
testcode Documentation

Release 2.0 (alpha)

James Spencer

May 27, 2012

CONTENTS

testcode is a python module for testing for regression errors in numerical (principally scientific) software. Essentially testcode runs a set of calculations, and compares the output data to that generated by a previous calculation (which is regarded to be “correct”). It is designed to be lightweight and highly portable: it can be used both as part of the development process and to verify the correctness of a binary on a new architecture. testcode requires python 2.4-3.2. If these are not available, then [pypy](#) is recommended—for this purpose pypy serves as a portable, self-contained python implementation but this is a tiny aspect of the pypy project.

testcode can run a set of tests and check the calculated data is within a the desired tolerance of results contained in previous output (using an internal data extraction engine, a user-supplied data extraction program or a user-supplied verification program). The programs to be tested can be run in serial and in parallel and tests can be run in either locally or submitted to a compute cluster running a queueing system such as PBS. Previous tests can be compared and diffed against other tests or benchmarks.

testcode provides access to these features via an API. The supplied command-line interface, [testcode.py](#), should be sufficient for most purposes. The command-line interface utilises simple [configuration files](#), wich makes it easy to customise to the local environment and to add new tests.

INSTALLATION

testcode2 is designed to be very lightweight and portable, so it can easily and quickly be used on a variety of machines. Typically only downloading the testcode2 package is required.

If the *testcode.py* script is used, then no additional installation steps are required assuming the directory structure is preserved. If the `testcode2` module is used or the files are split up and installed elsewhere, then the `testcode2` module must be able to be found by python (i.e. exists on `$PYTHONPATH`).

CONFIGURATION FILES

For convenience, tests can be specified via configuration files rather than using the testcode API directly. These configuration files are required for work with the command-line interface.

The two configuration files are, by default, *jobconfig* and *userconfig* in the working directory. Different names and/or paths can be specified if required.

Both configuration files take options in the ini format (as understood by Python's *configparser* module). For example:

```
[section_1]
a = 2
b = test_option
```

```
[section_2]
v = 4.5
hello = world
```

defines an ini file with two sections (named 'section_1' and 'section_2'), each with two variables set.

JOBCONFIG

The jobconfig file defines the tests to run. If a section named ‘categories’ exists, then it gives labels to sets of tests. All other sections are assumed to individually define a test.

3.1 Tests

A test is assumed to reside in the directory given by the name of the test section. For example:

```
[carbon_dioxide_ccsd]
inputs_args = ('co2.inp', '')
```

would define a test in the `carbon_dioxide_ccsd` subdirectory relative to the `jobconfig` configuration file, with the input file as `co2.inp` (in the `carbon_dioxide_ccsd` subdirectory) with no additional arguments to be passed to the test program. All input and output files related to the test are assumed to be contained within the test subdirectory.

The following options are permitted:

inputs_args [inputs and arguments format (see [below](#))] Input filename and associated arguments to be passed to the test program. No default.

nprocs [integer] Number of processors to run the test on. Zero indicates to run the test purely in serial, without using an external program such as `mpirun` to launch the test program. Default: 0.

output [string] Filename to which the output is written if the output is not written to standard output. The output file is moved to the specific testcode test filename at the end of the calculation before the test output is validated against the benchmark output. Wildcards are allowed so long as the pattern only matches a single file at the end of the calculation. Default: inherits from setting in [userconfig](#).

override_nprocs [boolean] True if the number of processors to run the test cannot be overridden by command-line options to [testcode.py](#). Useful to force certain tests to be executed on a given number of processors. Default: false.

test_program [string] Program name (appropriate section heading in [userconfig](#)) to use to run the test. Default: specified in the [user] section of [userconfig](#).

tolerance [tolerance format (see [Tolerance format](#))] Tolerances for comparing test output to the benchmark output. Default: inherits from the settings in [userconfig](#).

