)

The result is the standard output of the listed command. Deprecated; use GREP(FIELD) or LINE(FIELD) unless absolutely necessary.

`oct-run_regression_test.pl` will try to check validity of tags and matches and pass on meaningful error messages.

Options to the testsuite scripts

```
$ ./oct-run_testsuite.sh

 Copyright (C) 2005-2006 by Heiko Appel

Usage: oct-run_testsuite.sh [options]

     -h            this message
     -n            dry-run mode (show what would be executed)
     -g LIST       comma-separated list of test groups to run
     -q            query testfiles for the given groups (no tests are run)
     -d DIR        directory where to look for the testsuite
     -l            local run
     -e SUFFIX     exec suffix for octopus executable
     -p PREFIX     installation prefix [default: /usr]
     -c            delete all .log files and work directories after the run

$ ./oct-run_regression_test.pl

 Copyright (C) 2005-2014 H. Appel, M. Marques, X. Andrade, D. Strubbe

Usage: oct-run_regression_test.pl [options]

    -n        dry-run
    -v        verbose
    -h        this usage
    -D        name of the directory where to look for the executables
    -s        exec suffix for the executables
    -f        filename of testsuite [required]
    -p        preserve working directories
    -l        copy output log to current directory
    -m        run matches only (assumes there are work directories)

Exit codes:
    0         all tests passed
    1..253    number of test failures
    254       test skipped
    255       internal error

$ ../oct-run_regression_test.pl  -D ../../bin/ -f 01-quadratic_box.test
```

Use in parallel or with scheduler

- The runs will use MPI if the code was compiled with MPI, and either `` `which mpiexec` `` or the `$MPIEXEC` environment variable leads to a useable MPI.
- When running with MPI, the tests use up to 5 MPI tasks. Therefore you should typically have at least 4 cores available.
- To run in parallel with a scheduler, *e.g.* on a cluster or supercomputer with PBS, create an appropriate job script for the machine you are using.
- http://www.tddft.org/programs/octopus/wiki/index.php/Manual:Specific_architectures
- Running with scheduler for BuildBot requires use of `queue_monitor.pl` tool
- Example job script for `hopper.nersc.gov`, Cray supercomputer in Berkeley:

```
#!/bin/bash
#PBS -N pulpo
#PBS -m ae
#PBS -q regular
#PBS -l mppwidth=48
#PBS -l walltime=6:00:00

cd $HOME/hopper/octopus
echo $PBS_O_WORKDIR
export MPIEXEC=`which aprun`
export TEMPDIRPATH=$SCRATCH2/tmp
export OCT_TEST_NJOBS=15
make check-full &> $PBS_O_WORKDIR/makecheck-full
```

Job parallelism (OCT_TEST_NJOBS)

Example job script for `hopper.nersc.gov`:

```
…
export OCT_TEST_NJOBS=15
make check-full &> $PBS_O_WORKDIR/makecheck-full
```

```
=============

Starting test 100: real_time/01-propagators.test
In execution: periodic_systems/04-silicon.test periodic_systems/05-lithium.test periodic_systems/
06-h2o_pol_lr.test periodic_systems/07-tb09.test periodic_systems/08-benzene_supercell.test
periodic_systems/09-etsf_io.test periodic_systems/10-berkeleygw.test periodic_systems/11-
silicon_force.test periodic_systems/12-boron_nitride.test photo_electron/01-h1d_lin.test
photo_electron/02-restart.test photo_electron/03-h1d_ati.test photo_electron/04-nfft.test
photo_electron/05-pfft.test real_time/01-propagators.test

 ***** Silicon crystal *****

Using workdir    : /scratch2/scratchdirs/dstrubbe/tmp/octopus.MvPdR5
Using executable : /global/u1/d/dstrubbe/hopper/octopus/testsuite/../src/main/octopus_mpi
Using test file  : ./periodic_systems/04-silicon.test


Using input file : ./periodic_systems/04-silicon.01-gs.inp

Starting test run ...
Executing: cd /scratch2/scratchdirs/dstrubbe/tmp/octopus.MvPdR5;  /usr/bin/aprun -n 2   /global/
u1/d/dstrubbe/hopper/octopus/testsuite/../src/main/octopus_mpi > out

** Warning:
**   SymmetrizeDensity is under development.
**   It might not work or produce wrong results.
```

Creating tests

- All significant parts of code should be covered, including as many 'corner cases' as practical (i.e. spinors + periodic + hybrid).
- 'regression' = When you fix a bug, write a test to prevent recurrence!
- Usually not a realistic calculation, to save time: you can use small numbers of atoms, states, grid points, k-points, etc.
- Unless the goal is specifically to test what happens when one of these parameters is large (`28-carbon_big_box.test`).
- Current runtime: ~ 20 minutes, no test more than a few minutes. Need quick feedback, low barrier to testing, don't time-out buildslaves
- Use multiple input files in one test when runs depend on one another, or to do different things after a common `gs` run
- Keep independent runs separate unless very fast.
- "Whatever is not tested, is broken"!

