nrtest Documentation

Release 0.2.5

David Hall

Aug 16, 2021

Contents

Contents:

CHAPTER 1

nrtest

nrtest is an end-to-end regression testing framework, designed for scientific software that perform numerical calculations.

1.1 Features

nrtest aims to simplify your workflow:

- JSON files describe the software under test and the tests themselves
- result files are stored in a portable benchmark directory
- benchmarks are compared by iterating through tests and their results
- custom comparisons can easily be added through extensions

1.2 Basic Usage

As an example usage, we consider testing TOPAS. This is a Monte Carlo tool for particle simulation, designed for medical physics research. Of course, such a tool must be rigorously validated against experimental data. But it is also useful to frequently run shorter tests, checking for regressions by comparing results to a previous version.

First, we describe the software under test in a configuration file called apps/topas-2.0.3.json. Note that setup_script defines the environment needed to run the software.

```
{
    "name": "topas",
    "version" : "2.0.3",
    "setup_script" : "/path/to/topas-2.0.3/setup.sh",
    "exe" : "topas"
}
```

We then describe the test in second configuration file called `tests/Scoring_01.json`. In doing so, we define the command-line arguments presented to the executable and the input files needed for the test to run. Finally, we also specify the expected output files, and declare how they should be compared to a benchmark. Here we use `topas binned`, which is a custom comparison routine, though some comparison routines are bundled with nrtest. It is also easy to add your own.

```
{
    "name": "Scoring_01",
    "version": "1.0",
    "description": "Basic test shooting a 6cm diameter proton beam into a water␣
↪phantom.",
    "args": [
        "Scoring_01.txt"
    ],
    "input_files": [
        "Scoring_01.txt",
        "GlobalParameters.txt"
    ],
    "output_files": {
        "Dose.csv": "topas binned"
    }
}
```

To execute the test, we tell `nrtest` where to find the configuration files and where to output the benchmark. Note that `nrtest` will search `tests/` for tests, though we could have specified `tests/Scoring_01.json`.

```
$ nrtest execute apps/topas-2.0.3.json tests/ -o benchmarks/2.0.3
INFO: Found 1 tests
Scoring_01: pass
INFO: Finished
```

To compare to a previous benchmark:

```
$ nrtest compare benchmarks/2.0.3 benchmarks/2.0.2
Scoring_01: pass
INFO: Finished
```

More advanced usage is detailed in the documentation.

CHAPTER 2

User Documentation

2.1 Installation

The easiest way to install a stable release is using pip:

```
$ pip install nrtest
```

or easy_install:

```
$ easy_install nrtest
```

This will automatically fetch the package from PyPI and install it.

2.2 Usage

The package provides an `nrtest` script with two subcommands: **execute** and **compare**.

2.2.1 Execute

The minimal arguments required to execute a test are:

```
$ nrtest execute /path/to/software.json /path/to/test.json
```

The configuration files for the *software under test* and the *test itself* are documented elsewhere.

Results files and information about the test outcome are stored within a portable benchmark directory. By default, this is `benchmarks/new` but this can be specified:

```
$ nrtest execute /path/to/software.json /path/to/test.json -o benchmarks/v2.0
```

It is also possible to specify a directory containing multiple test configuration files instead of a single test:

```
$ nrtest execute /path/to/software.json /path/to/tests
```

2.2.2 Compare

The minimal arguments required to compare the newly created benchmark to a reference benchmark are:

```
$ nrtest compare benchmarks/new benchmarks/old
```

This will iterate through each of the tests contained in the benchmark and then compare each result file to its respective reference file. The type of comparison performed is determined by the output file type specified in the test configuration file (see *Result comparison*).

Comparisons of numerical result files might compare the difference to a tolerance in order to decide if the results are compatible. Relative and absolute tolerances can be specified:

```
$ nrtest compare benchmarks/new benchmarks/old --rtol=0.01 --atol=0.0
```

These are the default tolerances.