3.2 Test categories

Each test is automatically defined to reside in a category of the same name. Additional categories can be specified in the [categories] section. This makes it very easy to select subsets of the tests to run. For example:

```
[categories]
cat1 = t1 t2
cat2 = t3 t4
cat3 = cat1 t3
```

defines three categories (*cat*, *cat2* and *cat3*), each containing a subset of the overall tests. A category may contain another category so long as circular dependencies are avoided. There are two special categories, *_all_* and *_default_*. The *_all_* category contains, by default, all tests and should not be changed under any circumstances. The *_default_* category can be set; if it is not specified then it is set to be the *_all_* category.

3.3 Program inputs and arguments

The inputs and arguments must be given in a specific format. As with the *tolerance format*, the inputs and arguments are specified using a comma-separated list of python tuples. Each tuple (basically a comma-separated list enclosed in parantheses) contains two elements: the name of an input file and the associated arguments, in that order, represents a test. Both elements must be quoted. If the input filename contains wildcard, then those wildcards are expanded to find all files in the test subdirectory which match that pattern. A separate test (with the same arguments string) is then created for each file matching the pattern. used to construct the command to run A null string (") should be used to represent the absence of an input file or arguments. Tests within the same subdirectory are run in the order they are specified. For example:

```
inputs_args = ('test.inp', '')
```

defines a single test, with input filename `test.inp` and no arguments,

```
inputs_args = ('test.inp', ''), ('test2.inp', '--verbose')
```

defines two tests, with an additional argument for the second test, and

```
inputs_args = ('test*.inp', '')
```

defines a test for each file matching the pattern `test*.inp` in the test subdirectory.

USERCONFIG

The userconfig file must contain at least two sections. One section must be entitled ‘user’ and contains various user settings. Any other section is assumed to define a program to be tested, where the program is referred to internally by its section name. This makes it possible for a set of tests to cover multiple, heavily intertwined, programs. It is, however, far better to have a distinct set of tests for each program where possible.

4.1 [user] section

The following options are allowed in the [user] section:

benchmark [string] Specify the ID of the benchmark to compare to. This should be set running

The format of the benchmark files is ‘benchmark.out.ID.inp=INPUT_FILE.arg=ARGS’. The ‘inp’ and/or ‘arg’ section is not included if it is empty.

date_fmt [string] Format of the date string used to uniquely label test outputs. This must be a valid date format string (see *Python documentation*). Default: %d%m%Y.

default_program [string] Default program used to run each test. Only needs to be set if multiple program sections are specified. No default.

diff [string] Program used to diff test and benchmark outputs. Default: diff.

tolerance [tolerance format (see [below](#).)] Default tolerance(s) used to compare all tests to their respective benchmarks. Default: absolute tolerance 10^{-10} ; no relative tolerance set.

4.2 [program_name] section(s)

The following options are allowed to specify a program (called ‘program_name’) to be tested:

data_tag [string] Data tag to be used to extract data from test and benchmark output. No default.

ignore_fields [space-separated list of strings] Specify the fields (e.g. column headings in the output from the extraction program) to ignore. This can be used to include, say, timing information in the test output for performance comparison without causing failure of tests. No default.

exe [string] Path to the program executable. No default.

extract_args [string] Arguments to supply to the extraction program. Default: null string.

extract_cmd_template [string] Template of command used to extract data from output(s) with the following substitutions made:

tc.extract replaced with the extraction program.

tc.args replaced with `extract_args`.

tc.file replaced with (as required) the filename of the test output or the filename of the benchmark output.

tc.bench replaced with the filename of the benchmark output.

tc.test replaced with the filename of the test output.

Default: `tc.extract tc.args tc.file` if `verify` is `False` and `tc.extract tc.args tc.test tc.bench` if `verify` is `True`.

extract_program [string] Path to program to use to extract data from test and benchmark output. No default.

launch_parallel [string] Command used to run the test program in parallel. Default: `mpirun`.

run_cmd_template [string] Template of command used to run the program on the test with the following substitutions made:

tc.program replaced with the program to be tested.

tc.args replaced with the arguments of the test.

tc.input replaced with the input filename of the test.

tc.output replaced with the filename for the standard output. The filename is selected at runtime.

tc.error replaced with the filename for the error output. The filename is selected at runtime.