Creating tests

- Every input file should have match so you know if it went wrong.
- Matches are free but runs are not
- Set precision as low as you can to keep sensitivity, but must pass on all buildslaves and all other machines tested (except where there are known problems!)
- In certain cases, can create stand-alone runs of purely numerical operations in `oct-test` executable (*e.g.* orthogonalization, Hartree)

Making the code testable

- If you can't write a simple match, clarify the output.
- Results must be reproducible; do not allow variation even if it is physically equivalent.
- Test properties that do not depend on wavefunction phases or arbitrary linear combinations in degenerate subspaces. (*e.g.* wfn value, Casida dipole matrix element, GW summation over empty states breaking degenerate subspace)
- Fix seed of random numbers, and make each MPI task produce all the random numbers it would in serial.
- Make sure code does not test equality of floats, to ensure deterministic results. (*e.g.* sorting G-vectors in BerkeleyGW)
- Lower numerical tolerances of solvers and SCF if necessary, but variation in early iterations may be genuine issue to investigate.
- Have enough precision in output so that you can detect differences.

Setting the precision: `testsuite/buildbot_query.pl`

```
Using input file : ../../testsuite/functionals/06-rdmft.02-gs.inp

Starting test run ...
Executing: cd /tmp/octopus.mAzdmR;  /home/octopus/buildbot-slave/
babbage_x86_64_gfortran_test/build/_build/testsuite/../src/main/octopus > out

** Warning:
**   RDMFT theory level is under development.
**   It might not work or produce wrong results.

     Elapsed time:      3.0 s

Finished test run.

 Execution                              :     [   OK   ]

----------------------------------------

Match RDMFT energy after occupation numbers minimization           :

   Calculated value : -1.14361E+00
   Reference value  : -1.13257E+00
   Difference       : 0.0110399999999999
   Tolerance        : 0.0001

 RDMFT energy after occupation numbers minimization           :  [  FAIL  ]

----------------------------------------
```

Could update ref value with: `testsuite/fix_testsuite.py`

Setting the precision: testsuite/buildbot_query.pl

```
$ cat testsuite/buildbot_query.pl
...
# modify these to choose the input file and match you want to search for
$inputfile = "functionals/06-rdmft.02-gs.inp";
$match = "RDMFT energy after occupation numbers minimization";

# options specifying setup for Octopus
$bbpath = "http://www.tddft.org/programs/octopus/buildbot";
$shell_num = 4;
...

$ cat testsuite/functionals/06-rdmft.test
# -*- coding: utf-8 mode: shell-script -*-
# $Id: 06-rdmft.test 12664 2014-12-24 09:00:43Z theophilou $

Test       : RDMFT
Programs   : octopus; octopus_mpi
TestGroups : long-run, functionals
Enabled    : Yes

Processors: 1
# not implemented in parallel
Input: 06-rdmft.01-oep.inp
match ; OEP energy                    ; GREP(static/info, 'Total       =', 20)   ; -1.23959074
match ; OEP eigenvalue                ; GREP(static/info, '  1   --', 13) ;   -0.633895

Input: 06-rdmft.02-gs.inp
match ; RDMFT energy after occupation numbers minimization           ; GREP(out, 'occ minim = ',
30) ;  -1.13257E+00
match ; RDMFT energy after orbital minimization                      ; GREP(out, 'orbital minim =
', 34) ; -1.14698E+00
```

Setting the precision: `testsuite/buildbot_query.pl`

```
URL: http://www.tddft.org/programs/octopus/buildbot
Input file: functionals/06-rdmft.02-gs.inp
Match: RDMFT energy after occupation numbers minimization
…
Builder: babbage_x86_64_gfortran, at svn revision 12711
2015-01-25 11:23:36 URL:http://www.tddft.org/programs/octopus/buildbot/builders/
babbage_x86_64_gfortran/builds/1344/steps/shell_4/logs/stdio [231759] -> "stdio" [1]
Match not found.
…
Builder: hbar_x86_64_pgi_openmpi_test, at svn revision 12711
2015-01-25 11:23:47 URL:http://www.tddft.org/programs/octopus/buildbot/builders/
hbar_x86_64_pgi_openmpi_test/builds/570/steps/shell_4/logs/stdio [579172] ->
"stdio" [1]

   Calculated value : -1.14361E+00
   Reference value  : -1.13257E+00
   Difference       : 0.0110399999999999
   Tolerance        : 0.0001
…
=== SUMMARY ===
Based on 11 matches found.
Minimum   = -1.14361E+00
    (babbage_x86_64_gfortran_test)
Maximum   = -1.14361E+00
    (babbage_x86_64_gfortran_test)
Average   = -1.14361

Center    = -1.14361
Precision = 0.000000e+00
```

So: new reference value and precision (can be zero? write more digits!)

Adaptations for APE and BerkeleyGW

APE has precision set explicitly for every test.

Can automatically alter reference or precision to make the current run pass.

BerkeleyGW has many executables, not just one main one like Octopus, and different input file names.

Many utilities, including scripts in Bash, Perl, Python.

More commands, such as unpack tar.gz archive; copy files from testsuite directory to working directory; vary input files with ‘sed’

Match command with field rather than column (no more counting characters in a line…)

Unification planned soon, so common scripts can be used for all 3 codes, and more!