It is also possible to output the results of the comparison to a JSON file:

```
$ nrtest compare benchmarks/new benchmarks/old -o receipt.json
```

which can be helpful to post-process the comparison (e.g. display in dashboard, email notification).

2.3 Configuration: software

Metadata about the system under test is stored in a JSON configuration file. See *Basic Usage* for an example of the syntax.

The following fields can be used to describe the software under test.

2.3.1 Mandatory fields

name ***[string]*** Name of the software.

version ***[string]*** Version of the software.

exe ***[string]*** The executable name. This can also be a path.

2.3.2 Optional fields

description ***[string]*** A short description to help identification of this version.

setup_script ***[string]*** Path to a bash script that shall be sourced in order to create the environment needed to run the software.

timeout ***[float]*** The period in time [seconds] after which a test will be terminated and considered failed.

2.4 Configuration: test

Metadata about each test is stored in a JSON configuration file. See *Basic Usage* for an example of the syntax.

The following fields can be used to describe a single test.

2.4.1 Mandatory fields

name ***[string]*** Name of the test.

version ***[string]*** Version of the test.

args ***[list of strings]*** A list of command-line arguments that will be passed to the software under test.

2.4.2 Optional fields

description ***[string]*** A short description to help identification of this version.

minimum_app_version ***[string]*** The minimum software version required for the test to be executed (see *Configuration: software*). If the software under test does not satisfy this requirement, then the test is removed from the test suite before execution. This allows you to run the latest test suite on old software without test failures.

input_files ***[list of strings]*** A list of required input files. Each path is specified relative to the location of the configuration file itself.

output_files ***[dict of string-string pairs]*** A list of expected output files. The key is a path to the output file, relative to the working directory when the test is run. The value identifies the file type, which determines how it shall be compared to a benchmark (see *Result comparison*).

fail_strings ***[list of strings]*** If any of these strings are found in the stdout or stderr streams, the test is considered failed.

2.5 Result comparison

The routine used to compare a result file to its respective benchmark is determined by its file type. This is specified in the `output_files` field of the test, e.g. `"output_files": {"image.jpg": "default"}`.

Currently, the following output file types are bundled with nrtest:

default This is equivalent to the `diff` command-line utility

null No comparison is performed (used when a difference should not indicate a test failure).

If there are other comparison routines that are widely applicable I would be very happy to bundle these with nrtest too.

For more specialized comparisons, it is also possible to add custom comparison routines via extensions.

2.5.1 Custom comparison routines

An example of how to add custom comparison routines can be found in the nrtest-topas repository. This is a good resource when writing your own extensions.

Comparison functions must be written in Python and must have the following signature:

```
def xxx_compare(path_test, path_ref, rtol, atol):

    if compatible:
        return True
    else:
        return False
```

where path_test and path_ref are the paths to the results files produced by the software under test and the reference version, respectively. The relative tolerance rtol and the absolute tolerance atol are set at the command-line (see *Usage*), and should follow these interpretations in order to remain consistent with other comparisons.

In order to register the custom comparison function with nrtest, we pass entry_points to setuptools in our setup.py file. The syntax is

```
entry_points = {
    'nrtest.compare': [
        '<file type>=<module.path>:<function_name>'
    ]
}
```

where <file type> is the string that will be used in the configuration file to signify the use of this comparison function, <module.path> is the dotted module path where the comparison function is found, and <function_name> is xxx_compare in this case. More details can be found here.

CHAPTER 3

History

3.1 0.2.5 (2021-08-10)

- Add ability to skip tests based on *minimum_app_version* config

3.2 0.2.4 (2019-09-11)

- Fix sporadic failures on Windows when deleting temporary files

3.3 0.2.3 (2018-02-09)

- Add ability to write comparison results to JSON file

3.4 0.2.0 (2016-02-21)

- Add extensions for compare functions

3.5 0.1.0 (2016-02-21)

- First release on PyPI.