Default: `'tc.program tc.args tc.input > tc.output 2> tc.error'` in serial and `'launch_command -np nprocs tc.program tc.args tc.input > tc.output 2> tc.error'` in parallel, where `launch_command` is specified above and `nprocs` is the number of processors to run the test on. The parallel version is only used if the number of processors to run a test on is greater than zero.

submit_pattern [string] String in the submit to be replaced by the run command. Default: `testcode.run_cmd`.

submit_template [string] Path to a template of a submit script used to submit jobs to a queueing system. `testcode` will replace the string given in `submit_pattern` with the command(s) to run the test. The submit script must do all other actions (e.g. setting environment variables, loading modules, copying files from the test directory to a local disk and copying files back afterwards). No default.

tolerance [tolerance format (see [below](#).)] Default tolerance for tests of this type. Default: inherits from [user].

verify [boolean] True if the extraction program compares the benchmark and test outputs directly. See *verification* for more details. Default: `False`.

vcs [string] Version control system used for the source code. This is used to label the benchmarks. The program binary is assumed to be in the same directory tree as the source code. Supported values are: `hg`, `git` and `svn` and `None`. If `vcs` is set to `None`, then the version id of the program is requested interactively when benchmarks are produced. Default: `None`.

Most settings are optional and need only be set if certain functionality is required or the default is not appropriate. Note that either `data_tag` or `extract_program` must be supplied.

In addition, the following variables are used, if present, as default settings for all tests of this type:

- `inputs_args` (no default)
- `nprocs` (default: 0)
- `output` (no default)

See [jobconfig](#) for more details.

All other settings are assumed to be paths to other versions of the program (e.g. a stable version). Using one of these versions instead of the one listed under the `'exe'` variable can be selected by an option to [testcode.py](#).

4.3 Tolerance format

The format for the tolerance for the data is very specific. Individual tolerance elements are specified in a comma-separated list. Each individual tolerance element is a python tuple (essentially a comma-separated list enclosed in parentheses) consisting of, in order, the absolute tolerance, the relative tolerance and the label of the field to which the tolerances apply. The labels must be quoted. If no label is supplied then the setting is taken to be the default tolerance to be applied to all data. For example, the setting:

```
(1e-8, 1.e-6), (1.e-4, 1.e-4, 'Force')
```

uses an absolute tolerance of 10^{-8} and a relative tolerance of 10^{-6} by default and an absolute tolerance and a relative tolerance of 10^{-4} for data items labelled with 'Force' (i.e. in columns headed by 'Force' using an external data extraction program or labelled 'Force' by the internal data extraction program using data tags).

TEST VERIFICATION

testcode compares selected data from an output with previously obtained output (the ‘benchmark’); a test passes if all data is within a desired tolerance. The data can be compared using an absolute tolerance and/or a relative tolerance. testcode needs some way of knowing what data from the output files should be validated. There are three options.

- label output with a ‘data tag’

If a data tag is supplied, then testcode will search each output file for lines starting with that tag. The first numerical entry on those lines will then be checked against the benchmark. For example, if the data tag is set to be ‘[QA]’, and the line

[QA] Energy = 1.23456 eV

appears in the test output, then testcode will ensure the value 1.23456 is identical (within the specified tolerance) to the equivalent line in the benchmark output. The text preceding the value is used to label that data item; lines with identical text but different values are handled but it is assumed that such lines always come in the same (relative) order.

- user-supplied data extraction program

An external program can be used to extract data from the test and benchmark output; the program must print the data to be compared in an output file in a tabular format to standard output. A row of text is assumed to start a table. Multiple tables are permitted, but each table must be square (i.e. no gaps and the same number of elements on each row) and hence each column heading must contain no spaces. For example, a single table is of the format:

```
val_1  val_2  val3
1.2    2      3.32
8.7    4      17.2
```

and a table containing multiple subtables:

```
val_1  val_2  val3
1.2    2      3.32
8.7    4      17.2
val_4  val_5
11.22  221.0
```

Tables need not be beautifully presented: the amount of whitespace between each table cell is not important, so long as there’s at least one space separating adjacent cells.

Column headings are used to label the data in the subsequent rows. These labels can be used to specify different tolerances for different types of data.

Non-numerical values in a table are required to be equal (within python's definition of equality for a given object).

- user-supplied verification program

An external program can be used to validate the test output; the program must set an exit status of 0 to indicate the test passed and a non-zero value to indicate failure.

TESTCODE.PY

6.1 Synopsis

testcode.py [options] [action1 [action2...]]

6.2 Description

Run a set of actions on a set of tests.

Requires two configuration files, *jobconfig* and *userconfig*. See testcode documentation for further details.

testcode.py provides a command-line interface to testcode, a simple framework for comparing output from (principally numeric) programs to previous output to reveal regression errors or miscompilation.

6.3 Actions

“run” is th default action.

compare compare set of test outputs from a previous testcode run against the benchmark outputs.

compare diff set of test outputs from a previous testcode run against the benchmark outputs.

make-benchmarks create a new set of benchmarks and update the *userconfig* file with the new benchmark id. Also runs the ‘run’ action unless the ‘compare’ action is also given.

run run a set of tests and compare against the benchmark outputs.

tidy Remove files from previous testcode runs from the test directories.

6.4 Options

-h, --help show this help message and exit

-b BENCHMARK, --benchmark=BENCHMARK Set the file ID of the benchmark files. If BENCHMARK is in the format t:ID, then the test files with the corresponding ID are used. This allows two sets of tests to be compared. Default: specified in the [user] section of the *userconfig* file.

- c CATEGORY, --category=CATEGORY** Select the category/group of tests. Can be specified multiple times. Default: use the *_default_* category if run is an action unless make-benchmarks is an action. All other cases use the *_all_* category by default. The *_default_* category contains all tests unless otherwise set in the *jobconfig* file.
- e EXECUTABLE, --executable=EXECUTABLE** Set the executable(s) to be used to run the tests. Can be a path or name of an option in the *userconfig* file, in which case all test programs are set to use that value, or in the format *program_name=value*, which affects only the specified program. Only relevant to the run action. Default: exe variable set for each program listed in the *userconfig* file.
- jobconfig=JOBCONFIG** Set path to the job configuration file. Default: *jobconfig*.
- job-option=JOB_OPTION** Override/add setting to *jobconfig*. Takes three arguments. Format: *section_name option_name value*. Default: none.
- n NTHREADS, --nthreads=NTHREADS** Set the number of tests to run concurrently. Only relevant to the run action. Default: 1.
- older-than=OLDER_THAN** Set the age (in days) of files to remove. Only relevant to the tidy action. Default: 14 days.
- p NPROCS, --processors=NPROCS** Set the number of processors to run each test on. Only relevant to the run action. Default: run tests as serial jobs.
- q, --quiet** Print only minimal output. Default: False.
- s QUEUE_SYSTEM, --submit=QUEUE_SYSTEM** Submit tests to a queueing system of the specified type. Only PBS system is currently implemented. Only relevant to the run action. Default: none.
- t TEST_ID, --test-id=TEST_ID** Set the file ID of the test outputs. If TEST_ID is in the format *b:ID*, then the benchmark files with the corresponding ID are used. This allows two sets of benchmarks to be compared. Default: unique filename based upon date if running tests and most recent test_id if comparing tests.
- userconfig=USERCONFIG** Set path to the user configuration file. Default: *userconfig*.
- user-option=USER_OPTION** Override/add setting to *userconfig*. Takes three arguments. Format: *section_name option_name value*. Default: none.

6.5 License

Modified BSD License. See LICENSE in the source code for more details.

6.6 Bugs

Contact James Spencer (j.spencer@imperial.ac.uk) regarding bug reports, suggestions for improvements or code contributions.

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*